Radboud University Nijmegen

Faculty of Science

Solving set length history-based rewards for POMDPs

-

Thesis MSc Computing Science

Author:
Serena Rietbergen

Supervisor:
dr. Nils Jansen

Second reader:
- -

-

Contents

Abstract

Chapter 1

Introduction

Motivating Example

Problem Formulation

Given a POMDP with a history-based reward function, obtain a policy that maximizes the expected reward.

Contribution

Structure

Chapter 2

Preliminaries

Set Theory

Let S be any countable set, then $|S|$ denotes the cardinality. We let S^* and S^ω denote the set of finite and infinite sequences over S, respectively. For a sequence $\pi \in S^*$ we can denote the length by $|\pi|$.

Let $\epsilon \in S^*$ be the sequence of length zero, so $|\epsilon| = 0$.

TO WRITE: sequence with set length k will be denoted by S^k, of which an element can be rewritten to $s_1 s_2 \dots s_k$

Probability Theory

For any countable set S we can define a *discrete probability distribution* as $\psi : S \to [0, 1]$ where $\sum_{s \in S} \psi(s) = 1$. The set of all possible probability distributions over S is denoted as $\Pi(S)$. We denote the support of a *probability distribution* as $supp(\psi) = \{s \in S \mid \psi(s) > 0\}$.

TO WRITE: random variable, expected value

Chapter 3

Background

3.1 Markov decision processes

TO WRITE: introduction to MDP

Definition 3.1 (MDP)**.** A Markov decision process is a tuple $M = (S, s_I, A, T)$ where

- S, the finite set of states;
- $s_I \in S$, the initial state;
- A, the finite set of actions;
- $T_M : S \times A \to \Pi(S)$, the probabilistic transition function.

Note that given $s \in S, a \in A$, we assign a probability distribution over S through $T(s, a)$. To obtain the probability of ending up in a certain state s' when starting in state s and performing action a, we simply calculate $T(s, a, s')$ which we obtain through $T(s, a)(s')$.

The *available actions* for a state s are given by $A(s) = \{a \in A \mid \exists s' \in S : T(s, a, s') > 0\}$. We can give the *possible successors* of state s in a similar matter through $Succ(s) = \{s \in S \mid \exists a \in A : T(s, a, s') > 0\}$.

A finite *trajectory* or *run* π of an MDP is realization of the stochastic process performed by the MDP denoted by the finite sequence $s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n \in (S \times A)^* \times S$. To obtain the last state of a trajectory we can use the following

$$last(\pi) = last(s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n) = s_n$$

TO WRITE: Set notation for finite trajectories

Reward function

We can extend MDPs with a *reward function* R which assign a reward - usually in $\mathbb{R}$ for taking a certain action a in a state s.

TO WRITE: Intuitive explanation for reward function - including cost function, including a real-world example

Let us look at *Markovian reward functions*, which can determine a reward based on the current state, action and obtained state, independent of its history. The most conventional notation is $R : S \times A \to \mathbb{R}$, where we consider the current state and the taken action. Another possible definition is $R : S \times A \times S \to \mathbb{R}$, where in $R(s, a, s')$ we consider the specific transition from s to s' by using action a, or $R : A \to \mathbb{R}$ where in $R(a)$ we only consider the action taken.

TO WRITE: Real life example of reward function with history

A reward function wich is dependent of its history is called a *Non-Markovian reward function*. There are a number of different reward functions possible

- $R : A^* \to \mathbb{R}$ - which only needs all the finite actions used, or;
- $R : S^* \to \mathbb{R}$ - which only looks at the finite states visited, or;
- $R : (S \times A)^* \to \mathbb{R}$ - which looks at the finite (sub)trajectory without the last state, or;
- $R : (S \times A)^* \times S \to \mathbb{R}$ - which looks at the finite (sub)trajectory.

The reward function we will be using is the Non-Markovian reward function which looks at trajectories of specific length k, namely $R_k : (S \times A)^k \to \mathbb{R}$.

