Reactive Policies with Planning for Action Languages

Zeynep G. Saribatur and Thomas Eiter *

Technische Universität Wien, Vienna, Austria,
{zeynep,eiter}@kr.tuwien.ac.at

Abstract. Action languages are an important family of formalisms to represent action domains in a declarative manner and to reason about them. For this reason, the behavior of an agent in an environment may be governed by policies which take such action domain descriptions into account. In this paper, we describe a formal semantics for describing policies that express a reactive behavior for an agent, and connect our framework with the representation power of action languages. In this framework, we mitigate the large state spaces by employing the notion of indistinguishability, and combine components that are efficient for describing reactivity such as target establishment and (online) planning. Our representation allows one to analyze the flow of executing the given reactive policy, and lays foundations for verifying properties of policies. Additionally, the flexibility of the representation opens a range of possibilities for designing behaviors.

1 Introduction

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, which describes their capabilities and represents the world's model. After executing these 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.

For example, 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. 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.

As action languages [18] are a convenient tool to describe dynamic systems, one can use them in representing reactive agents and defining reactive policies. However,

* This work has been supported by Austrian Science Fund (FWF) project W1255-N23.

the shortage of representations of reactive policies using action languages with formal semantics prevents us from verifying such policies before putting them into use. We thus aim for a general model that allows for verifying the reactive behavior of agents. In that model, we want to use the representation power of the transition systems described by action languages and combine components that are efficient for describing reactivity.

We consider in this paper 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. This method matches the observe-think-act cycle in [20], but involves a planner that considers targets. The flexibility in the two components—target development and external planning—allows for a range of possibilities for designing behaviors. For example, one can use HEX [15] to describe a program that determines a target given the current agent state, and finds a suitable plan and execution schedule. ACTHEX programs [17], in particular, are a tool to define such reactive behaviors by allowing iterative program evaluation. Specifically, we make the following contributions:
(1) We introduce a novel framework for describing the semantics of a policy that follows a reactive behavior, by integrating components of target establishment and online planning. Our aim is not synthesis, but to lay foundations for verification of behaviors of (human-designed) reactive policies. The outsourced planning might also lend itself for modular, hierarchic planning, where macro actions (as targets) are turned into a plan of micro actions. Furthermore, outsourced planning may also be exploited to abstract from correct sub-behaviors (e.g., going always to the farthest point).
(2) We employ the notion of indistinguishable states and cluster states to reduce the large state spaces by omitting information irrelevant to the agent's behavior.
(3) We discuss complexity issues regarding the representation and show that verifying policy correctness over this framework is in PSPACE (with matching hardness instances).
(4) We connect the framework with action languages and discuss possibilities for policy formulation. In particular, we consider the action language $\mathcal{C}$ [19] for an application.

We proceed as follows. After some preliminaries in Section 2, we present a running example in Section 3. In Section 4, we introduce the general framework for modeling policies. Then, in Section 5, we show the relation with action languages. After some discussion and considering related work in Section 6, we conclude in Section 7 with issues for ongoing and future work. Throughout the paper, we consider (a fragment of) the action language $\mathcal{C}$ as a particular application, and provide example formulations.

2 Preliminaries

We define state transition systems as follows.

Definition 1. *An* (original) transition system *is a tuple* $\mathcal{T} = \langle S, S_0, \mathcal{A}, \Phi \rangle$ *where*

- S *is the finite set of states,*
- $S_0 \subseteq S$ *is the (finite) set of possible initial states,*
- $\mathcal{A}$ *is the finite set of possible actions, and*
- $\Phi : S \times \mathcal{A} \to 2^S$ *is the transition function, which returns the set of possible successor states after applying a possible action in the current state.*

For any states $s, s' \in S$, we say that there is a trajectory *between s and s', denoted by $s \rightarrow^{\sigma} s'$ for some action sequence $\sigma = \langle a_1, \ldots, a_n \rangle$ where $n \geq 0$, if there exist $s_0, \ldots, s_n \in S$ such that $s = s_0, s' = s_n$ and $s_{i+1} \in \Phi(s_i, a_{i+1})$ for all $0 \leq i < n$.*

If knowing the actions taken in the transitions is not necessary, then one can *project away* the actions and consider the transition function as $\Phi : S \rightarrow 2^S$, which returns the set of successor states after applying some action.

Action Languages Rooted in the work in knowledge representation, action languages [18] describe a particular type of transition systems that are based on action signatures. An *action signature* consists of a set **V** of value names, a set **F** of fluent names and a set **A** of action names. Any *fluent* has a *value* in any *state of the world*.

A transition system of an action signature $\langle \mathbf{V}, \mathbf{F}, \mathbf{A} \rangle$ is similar to Definition. 1, where $\mathcal{A} = \mathbf{A}$ and $\Phi \subseteq S \times \mathbf{A} \times S$ is the transition relation. In addition, we have a value function $V : \mathbf{F} \times S \rightarrow \mathbf{V}$, where $V(P, s)$ shows the *value of* P in state s. A transition system can be thought as a labeled directed graph, where a state s is represented by a vertex labeled with $P \rightarrow V(P, s)$, that gives the value of the fluents. Every triple $\langle s, a, s' \rangle \in \Phi$ is represented by an edge leading from a state s to s' and labeled by a.

