

# Learning Optimal Behavior in Environments with Non-Stationary Observations

Ilio Boone<sup>1</sup> and Gavin Rens<sup>1</sup> <sup>a</sup>

<sup>1</sup>*DTAI group, KU Leuven, Belgium*  
*ilio.boone@hotmail.com, gavin.rens@kuleuven.be*

**Keywords:** Markov Decision Process, Non-Markovian Reward Models, Mealy Reward Model (MRM), Learning MRMs, Non-Stationary

**Abstract:** In sequential decision-theoretic systems, the dynamics might be Markovian (behavior in the next step is independent of the past, given the present), or non-Markovian (behavior in the next step depends on the past). One approach to represent non-Markovian behaviour has been to employ deterministic finite automata (DFA) with inputs and outputs (e.g. Mealy machines). Moreover, some researchers have proposed frameworks for *learning* DFA-based models. There are at least two reasons for a system to be non-Markovian: (i) rewards are gained from temporally-dependent tasks, (ii) observations are non-stationary. Rens et al. (2021) tackle learning the applicable DFA for the first case with their ARM algorithm. ARM cannot deal with the second case. Toro Icarte et al. (2019) tackle the problem for the second case with their LRM algorithm. In this paper, we extend ARM to deal with the second case too. The advantage of ARM for learning and acting in non-Markovian systems is that it is based on well-understood formal methods with many available tools.

## 1 INTRODUCTION

The field of sequential decision-making has been extensively researched in academic literature, from developing optimal solution algorithms for the standard settings (Ross, 2014; White III and White, 1989), to learning how to apply these methods in a more complex environment (Bacchus et al., 1996; Singh et al., 1994). The traditional setting is the Markov Decision Process (MDP), where the environment is Markovian and all dynamics depend only on the current state. This assumption is often insufficient to present real-life problems, as this property may be violated.

Two reasons that might cause an environment (and its MDP model) to be non-Markovian are (i) temporal dependencies in the tasks to be solved and (ii) observations being non-stationary.

In the case where there are temporal dependencies in the tasks, we associate decision processes with *non-Markovian reward decision processes* (NMRDPs). Here, the reward received depends on the actions performed in the past. The reward function now has a *temporal* dependence. In order for the agent to know which action to perform to receive a higher reward, it thus has to keep track of its past actions and

states it visited.

The case where observations are non-stationary cannot occur in (fully observable) MDPs (nor in fully observable NMRDPs). However, this case is possible in partially observable MDPs (POMDPs), where an agent is unable to clearly distinguish states from each other and only receives an observation from the environment. In order for the agent to know which action to perform to receive a higher reward, it thus has to keep track of its past observations.

The ARM framework, as developed by Rens et al. (2021), has been shown to converge to an optimal solution for NMRDPs. In this paper, we re-evaluate the ARM algorithm for processes with non-stationarity and partial observability. We investigate to what extent Deterministic Finite Automata (DFA) can be used to learn the reward structure of POMDPs with non-stationary observations. The limitations of the ARM framework are investigated, using a version of the Cookie Domain, introduced by Toro Icarte et al. (2019), and we show how the ARM framework can be extended appropriately.

In the next three sections, we review the related work, we present the illustrative problem (the Cookie Domain), and some basic formal concepts are reviewed. Section 5 describes the ARM algorithm and Section 6 discusses how we extend ARM to overcome


<sup>a</sup>  <https://orcid.org/0000-0003-2950-9962>



Figure 1: The Cookie domain with a button in the yellow room and agent (A) in the blue room.

its shortcoming w.r.t. the Cookie Domain (and similar domains), and we proved theoretical results. The paper ends with a discussion and concluding remarks.

## 2 Related Work

As mentioned, this paper is concerned with learning non-Markovian reward functions with non-stationary observations, for which the ARM framework is used (Rens et al., 2021). The learning is based on Angluin’s  $L^*$  algorithm (Angluin, 1987), which asks a finite number of queries about the reward behaviour and stores the associated rewards to these queries. A DFA, in particular a Mealy Reward Machine (MRM) is then constructed. This MRM provides the memory necessary to maximize its rewards.

The use of traditional automata to represent the complex structure of reward functions in MDPs has been investigated by others as well. Gaon and Brafman (2020) and Xu et al. (2021) also create DFAs to represent the reward structure of complex MDPs, relying on the  $L^*$  algorithm. However, contrary to the implementation in the ARM framework, a tag for each trace/query is provided. This tag reflects whether the corresponding trace can be recreated in the environment within a certain number of attempts. The authors conclude that the  $L^*$  algorithm works well in short, simple reward models, but has difficulties in more complex environments.

Toro Icarte et al. (2019) employ a DFA in a non-stationary environment to represent the reward structure. The authors classify the problem in their work as a POMDP, as there are elements of the domain that are unobservable at certain moments. One of the problems that is investigated by Toro Icarte et al. (2019), is referred to as the ‘Cookie Domain’, which also has a non-Markovian reward structure. In Section 4 the problem is formally defined and the non-stationarity is illustrated.