TO WRITE: Increasing k creates increased reward

Policy

As stated above, we use reward functions over a MDP to usually argue over an optimized expected reward. After retrieving such an optimum, the question remains on how to actually obtain this value. We wish to know what strategy to apply path to take to obtain this value. For this we use strategies, or often called policies.

Definition 3.2. A policy for a MDP M is a function $\sigma : (S \times A)^* \times S \to \Pi(A)$, which maps a trajectory π to a probability distribution over all actions.

We call a policy *memoryless* if the function only considers $last(\pi)$. The notation $\sigma_k : (S \times A)^{k-1} \times S \to \Pi(S)$, gives a probability distribution when given the trajectory of length k - the k visited states and $k - 1$ actions. $\sigma_k(\pi, a) = \sigma_k(\pi)(a)$ gives the probability of using action a after the trajectory π.

TO WRITE: induced markov chain for removing non-determinism

3.2 Solve for optimal policy

Explain here or refer to paper(s)?

Value Iteration

Policy Iteration

Linear Programming

3.3 Partial observability

TO WRITE: Introduce pomdp

Definition 3.3 (POMDP)**.** A partially observable Markov decision process (POMDP) is a tuple $\mathcal{M} = (M, \Omega, O)$ where

- $M = (S, s_I, A, T)$, the hidden MDP;
- Ω, the finite set of observations;
- $O : S \to \Omega$, the observation function.

Let $O^{-1} : \Omega \to 2^S$ be the inverse function of the observation function, like so $O^{-1}(o) = \{s \in | \ O(s) = o\}$ in which we simply obtain all states in S that have observation o. Without loss of generality we assume that states with the same observations have the same set of available actions, thus $O(s_1) = O(s_2) \Rightarrow A(s_1) = A(s_2)$.

Since the actual states in a trajectory of the hidden MDP are not visible to the observes, we argue about an *observed trajectory* of the POMDP $\mathcal{M}$. This is not consist of a sequence of states and actions, but instead a sequence of observations are actions, thus an element of $(\Omega \times A)^* \times \Omega$. The set of all possible finite observed trajectories of will be denoted as $ObsSeq^{\mathcal{M}}$.

We can argue about the observed trajectory through the observation function, which will be extended over trajectories, like so

$$O(\pi) = O(s_1 a_1 s_2 a_2 \dots s_{n-1} a_{n-1} s_n) = O(s_1) a_1 O(s_2) a_2 \dots O(s_{n-1}) a_{n-1} O(s_n)$$

Policy

Definition 3.4. An observation-based strategy of a POMDP $\mathcal{M}$ is a function $\sigma : ObsSeq^{\mathcal{M}} \to \Pi(A)$ such that $supp(\sigma(O(\pi))) \subseteq A(last(\pi)) \ \forall \pi \in (S \times A)^* \times S$.

3.4 Mealy machine

Based on the definition as presented in [1].

Definition 3.5. A Mealy machine is a tuple $(Q, q_0, \Sigma, O, \delta, \lambda)$ where

- Q, the finite set of states;
- $q_0 \in Q$, the initial state;
- Σ, the finite set of input characters - the input alphabet;
- O, the finite set of output characters - the output alphabet;

- $\delta : Q \times \Sigma \to Q$, the input transition function, and;
- $\lambda : Q \times \Sigma \to O$, the output transition function.

Example

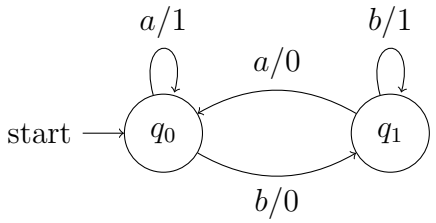

Figure 3.1: Simple Mealy machine

3.5 Reward machine

Based on the definition found in [2].

