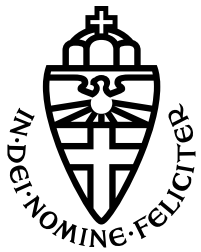
Radboud University Nijmegen



Faculty of Science

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 an alphabet Σ be a a finite set consisting of letters. A word is defined as a sequence of letters $w = w_1 w_2 \dots w_n \in \Sigma^*$. A language L is a subset of all possible words given an alphabet Σ, so $L \subseteq \Sigma^*$. Let ϵ denote the empty word, so $|\epsilon| = 0$.

A regular language is a language that can be defined by a regular expression. The language accepted by a regular expressions e is denoted as $L(e)$.

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 Finite Automata

TO WRITE: introduction to dfa

Definition 3.1 (DFA)**.** A Deterministic Finite Automata is a tuple $D = (Q, q_0, \Sigma, \delta, F)$ where

- Q, the finite set of states;
- q_0, the initial state;
- Σ the input alphabet;
- $\delta : Q \times \Sigma \to Q$, the deterministic transition function;
- $F \subseteq Q$, the set of final states.

TO WRITE: introduction to lambda star

Definition 3.2. We define $\delta^* : Q \times \Sigma^* \to Q$ where $\delta^*(q, w)$ denotes the state we end up after reading word w starting from state q as follows

$$\delta^*(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta^*(\delta(q, a_1), a_2 \dots a_n) & \text{if } w = a_1 a_2 \dots a_n \end{cases}$$

TO WRITE: introduction to accepted words

Definition 3.3. We say the language accepted by a DFA $D = (Q, q_0, \Sigma, \delta, F)$ consists of all the words that start in the begin state and finish in any final state. Thus $L(D) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F$.

3.2 Markov decision processes

TO WRITE: introduction to MDP

Definition 3.4 (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;
- $T : S \times A \to \Pi(S)$, 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 $s_1a_1s_2a_2 \dots s_{n-1}a_{n-1}s_n \in (S \times A)^* \times S$. To obtain the last state of a trajectory we can use the following

$$last(\pi) = last(s_1a_1s_2a_2 \dots s_{n-1}a_{n-1}s_n) = s_n$$

.

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 $R : S \to \mathbb{R}$ where in $R(s)$ we only consider the visited state s.

TO WRITE: Real life example of reward function with history

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

- $R : S^* \to \mathbb{R}$ - 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 3.5. 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 $last(\pi)$.

TO WRITE: induced markov chain for removing non-determinism

3.3 Partial observability

TO WRITE: Introduce pomdp

Definition 3.6 (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 - $O^{-1}(o) = \{s \in |\ O(s) = o\}$ - 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 3.7. 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 Belief MDP

3.5 Moore machine

Based on the definition as presented in [1].

Definition 3.8. A Mealy machine is a tuple $(Q, q_0, \Sigma, O, \delta, \sigma)$ 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;
- $\sigma : Q \times O$, the output transition function.

Example

TO WRITE: example moore machine – traditional sense

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 $R : \Omega^* \to \mathbb{R}$ 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 $(N, n_I, \Omega, \mathbb{R}, \delta, \lambda)$, where

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

TO WRITE: define δ^* just like DFA

4.2 Given a list of sequences and rewards

TO WRITE: it's a choice that you only get one reward after an input. if a seperate reward sequence is a strict substring of another sequence, they are not added. this can be done by adding the reward to every following node in the created sequence.!!!!!

Let's say we are designing a model for an engineer and they want certain observation sequences to connect to a reward. Thus we are given a number of observation sequences $\pi_1, \pi_2, \ldots, \pi_n$ together with their associated real valued rewards $r_1, r_2, \ldots, r_n$.

Definition 4.2. Given the observation sequences $\pi_1, \pi_2, \ldots, \pi_n$ and their associated rewards $r_1, r_2, \ldots, r_n$ we define the history-based reward function $R : \Omega^* \to \mathbb{R}$, which we create as follows

$$R(w) = \begin{cases} r_i & \text{if } w = \pi_i \text{ for } i \in \{1, \ldots, n\} \\ 0 & \text{otherwise} \end{cases}$$

In R we simply connect the observation sequence π_i to their respective reward r_i and every other sequence is connected to zero.

We only want to obtain any of the rewards if their associated observation sequence has been observed. Thus we create a reward controller in which we encode the reward in the state. The idea is as follows: if we read the observation sequence and we end up in a certain state n, we obtain the reward $\sigma(n)$ in that state.