In non-Markovian environments, memory is often required to make more accurate predictions regarding the (probability of the) current state the agent is in or to know what reward will be given to the agent. It is in this regard that POMDPs, NMRDPs and non-

stationary MDPs seem to share some properties, and some authors have alluded to their similarity, for instance, classifying POMDPs as a subclass of NMRDPs (Singh et al., 1994).

## 3 FORMAL PRELIMINARIES

A Markov decision process is traditionally defined as a tuple  $\langle S, A, T, R, s_0 \rangle$ , where  $S$  represents a finite set of states and  $A$  is a finite set of actions.  $T : S \times A \mapsto [0, 1]$  defines the state transition function such that  $T(s, a, s')$  is the probability that action  $a$  causes a system to transition from  $s$  to  $s'$ .  $R : A \times S \mapsto \mathbb{R}$  is the reward function such that  $R(a, s)$  is the immediate reward for performing  $a$  in state  $s$ .  $s_0$  is the initial state of the system. Bacchus et al. (1996) define an NMRDP as a tuple  $\langle S, A, T, R, s_0 \rangle$ , where all elements have the same meaning as in the MDP, except for the reward function. The reward function now takes as its domain a trace of states and actions, or a history, instead of individual states. Let  $H^{fo}$  be the set of all fully observed histories. A history  $\alpha_h \in H^{fo}$  is represented in the form  $s_1 a_1, s_2 a_2, \dots, a_{k-1} s_k$ . A non-Markovian reward function is  $nmR : \alpha_h \mapsto \mathbb{R}$ . It is sometimes useful to separate an NMRDP into a non-rewarding MDP (nrMDP)  $\langle S, A, T, s_0 \rangle$  and a non-Markovian reward function  $nmR$ .

A POMDP is a generalization of the traditional MDP and can be represented as a tuple  $\langle S, A, T, R, Z, \omega, b_0 \rangle$  (Kaelbling et al., 1998).  $S, A, T$ , and  $R$  are defined as in the traditional MDP.  $Z$  is a finite set of observations and  $\omega : Z \times A \times S \mapsto [0, 1]$  is the probability distribution over possible observations such that  $\omega(z, a, s)$  is the probability that observation  $z$  is perceived in state  $s$ , reached by performing action  $a$ . The initial probability distribution over  $S$  (i.e. belief state) is  $b_0$ . For POMDPs, we introduce partially observable histories. A history  $\alpha_h \in H^{po}$  is represented in the form  $a_1, z_1, a_2, z_2, \dots, a_k, z_k$ .

The notion of a belief state is often employed, that is, a probability distribution over all states. It is computed by  $k$  applications of belief update function  $BU : B \times A \times Z \mapsto B$ , where  $B$  is the space of belief states (Kaelbling et al., 1998).

**Definition 3.1.** Given belief state  $b_t$  and history  $\alpha_h = a_1, z_1, a_2, z_2, \dots, a_k, z_k$ , the history-based belief update function is  $BU_H(b_t, \alpha_h) = b_{t+k} \doteq BU(BU \dots BU(b_t, a_1, z_1), \dots, a_{k-1}, z_{k-1}), a_k, z_k)$ . The expected reward of performing action  $a$  in belief state  $b_{\alpha_h}$  is  $\sum_{s \in S} R(a, s) b_{\alpha_h}(s)$ .

A reward machine (defined below) uses a labelling function  $\lambda$  to map action-state pairs onto observations about the environment.  $\lambda : A \times S \mapsto Z \cup$

$\{\text{null}\}$ , where  $\lambda(a, s) = z$  means that  $z$  is observed in state  $s$ , reached via action  $a$ .<sup>1</sup>

**Definition 3.2** ((Rens et al., 2021)). *Given a set of states  $S$ , a set of actions  $A$  and a labeling function  $\lambda$ , a Mealy Reward Machine is a tuple  $\langle U, u_0, Z, \delta_u, \delta_r, c \rangle$ , where  $U$  is a finite set of MRM nodes,  $u_0 \in U$  is the start node,  $Z \uplus \{\text{null}\}$  is a set of observations.  $\delta_u : U \times Z \mapsto U$  is the transition function, such that  $\delta_u(u_i, \lambda(a, s)) = u_j$  for  $a \in A$  and  $s \in S$ , specifically,  $\delta_u(u_i, \text{null}) = u_i$ .  $\delta_r : U \times Z \mapsto \mathbb{R}$  is the reward-output function, such that  $\delta_r(u_i, \lambda(a, s)) = r'$  for  $r' \in \mathbb{R}$ ,  $a \in A$  and  $s \in S$ , specifically,  $\delta_r(u_i, \text{null}) = c$ .  $c$  is the default reward for the non-significant observation  $\text{null}$ .*

In order to create a process with Markovian behavior from an NMRDP with an MRM-based reward model, one can take the *synchronized product* of the NMRDP and the MRM. The product incorporates sufficient information regarding the environment in order to act optimally.

