Assured Reinforcement Learning with Formally Verified Abstract Policies

George Mason[1], Radu Calinescu[1], Daniel Kudenko[1] and Alec Banks[2]

[1]*Department of Computer Science, University of York, Deramore Lane, York, U.K.*

[2]*Defence Science and Technology Laboratory, Salisbury, U.K.*

Keywords: Reinforcement Learning, Safety Constraint Verification, Abstract Markov Decision Processes.

Abstract: We present a new reinforcement learning (RL) approach that enables an autonomous agent to solve decision making problems under constraints. Our *assured reinforcement learning* approach models the uncertain environment as a high-level, abstract Markov decision process (AMDP), and uses probabilistic model checking to establish AMDP policies that satisfy a set of constraints defined in probabilistic temporal logic. These *formally verified abstract policies* are then used to restrict the RL agent's exploration of the solution space so as to avoid constraint violations. We validate our RL approach by using it to develop autonomous agents for a flag-collection navigation task and an assisted-living planning problem.

1 INTRODUCTION

Reinforcement learning (RL) is a machine learning technique where an autonomous agent uses the rewards received from its interactions with an initially unknown Markov decision process (MDP) to converge to an optimal *policy*, i.e. the actions to take in the MDP states in order to maximise the obtained rewards (Wiering and Otterlo, 2012). Although successfully used in applications ranging from gaming (Szita, 2012) to robotics (Kober et al., 2013), standard RL is not applicable to problems where the policies synthesised by the agent must satisfy strict constraints associated with the safety, reliability, performance and other critical aspects of the problem.

Our work addresses this significant limitation of standard RL, extending the applicability of the technique to mission-critical and safety-critical systems. To this end, we present an *assured reinforcement learning* (ARL) approach that restricts the exploration of the RL agent to MDP regions guaranteed to yield solutions compliant with the required constraints. Given limited preliminary knowledge of the problem under investigation, ARL builds a high-level *abstract Markov decision process* (AMDP) (Marthi, 2007) model of the environment and uses probabilistic model checking (Kwiatkowska, 2007) to identify AMDP policies that satisfy a set of constraints formalised in *probabilistic computation tree logic* (PCTL) (Hansson and Jonsson, 1994). The constraint-compliant (i.e. "safe") *abstract policies* obtained in this way are then used to resolve some of the nondeterminism of the unknown problem MDP, inducing a restricted MDP that the RL agent can explore without violating any of the constraints.

As multiple safe abstract policies are generated during the probabilistic model checking stage of ARL, our approach retains only the abstract policies that are *Pareto-optimal* with respect to optimization objectives associated with the analysed constraints and/or specified additionally. For example, the constraints for the RL problem from one of the case studies described later in the paper specify a measure of the maximum healthcare cost permitted for an assisted-living system. The Pareto-optimal abstract policies that ARL retains for this RL problem (i) guarantee that this maximum threshold is not exceeded, and (ii) cannot be replaced by (known) policies that reduce both the healthcare cost and an additional optimization objective that reflects the level of distress for the patient using the system.

Our ARL approach complements the recent research on *safe reinforcement learning* (García and Fernández, 2015). The existing results from this area focus on specifying bounds for the reward obtained by the RL agent or for simple measures associated with this reward (Abe et al., 2011; Castro et al., 2012; Delage and Mannor, 2010; Geibel, 2006; Moldovan and Abbeel, 2012; Ponda et al., 2013). In contrast to these approaches, ARL uses probabilistic model checking to formally establish *safe AMDP policies* associated with the broad range of safety, reliability and perfor-

Mason G., Calinescu R., Kudenko D. and Banks A.
Assured Reinforcement Learning with Formally Verified Abstract Policies.
DOI: 10.5220/0006156001050117
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 105-117
ISBN: 978-989-758-220-2

mance constraints that can be formally specified in PCTL (Hansson and Jonsson, 1994) extended with rewards (Andova et al., 2004). To the best of our knowledge, these types of highly useful constraints are not supported by any existing safe RL solutions.

To assess the effectiveness and generality of ARL, we thoroughly evaluated its application through two case studies that addressed different types of RL problems within different application domains. The first case study tackles a navigation problem based on the benchmark RL flag-collection domain (Dearden et al., 1998), which we extended with an element of risk that requires the application of ARL. The second case study involves solving a planning problem to assist a dementia sufferer perform the task of hand-washing, and is adapted from a real-world application from the assisted-living domain (Boger et al., 2006). In both case studies, ARL generated Pareto-optimal sets of safe abstract policies, which were successfully used to drive the RL agents' exploration towards solutions satisfying a range of reliability and performance constraints associated with the two problems.

The rest of our paper is organized as follows. In Section 2, we compare our work with related research from the area of safe reinforcement learning. Next, Section 3 introduces the terminology and techniques that underpin the operation of ARL. Section 4 presents a motivating example that we use to illustrate the application of our ARL approach, which is described in Section 5. The two cases studies that we used to validate ARL are presented in Section 6, and Section 7 concludes the paper with a brief summary and a discussion of future work directions.

2 RELATED WORK

Our assured reinforcement learning approach belongs to a class of RL methods called *safe reinforcement learning* (García and Fernández, 2015). However, existing safe RL research focuses on enforcing bounds on the reward obtained by the RL agent or on simple measures related to this reward. Geibel (2006) proposes a safe RL method that supports an inequality constraint on the reward cumulated by the RL agent or a maximum permitted probability for such a constraint to be violated. Delage and Mannor (2010) and Ponda et al. (2013) introduce RL methods that enforce similar constraints through generalizing chance-constrained planning to infinite-horizon MDPs. Thanks to the wide range of safety, reliability and performance properties that can be expressed in PCTL, our ARL approach supports a much broader range of RL constraints, which includes those covered in (Delage and Mannor, 2010; Geibel, 2006; Ponda et al., 2013).

Other recent research on safe RL includes the work of (Moldovan and Abbeel, 2012), who introduce an approach that enforces the RL agent to avoid irreversible actions by entering only states from which it can return to the initial state. Similar approaches are proposed by Castro et al. (2012) and Abe et al. (2010). Thus, Castro et al. (2012) exploit domain knowledge from financial decision making and robust process control to enforce constraints on the cumulative reward obtained by the RL agent, on the variance of this reward, or on a combination of the two. Along the same lines, Abe et al. (2010) present a safe RL method that enforces high-level business and legal constraints during each value iteration step of the RL process, and apply their method to a tax collection optimization problem. ARL is complementary to the approaches proposed in (Abe et al., 2011; Castro et al., 2012; Moldovan and Abbeel, 2012) as it supports different types of constraints. In particular, the PCTL-encoded constraints used by our approach are not specific to any domain or application like those from (Castro et al., 2012) and (Abe et al., 2011).

ARL is unique in its use of an abstract MDP and of probabilistic model checking to constrain the RL agent's exploration to safe areas of the environment. Built with only limited knowledge about the problem to solve, this AMDP has a significantly smaller state space and action set than the (unknown) MDP model of the environment (Li et al., 2006; Marthi, 2007; Sutton et al., 1999), allowing the efficient analysis of problem aspects associated with the required constraints. In contrast, existing safe RL methods modify the reward function to "penalize" agent actions associated with high variance in the probability of attaining the reward (Mihatsch and Neuneier, 2002) or to avoid irreversible actions (Moldovan and Abbeel, 2012); or use domain knowledge to keep away from unsafe states (Driessens and Džeroski, 2004).