Definition 3.6. A reward machine $(V, v_I, \Sigma, \mathbb{R}, \delta, \sigma)$ consists of

- V, a finite not-empty set of states;
- $v_I \in V$, an initial state;
- Σ, the input alphabet;
- $\mathbb{R}$, the output alphabet;
- $\delta : V \times \Sigma \to V$, the deterministic transition function, and;
- $\sigma : V \times \Sigma \to \mathbb{R}$, the output function.

A reward machine is a special type of Mealy machine where the output alphabet consist of real values. Instead of simple transforming the input alphabet into an output alphabet, we obtain rewards for making transitions. After reading the input word, the final reward is the sum of all the rewards gathered from the transitions.

The obtained reward after reading a sequence $w \in \Sigma^*$ for an reward machine $(V, v_I, \Sigma, \mathbb{R}, \delta, \sigma)$ can be obtained by calculating $\texttt{FinalReward}(v_I, w)$

$$\texttt{FinalReward}(v, w) = \begin{cases} \sigma(v, a_0) + \texttt{FinalReward}(\delta(v, a_0), a_1 \dots a_k) & \text{if } w = a_0 a_1 \dots a_k \\ 0 & \text{if } w = \lambda \end{cases}$$

TO WRITE: Example

Chapter 4

Reward Controllers

4.1 Definition

TO WRITE: introduction

The idea is to transform the history-based reward function into something more tangible. We transform it is so that we can obtain the reward per step instead of only at the end of a sequence.

Based on the history-based reward function $R_k : (\Omega \times A)^k \to \mathbb{R}$ of a POMDP $\mathcal{M}$, we build a reward controller that mimics its behavior.

Definition 4.1. A reward controller $\mathcal{F}$ is a reward machine $(N, n_I, \Omega \times A, \mathbb{R}, \delta, \lambda)$, where

- N, the finite set of memory nodes;
- $n_I \in N$, the initial memory node;
- $\Omega \times A$, the input alphabet;
- $\mathbb{R}$, the output alphabet;
- $\delta : N \times \Omega \times A \to N$, the memory update;
- $\lambda : N \times \Omega \times A \to \mathbb{R}$, the reward update.

When starting in memory node n, where we observe observation o and then perform action a we will end up in the new memory node $\delta(n, o, a)$.

4.2 Creating a reward controller

Starting from a fully known R_k and POMDP $\mathcal{M}$, we know that R_k is defined over a number of sequences of elements in $\Omega \times A$ with length k. Since the reward of a sequence is only obtained after running through the entire sequence, we encode the reward of that specific sequence into the last transition. The idea is to make sure that every unique sequence has an unique path to the final state, so that their own reward is obtained.

Given all the sequences over which the Non-Markovian reward function is defined, let us create a reward controller through the following procedure.

Algorithm 1 Procedure for turning a list of sequences into a reward controller

```
1:  procedure CreateRewardController(sequences, R_k)
Require: sequences ∈ 2^{(Ω×A)^k}
Require: R_k ∈ 2^{(Ω×A)^k → ℝ}
2:      n_I ← new Node()                                   ▷ initial node
3:      n_F ← new Node()                                   ▷ "final" node
4:      N ← {n_I, n_F}
5:      for all π = π_1 π_2 ... π_k in sequences do
6:          n ← n_I
7:          for i ← 1 to k − 1 do
8:              if δ(n, π_i) is undefined then
9:                  n' ← new Node()                        ▷ create new memory node
10:                 N ← N ∪ {n'}
11:                 δ(n, π_i) ← n'
12:             n ← δ(n, π_i)                              ▷ update memory node
13:         δ(n, π_k) ← n_F                                ▷ set final transitions
14:         λ(n, π_k) ← R_k(π)                             ▷ set reward
15:     for all n ∈ N do                                   ▷ makes δ and λ deterministic
16:         for all (o, a) ∈ (Ω × A) do
17:             if δ(n, (o, a)) is undefined then          ▷ self-loop
18:                 δ(n, (o, a)) ← n
19:             if λ(n, (o, a)) is undefined then
20:                 δ(n, (o, a)) ← 0
21:     return (N, n_I, (Ω × A), ℝ, δ, λ)
```

We start by making an initial and final node in Lines 2 and 3. When reading a sequence we walk through it until we come across an undefined transition. This indicates that the sequence from that point on is unique again. For this transition we create a new node and connect it through Line 11.