**Definition 3.3** ((Rens et al., 2021)). *Given an n-MDP  $M = \langle S, A, T, s_0 \rangle$ , a labeling function  $\lambda : A \times S \mapsto Z \uplus \{\text{null}\}$  and an MRM  $\mathcal{R} = \langle U, u_0, Z, \delta_u, \delta_r \rangle$ , the synchronized product of  $M$  and  $\mathcal{R}$  under  $\lambda$  is defined as an (immediate reward) MDP  $P = M \otimes_{\lambda} \mathcal{R} = \langle S^{\otimes}, A^{\otimes}, T^{\otimes}, R^{\otimes}, s_0^{\otimes} \rangle$ , where  $S^{\otimes} = S \times U$ ,  $A^{\otimes} = A$ .  $T^{\otimes}((s, u), a, (s', u')) = T(s, a, s')$  if  $u' = \delta_u(u, \lambda(a, s'))$  and 0 otherwise.  $R^{\otimes}(a, (s, u)) = \delta_r(u, \lambda(a, s))$ ,  $s_0^{\otimes} = (s_0, u_0)$ .*

## 4 COOKIE DOMAIN

In the following sections, we investigate how to use an MRM to represent reward structures and how the ARM framework can learn this MRM. This is done with a particular problem in mind, namely the Cookie domain inspired by Toro Icarte et al. (2019). The Cookie domain is first presented in a fully observable and stationary environment in order to provide a full understanding of the dynamics. Then the Cookie domain with partial and non-stationary observations is described.

### 4.1 Fully Observable Cookie Domain

In its simplest form, the Cookie domain is represented as a world with three distinct rooms, which are blue, red and yellow (Fig. 1). An agent acts in the environment, where it can go to each of the three rooms. It can press a button in the yellow room, after which

<sup>1</sup>Observation *null* is added to denote non-significant states. The agent is able to distinguish those from significant observations, but this is not strictly necessary.

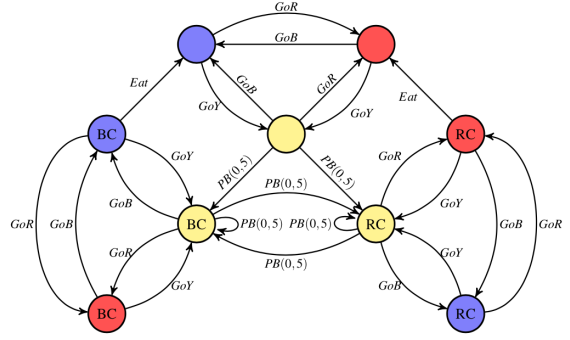


Figure 2: Transition function for fully observable Cookie domain.

a cookie will spawn in either the red or blue room, with equal probability. The agent has to go to the room where the cookie is located and when it eats the cookie, it will receive a reward. The non-stationarity in this problem is related to the need to press the button before a cookie spawns.

The agent starts in the blue room, without a cookie in the environment. Nine states can be distinguished, given that there can exist at most one cookie at any moment. The set of actions that the agent can perform consists of *goB*, *goR*, *goY* (Go to the Blue, Red or Yellow Room), *PressButton* (*PB*) (Try to press the button) and *Eat* (Try to eat the cookie). The states, actions and transitions of the domain are presented in Figure 2, where the color of the node corresponds to the room that the agent is in. *BC* and *RC* denote that a cookie is present is either the Blue or the Red room. The actions that result in the agent staying in the same state with probability one are not shown. The probability of all transitions between two different states is always equal to one, except for when the button in the yellow room is pressed, for which the probabilities is  $0,5$ .<sup>2</sup>

A reward is received when the agent performs the *Eat* action in the same room where a cookie is present. All other actions give no reward. This reward function is Markovian and the Cookie domain is defined as a traditional MDP. Traditional solving techniques can be used to derive an optimal policy for this problem (Bellman, 1957; Kolobov, 2012; Bertsekas, 2012).

### 4.2 Cookie Domain with Uncertainty

The Cookie domain can be made more complex, by withholding information about the environment from the agent. Suppose that the agent only knows what

<sup>2</sup>Transitions are allowed to be stochastic, but for clarity of illustration, we assume them to be deterministic (except for the button pressing action).

room it is in. It has no information regarding the location of the cookie and has no memory of his actions in the past. The state space is now represented by just three distinct states:  $B$  (Blue Room),  $R$  (Red Room) and  $Y$  (Yellow Room).