An action a is *executable* at a state s, if there is at least one state s' such that $\langle s, a, s' \rangle \in R$ and a is *deterministic* at state s, if there is at most one such state. Concurrent execution of actions can be defined by considering transitions $\langle s, A, s' \rangle$ with a set $A \subseteq \mathbf{A}$ of actions, where each action $a \in A$ is executable at s. Here we confine to *propositional* action signatures, which have truth values as value names, $\mathbf{V}=\{\mathtt{f}, \mathtt{t}\}$.

The transition system allows one to answer queries about the domain description. For example, one can find a plan to reach a goal state from an initial state, by searching for a path between the respective vertices. The properties about the paths can be expressed using an action query language.

The action language $\mathcal{C}$ [19] is based on *causality*, where one distinguishes the cases that a fact "holds" and that it is "caused". Its syntax consists of static and dynamic laws of the form

$$\textbf{caused } F \textbf{ if } G,$$
$$\textbf{caused } F \textbf{ if } G \textbf{ after } U$$

respectively, where F and G are formulas of fluents, and U is a formula containing fluents and elementary actions. For details, see [19, 18]. We focus on a fragment of the language $\mathcal{C}$ where the heads of the static and dynamic laws only consist of literals. This restriction on the laws reduces the cost of evaluating the transitions to polynomial time.

3 Running Example: Search Scenarios

Consider a memoryless agent that can sense horizontally and vertically, in an unknown $n \times n$ grid cell environment with obstacles, where a missing person needs to be found. Suppose we are given a policy of "always go to the farthest reachable point in visible distance (until a person is found)". Following this policy, the agent would determine a *target* (i.e., the farthest point) at its current state, compute the course of actions to reach the target, execute it and observe the outcomes.

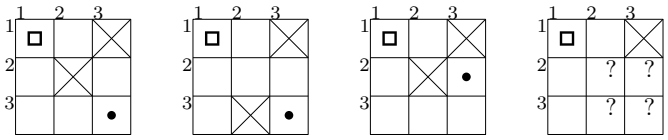

Fig. 1: (a),(b),(c): Possible instances of a search scenario in a grid-cell environment, (d): Agent's observation in the instances, □:agent, •:person, ×:obstacle, ?:unknown

Target determination at the states according to the given policy can be done using a logic program as shown below.

$$\begin{aligned} &\mathit{targetCell}(X1,\, Y1) \leftarrow \mathit{farthest}(X,\, Y,\, X1,\, Y1), \mathit{robotAt}(X,\, Y), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{not}\ \mathit{personDetected}. \\ &\mathit{personDetected} \leftarrow \mathit{personDetected}(X,\, Y). \\ &\mathit{targetPerson}(X,\, Y) \leftarrow \mathit{personDetected}(X,\, Y). \\ &\mathit{personFound} \leftarrow \mathit{personDetected}(X,\, Y), \mathit{robotAt}(X,\, Y). \end{aligned} \tag{1}$$

The target of a state can be computed through joint evaluation of these rules over the state with the *known/observed* fluents about the agent's location and the reachable points. The target can either be moving to the farthest cell, $\mathit{targetCell}(X1,\, Y1)$, if the person is not detected, or moving to the cell of the person, $\mathit{targetPerson}(X,\, Y)$, if the person is detected. Then, an outsourced planner can be used to determine the course of actions from the agent's current location to the target location.

Given such a policy, it needs to be checked whether or not the agent can always find the person, in all instances of the environment. Note that we assume that the obstacles are always placed in a way that the person is reachable.

Figures 1(a) to 1(c) show some instances for n=3 to demonstrate that the given policy might not always work. Firstly, notice that these initial states provide the same observations for the agent, which is shown in Figure 1(d), since it can only observe horizontally and vertically. In these states, the agent only sees that the first column is clear of obstacles, and the first row has one obstacle. Since the rest of the environment can not be observed, these states are *indistinguishable* to the agent.

The farthest reachable point in these states is (3,1), which is determined as the target. Then the policy computes the course of actions to reach this target. Clearly, in Figure 1(a) the person will be found when moved to (3,1). However, in Figure 1(b) after reaching (3,1), the agent/policy will decide to move to (1,1) again, which results in a loop. Also, in Figure 1(c), after reaching (3,1), the agent/policy can either choose to move to (3,3) (which results in seeing the person), or to move back to (1,1). So there is a possibility for the agent to go in a loop. Hence, the policy does not work for the last two instances.

4 Modeling Policies in Transition Systems

We consider a general notion of a policy, that guides the agent by setting up targets and determining the course of actions to bring about these targets, and describe how such a policy can be represented with transition systems.

Definition 2. *A* policy *is a function $\mathcal{P}_{g_\infty,KB} : S \rightarrow 2^{\Sigma}$ 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 define a transition system that shows the policy execution, while also employing the notion of *indistinguishability* to do state clustering. The determination of targets for a given state is done by a *target component*, while the (higher level) transition between states is determined by the course of actions computed by a *(online) planner component*.

Having a classification on states and defining higher level transitions helps in reducing the state space/the number of transitions. Furthermore, it aids in abstraction and allows one to emulate a modular hierarchic approach, in which a higher level (macro) action, expressed by a target, is realized by a sequence of (micro) actions that is compiled by the external planner, which may use different ways (planning on the fly, using scripts etc.)

4.1 State Profiles According to the Policy

Large state spaces are a major issue for the (original) transition system when dealing with large environments. However, depending on the agent's designed behavior, and its determination of its course of actions at a state, some information in the state may not be necessary, relevant or even observable. In this sense, the states that contain different facts about such information can be seen as *indistinguishable* to the agent. Such indistinguishable states can be clustered into one with respect to the *profiles* they provide and only the relevant information to the agent/policy can be kept.