Unlike existing safe RL approaches, ARL synthesises a Pareto-optimal set of safe AMDP policies that correspond to a wide range of trade-offs between relevant attributes of the optimisation problem. Although *multi-objective RL* (MORL) research (Liu et al., 2015; Vamplew et al., 2011) has considered the problem of learning a policy that satisfies conflicting objectives, existing MORL methods, e.g. (Barrett and Narayanan, 2008; Gábor et al., 1998; Mannor and Shimkin, 2004; Moffaert and Nowé, 2014), do not support the broad range of optimisation objectives that can be specified in ARL using reward-augmented PCTL constraints.

Finally, our ARL approach builds on preliminary results that we described in a recent work-in-progress paper (Mason et al., 2016), which it significantly extends through formalising the stages of the approach, through the addition of an assisted-living case study that confirms the applicability of ARL to planning problems, and through the inclusion of new experimental results and insights from these experiments.

3 PRELIMINARIES

Markov Decision Processes (MDPs) (Wiering and Otterlo, 2012) are a formalism for modelling sequential decision-making problems where an autonomous agent can perceive the current environment state s and select an action a from a set of actions. Performing the selected action a results in the agent transitioning to a new state s' and receiving an immediate reward $r \in \mathbb{R}$.

An MDP is formally defined as a tuple (S, A, T, R), where: S is a finite set of states; A is a finite set of actions; $T : S \times A \times S \to [0,1]$ is a state transition function such that for any $s, s' \in S$ and any action $a \in A$ that is allowed in state s, $T(s,a,s')$ gives the probability of transitioning to state s' when performing action a in state s; and $R : S \times A \times S \to \mathbb{R}$ is a reward function such that $R(s,a,s') = r$ is the reward received by the agent when action a performed in state s leads to state s'.

Given an MDP (S, A, T, R), we are interested in finding a *deterministic policy* $\pi : S \to A$ that maps each state in S to one of the actions permitted in that state and maximizes the reward accrued by an autonomous agent that follows the policy. When all MDP elements are known, the problem can be solved using dynamic programming, e.g. by means of value or policy iteration algorithms. In scenarios where the transition function T and/or the reward function R are unknown a priori, RL is employed as described below.

Reinforcement Learning (RL) is a machine learning technique where an autonomous agent with no initial knowledge of an environment must learn about it through *exploration*, i.e. by selecting initially arbitrary actions while moving from one state of an unknown MDP to another. By receiving rewards after each state transition, the RL agent learns about the quality of its action choices. The agent stores this knowledge it gains about the quality of a state-action pair $(s,a) \in S \times A$ in the form of a *Q-value* $Q(s,a) \in \mathbb{R}$. Updates to Q-values are done using a temporal difference learning algorithm such as Q-learning (Watkins and Dayan, 1992), whose formula

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \cdot \max_{a' \in A} Q(s',a') - Q(s,a)],$$

shows the Q-value $Q(s,a)$ being updated after selecting action a in state s earned the agent a reward r and took it to state s', where $\alpha \in (0,1]$ is the learning rate and $\gamma \in [0,1]$ is the discount factor.

The Q-value updates propagate information about rewards within the environment over the state-action pairs, enabling the RL agent to exploit the knowledge it has learned. Thus, when the agent revisits a state, it can perform an action based on a pre-defined policy instead of a randomly selected action. As an example, with the ε-*greedy policy* the agent acts randomly with probability ε and selects the highest Q-value action with probability $1 - \varepsilon$ (Wiering and Otterlo, 2012). Provided that each MDP state is visited infinitely many times and the learning rate α satisfies certain conditions, the algorithm is guaranteed to converge to an optimal policy. In practice, a finite number of learning iterations (i.e. *episodes*) is typically enough to obtain a policy sufficiently close to the optimal policy.

Abstract MDPs (AMDPs) are high-level representations of MDPs in which multiple MDP states are aggregated, e.g. based on their similarity (Li et al., 2006), and the MDP actions are replaced by temporally abstract *options* (Sutton et al., 1999). As an example, instead of an agent performing a sequence of stepwise movements to transition through a series of Cartesian coordinates from room A to enter room B in a navigation problem, in an associated AMDP each location would be a single state and the option would simply be to "move" from room A to room B. Accordingly, an AMDP is orders of magnitude smaller than its MDP counterpart, can often be built with very limited knowledge about the environment, and can be solved and reasoned about much faster (Marthi, 2007).

Consider an MDP (S, A, T, R) and a function $z : S \to \bar{S}$ that maps each state $s \in S$ to an abstract state $z(s) \in \bar{S}$ such that $\bar{S} = z(S)$. Then, the AMDP induced by the state-mapping function z is a tuple $(\bar{S}, \bar{A}, \bar{T}, \bar{R})$, where: $\bar{S}$ is the set of abstract states; $\bar{A}$ is the set of options; $\bar{T} : \bar{S} \times \bar{A} \times \bar{S} \to [0,1]$ is a state transition function such that

$$\bar{T}(\bar{s}, o, \bar{s}') = \sum_{s \in S, z(s) = \bar{s}} w_s \sum_{s' \in S, z(s') = \bar{s}'} P(s'|s,o)$$

for any $\bar{s}, \bar{s}' \in \bar{S}$ and any option $o \in \bar{A}$; and $\bar{R} : \bar{S} \times \bar{A} \times \bar{S} \to \mathbb{R}$ is a reward function such that

$$\bar{R}(\bar{s}, o) = \sum_{s \in S, z(s) = \bar{s}} w_s R(s,o)$$

for any $\bar{s} \in \bar{S}$ and any $o \in \bar{A}$, where $w_s \geq 0$ is the weight of state s, calculated based on the expected frequency of occurrence of state s in the abstract state $z(s)$ (Marthi, 2007).

A *parameterised* AMDP uses parameters to specify which option to perform in each AMDP state (Xia and Jia, 2013). An *abstract policy* selects the values of these parameters, resolving the nondeterminism of the AMDP, and thus transforming it into a discrete-time Markov chain with a single option for each state.

Probabilistic model checking is a mathematically based technique for establishing reliability, performance and other nonfunctional properties of systems with stochastic behaviour (Kwiatkowska, 2007). Given a Markovian model of the analysed system, probabilistic model checking tools, such as PRISM (Kwiatkowska et al., 2011) and MRMC (Katoen et al., 2011), use efficient symbolic algorithms to efficiently examine its entire state space, producing results that are guaranteed to be correct. The technique has been successfully used to analyse nonfunctional properties of systems ranging from cloud infrastructure (Calinescu et al., 2012) and service-based systems (Calinescu et al., 2013) to unmanned vehicles (Gerasimou et al., 2014). Typical properties that can be established using the probabilistic model checking include: '*What is the probability that the agent will safely reach the goal area?*' and '*What is the expected level of distress for the dementia patient?*' in the flag-collection navigation problem and in the assisted-living planning problem from our case studies, respectively.