When we get to the end of the transition (so when $i = k-1$), we ensure that the final transition is connected to the final node n_F in Line 13. This is also where we encode the actual reward value through Line 14 by calling upon the reward value given by R_k.

Since all functions of the reward controller need to be deterministic, we need to set the remaining values. All the memory nodes that have not been defined yet, will point towards themselves and thereby ensuring a dead-end through Line 18. After all, when reading a sequence and if the transition ends up in a state with a self-loop, there is already a deviation from the sequences over which the reward function is defined. Which in turn means that there will be no reward connected to it. Furthermore, since we only encode the reward in the final transition all the other transitions do not contribute to the final reward. This is solved by simple setting the reward value to zero in Line 19.

We observe that the amount of memory nodes $|N|$ of the newly created reward controller $\mathcal{F}$ is bounded by $|\Omega| \times |A| \times k$.

Example

Let R_k be defined over the following sequences

- $\square\ a\ \square\ b\ \square\ c$
- $\square\ b\ \square\ a\ \blacksquare\ c$
- $\square\ b\ \blacksquare\ c\ \blacksquare\ c$

TO WRITE: Notes for the model: missing actions/states

Following the procedure 1 we can create the reward controller as seen in Figure 4.2. Note that everywhere where the red lines are depicted, those paths are obsolete. And following the procedure those end-markings should be transitions towards the memory node they originated from. The green highlighted are the rewards that are set in Line 14. The elaborated version shown in the figure first checks the observation and then check the action to then direct it to a new memory node. A more compact version can be seen in Figure 4.2, where the observation and the action are condensed into one transition. Note that in this figure, the self-loops are also not shown.