Definition 3. *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.*

Note that the agent has the capability to gain knowledge, and this knowledge can eventually become relevant to the policy. So it would be useful to keep such potentially relevant knowledge in the states to pass on to the successor states even though this knowledge might not be currently relevant to the policy. Therefore the profile scheme consists of all attributes that may be relevant to the policy. A profile at a state consists of values of attributes that are partitioned as *currently relevant*, *irrelevant* and *not yet observed*, depending on the observability of the environment and the policy. Currently relevant attributes at a state can be regarded as the *active profile*.

Example 1. Reconsider Figure 1. Due to partial observability, the agent is unable to distinguish its state, and the policy does not consider the unobservable parts. The agent's observation, "$robotAt(1,1)$, $obstacleAt(1,3)$, $reachable(1,2)$, $reachable(2,1)$, $reachable(3,1)$" that is *currently relevant* and the rest of the environment that is *not yet observed*, is viewed as a profile, and the states with this profile can be clustered in one group (Fig. 1(d)).

The profile of a state is determined by evaluating a set of formulas that yield the attribute values. We consider a *classification function*, $h : S \rightarrow \Omega_h$, 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. However, in fully observable environments, observability is not of concern. One needs to check the policy to determine profiles.

Definition 4. *An* equalized state *relative to the classification function h is a state $\hat{s} \in \Omega_h$.*

The term *equalized* comes from the fact that the states in the same cluster are considered as the same, i.e., equal. We abuse the notation $s \in \hat{s}$ when talking about a state s that is clustered into an equalized state $\hat{s}$, and identify $\hat{s}$ with its pre-image (i.e., the set of states that are mapped to $\hat{s}$ according to h).

4.2 Transition Systems According to the Policy

We now define the notion of a transition system that is able to represent the evaluation of the policy on the state clusters.

Given a set of equalized states $\widehat{S}$, for an equalized state $\hat{s} \in \widehat{S}$, the policy $P_{g_\infty,KB}$ uses a *target function* $\mathcal{B}(\hat{s})$ to determine a target g_B from a *set of possible targets*, G_B, and then an *outsourced planner* $Reach(\hat{s}, g_B)$ to compute a plan to reach the target from the current equalized state, i.e., $P_{g_\infty,KB}(\hat{s}) = \{\sigma \mid \sigma \in Reach(\hat{s}, g_B), g_B \in \mathcal{B}(\hat{s})\}$.

Definition 5. *$Reach$ is an outsourced function that returns a set of plans needed to reach a state that meets the target condition g_B from the current equalized state $\hat{s} \in \widehat{S}$:*

$$Reach(\hat{s}, g_B) \subseteq \{\sigma \mid \forall \hat{s}' \in Res(\hat{s}, \sigma) : \hat{s}' \models g_B\}$$

where $\hat{s} \models g_B \Leftrightarrow \forall s \in \hat{s} : s \models g_B$, and Res gives the resulting states of executing a sequence of actions at a state $\hat{s}$: $Res(\hat{s}, \langle\rangle) = \{\hat{s}\}$, and

$$Res(\hat{s}, \langle a_1, \ldots, a_n\rangle) = \begin{cases} \bigcup_{\hat{s}' \in \hat{\Phi}(\hat{s}, a_1)} Res(\hat{s}', \langle a_2, \ldots, a_n\rangle) & \hat{\Phi}(\hat{s}, a_1) \neq \emptyset \\ \{\hat{s}_{err}\} & \hat{\Phi}(\hat{s}, a_1) = \emptyset \end{cases}$$

for $n \geq 1$. Here $\hat{s}_{err}$ is an artifact state that does not satisfy any target, and $\hat{\Phi}$ is a transition relation of executing an action at a state $\hat{s}$:

$$\hat{\Phi}(\hat{s}, a) = \{\hat{s}' \mid \exists s' \in \hat{s}' \, \exists s \in \hat{s} : s' \in \Phi(s, a)\}.$$

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.

Definition 6. *An* equalized (higher level) transition system $\mathcal{T}_{h,P_{g_\infty,KB}}$*, with respect to the classification function h and the policy $P_{g_\infty,KB}$, is defined as $\mathcal{T}_{h,P_{g_\infty,KB}} = \langle \widehat{S}, \widehat{S}_0, \Sigma, G_B, \mathcal{B}, \Phi_{\mathcal{B}} \rangle$, where*

- *$\widehat{S}$ is the finite set of equalized states;*
- *$\widehat{S}_0 \subseteq \widehat{S}$ is the finite set of initial equalized states, where $\hat{s} \in \widehat{S}_0$ if there is some $s_i \in \hat{s}$ such that $s_i \in S_0$ holds;*
- *Σ is the set of possible plans $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$ where $a_i \in \mathcal{A}$, for all $i, 1 \leq i \leq n$.*
- *G_B is the finite set of possible* targets *relative to the behavior, where a target can be satisfied by more than one equalized state;*
- *$\mathcal{B} : \widehat{S} \to 2^{G_B}$, is the* target function *that returns the possible targets to achieve from the current equalized state, according to the policy;*
- *$\Phi_{\mathcal{B}} : \widehat{S} \times \Sigma \to 2^{\widehat{S}}$ is the transition function according to the policy, called the* policy execution function*, where*

  $$\Phi_B(\hat{s}, \sigma) = \{\hat{s}' \mid \hat{s}' \in Res(\hat{s}, \sigma), \sigma \in Reach(\hat{s}, g_B), g_B \in \mathcal{B}(\hat{s})\};$$

  it returns the possible resulting equalized states after applying the plan determined by the policy in the current equalized state.