Probabilistic model checking operates with MDPs whose states are labelled with *atomic propositions* that specify basic properties of interest that hold in each MDP state, e.g. *start*, *goal*, or *RoomA*. MDPs labelled with atomic propositions enable the analysis of properties that express probabilities and temporal relationships between events and are specified in a probabilistic variant of temporal logic called probabilistic computational tree logic (PCTL) (Hansson and Jonsson, 1994). Given a set of atomic propositions AP, a *state formula* Φ and a *path formula* Ψ in PCTL are defined by the grammar:

$$\begin{array}{l} \Phi ::= true \mid a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathrm{P}_{\bowtie p}[\Psi] \\ \Psi ::= \mathrm{X}\Phi \mid \Phi_1 \,\mathrm{U}\, \Phi_2 \mid \Phi_1 \,\mathrm{U}^{\leq k}\, \Phi_2 \end{array}, \qquad (1)$$

where $a \in AP$, $\bowtie \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ and $k \in \mathbb{N}$; and a PCTL *reward state formula* (Kwiatkowska et al., 2007) is defined by the grammar:

$$\Phi ::= \mathrm{R}_{\bowtie r}[\mathrm{I}^{=k}] \mid \mathrm{R}_{\bowtie r}[\mathrm{C}^{\leq k}] \mid \mathrm{R}_{\bowtie r}[\mathrm{F}\Phi] \mid \mathrm{R}_{\bowtie r}[\mathrm{S}], \qquad (2)$$

where $r \in \mathbb{R}_{\geq 0}$. State formulae include the logical operators $\wedge$ and $\neg$, which allow the formulation of disjunction ($\vee$) and implication ($\Rightarrow$).

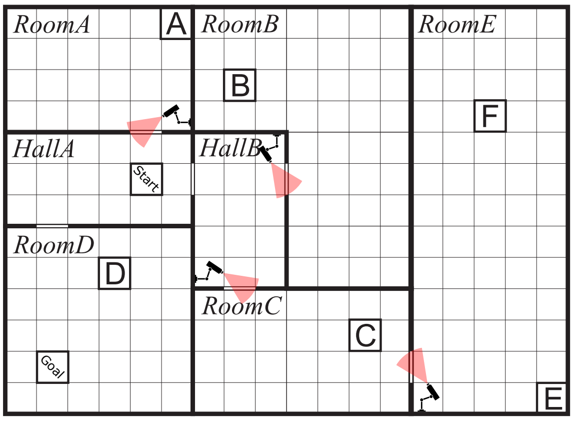

Figure 1: Flag-collection mission. (Dearden et al., 1998) extended with security cameras. The diagram shows the flag positions A–F, the start and goal positions for the agent, and the cameras and their field of view.

The semantics of PCTL are defined with a satisfaction relation $\models$ over the states and paths of an MDP (S, A, T, R). Thus, $s \models \Phi$ means Φ is satisfied in state s. For any state $s \in S$, we have: $s \models true$; $s \models a$ iff s is labelled with the atomic proposition a; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. A state formula $\mathcal{P}_{\bowtie p}[\Psi]$ is satisfied in a state s if the probability of the future evolution of the system satisfying Ψ satisfies $\bowtie p$. For an MDP path $s_1 s_2 s_3 \ldots$, the "next state" formula $\mathrm{X}\,\Phi$ holds iff Φ is satisfied in the next path state (i.e. in state s_2); the *bounded until* formula $\Phi_1 \,\mathrm{U}^{\leq k}\, \Phi_2$ holds iff before Φ_2 becomes true is some state s_x, $x < k$, Φ_1 is satisfied for states s_1 to s_{x-1}; and the *unbounded until* formula $\Phi_1 \,\mathrm{U}\, \Phi_2$ removes the constraint $x < k$ from the bounded until. For instance, the PCTL formula $\mathrm{P}_{\geq 0.95}[\neg RoomA \,\mathrm{U}\, goal]$ formalises the constraint 'the probability of reaching the *goal* without entering *RoomA* is at least 0.95'.

The notation $\mathrm{F}^{\leq k}\Phi \equiv true\mathrm{U}^{\leq k}\Phi$, and $\mathrm{F}\Phi \equiv true\mathrm{U}\Phi$ is used when the first part of a bounded until, and until formula, respectively, are *true*. The reward state formulae (2) express the expected cost at timestep k ($\mathcal{R}_{\bowtie r}[\mathrm{I}^{=k}]$), the expected cumulative cost up to time step k ($\mathcal{R}_{\bowtie r}[\mathrm{C}^{\leq k}]$), the expected cumulative cost to reach a future state that satisfies a property Φ ($\mathcal{R}_{\bowtie r}[\mathrm{F}\Phi]$), and the expected steady-state reward in the long run ($\mathcal{R}_{\bowtie r}[\mathrm{S}]$).

Finally, probabilistic model checkers also support PCTL formulae in which the bounds '$\bowtie p$' and '$\bowtie r$' are replaced with '=?', to indicate that the computation of the actual bound is required. For example, $\mathrm{P}_{=?}[\mathrm{F}^{\leq 20} goal]$ expresses the probability of succeeding (i.e. of reaching a state labelled *goal*) within 20 time steps.

4 MOTIVATING EXAMPLE

We motivate the need for assured reinforcement learning using an extension of the benchmark RL flag-collection mission from (Dearden et al., 1998). In the original mission, an agent needs to find and collect flags scattered throughout a building by learning to navigate through its rooms and hallways. In our extension, certain doorways between areas are provided with security cameras, as shown in Figure 1. Detection by a camera results in the capture of the agent and the termination of its flag-collection mission.

Unknown to the agent, the detection effectiveness of the cameras decreases towards the boundary of their field of view, so that the camera-monitored doorways comprise three areas with decreasing probabilities of detection: direct view by the camera, partial view, and hidden. We assume that the detection probabilities for the camera-monitored doorways from Figure 1 and the camera-view areas have the values from Table 1.

Table 1: Agent detection probabilities.

Camera	**Camera view**		
	Direct	Partial	Hidden
HallA ↔ RoomA	0.18	0.12	0.06
HallB ↔ RoomB	0.15	0.1	0.05
HallB ↔ RoomC	0.15	0.1	0.05
RoomC ↔ RoomE	0.21	0.14	0.07

Consider now a real-world application where the agent is an expensive autonomous robot pursuing a surveillance mission or a search-and-rescue operation. In this scenario, its owners are interested in the safe return of robot, but do not want it to behave "too safely" or it will not collect enough flags. Therefore, they specify the following constraints for the agent:

C_1 The agent should reach the 'goal' area with probability at least 0.75.

C_2 The agent should cumulate more than two flags before it reaches the 'goal' area.

Subject to these constraints being satisfied, they are interested to maximise:

O_1 The probability that the agent reaches the 'goal'.

O_2 The number of collected flags.

As a result, the agent owners additionally want to know the range of possible trade-offs between these two conflicting *optimization objectives*. In this way, the right level of trade-off can be selected for each instance of the mission. Note that formulating the constraints C_1 and C_2 into a reward function and using standard RL to solve the problem is not possible because an RL agent aims to maximize its reward rather than to maintain it within a specified range.

5 THE ARL APPROACH

Our ARL approach takes as input the following information about problem to solve:

1. Partial knowledge about the problem;
2. A set of constraints $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ that must be satisfied by the policy learnt by the RL agent;
3. A set of objectives $O = \{O_1, O_2, \ldots, O_m\}$ that the RL policy should optimise (i.e. minimise or maximise) subject to all constraints being satisfied.