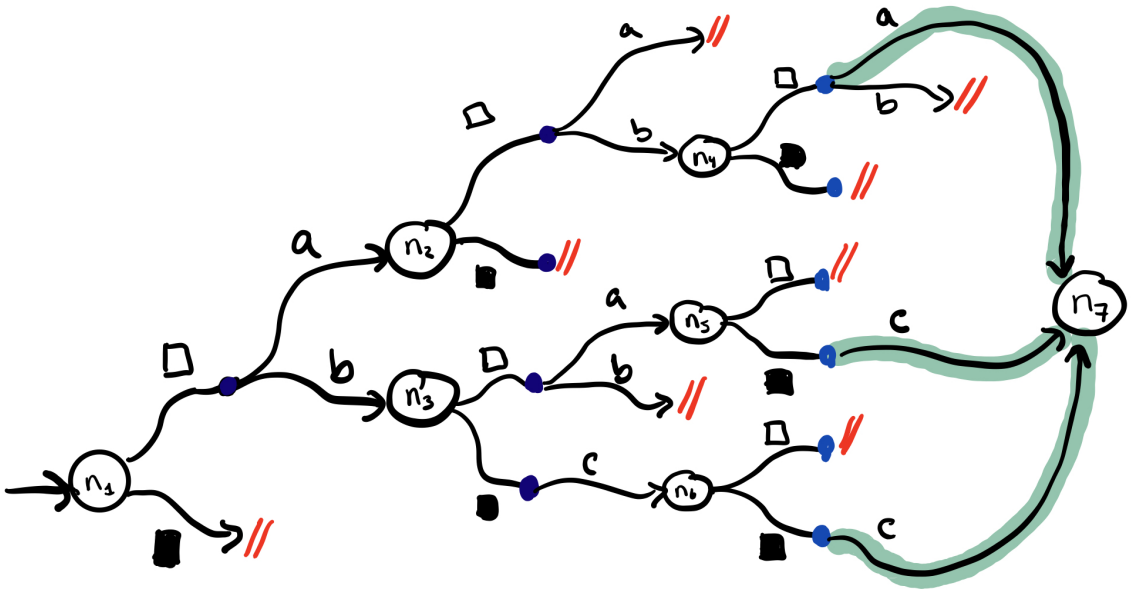

Figure 4.1: Elaborated reward controller

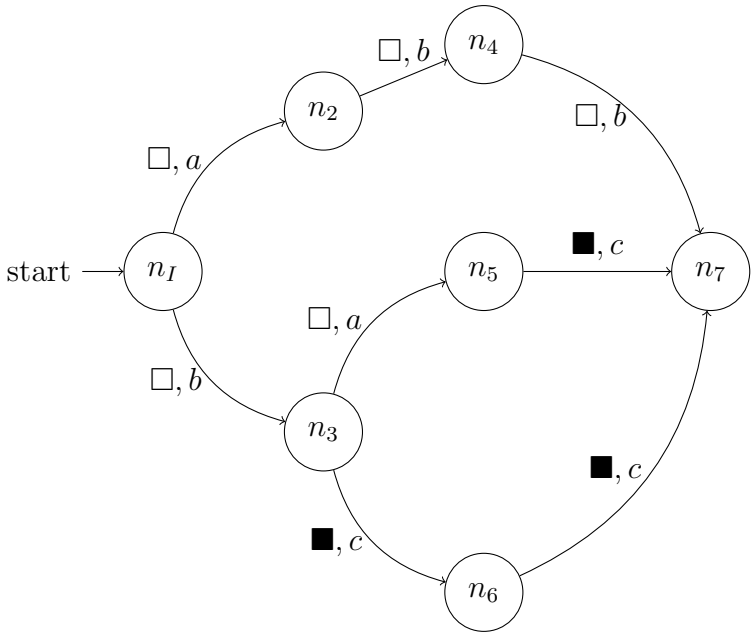

Figure 4.2: Compact reward controller

Corollary 4.1.1. *For every path $\pi \in (\Omega \times A)^k$ that is defined for R_k, we have that $R_k(\pi) =$ `FinalReward`(n_I, π)*

Chapter 5

Obtaining policy

The next step is defining the MDP with a new reward function based on the given POMDP together with it's reward controller. Since the reward function of the POMDP is bound by length k, we can define a Markov decision process limited to k steps. The reward function will be presented as a Markovian reward function, since the reward controller bound the rewards per step instead of over the entire history through it's construction.

Definition 5.1. The induced Reward Markov Decision Process for reward controller $\mathbb{F} = (N, n_I, A, \mathbb{R}, \delta, \lambda)$ on a POMDP $\mathcal{M} = (M, \Omega, O)$ where $M = (S, s_I, A, T_M)$ extended with R_k is a tuple $\mathcal{A} = (V, v_I, A, T_\mathcal{A}, \mathcal{R})$ where

- $V = \Omega \times N$, the finite set of states;
- $v_I = \langle O(s_I), n_I \rangle$, the initial state;
- A, the available actions;
- $T_\mathcal{A} : V \times A \to \Pi(V)$ where

$$T_\mathcal{A}(\langle o, n\rangle, a)(\langle o', n'\rangle) = \begin{cases} \frac{\sum_{s \in O^{-1}(o)} \sum_{s' \in O^{-1}(o')} T_\mathcal{M}(s,a,s')}{\sum_{s \in O^{-1}(o)} \sum_{s' \in S} T_\mathcal{M}(s,a,s')} & \text{if } \delta(n, o, a) = n' \\ 0 & \text{otherwise} \end{cases}$$

- $\mathcal{R} : V \times A \to \mathbb{R}$ defined as $\mathcal{R}(\langle o, n\rangle, a) = \lambda(n, (o, a))$

The different transitions to the final state are irrelevant, since the reward is encoded in the action, not including the obtained state

Since we have now obtained an MDP with a reward function that is only dependent of it's current state and the action, we can use known methods to compute an optimal policy.