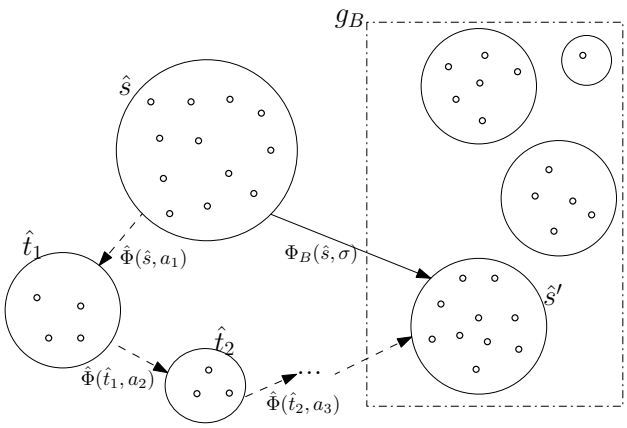

Fig. 2: A transition in the equalized transition system

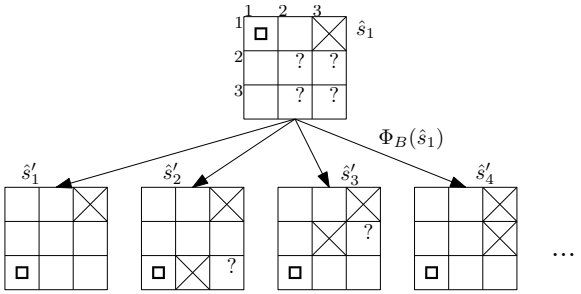

Fig. 3: Parts of an equalized transition system

The target function gets the equalized state as input and produces the possible targets to achieve. These targets may be expressed as formulas over the states (in particular, of states that are represented by fluents or state variables), or in some other representation. The aim is to intend to reach a state that satisfies the conditions of the target.

The equalized transition system $\langle \widehat{S}, \widehat{S}_0, \Sigma, \Phi_B \rangle$ can be viewed as a transition system $\langle S, S_0, \mathcal{A}, \Phi \rangle$ with an infinite set of actions. Additionally, it contains auxiliary definitions $\langle G_B, \mathcal{B} \rangle$ that are used in defining the policy.

Figure 2 demonstrates a transition in the equalized transition system. Depending on the current state, $\hat{s}$, a plan σ can be executed if it is returned by $Reach$ to reach the target g_B that is determined by the policy. There may be more than one equalized state satisfying g_B, and the policy execution function $\Phi_B(\hat{s}, \sigma)$ executes σ and finds a transition into one of these states, $\hat{s}'$. In our case, the actions taken in the transitions are not of concern. Therefore, we project away the knowledge of the executed action sequences, and only consider $\Phi_B : \widehat{S} \to 2^{\widehat{S}}$. Thus, the transition Φ_B becomes a big jump between states, where the actions taken and the states passed in between are omitted.

Example 2. Figure 3 shows a part of the equalized transition system constructed according to the policy. The indistinguishable states due to partial observability are clustered into one. The policy is applied according to current observations, and the possible suc-

cessor states are shown. The policy is targeting the farthest reachable point, which for $\hat{s}_1$ is $(3, 1)$. Since the agent gains knowledge about the environment while moving, there are several possibilities for the resulting state that satisfy the target $g_B{=}robotAt(3, 1)$.

Notice that we assume that the outsourced $Reach$ function is able to return conformant plans that guarantee to reach a state that satisfies the determined targets. For practical reasons, we consider $Reach$ to be able to return a subset of all conformant plans. The maximal possible $Reach$, where we have equality, is denoted with $Reach_0$.

Consider the case of uncertainty, where the agent requires to do some action, e.g., $checkDoor$, to gain further knowledge about its state. The target function can be modified to return dummy fluents as targets to ensure that the action is made, e.g., $doorIsChecked$, and given this target, the $Reach$ function can return the desired action as the plan. The nondeterminism of the environment is modeled through the possible outcomes of Res.

Our generic definition allows for the possibility of representing well-known concepts like purely reactive systems or conformant planning. Reactive systems can be represented with the policy "pick some action", which models systems that immediately react to the environment without reasoning. As for conformant planning, one can set the target as the main goal. Then, $Reach$ would have the difficult task of finding a plan that quarantees reaching the main goal. If however, such a plan is available, then we have the following.

Proposition 1. *Let $P = \langle a_1, \ldots, a_n \rangle$,$n \geq 1$, be a conformant plan that reaches a goal state g from the initial states $s_{01}, \ldots, s_{0r}$ in the original transition system. The plan P can be polynomially expressed in an equalized transition system.*

Proof (Sketch). One can mimic the plan by modifying the targets G_B and the target function $\mathcal{B}$ in a way that at each point in time the next action in the plan is returned by $Reach$, and the corresponding transition is made. For that, one needs to record information in the states and keep track of the targets.