The optimisation objectives O can be associated with problem properties that appear in the constraints $\mathcal{C}$ (like in our motivating example), or also with additional problem properties (as in the assisted-living planning problem from Section 6). The partial knowledge must contain sufficient information for the assembly of an abstract MDP supporting the formalisation in PCTL and the probabilistic model checking of the $n > 0$ constraints and $m \geq 0$ optimisation objectives. For instance, for constraint C_1 from our motivating example, it is enough to know the hidden-view detection probabilities of the cameras (as the agent should be able to learn the best area for crossing the doorway). Note that the partial knowledge about the environment assumed by ARL is necessary: no constraints could be ensured during RL exploration in the absence of any information about the environment. Furthermore, ARL assumes that sufficient learning is undertaken by the RL agent to find an optimal policy for safety requirements to be assured; suboptimal RL policies may not satisfy the safety requirements.

Under these assumptions, our ARL approach: (i) generates a Pareto-optimal set of "safe" abstract policies that satisfy the constraints $\mathcal{C}$ and are Pareto non-dominated with respect to the optimisation objectives O; and (ii) learns a (concrete) policy that satisfies the constraints $\mathcal{C}$ and meets trade-offs between objectives O given by a Pareto-optimal abstract policy selected by the user. ARL comprises three stages:

1. **AMDP construction** – This stage devises a parameterised AMDP model of the RL problem that supports the probabilistic model checking of PCTL-formalised versions of the constraints $\mathcal{C}$ and of the optimisation objectives O.
2. **Abstract policy synthesis** – This stage generates the Pareto-optimal set of "safe" abstract policies.
3. **Safe learning** – This stage uses a user-selected abstract policy from the Pareto-optimal set to enforce state-action constraints for the exploration of the environment by the RL agent, which subsequently learns an optimal policy that complies

with the problem constraints and meets the optimisation objective trade-offs associated with the selected abstract policy.

The three ARL stages are described in detail in the remainder of the section.

Stage 1: AMDP Construction. In this ARL stage, all features that are relevant for the problem constraints and optimisation objectives must be extracted from the available partial knowledge about the RL environment. This could include locations, events, rewards, actions or progress levels. The objective is to abstract out the features that have no impact on the solution attributes that the constraints $\mathcal{C}$ and objectives O refer to, whilst retaining the key features that these attributes depend on. This ensures that the AMDP is sufficiently small to be analysed using probabilistic model checking, while also containing the necessary details to enable the analysis of all constraints and optimisation objectives.

In our motivating example, the key features are the locations and connections of rooms and halls, the detection probabilities of the cameras and the progress of the flags collected. Instead of having each Cartesian coordinate within a room or hall as a separate state, the room or hall as a whole is considered a single state in the AMDP. Also, we only consider the hidden-view detection probability per camera since these are the probabilities that the RL agent will learn for the optimal points to traverse the doorways. These abstractions yield a 448-state AMDP for our flag-collection problem, compared to 14,976 states for the RL MDP (which is unknown to the agent). Note that the number of AMDP states is larger than the number of locations (i.e. rooms and halls) because different AMDP states are used for each possible combination of a location and a number of flags collected so far.

The actions of the full RL MDP are similarly abstracted. For example, instead of having the cardinal movements at each location of the building from our motivating example, abstract actions (i.e. *options* – cf. Section 3) are specified as simply the movement between locations. Thus, instead of the four possible actions for each of the 14,976 MDP state, the 448 AMDP states have only between one and four possible options each. The N options that are available for an AMDP state correspond to the $N \geq 1$ passageways that link the location associated with that state with other locations, and can be encoded using a *state parameter* that takes one of the discrete values 1, 2, $\ldots, N$. The parameters for AMDP states with a single passageway (corresponding to rooms A, B and E from Figure 1) can only take the value 1 and are therefore discarded. This leaves a set of 256 parameters that correspond to approximately 4×10^{99} possible abstract policies.

Algorithm 1: Abstract policy synthesis heuristic.

```
function GenAbstractPolicies($\mathcal{M}, \mathcal{C}, O$)
    $PS \leftarrow \{\}$
    while $\neg$Done($PS$) do
        $P \leftarrow$ GetCandidatePolicies($PS, \mathcal{M}$)
        for $\pi \in P$ do
            if $\bigwedge_{c \in \mathcal{C}} \mathrm{PMC}_1(\mathcal{M}, \pi, c)$ then
                $dominated = \mathsf{false}$
                for $\pi' \in PS$ do
                    if Dom($\pi, \pi', \mathcal{M}, O$) then
                        $PS \leftarrow PS \setminus \{\pi'\}$
                    else if Dom($\pi', \pi, \mathcal{M}, O$) then
                        $dominated = \mathsf{true}$
                        break
                    end if
                end for
                if $\neg dominated$ then
                    $PS \leftarrow PS \cup \{\pi\}$
                end if
            end if
        end for
    end while
    return $PS$
end function

function Dom($\pi_1, \pi_2, \mathcal{M}, O$)
    return
        $\forall o \in O \cdot \mathrm{PMC}_2(\mathcal{M},\pi_1,o) \geq \mathrm{PMC}_2(\mathcal{M},\pi_2,o) \wedge$
        $\exists o \in O \cdot \mathrm{PMC}_2(\mathcal{M},\pi_1,o) > \mathrm{PMC}_2(\mathcal{M},\pi_2,o)$
end function
```

Finally, this ARL stage is also responsible for labelling the AMDP with atomic propositions enabling its probabilistic model checking, and for the PCTL formalisation of the constraints $\mathcal{C}$ and optimisation objectives O in terms of these atomic propositions. For our flag-collection mission, this involves associating an atomic proposition '*goal*' with the AMDP states corresponding to the agent reaching the 'goal' area (with any number of collected flags), and formalising the constraints and optimisation objectives from Section 4 as follows:

C_1: $\mathrm{P}_{\geq 0.75}[\mathrm{F}\, goal]$ $\quad$ O_1: maximize $\mathrm{P}_{=?}[\mathrm{F}\, goal]$
C_2: $\mathrm{R}_{>2}[\mathrm{F}\, goal]$ $\quad$ O_2: maximize $\mathrm{R}_{=?}[\mathrm{F}\, goal]$

Stage 2: Abstract Policy Synthesis. In this ARL stage, the generic heuristic from Algorithm 1 is used to find constraint-compliant abstract policies for the RL problem. Given an AMDP $\mathcal{M}$, a set of constraints $\mathcal{C}$ and a set of optimisation objectives O (all obtained in Stage 1 of ARL), the function GenAbstractPolicies from Algorithm 1 synthesises an approximate Pareto-optimal set of abstract policies that satisfy the constraints $\mathcal{C}$ and are Pareto non-dominated with respect to the optimisation objectives O. The abstract policy set *PS* returned by this function in line 22 starts empty (line 2), and is assembled iteratively by the while loop in lines 3–21 until a termination criterion $\neg$DONE(PS) is satisfied. This criterion (not shown in Algorithm 1) may involve ending the while loop after a fixed number of iterations, or after several consecutive iterations during which *PS* is left unchanged. Each iteration of the while loop first identifies a set *P* of "candidate" abstract policies in line 4, and then updates the Pareto-optimal policy set in the for loop from lines 5–20. Our algorithm is not prescriptive about the method used to get new candidate policies. As such, the function GETCANDIDATEPOLICIES from line 4 can be implemented using a metaheuristic such as the genetic algorithm used to synthesise Markovain models in (Gerasimou et al., 2015), a simple heuristic like hill climbing, or just random search.