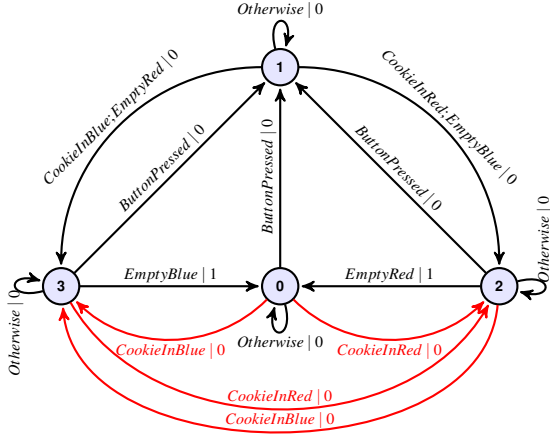


Figure 3: MRM learned in the Extended ARM Framework

The set of actions remains the same as in the original problem, with an extra action  $Look$ , which allows the agent to see whether a cookie is present in the current room. The transitions in this problem remain as before and the agent stays in the same state when performing the  $Look$  action.

By withholding information about the cookie in the states, it becomes impossible to define a Markovian reward function, as it now has a temporal dependence, a non-Markovian behavior. Furthermore, the agent is uncertain about the location of the cookie once it has spawned and has to find out where it is through observations, which are non-stationary due to the dependence on when the button is pressed.

Toro Icarte et al. (2019) define a 'perfect MRM' as the smallest DFA sufficient to represent the reward structure of a problem. The 'perfect MRM' for our implementation of the Cookie domain is given in Figure 3 (ignore the transitions in red; they are for a later discussion).

In order to formally analyze this kind of environment, we define the non-stationary and deterministic observation MDP below. It relies critically on the notion of a *non-stationary* labeling function:

**Definition 4.1.** A non-Markovian, deterministic labeling function  $\lambda : A \times H \mapsto Z \uplus \{null\}$  is a mapping from an action-history pair to an observation. A non-stationary labeling function  $\lambda^{ns} : A \times S \mapsto 2^{Z \uplus \{null\}}$  is a mapping from an action-state pair to one observation in one of the sets in  $2^{Z \uplus \{null\}}$ . Furthermore, for any history  $\alpha_h \in H$ ,  $\lambda(a, \alpha_h) \in \lambda^{ns}(a, last(\alpha_h))$ , where

$$last(s_1 a_1, s_2 a_2, \dots, a_{k-1} s_k) = s_k.$$

**Definition 4.2.** A non-stationary and deterministic observation MDP (NS-DOMDP) is a tuple  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$ , where all elements of the tuple except for  $\lambda^{ns}$  are as usual for a (regular) POMDP.

Because of the partial observability in the uncertain Cookie domain, the labelling function becomes non-stationary:  $\lambda^{ns}(Look, B) \in \{EmptyBlue, CookieInBlue\}$ ,  $\lambda^{ns}(Look, R) \in \{EmptyRed, CookieInRed\}$ ,  $\lambda^{ns}(Eat, B) = EmptyBlue$ ,  $\lambda^{ns}(Eat, R) = EmptyRed$ ,  $\lambda^{ns}(PressButton, Y) = ButtonPressed$ ,  $\lambda^{ns}(a, s) = null$  otherwise.

The synchronized product (cf. Sect. 3) can then be computed, using the MRM and the state space. If the ARM algorithm is able to learn this MRM, then the synchronized product and a corresponding optimal policy can be computed for the Cookie domain (cf. Sect. 6.2). In the next section, it is discussed where the ARM has difficulties with the dynamic and uncertainty aspects and how it can be extended in order to deal with them.

## 5 THE ARM ALGORITHM

The ARM framework for learning in NMRDPs, as developed by Rens et al. (2021), consists of a learning and an exploitation step. The learning step uses the L\* algorithm Angluin (1987), in order to build an Observation Table (OT) from a finite set of Membership Queries (MQ), from which an MRM is built. In what follows, the fundamentals of the ARM framework are explained, with focus on how it answers MQs. Two procedures of the ARM algorithm are discussed: (i) generating MQs and (ii) collecting their rewards. We refer to Rens et al. (2021) for details.

**Generating Membership Queries** This procedure generates observation traces for the OT, which have not been queried yet and are required to make the OT complete. To achieve this, it requires an input alphabet  $I$ , which consists of a finite set of letters (Angluin, 1987). In the context of NMRDPs, these letters correspond to observations in the environment, which are the output of a labelling function  $\lambda$ . New observation traces get generated, with different orderings and lengths, until the OT is complete.

**Collecting Reward Traces** As transitions in MDPs are stochastic, there is no sequence of actions that guarantees the occurrence of a particular history  $\alpha_h$ . MQs correspond to observation traces  $\alpha_z$ , which can

be extracted from a history  $\alpha_h$  using a labelling function, answering MQ’s essentially comes down to a planning problem. The procedure is the implementation of this planning problem in the ARM algorithm, which forces a particular  $\alpha_z$  to occur. The procedure derives a policy  $\pi(MIN)$  for which the expected number of steps to reach the sequence of observations is minimized.<sup>3</sup> The output of this procedure is a deterministic stationary policy, stored in a scheduler, which provides an action to perform in each state of the environment, dependent on the observation that is being queried. When a new sequence of observations is queried, the procedure creates a temporary MDP. This temporary MDP is created by using the states of the problem and iteratively going over all actions that can be performed in each of the states and investigating the corresponding observations. If a state-action pair leads to the wrong observation, this state is not added to the temporary MDP. States in the temporary MDP consist of the original state plus an index. This index starts as 0 and reflects the number of observations of  $\alpha_z$  that have been made, in the correct order. The created scheduler provides actions to reach a state with an index equal to the length of  $\alpha_z$ , meaning the trace was successfully recreated.