4.3 Complexity Issues

As the function $Reach$ is outsourced, we rely on an implementation that returns conformant plans to achieve transitions in the equalized transition systems. This raises the issue whether a given such implementation is suitable, and leads to the question of soundness (only correct plans are output) and completeness (some plan will be output, if one exists). We next assess how expensive it is to test this, under some assumptions about the representation and computational properties of (equalized) transition systems, which will then also be used for assessing the cost of policy checking.

Assumptions We assume that given a state $s \in S$ which is implicitly given using a binary encoding, the cost of evaluating the classification $h(s)$, the (original) transition $\Phi(s, a)$ for some action a, and recognizing the initial state, say with $\Phi_{init}(s)$, is polynomial. The cost could also be in NP, if projective (i.e., existentially quantified) variables are allowed. Furthermore, we assume that the size of the representation of a "target" in G_B is polynomial in size of the state, so that given a string, one can check in polynomial time if it is a correct target description g_B. This test can also be relaxed to be in NP by allowing projective variables.

Given these assumptions, we have the following two results on the cost of checking whether a given implementation of $Reach$ is sound and complete; we assume here that testing whether $\sigma \in Reach(\hat{s}, g_B)$ is feasible in Π_2^p (i.e., it is no worse than a naive guess and check algorithm that verifies conformant plans).

Theorem 1 (soundness of $Reach$). *Let $\mathcal{T}_h{=}\langle \widehat{S}, \widehat{S}_0, G_B, \mathcal{B}, \Phi_{\mathcal{B}} \rangle$ be a transition system w.r.t. a classification function h. Checking whether every transition found by the policy execution function $\Phi_{\mathcal{B}}$ induced by a given implementation $Reach$ is correct is in Π_3^p.*

The result for soundness of $Reach$[1] is complemented with another result for completeness with respect to short (polynomial size) conformant plans that it returns.

Theorem 2 (completeness of $Reach$). *Let $\mathcal{T}_h = \langle \widehat{S}, \widehat{S}_0, G_B, \mathcal{B}, \Phi_{\mathcal{B}} \rangle$ be a transition system w.r.t. a classification function h. Deciding whether for a given implementation $Reach$, Φ_B fulfills $\hat{s}' \in \Phi_B(\hat{s})$ whenever a short conformant plan from $\hat{s}$ to some $g_B \in \mathcal{B}(\hat{s})$ exists and $\hat{s}'$ is the resulting state after the execution of the plan in T_h, is in Π_4^p.*

The complexities drop if checking the output of $Reach$ is lower (e.g., it drops to Π_2^p for soundness and to Π_3^p for completeness, if output checking is in co-NP).

Throughout the paper we assume that $Reach$ is complete. We also restrict the plans σ that are returned by $Reach$ to have polynomial size. This constraint would not allow for exponentially long conformant plans (even if they exist). Thus, the agent is forced to develop targets that it can reach in polynomially many steps. Informally, this does not limit the capability of the agent in general. The "long" conformant plans can be split into short plans with a modified policy and by encoding specific targets into the states, such that at each state, one chooses the next action with respect to the conformant plan. The targets can be encoded to give the stage of the plan execution so that the respective action is taken, or they can be encoded to assign the latest action in the conformant plan that is done from the current 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. Note that the policy could be easily modified to stop or to loop in any state $\hat{s}$ that satisfies the goal.

Definition 7. *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*)) and then check whether the initial states in $\widehat{S}_0$ satisfy the formula.

Under the assumptions from above, we obtain the following.

Theorem 3. *The problem of determining whether the policy works is in* PSPACE.

In the proof of Theorem 3, for a counterexample, a run of at most exponential length from some initial state in which the main goal is not satisfied can be nondeterministically built in polynomial space.

Note that in this formulation, we have tacitly assumed that the main goal can be established in the original system, thus at least some trajectory from some initial state to a state fulfilling the goal exists. In a more refined version, we could define the working

[1] Proof sketches of this and further results are in the extended version at http://goo.gl/FXktqP.

of a policy relative to the fact that some abstract plan would exist that makes g_∞ true; naturally, this may impact the complexity of the policy checking.

The results in Theorems 1-3 are all complemented by lower bounds for realistic parameter instantiations (notably, for action languages such as fragments of $\mathcal{C}$).

4.4 Constraining Equalization

The definition of $\hat{\Phi}$ allows for certain transitions that do not have corresponding concrete transitions in the original transition system. However, the aim of defining such an equalized transition system is not to introduce new features, but to keep the structure of the original transition system and discard the unnecessary parts with respect to the policy. Therefore, one needs to give further restrictions on the transitions.

Let us consider the following condition.

$$\hat{s}' \in \hat{\Phi}(\hat{s}, a) \Leftrightarrow \forall s' \in \hat{s}',\ \exists s \in \hat{s} :\ s' \in \Phi(s, a) \tag{2}$$

This condition ensures that a transition between two states $\hat{s}_1, \hat{s}_2$ in the equalized transition system represents that any state in $\hat{s}_2$ has a transition from some state in $\hat{s}_1$. An equalization is called *proper* if condition (2) is satisfied.

Theorem 4. *Let $\mathcal{T}_h{=}\langle \widehat{S}, \widehat{S}_0, G_B, \mathcal{B}, \Phi_\mathcal{B}\rangle$ be a transition system w.r.t. a classification function h. Let $\hat{\Phi}$ be the transition function that the policy execution function $\Phi_\mathcal{B}$ is based on. The problem of checking whether $\hat{\Phi}$ is proper is in Π_2^p.*

This result is also complemented by a lower bound similar to the results in Theorem 1-3.