To decide how to update *PS*, the for loop in lines 5–20 examines each candidate abstract policy π as follows. First, the boolean function PMC$_1$ (which invokes a probabilistic model checking tool) is used to establish whether using policy π for the AMDP $\mathcal{M}$ satisfies every constraint $c \in \mathcal{C}$ (line 6). If it does, π is deemed "safe" and the inner for loop in lines 8–15 compares it to each of the abstract policies already in *PS* by using the Pareto-dominance comparison function DOM defined in lines 24–26, where the probabilistic model checking function PMC$_2(\mathcal{M},\pi,o)$ computes the value of the optimisation objective $o \in O$ for the policy π of $\mathcal{M}$.[1] Every policy $\pi' \in PS$ that is Pareto dominated by π is removed from *PS* (lines 9–10). If π is itself Pareto-dominated (line 11), the flag *dominated* (initially false, cf. line 7) is set to true in line 12 and the inner for loop is terminated early in line 13. Finally, the new abstract policy is added to the Pareto-optimal policy set if is not dominated by any known policy (lines 16–18).

Stage 3: Safe Learning. The final stage of ARL exploits the previously obtained approximate Pareto-optimal set of abstract policies. A policy is selected from this set by taking into account the trade-offs that different policies achieve for the optimisation objectives used to assemble the set. The high-level *options* from the abstract policy are used as rules for which of the corresponding low-level MDP actions the RL agent should, or should not, perform in order to achieve the required constraints. For instance, assume that the selected abstract policy for our motivating example requires the agent to never enter RoomA. In this case, should the agent be at Cartesian coordinates (5,9) (i.e. the position immediately to the North of the Start position), the action to move North and thus to enter RoomA is removed from the agent's action set, for this specific state. Disallowing actions that are not associated with the safe options of the abstract policy results in the RL agent learning low-level behaviours that are guaranteed to satisfy the safety constraints.

This restriction of actions necessarily reduces the RL agent's autonomy, however, it is not removed entirely. Specifically, to ensure that the agent behaves according to the safety requirements, exploration of actions that can result in safety violations, i.e. those actions which contradict the abstract policy, are restricted. Otherwise, the agent is free to explore its environment as it normally would. For example, in the motivating example, the agent's exploration is restricted only by which rooms it can enter. The agent must still explore the environment to learn the flag locations within the rooms as well as the doorway areas safest to cross, information which is unknown a priori and therefore not contained within the abstract policies. Although abstract policy constraints may yield suboptimal RL policies with respect to the RL model in its entirety, this key feature assures safety.

6 EVALUATION

To evaluate the efficacy and generality of ARL we applied it to two case studies from different domains. The first case study is based on the navigation task described in Section 4, where the learning agent must navigate a guarded building with the objective of collecting flags distributed throughout. The second case study is a planning problem adapted from (Boger et al., 2006), where a system has been designed to assist a dementia sufferer perform the task of washing their hands.

For each case study we conducted a set of four experiments. An initial experiment was first conducted which was a traditional RL implementation of the case study problem. This experiment serves as a baseline which we contrast with the ARL experiments in order to determine the effects of our method. Following the baseline experiment a further three experiments were undertaken where RL in Stage 3 of ARL was applied using a different abstract policy from the Pareto-optimal set of abstract policies constructed in

[1] A policy π_1 is said to Pareto-dominate another policy π_2 with respect to a set of objectives O iff π_1 gives superior results to π_2 for at least one objective from O, and for all other objectives π_1 it is at least as good as π_2 (Liu et al., 2015). Without loss of generality, the definition of DOM from Algorithm 1 assumes that all objectives from O are maximising objectives.

Stage 2 using an implementation based on random search for function GETCANDIDATEPOLICIES from Algorithm 1.

For all experiments we use a discount factor $\gamma = 0.99$ and a learning rate $\alpha = 0.1$ which decays to 0 over the learning run, experiment-specific parameters are shown where relevant in the remainder of this section. All parameters have been chosen empirically in line with standard RL practice. As is convention when evaluating stochastic processes, we repeated each experiment multiple times (i.e. five times) and we evaluated the final policy for each experiment many times (i.e. 10,000 times) in order to ensure that the results are suitably significant (Arcuri and Briand, 2011).

We evaluate the learning progress of each experiment after each learning episode during a run. Error bars for the standard error of the mean show the statistical significance of the learning over the five learning runs that we performed for each experiment (Figures 3, 4, 7 and 8). Evaluation of the learned RL policies was done once a learning run had finished (Tables 3 and 7).

6.1 Guarded Flag Collection

This case study is based on the scenario described in Section 4 and referred to throughout Section 5. In the interest of brevity, the details presented in these two previous sections will not be repeated here.

In our RL implementation, the reward structure was defined as follows: the agent receives a reward of 1 for each flag it collects and an additional reward of 1 for reaching the 'goal' area of the building. If the agent is captured it receives a reward of -1 and any flags that have been previously collected are disregarded.

We used the AMDP constructed during the first ARL stage as described in Section 5. In the second ARL stage, we generated 10,000 abstract policies with parameter values (i.e. state to action mappings) drawn randomly from a uniform distribution. Out of these abstract policies, probabilistic model checking using the tool PRISM identified 14 policies that satisfied the two constraints from Section 4. Figure 2 shows the QV results obtained for these 14 abstract policies, i.e. their associated probability of reaching the 'goal' area and expected number of flags collected. The approximate Pareto-front depicted in this figure was obtained using the two optimization objectives described in Section 5, i.e. maximizing the expected number of flags collected and the probability of reaching the 'goal' area of the building.

Three abstract policies were selected to use in each of the ARL experiments during the safe learning stage, as explained in Section 5. The properties of these three abstract policies are shown in Table 2.

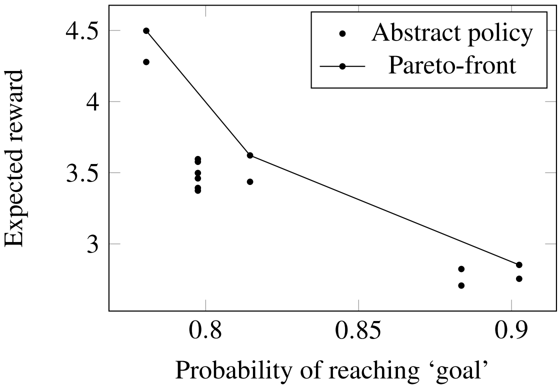

Figure 2: Pareto-front of abstract policies that satisfy the constraints from Section 4.

Table 2: Selected abstract policies to use for ARL in the guarded flag-collection.