Given all the sequences over which the Non-Markovian reward function is defined, let us create a reward controller through the following procedure. Note that we assume that all the sequences are unique.

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

```
1:  procedure CreateRewardController(sequences, R)
Require: sequences
Require: R : Ω* → ℝ
2:      n_I ← new Node()                                   ▷ initial node
3:      n_F ← new Node()                                   ▷ dump node
4:      path(n_I) = ϵ
5:      N ← {n_I, n_F}
6:      for all π = o_1 o_2 ... o_k in sequences do
7:          n ← n_I
8:          for i ← 1, ..., k do
9:              if δ(n, o_i) is undefined then
10:                 n' ← new Node()                        ▷ create new memory node
11:                 path(n) = o_1 ... o_i
12:                 N ← N ∪ {n'}
13:                 δ(n, o_i) ← n'
14:             n ← δ(n, o_i)                              ▷ update memory node
15:         σ(n) ← R(π)                                    ▷ set reward
16:     for all n ∈ N do                                   ▷ makes δ and σ deterministic
17:         for all o ∈ Ω do
18:             if δ(n, o) is undefined then               ▷ useless transition
19:                 δ(n, o) ← n_F
20:         if σ(n) is undefined then
21:             σ(n) ← 0
22:     return (N, n_I, Ω, ℝ, δ, σ)
```

We start by creating an initial node in Line 2 and a dump node in Line 3. The idea is that, since the reward controller is deterministic, if we need to determine

the reward of a sequence that is (for example) longer than a known sequence (with reward), we don't want to end in the state in which the reward is encoded. Thus these zero-reward sequences are passed along to a node which will only consist of self-loops and will have a reward of zero encoded to them.

Then for every sequence which we are given, we walk through it. If we then come across a transition which isn't defined yet, we define it by making a new memory node in Line 10, adding it to N, and setting the transition to this new node. If the transition already existed, we simply update the memory node. After we are done with reading the sequence, we simply encode the reward into the state itself in Line 15.

Then since the reward controller needs to be deterministic, we set the other undefined values. Every other transition that hasn't been made yet, will be transferred to the dump node as mentioned above in Line 19. Furthermore, there are still nodes in which the reward is undefined. None of the given sequences ended up in these states, so per Definition 4.2 we encode those to zero in Line 21.

We observe that the number of memory nodes $|N|$ of the newly created reward controller $\mathcal{F}$ is bounded by Ω^k, where $k = \max_{seq \in sequences} |seq|$.

Note that the set of nodes N without n_F together with the memory update function is represents a directed acyclic graph. This indicates for every node n there is an unique path from the initial node n_I to node n. This unique path is encoded in the function `path`: $N \setminus n_F \to \Omega^*$. This function is well-defined, since it's defined for n_I in Line 4. Every other time a new node is neccesary, it is created in Line 10, and `path` is then immediately defined for the new node. This `path` function is needed for proving the following lemma.

let maarten read this

Lemma 4.3. *For any sequence $\pi \in \Omega^*$, let $r = R(\pi)$ be its associated reward. Then $\sigma(\delta^*(n_I, \pi)) = r$.*

Proof. Let us state that after reading π, we end up in state n, so $n = \delta^*(n_I, \pi)$.
Now if $n = n_F$, we know that the associated reward is zero since $\sigma(n_F) = 0$. A sequence can only end up in n_F if it was not a part of the pre-defined sequences and following Definition 4.2 the reward is then zero.
If $n \in N \setminus n_F$, we can obtain the unique path to node n through $\texttt{path}(n)$. We know that this is equal to π, so the associated reward is thus $R(\texttt{path}(n)) = R(\pi) = r$. □

Example

Say we are given the following sequences and rewards

1. □ □ with a reward of 15
2. ■ □ ■ with a reward of 20
3. □ □ ■ □ with a reward of 12
4. ■ with a reward of 2

Following the procedure 1 we create the associated reward controller. To show how the procedure works, we will show you the intermediate reward controller after processing every sequence.