The following proposition shows that the policy execution function is sound.

Proposition 2 (soundness). *Let $\mathcal{T}_h{=}\langle \widehat{S}, \widehat{S}_0, G_B, \mathcal{B}, \Phi_\mathcal{B}\rangle$ be a transition system w.r.t. a classification function h. Let $\hat{s}_1, \hat{s}_2 \in \widehat{S}$ be equalized states that are reachable[2] from some initial states, and $\hat{s}_2 \in \Phi_B(\hat{s}_1)$. For any concrete state $s_2 \in \hat{s}_2$, assuming* (2)*, there is a concrete state $s_1 \in \hat{s}_1$ such that $s_1 \rightarrow^\sigma s_2$ for some action sequence σ, in $\mathcal{T}$.*

Proof of Proposition 2 is based on the possibility of backwards tracking with any of the plans σ executed to reach $\hat{s}_2$ from $\hat{s}_1$.

Thus, we obtain the following corollary, with the requirement of only having initial states clustered into the equalized initial states (i.e., no "non-initial" state is mapped to an initial equalized state). Technically, it should hold that $\forall s \in S_0 : h^{-1}(h(s)) \subseteq S_0$.

Corollary 1. *If there is a trajectory in the equalized transition system with initial state clustering from an equalized initial state $\hat{s}_0$ to g_∞, then for any $g \in g_\infty$ a trajectory can be found in the original transition system from some concrete initial state $s_0 \in \hat{s}_0$.*

Our aim is to analyze the reactive policy through the equalized transition system. If the policy does not work as expected, there will be trajectories showing the failure. Knowing that any such trajectory found in the equalized transition system exists in the original transition system is enough to conclude that the policy indeed does not work.

[2] For a formal definition of reachability, see the extended version at http://goo.gl/FXktqP.

Current assumptions can not avoid the case where a plan σ returned by $Reach$ on the equalized transition system does not have a corresponding trajectory from some initial state in the original transition system. Therefore, we consider as an additional condition

$$\hat{s}' \in \hat{\Phi}(\hat{s}, a) \Leftrightarrow \forall s \in \hat{s},\ \exists s' \in \hat{s}' : s' \in \Phi(s, a) \tag{3}$$

that strengthens the properness condition (2). Under this condition, every plan returned by $Reach$ can be successfully executed from any initial state in the original transition system $\mathcal{T}$. However, still we may lose trajectories of $\mathcal{T}$ as clustering the states might restrain conformant plans; for this, also stronger conditions like exact approximation [8], $\hat{s}' \in \hat{\Phi}(\hat{s}, a) \Leftrightarrow \forall s \in \hat{s},\ \forall s' \in \hat{s}' : s' \in \Phi(s, a)$, is not enough. One would need to modify the target determination, i.e., the set of targets G_B and the function $\mathcal{B}$.

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.

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:

Type 1: Extend the set of truth values by $\mathbf{V}' = \mathbf{V} \cup \{\mathsf{u}\}$, where u denotes the value to be *unknown*. Consider an *observability relation* $\mathcal{O} : \mathbf{F} \times S \to \mathbf{V}'$ which returns how the fluents' values are observed at the states. Then, consider a set of clusters, $\widehat{S}$, where a cluster $\hat{s}_i \in \widehat{S}$ contains all the states $s \in S$ that have the same observed values, i.e., $\widehat{S} = \{\hat{s} \mid \forall d, e \in S,\ d, e \in \hat{s} \iff \forall p \in \mathbf{F} : \mathcal{O}(p, d) {=} \mathcal{O}(p, e)\ \}$. The value function for the clusters is $\widehat{V} : \mathbf{F} \times \widehat{S} \to \mathbf{V}'$.

Type 2: Consider a set of (auxiliary) fluent names $\mathbf{F}_a$, where each fluent $p \in \mathbf{F}_a$ is *related* with some fluents of $\mathbf{F}$. The relation can be shown with a mapping $\Delta : 2^{\mathbf{F} \times \mathbf{V}} \to \mathbf{F}_a \times \mathbf{V}$. Then, consider a new set of clusters, $\widehat{S}$, where a cluster $\hat{s}_i \in \widehat{S}$ contains all the states $s \in S$ that give the same values for all $p \in \mathbf{F}_a$, i.e., $\widehat{S} = \{\hat{s} \mid \forall d, e \in S,\ d, e \in \hat{s} \iff \forall p \in \mathbf{F}_a : V(p, d) {=} V(p, e)\ \}$. The value function for the clusters is $\widehat{V} : \mathbf{F}_a \times \widehat{S} \to \mathbf{V}$.

We can consider the states in the same classification to have the same *profile*, and the classification function h as a membership function that assigns the states into groups.

Remarks: (1) In Type 1, introducing the value *unknown* allows for describing sensing actions and knowing a fluent's true value later. Also, one needs to impose constraints; e.g., a fluent related to a grid cell can not be unknown while the robot can observe it.
(2) In Type 2, one needs to modify the action descriptions according to the newly defined fluents and define *abstract actions*. However, this is not necessary in Type 1, assuming that the action descriptions only use fluents that have *known* values.

Example 3. In $\mathcal{C}$, we introduce unknown values by auxiliary fluents as follows.

caused $uReachable(X, Y)$ **if** $not\ reachable(X, Y)\ \wedge\ not\ \neg reachable(X, Y)$.

i.e. if it is not known that a grid cell is reachable or not, then the fluent $uReachable$ becomes true. Additional rules are added to express that it becomes false otherwise.