Abstract Policy	Probability of Reaching 'goal'	Expected Reward
A	0.9	2.85
B	0.81	3.62
C	0.78	4.5

The baseline experiment, which was a standard RL implementation of the case study, used an $\varepsilon = 0.8$ and performed 2×10^7 learning episodes, each with 10,000 steps. This did not, however, reach a global optimum. Even after extensive learning runs, in excess of 10^9 learning episodes, conventional RL did not attain a superior solution. In contrast, in our experiments where ARL was used (cf. abstract policy C, Table 3), a superior policy was learned much faster, further demonstrating the advantages of our approach. Figure 3 shows the learning progress for this experiment.

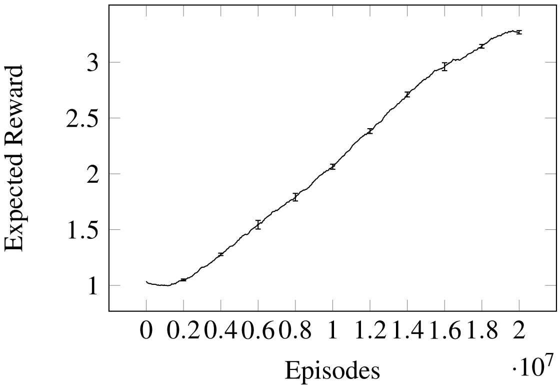

Figure 3: Learning for guarded flag-collection with no ARL applied.

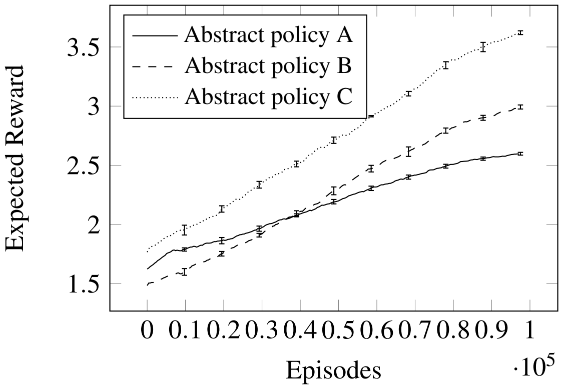

Figure 4: Learning for guarded flag-collection with ARL applied using the selected abstract policies A, B and C.

Table 3: Results for baseline and ARL experiments for guarded flag-collection.

Abstract Policy	Probability of Reaching 'goal'	Standard Error	Expected Reward	Standard Error
None	0.72	0.0073	4.01	0.031
A	0.9	0.0012	2.85	0.0029
B	0.81	0.0019	3.62	0.0037
C	0.78	0.0012	4.5	0.0041

Next, we present the three ARL experiments, one for each of the abstract policies from Table 2. Since the abstract policy had the effect of guiding the agent with regard to the locations to enter next, less exploration by the agent was required and fewer learning episodes were necessary. Therefore, we used $\varepsilon = 0.6$ which decayed to zero over the learning run and 10^5 episodes were needed for the learning to converge. Figure 4 shows the RL learning progress for each of the abstract policies used for ARL.

The learned policies for each of the experiments were evaluated and the results summarized in Table 3. The experiments where an abstract policy was applied resulted in an RL policy that: (a) satisfied the problem constraints and optimisation objectives specified in Section 4; and (b) matched the probability of reaching the 'goal' area and the expected reward of the abstract policy (cf. Table 2). The baseline experiment gave results that do not satisfy our constraints, which was expected given that only 14 of the 10,000 abstract policies synthesised by ARL satisfied these constraints.

6.2 Autonomous Assistance for Dementia Sufferer

Dementia is a common chronic illness with significantly debilitating consequences. As the illness progresses, it becomes increasingly difficult for the sufferer to perform even simple tasks, making it necessary for a caregiver to provide assistance with such tasks.

Table 4: Hand washing subtasks.

Subtask	Atomic proposition
Turn tap on	on
Apply soap	soaped
Wet hands under tap	wet
Rinse washed hands	rinsed
Dry hands	dried

To alleviate the duties of the caregiver and the cost to healthcare, the project described in (Boger et al., 2006) has developed an automated system that helps a dementia patient perform the task of washing their hands. For our second case study we used a simulated version of this assisted-living system. For the purpose of our system, the hand-washing task can be decomposed into the subtasks listed in Table 4. This table also shows the atomic propositions (i.e. boolean labels, cf. Section 3) that we will use in this section to indicate whether each of the subtasks has been completed.

It is possible for the dementia sufferer to regress in this task by repeating subtasks they have already performed, or by performing the wrong subtask for the stage of the hand-washing process they have reached. Figure 5 depicts the workflow carried out by a healthy person while progressing with the task (black, continuous-line nodes and arrows) and the possible regressions that a dementia sufferer could make (red dashed-line nodes and arrows). For ease of reference, the states of the workflow is labelled with a state ID (s_1 to s_{12}) and with the atomic propositions that hold in that state.

The probabilities of the dementia sufferer progressing and regressing (not shown in Figure 5) vary at each stage of the task and between sufferers. For the purpose of our evaluation, we carefully decided these probabilities based on the subtask complexity, as indicated in (Boger et al., 2006).

The system is designed so that if the user fails to perform one of the next correct subtasks then it may provide a voice prompt instructing the user what subtask to do next. The system learns what style of voice is most appealing to the user based on how conducive different styles of prompt are at the user succeeding with the overall task. Voice styles vary in gender, sternness of the instructions (mild, moderate or strict) and volume (soft, medium or loud). The appeal of the voice style will induce an increase in the probability that the dementia sufferer progresses compared to no prompt being given, with the least appealing voice

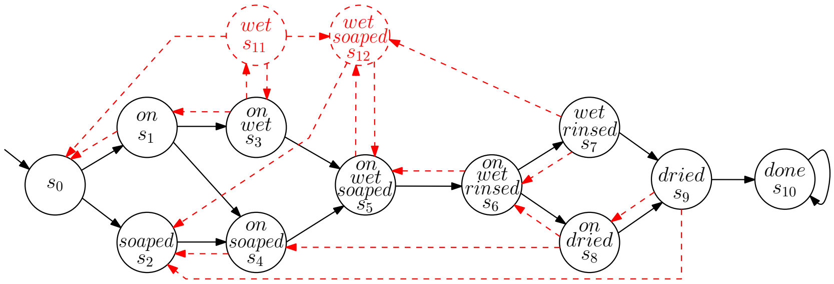

Figure 5: Workflow of washing hands, showing the subtasks at each stage of progress with the progression of a healthy person in black continuous lines and the possible regressions of a dementia sufferer in red dashed lines.

Table 5: Constraints and optimisation objectives for the assisted-living system.

ID	Constraint (C) or optimisation objective (O)	PCTL
C_1	The probability that the caregiver provides assistance should be at least 0.05	$\text{P}_{\geq 0.05}[\text{F}\, m = \mathit{MAX\_MISTAKES}]$
C_2	The probability that the caregiver provides assistance should be at most 0.2	$\text{P}_{\leq 0.2}[\text{F}\, m = \mathit{MAX\_MISTAKES}]$
O_1	The level of dementia sufferer distress due to multiple voice prompts should be minimised	minimise $R^{\text{distress}}_{=?}[\text{F}\, \mathit{done} \vee m = \mathit{MAX\_MISTAKES}]$
O_2	The probability of calling the caregiver should be minimised (subject to C_1 and C_2 being satisfied)	minimise $P_{=?}[\text{F}\, m = \mathit{MAX\_MISTAKES}]$

yielding the smallest increase and the most appealing yielding the largest increase.