After sequence (1)

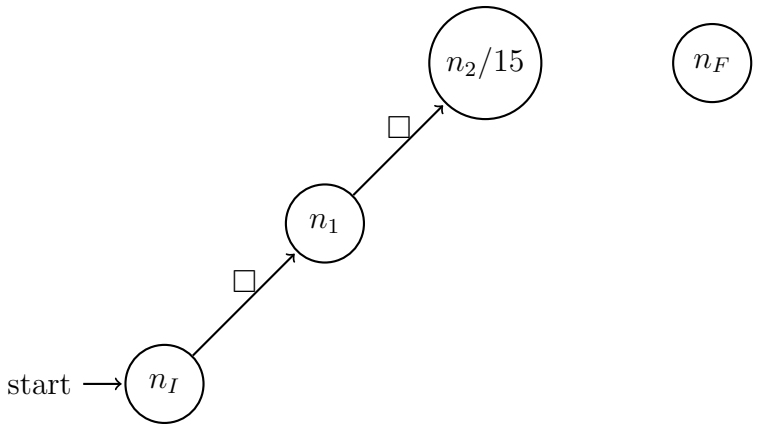


Figure 4.1: Reward controller after sequence (1)

After sequence (2)

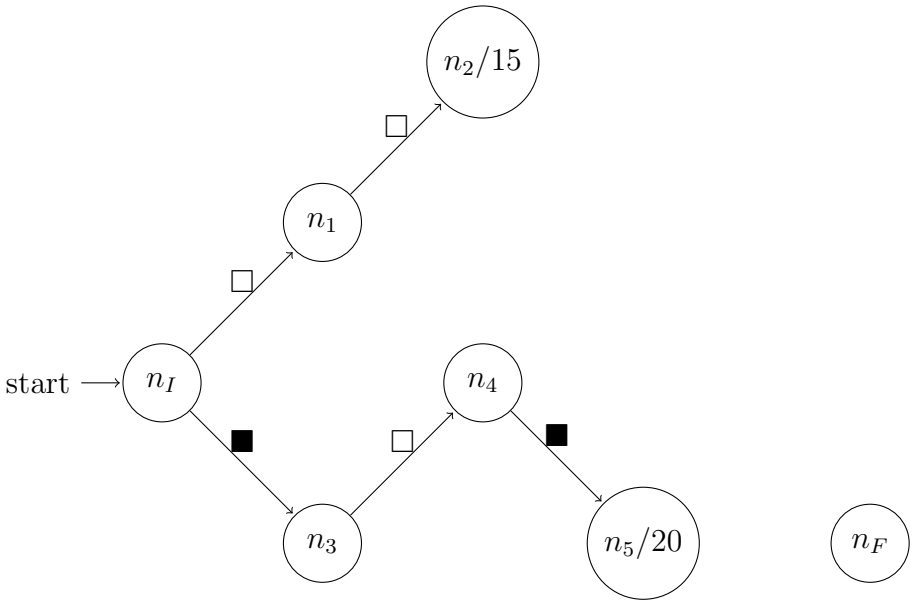


Figure 4.2: Reward controller after sequence (1) and (2)

After sequence (3)

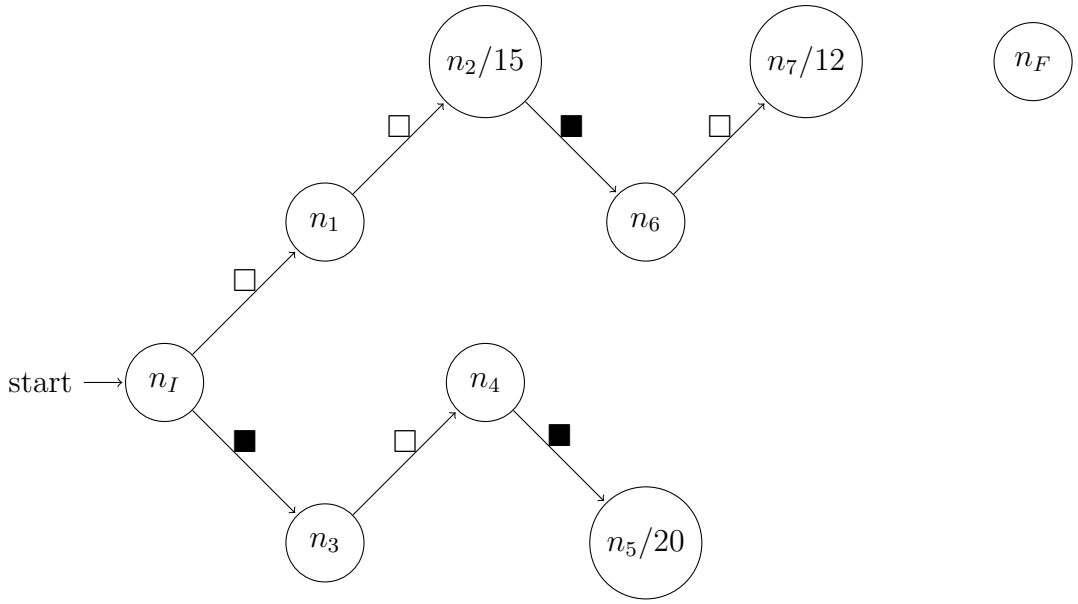


Figure 4.3: Reward controller after sequence (1), (2) and (3)