## 6 EXTENDING THE ARM ALGORITHM

In this section, the difficulties of the ARM algorithm with respect to non-stationary observations / partial observability are explored. The ARM algorithm is extended in some regards in order to be able to deal with these complexities. The challenge for the ARM algorithm and the labeling function we define is to deal with MQs in a principled way. The pseudo-code for the learning phase of the extended ARM algorithm is shown in Algorithm 1; it is only fully defined by the time  $M_{\otimes}(\mathcal{P}, \mathcal{R})$  is defined in Section 6.2. Once the MRM is built, the exploitation phase remains as in the original framework. Finally, the advantage of using the extended ARM algorithm for environments similar to the Cookie domain is discussed, including some theoretical results.

*We confirmed experimentally that the ‘perfect’ MRM is learned in the Cookie domain by the extended ARM framework, whereas the original framework was unable to learn and exploit the Cookie domain.*

<sup>3</sup>There is also a *MAX* mode which we do not consider in this work.

---

### Algorithm 1 Extended ARM learning

---

```

Initialize observation table  $OT$ 
if  $OT$  is not complete then
   $\alpha_z \leftarrow \text{getMQ}(OT)$ 
   $\alpha'_z \leftarrow \text{makeFeasible}(\alpha_z)$ 
  while  $\mu = \text{false}$  do
     $s \leftarrow \text{getExperience}(\alpha'_z, s_0)$ 
    if Last observation in  $\alpha'_z$  is found then
       $\mu = \text{true}$ 
    end if
  end while
   $\alpha_r \leftarrow \text{resolveMQ}(s, \alpha'_z)$ 
   $\text{addToOT}(\alpha'_z, \alpha_r)$ 
else
   $\mathcal{H} \leftarrow \text{buildRewardMachine}(OT)$ 
   $M \leftarrow M_{\otimes}(\mathcal{P}, \mathcal{H})$  where  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$ 
  Exploitation phase using  $M$  ...
end if

```

---

## 6.1 Extensions

### 6.1.1 Extension 1

As discussed in the previous section, the learning of MRMs in the ARM framework comes down to answering a finite number of MQs. MQs are answered in the ARM algorithm by creating a scheduler, which is used to recreate  $\alpha_z$  in the environment and storing the received reward. With non-stationary observations, it may be impossible to find a correct scheduler. In the Cookie domain, for example, a cookie can only be observed after pressing the button. If  $\alpha_z$  starts with the observation *CookieInRed*, it is impossible to recreate this  $\alpha_z$ . A *ButtonPressed* observation has to occur first.

To allow for the learning of the MRM in environments with non-stationary observations, some additions are made to the original ARM L\* algorithm. In algorithm 1, the `getMQ()` procedure corresponds to the generation of MQs. MQs get generated by combining all of the observations into observations traces, with different orderings and lengths, if they are not already present in the OT. In order to deal with non-stationarity, each observation trace  $\alpha_z$  is now checked for feasibility and if necessary, observations are inserted in  $\alpha_z$ . In general, this strategy requires domain knowledge to identify infeasible MQs and knowing which observations, when inserted, would make the MQ feasible and where they have to be inserted. Although it was not investigated for this paper, it should be possible to implement a procedure that learns (with certain probability) how to make infeasible MQs feasible, while answering MQs simultaneously.

Hence, a new function `makeFeasible()` was defined specifically for the Cookie domain, which takes as input an observation trace  $\alpha_z$  and returns trace  $\alpha'_z$ .

Note that  $\alpha'_z$  is not always different from  $\alpha_z$ , only when  $\alpha_z$  is not feasible/realistic. However, due to this extension, special attention is needed for interpreting the constructed MRM. The `makeFeasible()` procedure ensures that all MQs can be answered, but it does not change the assumption that each observation can be made at any moment. Consequently, the constructed MRM still creates a transition for each observation from every node. In this manner, there are some 'impossible' transitions present in the MRM. This affects the interpretability of the created MRM, as it might not be clear which transitions are impossible. The presence of these transitions, however, does not affect the derivation of an optimal policy from the synchronized product: Transitions made in the synchronized product are based on actual observations made in the environment and thus they will never occur when playing an episode. For the Cookie domain, these 'impossible' transitions are displayed in red in Figure 3.