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 $\mathcal{F}_{G_B}(\widehat{\mathbf{F}})$.

Example 4. $\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$ corresponds to the set of causal laws in (1) and $\mathcal{F}_{G_B}(\widehat{\mathbf{F}})$ consists of all atoms $targetCell(X, Y)$ and $targetPerson(X, Y)$ for $1 \leq X \leq n, 1 \leq Y \leq n$.

Notice that the separation of formulas $\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$ and the targets $\mathcal{F}_{G_B}(\widehat{\mathbf{F}})$ is to allow for outsourced planners that understand simple target formulas. These planners need no knowledge to find plans. However, if one is able to use planners that are powerful enough, then the target language can be given as input to the planner, so that the planner determines the target and finds the corresponding plan.

Transition Between States The transitions in the (projected) equalized transition system can be denoted with $\widehat{R} \subseteq \widehat{S} \times \widehat{S}$, where $\widehat{R}$ corresponds to the projection of the policy execution function Φ_B that uses (a) the target language to determine targets, (b) an outsourced planner (corresponding to the function $Reach$) to find conformant plans and (c) the computation of executing the plans (corresponding to the function Res). Thus, $\widehat{R}$ shows the resulting states after applying the policy.

Equalized Transition System over Action Language $\mathcal{C}$ The equalized transition system $\langle \widehat{S}, \widehat{V}, \widehat{R} \rangle$ that describes a policy is defined as follows:

(i) $\widehat{S}$ is the set of all interpretations of $\widehat{\mathbf{F}}$ such that, $\hat{s}$ satisfies every static law in $\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$.

(ii) $\widehat{V}(P, \hat{s}) = \hat{s}(P)$, where $P \in \widehat{\mathbf{F}}$,

(iii) $\widehat{R} \subseteq \widehat{S} \times \widehat{S}$ is the set of all $\langle \hat{s}, \hat{s}' \rangle$ such that
  (a) for every $s' \in \hat{s}'$ there is a trajectory from some $s \in \hat{s}$ of the form $s, A_1, s_1, \ldots, A_n, s'$ in the original transition system;
  (b) for static laws $f_1, f_2, \ldots, f_m \in \mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})$ for which $\hat{s}$ satisfies the body, it holds that $\hat{s}' \models g$ for some $g \in \mathcal{M}(f_1, \ldots, f_m)$, where $\mathcal{M}$ is a mapping $\mathcal{M} : 2^{\mathcal{F}_{\mathcal{B}}(\widehat{\mathbf{F}})} \rightarrow 2^{\mathcal{F}_{G_B}(\widehat{\mathbf{F}})}$, that gives the relation between the formulas and the targets.

Notice that $\widehat{R}$ in (iii) has no prescription of (a) how a trajectory is computed or (b) how a target is determined. This makes the implementation of these components flexible.

By focusing on a fragment of $\mathcal{C}$, we match the above conditions on complexity. Furthermore, by well-known results on the complexity of action language $\mathcal{C}$ [27, 14], the results in Theorems 1-4 can be turned into completeness results already for this fragment. Other languages can be similarly used to describe the equalized transition system, as long as they are powerful enough to express the concepts in the previous section.

6 Discussion

The notions of profiles and state clustering help in reducing the state space by omitting irrelevant information. This also comes in handy when dealing with partial observability, since it omits the unobservable information that is irrelevant to the policy.

In the equalized transition system, the trajectories from the initial states correspond to the policy execution, where one can check and verify properties of the policy. The properness condition ensures that any counterexample found in the equalized transition system stating a failure of the policy has a concrete trajectory in the original transition system. This way, the shortcomings of the policy can be detected, and thus improved.

For target language definitions, we can use other formalisms with different expressiveness capabilities, e.g., answer set programming. Target descriptions can be made more complex by considering formulas. In particular, target formulas with disjunctions would express nondeterminism in the environment that affects the target determination. Handling this within the framework requires further study.

It is also possible to use other plans, e.g., short conditional plans, in the planner component. Furthermore, this component can be extended by considering a plan library of precomputed plans. This offline planning component can provide the frequently used plans and reduce the calls to the online planner.

6.1 Related Work

There are works being conducted on the verification of GOLOG programs [22], a family of high-level action programming languages defined on top of action theories expressed in the situation calculus. The method of verifying properties of non-terminal processes are sound, but incomplete as the verification problem is undecidable [11, 9]. By resorting to action formalisms based on description logic, decidability can be achieved [1].

Verifying temporal properties of dynamic systems in the context of data management is studied by [5] for description logic knowledge bases. However, target establishment and planning components, and real-life environment settings are not considered.

The BDI model [24] is based on beliefs, desires and intentions, in which agents are viewed as being rational and acting in accordance with their beliefs and goals. There are many different agent programming languages and platforms based on it. Some works considered verifying properties of agents represented in these languages [4, 12]. These approaches consider very complex architectures that even contain a plan library where plans are matched with the intentions or the agent's state and manipulate the intentions. Verification for such complex BDI architecture gets very challenging.

Verification of multi-agent systems with specifications defined in the epistemic logic is studied by [23], while our focus is on single agents with target determination and planning components which help in reasoning about the behavior of the agent in the environment.