After sequence (4)

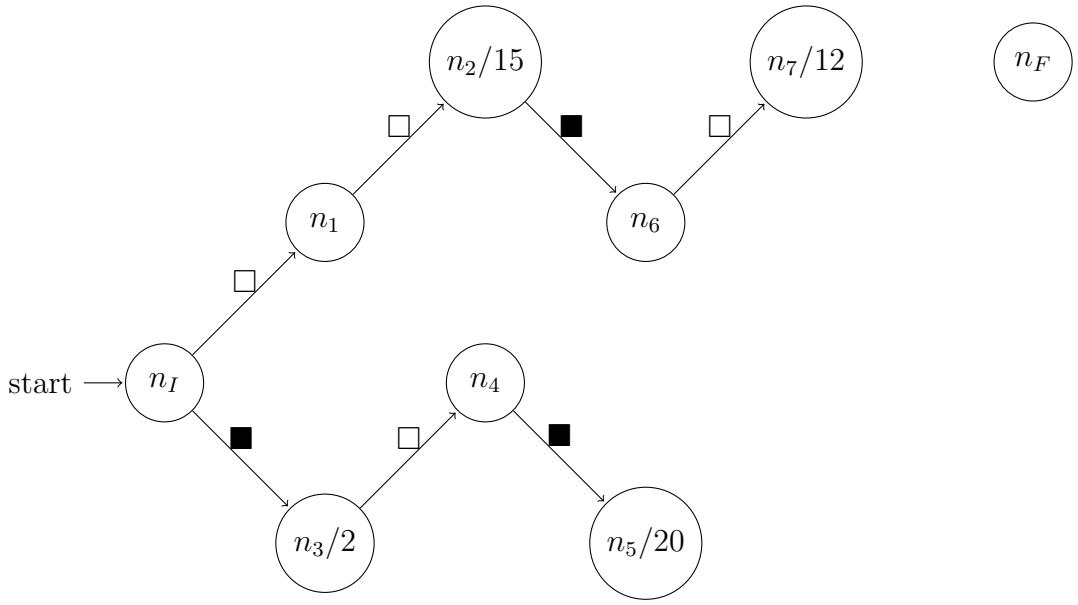


Figure 4.4: Reward controller after sequence (1), (2) and (3)

Finalized Reward Controller

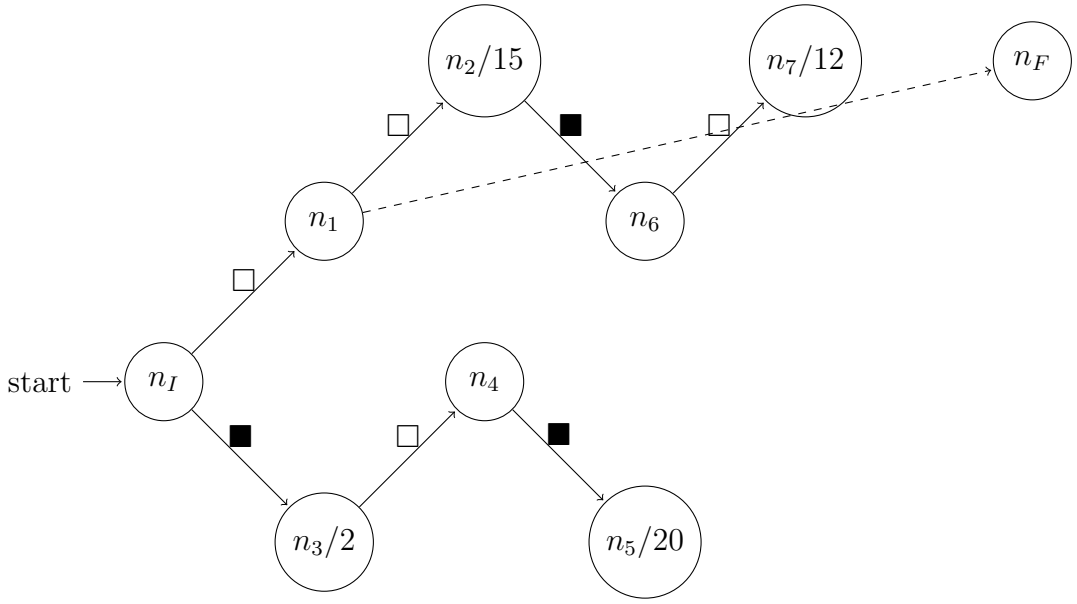


Figure 4.5: Reward controller after sequence (1), (2) and (3)