### 6.1.2 Extension 2

In the Observation Table (OT), the reward of each MQ is stored, as received during the recreation of the observation traces  $\alpha'_z$ . Due to the non-stationarity of observations in the Cookie domain, it becomes more difficult to recreate  $\alpha'_z$ . If the observation `CookieInRed` is being searched for, but the cookie spawns in the blue room, the agent will never be able to make the `CookieInRed` observation.

As discussed earlier, a temporary MDP is constructed, which is used to create a scheduler that suggests how to find the observations in  $\alpha'_z$ . In algorithm 1, this is done in the procedure `getExperience()`, which returns a scheduler  $s$ , used to find the reward associated with the MQ in procedure `resolveMQ()`. Even when the necessary observations were not found in the temporary MDP, a scheduler is output. When this happens, procedure `resolveMQ` enters an endless loop, as the scheduler always provides some action, but the correct observation cannot be reached.

Again, the ARM algorithm is extended here such that it will keep creating temporary MDPs until it succeeds in finding all the observations of  $\alpha'_z$  in the right order, whereas it only used one temporary MDP before. An auxiliary boolean variable  $\mu$  is created, which is *true* when every observation in  $\alpha'_z$  was found during the creation of the temporary MDP and *false* otherwise. The initial value of  $\mu$  is *false* and every time a temporary MDP has been fully created and  $\mu$  is still *false*, a new temporary MDP is constructed. The algorithm succeeds in finite time, as each observation trace that is added to the OT is feasible, due to the additions made in the previous procedure. Using the

correct scheduler, `resolveMQ` finds the reward associated with the MQ, and `addToOT()` stores reward and MQ in the OT afterwards, as in algorithm 1.

## 6.2 Theoretical Considerations

Whether the system is modeled as a NMRDP or a POMDP with a non-stationary and deterministic observation function (NS-DOMDP), if the system's behavior is fully described with the addition of a MRM, then the model can be transformed into a (regular) MDP by taking the synchronized product of the non-rewarding MDP (nrMDP), respectively, NS-DOMDP and the MRM. An optimal policy can then be computed for the resulting MDP. The synchronized product of a nrMDP and a MRM was defined by Rens et al. (2021) and in Section 3. Now we show how to construct a (regular) MDP from a NS-DOMDP and an MRM. First, we construct a (regular) POMDP:

**Definition 6.1.** Given a NS-DOMDP  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$  and an MRM  $\mathcal{R} = \langle U, u_0, Z, \delta_u, \delta_r \rangle$ , the synchronized product of  $\mathcal{P}$  and  $\mathcal{R}$  is defined as an (immediate reward) POMDP  $\mathcal{P}_{\otimes} = \mathcal{P} \otimes \mathcal{R} = \langle S^{\otimes}, A^{\otimes}, T^{\otimes}, R^{\otimes}, Z^{\otimes}, \omega^{\otimes}, b_0^{\otimes} \rangle$ , where  $S^{\otimes} = S \times U$ ,  $A^{\otimes} = A$ .  $T^{\otimes}((s, u), a, (s', u')) = T(s, a, s')$  if  $u' = \delta_u(u, \lambda^{ns}(a, s'))$  and 0 otherwise.  $R^{\otimes}(a, (s, u)) = \delta_r(u, \lambda^{ns}(a, s))$  for all  $\alpha_h \in H^{po}$  that lead to  $u$  from  $u_0$ ,  $\delta_r(u, \lambda^{ns}(a, s)) = \sum_{s' \in S} R(a, s') b_{\alpha_h}(s')$ , where  $b_{\alpha_h} = BU_H(b_0, \alpha_h)$ .  $Z^{\otimes} = Z$ ,  $\omega^{\otimes}(z, a, (s, u)) = 1$  if  $z = \lambda^{ns}(a, s)$  and 0 otherwise.  $b_0^{\otimes} = (b_0, u_0)$

Note that the observation function  $\omega^{\otimes}$  is deterministic and Markovian. This means that  $\mathcal{P}_{\otimes}$  can be viewed as a (regular) MDP  $\langle S^{\otimes}, A^{\otimes}, T^{\otimes}, R^{\otimes}, s_0^{\otimes} \rangle$ , if we assume  $b_0 = \{(s_0^{\otimes}, 1)\}$  and where  $T^{\otimes}((s, u), a, (s', u')) = T(s, a, s') \omega^{\otimes}(\lambda^{ns}(a, s'), a, (s', u'))$  if  $u' = \delta_u(u, \lambda^{ns}(a, s'))$ , and 0 otherwise. We denote the MDP constructed from NS-DOMDP  $\mathcal{P}$  and MRM  $\mathcal{R}$  as  $M_{\otimes}(\mathcal{P}, \mathcal{R})$ .