Synthesizing and Verifying Plans. Synthesizing plans via symbolic model checking techniques was considered, e.g., in [7, 6, 3]. The approaches could solve difficult planning problems like strong planning and strong cyclic planning. Son and Baral [25] 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. The equalization of states allows for omitting the details that are irrelevant to the behavior of 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.

Verifying whether a given plan is a solution to a planning problem considering knowledge-based programs as plans [21] or HTN plans [2] has been studied, while the policies that we focus on are more enriched, making use of target determination and outsourced planning.

Execution Monitoring. There are logic-based monitoring frameworks for plan execution and recovery in case of failure. Some of the approaches are replanning [10], backtracking to the point of failure and continuing from there [26], or diagnosing the failure and recovering from the failure situation [16, 13]. 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.

7 Conclusion and Future Work

In this paper, we described a high-level representation that models reactive behaviors, and integrates target development and online planning capabilities. Flexibility in these components does not bound one to only use action languages, but allows for the use of other formalizations as well. For future work, one could imagine targets to depend on further parameters or to incorporate learning from experience in the framework. Furthermore, to instantiate the framework for a range of action languages besides $\mathcal{C}$.

The long-term goal of this work is to check and verify properties of the reactive policies for action languages. In order to solve these problems practically, it is necessary to use techniques from model checking, such as abstraction, compositional reasoning and parameterization. Also, the use of temporal logic formulas is needed to express complex goals such as properties of the policies. Our main target is to work with action languages, and to incorporate their syntax and semantics with such model checking techniques. The general structure of our framework allows one to focus on action languages, and to investigate how to merge these techniques.

References

1. Baader, F., Zarrieß, B.: Verification of Golog programs over description logic actions. Frontiers of Combining Systems pp. 181–196 (2013)
2. Behnke, G., Höller, D., Biundo, S.: On the complexity of htn plan verification and its implications for plan recognition. In: Proc. of ICAPS. pp. 25–33 (2015)
3. Bertoli, P., Cimatti, A., Riveri, M., Traverso, P.: Strong planning under partial observability. Artificial Intelligence 170(4), 337–384 (2006)
4. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous agents and multi-agent systems 12(2), 239–256 (2006)

5. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems. In: Web Reasoning and Rule Systems, pp. 50–64. Springer (2013)
6. Cimatti, A., Riveri, M., Traverso, P.: Automatic OBDD-based generation of universal plans in non-deterministic domains. In: Proc. of AAAI/IAAI. pp. 875–881 (1998)
7. Cimatti, A., Riveri, M., Traverso, P.: Strong planning in non-deterministic domains via model checking. AIPS 98, 36–43 (1998)
8. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Transactions on Programming Languages and Systems (TOPLAS) pp. 1512–1542 (1994)
9. Claßen, J., Lakemeyer, G.: A logic for non-terminating Golog programs. In: Proc. of KR. pp. 589–599 (2008)
10. De Giacomo, G., Reiter, R., Soutchanski, M.: Execution monitoring of high-level robot programs. In: Proc. of KR. pp. 453–465 (1998)
11. De Giacomo, G., Ternovskaia, E., Reiter, R.: Non-terminating processes in the situation calculus. In: Working Notes of Robots, Softbots, Immobots: Theories of Action, Planning and Control, AAAI97 Workshop (1997)
12. Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: Model checking agent programming languages. Automated Software Engineering 19(1), 5–63 (2012)
13. Eiter, T., Erdem, E., Faber, W., Senko, J.: A logic-based approach to finding explanations for discrepancies in optimistic plan execution. Fundamenta Informaticae 79(1-2), 25–69 (2007)
14. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning: Semantics and complexity. ACM Trans. Comput. Log. 5(2), 206–263 (2004), http://doi.acm.org/10.1145/976706.976708
15. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: Proc. of IJCAI. pp. 90–96 (2005)
16. Fichtner, M., Großmann, A., Thielscher, M.: Intelligent execution monitoring in dynamic environments. Fundamenta Informaticae 57(2-4), 371–392 (2003)
17. Fink, M., Germano, S., Ianni, G., Redl, C., Schüller, P.: Acthex: Implementing HEX programs with action atoms. Logic Programming and Nonmonotonic Reasoning pp. 317–322 (2013)
18. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on AI 3(16) (1998)
19. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. of AAAI/IAAI. pp. 623–630 (1998)
20. Kowalski, R.A., Sadri, F.: From logic programming towards multi-agent systems. Ann. Math. Artif. Intell. 25(3-4), 391–419 (1999), http://dx.doi.org/10.1023/A:1018934223383
21. Lang, J., Zanuttini, B.: Knowledge-based programs as plans - the complexity of plan verification. In: Proc. of ECAI. pp. 504–509 (2012)
22. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. The Journal of Logic Programming 31(1), 59–83 (1997)
23. Lomuscio, A., Michliszyn, J.: Verification of multi-agent systems via predicate abstraction against ATLK specifications. In: Proc. of AAMAS. pp. 662–670 (2016)
24. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: Proc. of KR. pp. 473–484 (1991)
25. Son, T.C., Baral, C.: Formalizing sensing actions – a transition function based approach. Artificial Intelligence 125(1), 19–91 (2001)
26. Soutchanski, M.: High-level robot programming and program execution. In: Proc. of ICAPS Workshop on Plan Execution (2003)
27. Turner, H.: Polynomial-length planning spans the polynomial hierarchy. In: Proc. of JELIA. pp. 111–124. Springer (2002)