finish this

4.3 Given regular expressions and their rewards

find ref for product construction

Given a number of regular expressions over observations defined as $e_1, e_2, \ldots, e_n$ together with their respective rewards $r_1, r_2, \ldots, r_n$. For all $i = \{1, \ldots, n\}$ let the DFA $D_i = (Q_i, q_{0,i}, \Omega, \delta_i, F_i)$ be a DFA that accepts the language generated by e_i, so $L(D_i) = L(e_i)$. From all these seperate DFAs, we now create the product DFA as follows.

Definition 4.4. The induced product DFA for given DFAs $D_1, D_2, \ldots, D_n$ is $N = (Q, q_0, \Sigma, \delta, F)$ where

- $Q = Q_1 \times Q_2 \times \cdots \times Q_n$
- $q_0 = \langle q_{0,1}, q_{0,2}, \ldots, q_{0,n} \rangle$
- Ω, the same input alphabet
- $\delta(\langle q_1, q_2, \ldots, q_n \rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a), \ldots, \delta_n(q_n, a) \rangle$
- $F = \{\langle q_1, q_2, \ldots, q_n \rangle \mid \exists i \in \{1, 2, \ldots, n\} : q_i \in F_i\}$

Lemma 4.5. *Given n DFAs where $D_i = (Q_i, q_{0,i}, \Sigma, \delta_i, F_i)$, let N be the product automaton as obtained in Definition 4.4. Then we $L(N) = L(D_1) \cup L(D_2) \cup \ldots L(D_n)$.*

Proof.

$$\begin{aligned} w \in L(N) &\iff \delta_N^*(q_0, w) \in F \\ &\iff \langle \delta_1^*(q_{0,1}, w), \delta_2^*(q_{0,2}, w), \ldots, \delta_n^*(q_{0,n}, w) \rangle \in F \\ &\iff \exists i \in \{1, \ldots, n\} : \delta_i^*(q_{0,i}, w) \in F_i \\ &\iff \delta_1^*(q_{0,1}, w) \in F_1 \text{ or } \delta_2^*(q_{0,2}, w) \in F_2 \text{ or } \ldots \text{ or } \delta_n^*(q_{0,n}, w) \in F_n \\ &\iff w \in L(D_1) \text{ or } w \in L(D_2) \text{ or } \ldots \text{ or } w \in L(D_n) \\ &\iff w \in L(D_1) \cup L(D_2) \cup \cdots \cup L(D_n) \end{aligned}$$

□

TO WRITE: R_A keeps track of all the rewards associated with their respective regular expression

$$R_A(q) = \begin{cases} R(e_i) & \text{if } q \in F_i \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.6. So having this DFA $N = (Q, q_0, \Omega, \delta, F)$ and associated reward function R_A, we define the induced reward controller $\mathcal{F} = (N, n_I, \Omega, \mathbb{R}, \delta_{\mathcal{F}}, \sigma)$ as follows

- $N = Q$
- $n_I = q_0$
- $\delta_{\mathcal{F}} = \delta$
- $\sigma : Q \to \mathbb{R}$ where $\sigma(\langle q_1, q_2, \ldots, q_n \rangle) = \sum_{i=1}^{n} R_A(q_i)$

fill this in please

Lemma 4.7. *Given X and Y etc For all* $\pi \in \Omega^*$

$$\sigma(delta^*(n_I, \pi)) = \sum_{e \in \{e_i | \pi \in L(e_i)} R(e_i)$$

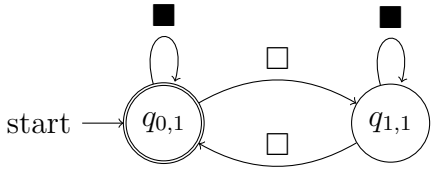
dit

Proof. zie schrift □

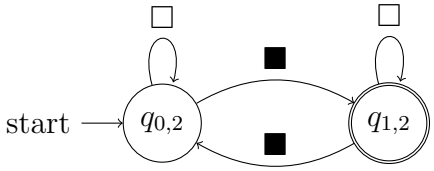
Example

are the regular expressions really neccesary?