For our system we wish to determine when to give a prompt to the user and when it becomes necessary to call the caregiver (because the user is not making progress despite repeated prompts). Overloading the user with prompts can become stressful, and therefore each prompt has a negative reward of -1. Whilst calling the caregiver will be of relief to the user as well as ensuring the completion of the task, doing it too frequently will become stressful to the caregiver or, in a care home, will overstretch the personnel resources available. Therefore, the caregiver should assist on some occasions, but most of the time not; thus the action to call the caregiver has a negative reward of -300. Completing the task results in a reward of 500. Note that the rewards for calling the caregiver and for completing the task are only necessary in the RL simulation for learning to appropriately progress and are not necessary in the AMDP.

Finally, we desire that the caregiver be present at least once every one-to-four days, to ensure that the sufferer receives the caregiver's attention regularly. Assuming that a person washes their hands approximately five times a day, the probability that the caregiver should assist the dementia sufferer during any one hand-washing should be between $^1/_{20}$ and $^1/_5$, i.e. between 0.05 and 0.2. This constraint, and an additional, manually-specified optimisation objective for the abstract policy synthesis stage of ARL can be formalised in PCTL as shown in Table 5, where m is the number of mistakes made at any given time, *MAX_MISTAKES* is the threshold for the maximum number of mistakes that result in calling the caregiver, *distress* is the reward structure for stress to the dementia sufferer, and *done* is the atomic proposition associated with the completion of the hand-washing task by the user (cf. Figure 5).

We constructed the AMDP for this system based on the workflow shown in Figure 5, where each workflow stage represents a different AMDP state. To abstract the RL MDP we only used in the AMDP the transition probabilities and the best style of prompt which the RL agent aims to learn. We encoded an abstract policy for this AMDP using an array of 12 parameters, one for each stage of the task from Figure 5 other than stage 10 (where the task is complete). The parameter associated with each workflow stage represents the minimum number of total user mistakes that warrant giving a prompt at that stage. Each parameter can take values between zero (always give a voice prompt) and the maximum number of mistakes allowed before calling the caregiver (never give a voice prompt).

We generated 10,000 abstract policies using random search, and we used the probabilistic model

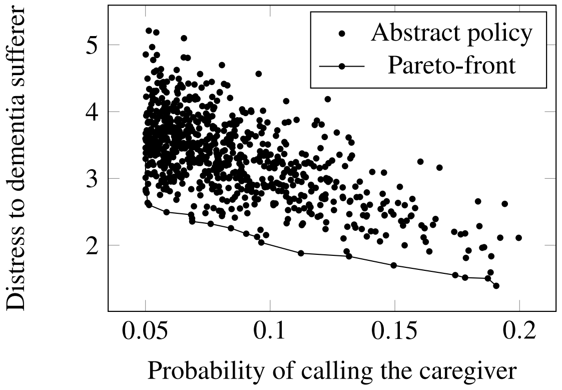

Figure 6: Abstract policies and Pareto-front for the assisted-living system.

Table 6: Selected abstract policies used during the safe learning stage of ARL for the assisted-living system.

Abstract Policy	**Probability of Calling the Caregiver**	**Distress to Dementia Sufferer**
A	0.08	2.17
B	0.13	1.70
C	0.17	1.38

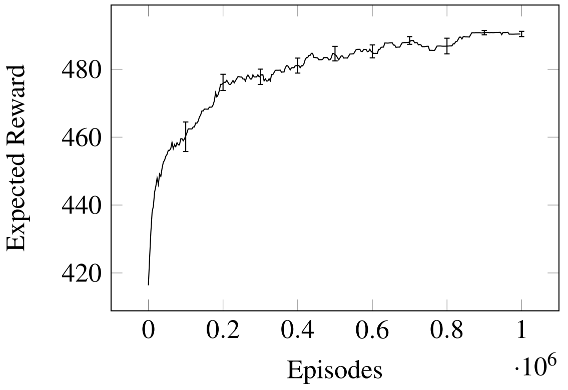

Figure 7: Learning for assisted-living system with no ARL applied.

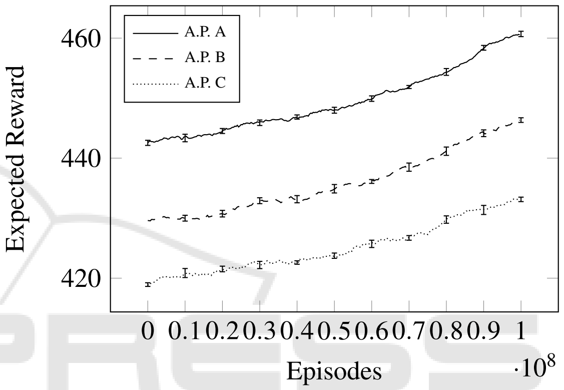

Figure 8: Learning for assisted-living system with ARL applied using the selected abstract policies A, B and C.

checker PRISM to identify the 786 abstract policies that satisfied constraints C_1 and C_2 from Table 5. Two optimisation objectives were used to assemble the approximate Pareto-front (and the set of Pareto-optimal abstract policies) in the abstract policy synthesis stage of ARL. The former objective was objective O_1 from Table 5. The latter objective (O_2 from Table 5) was derived from constraint C_2, i.e. we aimed to minimise the probability of calling the caregiver. Figure 6 shows the entire set of safe abstract policies, as well as the Pareto-front. For the last stage of ARL (safe learning), we carried out experiments starting from three abstract policies from different areas of the Pareto-front shown in Figure 6. Table 6 lists these three abstract policies with their associated attributes (i.e. the probability to call the caregiver and the level of distress to the dementia sufferer).

We chose a value of $\varepsilon = 0.5$ for all experiments in this case study. Figure 7 shows the average learning of all five learning runs of the baseline experiment (without ARL), with error bars used to show the standard error of the mean. For this experiment 1×10^6 episodes were necessary to reach an optimal policy and each episode had a maximum of 1,000 steps.

Following the baseline experiment, we carried out a series of experiments for each of the three selected abstract policies from Table 6, the learning progress of these experiments is shown in Figure 8. More learning episodes were necessary for the ARL experiments since for many states the abstract policies prevented a prompt being given, delaying the agent's ability to explore and learn about different prompt styles.

Contrasting the results from the baseline experiment and the ARL experiments, it is clear that the action constraints are having the expected effect on the learned policy. In particular, comparing the probabilities of calling the caregiver and the level of distress to the dementia sufferer against those that were verified for the abstract policies, the action constraints are having the desired effect, with all the results being close to or matching the values shown in Table 6. The slight difference from the abstract policy C's probability of calling the caregiver can be attributed to the policy not being entirely optimal and further learning should reduce the variance to zero.

7 CONCLUSION

For assured RL we proposed the use of an abstract MDP formally analysed using quantitative verification. Safe abstract policies are identified and used as

Table 7: Results from baseline and ARL experiments for the assisted-living system.