We adapt the definition of a "perfect reward machine" from Toro Icarte et al. (2019) for our definitions of an MRM and labeling function:

**Definition 6.2.** A Mealy reward machine  $\mathcal{R} = \langle U, u_0, Z, \delta_u, \delta_r, c \rangle$  is considered perfect for a NS-DOMDP  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$  if and only if for every trace  $s_0, a_0, \dots, s_t, a_t$  generated by any policy over  $\mathcal{P}$ , the following holds:  $P(s_{t+1}, r_t | s_0, a_0, \dots, s_t, a_t) = P(s_{t+1}, r_t | s_t, x_t, a_t)$  where  $x_0 = u_0$  and  $x_t = \delta_u(x_{t-1}, \lambda^{ns}(a_{t-1}, s_{t-1}))$ .

Recall that  $\lambda^{ns}$  is deterministic, and note that its non-stationarity is 'resolved', given the trace (history)  $s_0, a_0, \dots, s_t, a_t$  (cf. Def. 4.1).

It is known that a POMDP can be viewed as a ‘belief-MDP’ if states represent beliefs (i.e. probability distributions over states), and methods exist for computing optimal policies for POMDPs (Cassandra et al., 1994; Kaelbling et al., 1998). The following theorem (adapted from Toro Icarte et al. (2019) for this work) follows from Definition 6.2.

**Theorem 6.1.** *Given any NS-DOMDP  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$  with a finite reachable belief space, there will always exist at least one perfect MRM for  $\mathcal{P}$  with respect to  $\lambda^{ns}$ .*

*Proof sketch.* By defining  $\omega(z, a, s) = 1$  if  $z = \lambda^{ns}(a, s)$  and 0 otherwise,  $\mathcal{P}$  can be viewed as a (regular) POMDP  $\mathcal{P}' = \langle S, A, T, R, Z, \omega, b_0 \rangle$ . If the belief space  $B$  of  $\mathcal{P}$  reachable from  $b_0$  is finite, we can construct an MRM that keeps track of the current belief state using one MRM state per belief state and emulating their progression using  $\delta_u$  and one propositional symbol for every action-observation pair. Thus, the current belief state  $b_t$  can be inferred from the last observation, last action, and the current MRM state. As such, the equality from Definition 6.2 holds.  $\square$

Another theorem (adapted from Toro Icarte et al. (2019) for this work) follows from Definition 6.2.

**Theorem 6.2.** *Let  $\mathcal{R}$  be a perfect MRM for a NS-DOMDP  $\mathcal{P} = \langle S, A, T, R, Z, \lambda^{ns}, b_0 \rangle$ . Then any optimal policy for  $\mathcal{R}$  w.r.t. the environmental reward is also optimal for  $\mathcal{P}$ .*

*Proof sketch.* Let  $M = M_{\otimes}(\mathcal{P}, \mathcal{R}) = \langle S^{\otimes}, A^{\otimes}, T'^{\otimes}, R^{\otimes}, s_0^{\otimes} \rangle$  be the MDP constructed from  $\mathcal{P}$  and  $\mathcal{R}$  as described above. An optimal policy for  $\mathcal{R}$  is a function  $\pi : S \times U \mapsto A$  such that if  $\pi(s, u) = a$ , then  $z = \lambda^{ns}(a, s)$  is the optimal observation at  $u$ .  $R^{\otimes}$  is based on  $\mathcal{R}$ , and  $S^{\otimes} = S \times U$ . Therefore,  $\pi(s, u) = \pi(s^{\otimes})$  for all  $s^{\otimes} = (s, u)$ , and  $\pi$  is optimal for  $M$ . Since  $T'^{\otimes}$  is defined on  $T$  and  $\delta_u$  of  $\mathcal{R}$ ,  $\pi$  must be optimal for  $\mathcal{P}$ .  $\square$

For the extended ARM framework, the guarantees of the original framework still hold. This means that the algorithm is guaranteed to either find the optimal value for a synchronized product or find a counter-example to the hypothesized MRM. The additions to the exploitation phase ensure that counter-examples are only found if the underlying MRM and the hypothesized MRM are not equivalent.

The main result of Rens et al. (2021) transfers to the extended ARM framework:

**Theorem 6.3.** *Let  $V(\pi_{\mathcal{H}}^*)$  be the value of the optimal policy under the current/hypothesized MRM  $\mathcal{H}$ , let  $\mathcal{P}$  be a NS-DOMDP and let  $\mathcal{R}$  be the perfect MRM for the environment under consideration. Given an expert value  $V_{expert}$ , which is an underestimation of the optimal value  $V(\pi_{\mathcal{R}}^*)$  of  $M_{\otimes}(\mathcal{P}, \mathcal{R})$ , the algorithm will*

*learn  $\mathcal{H}$  such that  $V(\pi_{\mathcal{H}}^*) \geq V_{expert}$  with a probability of one in finite time.*

The proof of Theorem 6.3 follows the same reasoning as given by Rens et al. (2021).

## 7 DISCUSSION AND CONCLUSION

Although the ARM framework could previously learn (and exploit) Mealy reward machines (MRMs) for tasks with temporal dependencies, the framework could not learn MRMs where observations are partially observable and non-stationary. In this paper, we showed how to extend ARM to deal with the latter.