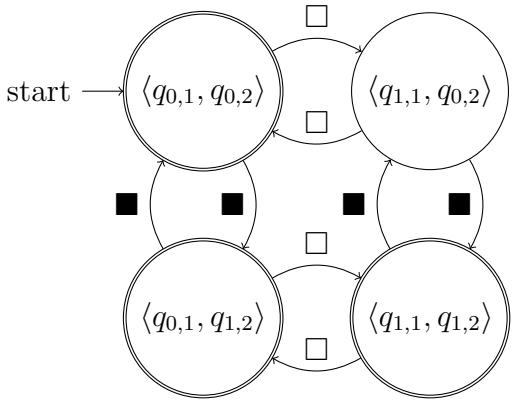
Let's say an even number off □ gives a reward of 10 and an uneven number of ■ gives a reward of 15. In other words $R((■^*□■^*□■^*)^*) = 10$ and $R(□^*■□^*(■□^*■□^*)^*) = 15$



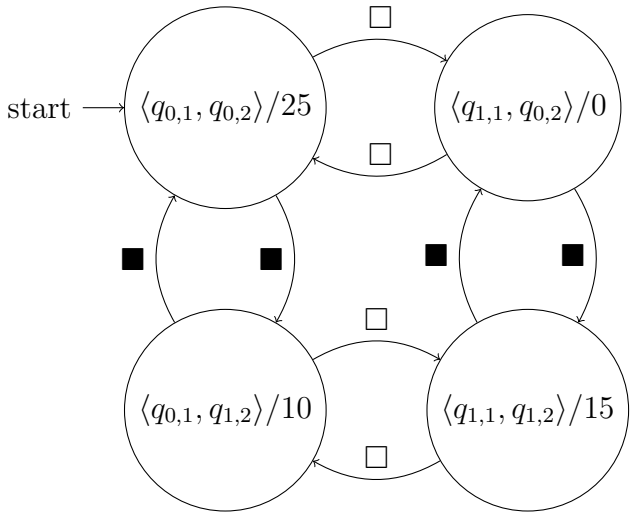
(a) DFA for regular expression even number of □



(b) DFA for regular expression for odd number of ■



(c) Product DFA for both regular expressions



(d) Reward Controller for R

Note that
$R_A(q_{0,1}) = 10$
$R_A(q_{1,1}) = 0$
$R_A(q_{0,2}) = 0$
$R_A(q_{1,2}) = 15$

Chapter 5

Obtaining policy

TO WRITE: introduction

Definition 5.1. The induced POMDP for reward controller $\mathcal{F} = (N, n_I, \Omega, \mathcal{R}, \delta, \lambda)$ on a POMDP $\mathcal{M} = (M, \Omega, Obs)$ where $M = (S, s_I, A, T_M)$ is a tuple $\mathcal{M}' = (M', \Omega', Obs')$ where

- $M' = (S', s'_I, A', T_{M'}, \mathcal{R})$, the hidden MDP defined as follows
  - $S' = S \times N \cup \{s_F\}$
  - $s'_I = \langle s_I, \delta(n_I, O(s_I))\rangle$
  - $A' = A \cup \{\texttt{end}\}$
  - $T_{M'} : S' \times A \to \Pi(S)$ where

$$T_{M'}(\langle s, n\rangle, a, \langle s', n'\rangle) = \begin{cases} T_M(s, a, s') & \text{if } \delta(n, O(s') = n') \\ 0 & \text{otherwise} \end{cases}$$

$$T_{M'}(s, \texttt{end}, s_F) = 1 \text{ for all} s \in S'$$

  - $R : S' \times A' \times S' \to \mathcal{R}$ where

$$R(s', \texttt{end}, s_F) = \begin{cases} \sigma(n) & \text{if } s' = \langle s, n\rangle \\ 0 & \text{if } s' = s_F \end{cases}$$

$$R(s, a, s') = 0 \text{ for all } s' \in S' \setminus \{s_F\}$$

- $\Omega' = \Omega \cup \{o_F\}$, the observation state
- $Obs : S' \to \Omega'$ where

$$Obs'(s) = \begin{cases} Obs(s') & \text{if } s = \langle s', n\rangle \\ o_F & \text{if } s = s_F \end{cases}$$

Bibliography

[1] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.