Abstract Policy	Probability of Calling the Caregiver	Standard Error	Distress to Patient	Standard Error
None	4.02×10^{-4}	4.28×10^{-4}	8.31	4.03×10^{-3}
A	0.08	4.95×10^{-4}	2.17	3.25×10^{-3}
B	0.13	5.17×10^{-4}	1.70	2.22×10^{-3}
C	0.18	4.27×10^{-4}	1.38	1.84×10^{-3}

a means to restrict the action set of an RL agent to those actions that were proven to satisfy a set of requirements, adding to the growing research on safe RL. Through two qualitatively different case studies, we demonstrated that the ARL technique can be applied successfully to multiple problem domains. ARL assumes that partial knowledge of the problem is provided a priori, and makes the typical assumption that with sufficient learning the RL agent will converge towards an optimal policy.

ARL supports a wide range of safety, performance and reliability constraints that cannot be expressed using a single reward function in standard RL and are not supported by existing safe RL techniques. Furthermore, the use of an AMDP allows the application of ARL where only limited knowledge of the problem domain is available, and ensures that ARL scales to much larger and complex models than would otherwise be feasible. Additionally, the expressive nature of PCTL formulae and the construction of the AMDP enables convenient on the fly experimentation of constraints and properties without requiring modification of the underlying model.

Future work on ARL includes researching a means of updating the AMDP should it not accurately reflect the RL MDP. In the event that the RL agent encounters information in the RL MDP that does not correlate with the AMDP, or should the RL system dynamics change during runtime, a means of feeding back this information to update the AMDP can be developed, e.g. based on (Calinescu et al., 2011; Calinescu et al., 2014; Efthymiadis and Kudenko, 2015). After updating the AMDP the constraints will need to be reverified and, if necessary, a new abstract policy will be generated.

Additionally, we intend to exploit some of the more sophisticated constraints that can be specified in PCTL. For example, bounded until PCTL formulae can place constraints on the number of time steps taken to achieve a certain outcome, e.g. for our assisted-living case study the formula $\mathrm{P}_{\geq 0.75}[\neg(s_{11} \vee s_{12})\,\mathrm{U}^{\leq 8}\,s_6]$ requires the agent to ensure, with a probability of at least 0.75, that the user washes their hands with water and soap and then rinses them (thus reaching stage s_6 of the workflow from Figure 5) within eight subtasks, without switching the tap on and off unnecessarily (by entering stages s_{11} or s_{12} of the workflow).

ACKNOWLEDGEMENTS

This paper presents research sponsored by the UK MOD. The information contained in it should not be interpreted as representing the views of the UK MOD, nor should it be assumed it reflects any current or future UK MOD policy.

REFERENCES

Abe, N., Melville, P., Pendus, C., et al. (2011). Optimizing debt collections using constrained reinforcement learning. In *16th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining*, pages 75–84.

Andova, S., Hermanns, H., and Katoen, J.-P. (2004). Discrete-time rewards model-checked. In *Formal Modeling and Analysis of Timed Systems*, pages 88–104.

Arcuri, A. and Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *33rd Intl. Conf. Software Engineering*, pages 1–10.

Barrett, L. and Narayanan, S. (2008). Learning all optimal policies with multiple criteria. In *25th Intl. Conf. Machine learning*, pages 41–47.

Boger, J., Hoey, J., Poupart, P., et al. (2006). A planning system based on markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 10(2):323–333.

Calinescu, R., Johnson, K., and Rafiq, Y. (2011). Using observation ageing to improve Markovian model learning in QoS engineering. In *2nd Intl. Conf. Performance Engineering*, pages 505–510.

Calinescu, R., Johnson, K., and Rafiq, Y. (2013). Developing self-verifying service-based systems. In *28th IEEE/ACM Intl. Conf. on Automated Software Engineering*, pages 734–737.

Calinescu, R., Kikuchi, S., and Johnson, K. (2012). Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In *Large-*

Scale Complex IT Systems. Development, Operation and Management, pages 303–329. Springer.

Calinescu, R., Rafiq, Y., Johnson, K., and Bakir, M. E. (2014). Adaptive model learning for continual verification of non-functional properties. In *5th Intl. Conf. Performance Engineering*, pages 87–98.

Castro, D. D., Tamar, A., and Mannor, S. (2012). Policy gradients with variance related risk criteria. In *29th Intl. Conf. Machine Learning*, pages 935–942.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian Q-learning. In *15th National Conference on Artificial Intelligence*, pages 761–768.

Delage, E. and Mannor, S. (2010). Percentile optimization for Markov decision processes with parameter uncertainty. *Operations Research*, 58(1):203–213.

Driessens, K. and Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304.

Efthymiadis, K. and Kudenko, D. (2015). Knowledge revision for reinforcement learning with abstract MDPs. In *14th Intl. Conf. Autonomous Agents and Multiagent Systems*, pages 763–770.

Gábor, Z., Kalmár, Z., and Szepesvári, C. (1998). Multicriteria reinforcement learning. In *15th Intl. Conf. Machine Learning*, pages 197–205.

García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

Geibel, P. (2006). Reinforcement learning for MDPs with constraints. In *17th European Conference on Machine Learning*, volume 4212, pages 646–653.

Gerasimou, S., Calinescu, R., and Banks, A. (2014). Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In *9th Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 115–124.

Gerasimou, S., Tamburrelli, G., and Calinescu, R. (2015). Search-based synthesis of probabilistic models for quality-of-service software engineering. In *30th IEEE/ACM Intl. Conf. Automated Software Engineering*, pages 319–330.

Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535.

Katoen, J.-P., Zapreev, I. S., Hahn, E. M., et al. (2011). The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721.

Kwiatkowska, M. (2007). Quantitative verification: Models, techniques and tools. In *6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 449–458.

Kwiatkowska, M., Norman, G., and Parker, D. (2007). Stochastic model checking. In *7th Intl. Conf. Formal Methods for Performance Evaluation*, volume 4486, pages 220–270.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *23rd Intl. Conf. Computer Aided Verification*, volume 6806, pages 585–591.

Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *9th International Symposium on Artificial Intelligence and Mathematics*, pages 531–539.

Liu, C., Xu, X., and Hu, D. (2015). Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398.

Mannor, S. and Shimkin, N. (2004). A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, 5:325–360.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *24th Intl. Conf. Machine learning*, pages 601–608.

Mason, G., Calinescu, R., Kudenko, D., and Banks, A. (2016). Combining reinforcement learning and quantitative verification for agent policy assurance. In *6th Intl. Workshop on Combinations of Intelligent Methods and Applications*, pages 45–52.

Mihatsch, O. and Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine Learning*, 49(2):267–290.

Moffaert, K. V. and Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15(1):3663–3692.

Moldovan, T. M. and Abbeel, P. (2012). Safe exploration in Markov decision processes. In *29th Intl. Conf. Machine Learning*, pages 1711–1718.

Ponda, S. S., Johnson, L. B., and How, J. P. (2013). Risk allocation strategies for distributed chance-constrained task allocation. In *American Control Conference*, pages 3230–3236.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

Szita, I. (2012). Reinforcement learning in games. In *Reinforcement Learning*, pages 539–577. Springer.

Vamplew, P., Dazeley, R., Berry, A., et al. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1):51–80.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Wiering, M. and Otterlo, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement Learning: State-of-the-art*, volume 12, pages 3–42. Springer.

Xia, L. and Jia, Q.-S. (2013). Policy iteration for parameterized markov decision processes and its application. In *9th Asian Control Conference*, pages 1–6.