

# FSM Learning

Radek Mařík

Czech Technical University  
Faculty of Electrical Engineering  
Department of Telecommunication Engineering  
Prague CZ

December 12, 2019



- 1 **FSM Learning**
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 **Hidden Markov Model**
  - A Brief Overview
- 3 **Markov Decision Process**
  - Introduction
  - Utility Function, Policy
  - Value Iteration
  - Policy Iteration
  - Conclusions

# Outline

- 1 **FSM Learning**
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - Utility Function, Policy
  - Value Iteration
  - Policy Iteration
  - Conclusions

# Finite State Machine

A **finite-state machine** is a sextuple  $(S, \Sigma, \Gamma, s_0, \delta, \lambda)$ , where

- $S$  is a finite nonempty set of states,
- $\Sigma$  is an input alphabet (a finite nonempty set of symbols),
- $\Gamma$  is an output alphabet (a finite nonempty set of symbols),
- $s_0$  is an initial state,  $s_0 \in S$ ,
- $\delta$  is a state-transition function:  $\delta : S \times \Sigma \rightarrow S$ ,
- $\lambda$  is an output function:  $\lambda : S \times \Sigma_\epsilon \rightarrow \Gamma_\epsilon$ .

Additional designations:

- $\Sigma^*$  is the set of all strings (words) over the input alphabet,
- $\Gamma^*$  is the set of all strings (words) over the output alphabet,
- Alphabet  $X^*$  always contains  $\epsilon$  and  $\forall x \in X^* : \epsilon \cdot x = x = x \cdot \epsilon$ .
- Thus  $X^*$  is always nonempty and it is also countable because  $X$  is countable.



# Goal

- A system trying to figure out the effects its actions have on its environment...
  - It performs actions.
  - It gets observations.
  - It tries to make an internal model of what is happening.
- Let's model the world as a DFA.

## Applications

- Communication protocol learning,
- Hidden process learning,
- WWW application learning,
- Black box proprietary behavior identification,
- Software implementation identification.



# Learning a Language

- Inferring finite automata is analogous to learning a language
- There is no way to distinguish between two automata that recognize the same language, without examining the state structure.
- We focus on finding the minimum equivalent automata.
- It has been shown that the only classes of languages that can be learned from **positive data only** are classes which include no infinite language.

# Active Learning <sup>[Hon13]</sup>

- **Passive learning** - a set  $X$  is given and we cannot modify it.
  - NP problem
- **Active learning** - a set  $X$  can be selected and it can be modified during a learning process.
  - P problem



# Outline

## 1 FSM Learning

- FSM Learning Overview
- **Angluin's Algorithm**
- Example

## 2 Hidden Markov Model

- A Brief Overview

## 3 Markov Decision Process

- Introduction
- Utility Function, Policy
- Value Iteration
- Policy Iteration
- Conclusions



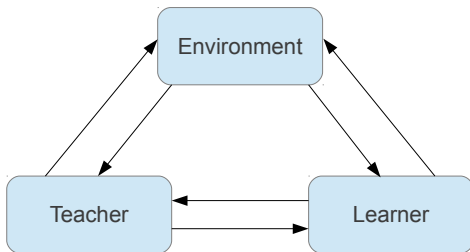
The teacher has to be able to answer two kinds of queries

- **Membership query** - Yes/No.
  - In a membership query the learner selects a word  $w \in \Sigma^*$  and
  - the teacher gives the answer whether or not  $w \in L$ .
- **Equivalence query (counterexamples)** - Yes/a counterexample string.
  - In an equivalence query the learner selects a hypothesis automaton  $\mathcal{H}$ , and the teacher answers whether or not  $L$  is the language of  $\mathcal{H}$ .
  - If yes, then the algorithm terminates.
  - If no, then the teacher gives a counterexample, i.e., a word in which  $L$  differs from the language of  $\mathcal{H}$ .

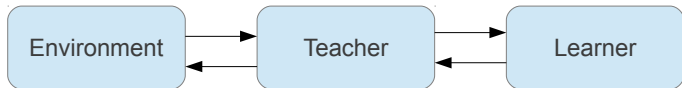
An issue of whether or not we have a **reset** button.



# Active Learning with a Teacher <sup>[Hon13]</sup>



A learning architecture with a minimally adequate teacher.



An architecture with a degraded teacher working as an interface.

# Angluin's Algorithm - Top Level View

- Iteratively, the algorithm builds a DFA using membership queries, then presents the teacher with the DFA as a solution.
- If the DFA is accepted, the algorithm is finished. Otherwise, the teacher responds with a counter-example, a string that the DFA presented would either accept or reject incorrectly.
- The algorithm uses the counter-example to refine the DFA, going back to the first step.



# Angluin's Algorithm - Control Structures

## States and Experiments

The algorithm uses two sets,

- $S$  for states,
  - $S \dots$  access sequences to states
  - $S \bullet A \dots$  sequences to exercise all transitions
- $E$  for experiments (distinguishing sequences), and
- one observation table,  $T$ , where
  - elements of  $S \cup S \bullet A$  form rows, and
  - elements of  $E$  form columns – the values of each cell is the outcome of a membership test for the concatenation of the row and column strings.



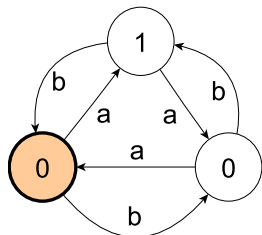
# Observation Table [Ang86, Sha08, Hon13]

## Definition 1.1

Let  $\mathcal{E} = (A, \text{accept})$  be an accepting environment.