Corollary 5.1.1. *Given a POMDP $\mathcal{M}$ and the reward controller $\mathcal{F}$ obtained through the history-based reward function R_k of $\mathcal{M}$, we create the Reward Machine $\mathbb{A}$ with the help of Definition 5.1. Given a policy σ, applying this to $\mathcal{M}$ will give the same expected reward as applying it to $\mathcal{A}$.*

Example

Given the reward controller as found in Figure 3.4 and the POMDP in Figure 5, we can create the induced reward MDP. Note that there are a number of missing self-loop transitions. Since there are irrelevant to the final product, they are not shown.

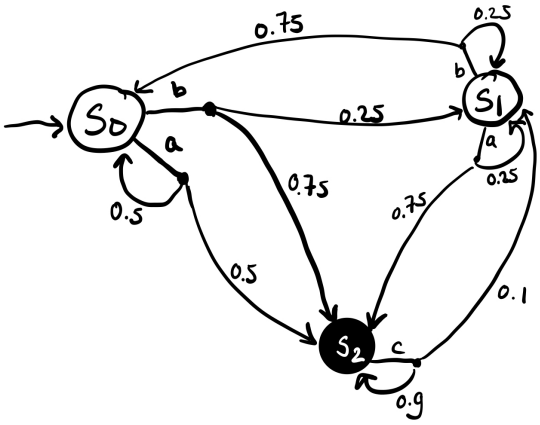

Figure 5.1: POMDP where $\Omega = \{\square, \blacksquare\}$

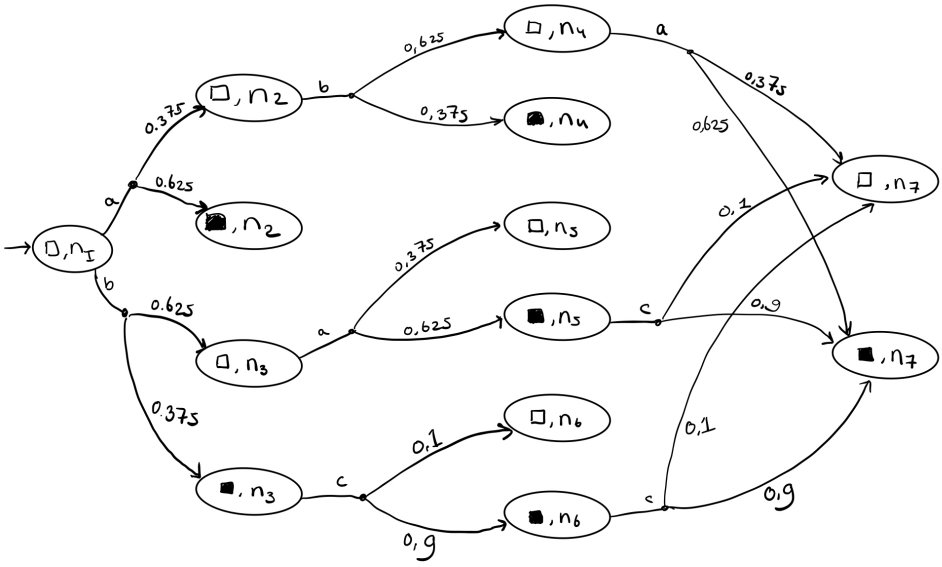

Figure 5.2: Resulting reward MDP

A few of the transitions calculated:

$$T_{\mathcal{A}}(\langle\square, n_I\rangle, a)(\langle\square, n_2\rangle) = \frac{\sum_{s\in O^{-1}(\square)}\sum_{s'\in O^{-1}(\square)} T_{\mathcal{M}}(s,a,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,a,s')}$$
$$= \frac{T(s_0,a,s_0) + T(s_0,a,s_1) + T(s_1,a,s_0)T(s_1,a,s_1)}{2}$$
$$= \frac{0,5+0+0+0,25}{2} = 0,375$$
$$T_{\mathcal{A}}(\langle\square, n_I\rangle, a)(\langle\blacksquare, n_2\rangle) = \frac{\sum_{s\in O^{-1}(\square)}\sum_{s'\in O^{-1}(\blacksquare)} T_{\mathcal{M}}(s,a,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,a,s')}$$
$$= \frac{T(s_0,a,s_2) + T(s_1,a,s_2)}{2}$$
$$= \frac{0,5+0,75}{2} = 0,625$$
$$T_{\mathcal{A}}(\langle\square, n_I\rangle, b)(\langle\square, n_3\rangle) = \frac{\sum_{s\in O^{-1}(\square)}\sum_{s'\in O^{-1}(\square)} T_{\mathcal{M}}(s,a,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,b,s')}$$
$$= \frac{T(s_0,b,s_0) + T(s_0,b,s_1) + T(s_1,b,s_0)T(s_1,b,s_1)}{2}$$
$$= \frac{0+0,25+0,75+0,25}{2} = 0,6255$$
$$T_{\mathcal{A}}(\langle\square, n_I\rangle, b)(\langle\blacksquare, n_3\rangle) = \frac{\sum_{s\in O^{-1}(\blacksquare)}\sum_{s'\in O^{-1}(\square)} T_{\mathcal{M}}(s,a,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,b,s')}$$
$$= \frac{T(s_0,b,s_2) + T(s_1,b,s_2)}{2}$$
$$= \frac{0,75+0}{2} = 0,375$$
$$T_{\mathcal{A}}(\langle\blacksquare, n_3\rangle, c)(\langle\square, n_6\rangle) = \frac{\sum_{s\in O^{-1}(\square)}\sum_{s'\in O^{-1}(\square)} T_{\mathcal{M}}(s,c,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,c,s')}$$
$$= \frac{T(s_2,c,s_0) + T(s_2,c,s_1)}{1}$$
$$= \frac{0+0,1}{1} = 0,1$$
$$T_{\mathcal{A}}(\langle\blacksquare, n_3\rangle, c)(\langle\blacksquare, n_6\rangle) = \frac{\sum_{s\in O^{-1}(\blacksquare)}\sum_{s'\in O^{-1}(\blacksquare)} T_{\mathcal{M}}(s,c,s')}{\sum_{s\in O^{-1}(\square)}\sum_{s'\in S} T_{\mathcal{M}}(s,c,s')}$$
$$= \frac{T(s_2,c,s_2)}{1}$$
$$= \frac{0,9}{1} = 0,9$$

5.1 Important Note

I noted a bit too late that the reward function was $(\Omega \times A)^k \to \mathbb{R}$ instead of the more generic $(S \times A)^k \to \mathbb{R}$. From the top of my head, everything should still work, save for a few changes:

- Definition of the reward controller $\mathcal{F}$ we change Ω to S.
- Procedure 1, change Ω to S
- The induced MDP in Definition 5.1 should change to $\mathcal{A} = (V, v_I, A, T_\mathcal{A}, \mathcal{R})$ where
  - $V = S \times N$, the finite set of states;
  - $v_I = \langle s_I, n_I \rangle$, the initial state;
  - A, the available actions;
  - $T_\mathcal{A} : V \times A \to \Pi(V)$ where $o_1 = O(s)$ and $o_2 = O(s')$ in

$$T_\mathcal{A}(\langle s, n\rangle, a)(\langle s', n'\rangle) = \begin{cases} \frac{\sum_{s_1 \in O^{-1}(o_1)} \sum_{s_2 \in O^{-1}(o_2))} T_\mathcal{M}(s_1, a, s_2)}{\sum_{s_1 \in O^{-1}(o_1)} \sum_{s_2 \in S} T_\mathcal{M}(s, a, s')} & \text{if } \delta(n, s, a) = n' \\ 0 & \text{otherwise} \end{cases}$$

  - $\mathcal{R} : V \times A \to \mathbb{R}$ defined as $\mathcal{R}(\langle s, n\rangle, a) = \lambda(n, (s, a))$

I haven't changed it yet, because I wasn't completely sure.

Bibliography

[1] George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, Jan 1955.

[2] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 590–598. AAAI Press, 2020.