We used the Cookie domain of Toro Icarte et al. (2019) to illustrate the issues arising from partially observable and non-stationary observations. We confirmed experimentally that the perfect MRM for the Cookie domain is learnt with the extended ARM framework, whereas previously, it could not be learned.

The extended framework offers a way to derive optimal policies using model-checking techniques. Currently, no other works in the literature has developed techniques to do so in dynamic and partially observable environment. Toro Icarte et al. (2019) do not offer a concise way to derive the optimal policy. Additionally, their approach is approximate, based on sampling, contrary to the extended ARM framework.

From a theoretical perspective, we defined the non-stationary and deterministic observation MDP (NS-DOMDP) and proved several properties with respect to NS-DOMDPs. This includes that given an underestimate ( $V_{expert}$ ) of the value of the optimal policy for an environment (with non-stationary observations), the extended ARM framework is guaranteed to learn an MRM that will allow a policy with value at least  $V_{expert}$  to be computed.

The proposed framework provides a minimal MRM, which results in a minimal extension of the state space that fully entails all of the necessary information to transform the problem into a process with Markov behavior. In this way, the algorithm is more principled than other developed techniques for MDPs with uncertainty, that suffer from a trade-off between minimal extension and computational efficiency for policy derivation (Thiébaux et al., 2006).

An interesting direction for future work is to make the learning phase more efficient. In the extended ARM algorithm the structure of the labelling function is assumed to be known in order to make MQs feasible. Simultaneous learning of the labelling function

definition could be possible, when MQs can be tagged as 'impossible' if they are not answered in reasonable time (Gaon and Brafman, 2020; Xu et al., 2021). Traditionally, actions in MDPs are stochastic, so it is possible that a feasible MQ is tagged as negative. However, a large number of MQs is answered, so the probability of including false negatives is low.

Since the labeling function in this work is always deterministic, we could transform POMDPs into (fully observable) MDPs. If observations are stochastic (i.e. noisy), this would be impossible. Unfortunately, we are dealing with non-stationarity, which make traditional techniques for solving POMDPs inapplicable. Literature on non-stationary POMDPs with stochastic observatin functions does exist (Peshkin et al., 1999; Shani et al., 2005; Jaulmes et al., 2005; Chatzis and Kosmpoulos, 2014). One could also consider literature regarding partial observability, proposing the use of stochastic action plans (Meuleau et al., 1999) and stochastic policies (Sutton and Barto, 2018).

## ACKNOWLEDGEMENTS

The second author was supported by the EOS project (No. 30992574).

## REFERENCES

- Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106.
- Bacchus, F., Boutilier, C., and Grove, A. (1996). Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1160–1167.
- Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684.
- Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific.
- Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Aaai*, volume 94, pages 1023–1028.
- Chatzis, S. P. and Kosmpoulos, D. (2014). A non-stationary partially-observable markov decision process for planning in volatile environments. *OPT-i 2014 - 1st International Conference on Engineering and Applied Sciences Optimization, Proceedings*, 2014:3020–3025.
- Gaon, M. and Brafman, R. (2020). Reinforcement learning with non-markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3980–3987.
- Jaulmes, R., Pineau, J., and Precup, D. (2005). Learning in non-stationary partially observable markov decision processes. In *ECML Workshop on Reinforcement Learning in non-stationary environments*, volume 25, pages 26–32.
- Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.
- Kolobov, A. (2012). Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210.
- Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*.
- Peshkin, L., Meuleau, N., and Kaelbling, L. P. (1999). Learning policies with external memory. In *ICML*.
- Rens, G., Raskin, J.-F., Reynouard, R., and Marra, G. (2021). Online learning of non-markovian reward models. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 74–86. INSTICC, SciTePress.
- Ross, S. M. (2014). *Introduction to stochastic dynamic programming*. Academic press.
- Shani, G., Brafman, R., and Shimony, S. (2005). Adaptation for changing stochastic environments through online pomdp policy learning. In *Proc. Eur. Conf. on Machine Learning*, pages 61–70. Citeseer.
- Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning Proceedings 1994*, pages 284–292. Elsevier.
- Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge and London, 2nd edition.
- Thiébaux, S., Gretton, C., Slaney, J., Price, D., and Kabanza, F. (2006). Decision-theoretic planning with non-markovian rewards. *Journal of Artificial Intelligence Research*, 25:17–74.
- Toro Icarte, R., Waldie, E., Klassen, T., Valenzano, R., Castro, M., and McIlraith, S. (2019). Learning reward machines for partially observable reinforcement learning. *Advances in Neural Information Processing Systems*, 32:15523–15534.
- White III, C. C. and White, D. J. (1989). Markov decision processes. *European Journal of Operational Research*, 39(1):1–16.
- Xu, Z., Wu, B., Ojha, A., Neider, D., and Topcu, U. (2021). Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 115–135. Springer.