**Observation table of environment  $\mathcal{E}$**  is an ordered triple  $OT = (S, E, T)$ , where

- $S \subseteq A^*$ ,  $S \neq \emptyset$ ,  $S$  finite,  $S$  is prefix closed.
  - $E \subseteq A^*$ ,  $E \neq \emptyset$ ,  $E$  finite,  $E$  is suffix closed.
  - $T$  is a function  $(S \cup S \bullet A) \times E \rightarrow \{0, 1\}$ .
- The set  $S$  is called *input set*.
  - $E$  is a *distinguishing set*.



		$E$	
		$\epsilon$	$a$
$S$	$\epsilon$	0	1
	$a$	1	0
	$b$	0	0
$S \bullet A$	$aa$	0	0
	$ab$	0	1
	$ba$	0	1
	$bb$	1	0

# Initial Observation Table <sup>[Hon13]</sup>

$L^*$  algorithm initialization:

- Init the observation table  $OT = (S, E, T)$ , where  $S = \{\epsilon\}$ ,  $E = \{\epsilon\}$ .
- Create a *queue of membership queries*: all pairs  $s \cdot e$ , where  $s \in S \cup S \cdot A$  and  $e \in E$ .
- Get the answers from the set of  $\{0, 1\}$  provided by the teacher, if  $s \cdot e$  belongs to the learned language. Insert the answer value to the place  $T(s, e)$  in the observation table.
- Different rows in the section  $S$  of the table define states of the a possible automaton.

		$E$
		$\epsilon$
$S$	$\epsilon$	1
$S \cdot A$	$a$	0
	$b$	0



# Observation Table - Closeness, Consistency <sup>[Hon13]</sup>

$$(s_1, s_2 \in S)$$

$$(s_1 \stackrel{E}{\sim} s_2) \iff (\forall e \in E)(\lambda(s_1, e) = \lambda(s_2, e))$$

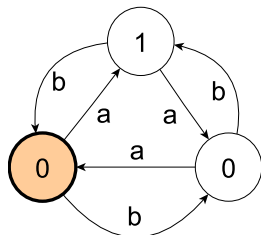
## Definition 1.2

**An observation table**  $OT = (S, E, T)$  is **closed**, if  $(\forall t \in S \cdot A)(\exists s \in S)(s \stackrel{E}{\sim} t)$ .

**The table is consistent**, if

$$(\forall s, t \in S, s \stackrel{E}{\sim} t) \implies (\forall a \in A)(s \cdot a \stackrel{E}{\sim} t \cdot a).$$

- The closeness and consistency checking is performed when the queue of queries becomes empty.



		$E$	
		$\epsilon$	$a$
$S$	$\epsilon$	0	1
	$a$	1	0
	$b$	0	0
$S \bullet A$	$aa$	0	0
	$ab$	0	1
	$ba$	0	1
	$bb$	1	0

# Observation Table - Modifications <sup>[Hon13]</sup>

- If  $OT = (S, E, T)$  is not closed, then
  - ① Search for  $t \in S \cdot A$ , so that  $s \not\stackrel{E}{\sim} t$  for all  $s \in S$ .
  - ② This  $t$  is added to the set  $S$  and the queue of membership queries is extended with  $t \cdot a \cdot e$  for all  $a \in A$  and  $e \in E$ .
- If  $OT$  is not consistent,
  - ① Search for  $s, t \in S$ ,  $e \in E$  and  $a \in A$ , so that  $s \stackrel{E}{\sim} t$ , but  $T(s \cdot a, e) \neq T(t \cdot a, e)$ .
  - ② The word  $a \cdot e$  is added to the distinguishing set  $E$ .
  - ③ The queue of membership queries is extended with  $s' \cdot e$  for all  $s' \in S \cup S \cdot A$ .
  - ④ It is obvious that  $s \stackrel{E}{\sim} t$  is not satisfied in the new observation table.





# $L^*$ algorithm [Ang86, Sha08, Hon13]

- 1 Init the observation table  $OT = (S, E, T)$ .
- 2 Fill the observation table using the membership query queue.
- 3 Check if  $OT$  is closed and consistent:
  - 1 If  $OT$  is not closed, extend the set  $S$  with  $t \in S \cdot A$ , so that  $s \stackrel{E}{\not\sim} t$  for all  $s \in S$ . Extend the queue of membership queries and continue to 2.
  - 2 If  $OT$  is not consistent, extend the set  $E$  with the word  $a \cdot e$ ,  $e \in E$ , and  $a \in A$  so that there are  $s, t \in S$ , that  $s \stackrel{E}{\sim} t$ , but  $T(s \cdot a, e) \neq T(t \cdot a, e)$ . Extend the queue of membership queries and continue to 2.
- 4 Make the conjecture  $\mathcal{A}$  and ask the teacher for its correctness.
- 5 If the teacher returns a counterexample  $c \in A^+$ , delete the conjecture  $\mathcal{A}$ , add all elements of the set  $\text{pref}(c)$  to the set  $S$ , extend the queue of membership queries and continue to 2.
- 6 Accept the conjecture  $\mathcal{A}$  as the automaton modeling the environment  $\mathcal{E}$ .



# FSM Conjecture Example [Ang86, Ang87]

- An acceptor  $M(S, E, T)$ 
  - over the alphabet  $A$ ,
  - with state set  $Q$ ,
  - initial state  $q_0$ ,
  - accepting states  $F$ , and
  - transition function  $\delta$ :

$$Q = \{\text{row}(s) : s \in S\}, \quad (1)$$

$$q_0 = \text{row}(\epsilon), \quad (2)$$

$$F = \{\text{row}(s) : s \in S \text{ and } T(s) = T(s \bullet \epsilon) = 1\}, \quad (3)$$

$$\delta(\text{row}(s), a) = \text{row}(s \bullet a). \quad (4)$$

- $S = \{\epsilon, a, b, bb\}$ ,  $E = \{\epsilon, a\}$

$T_4$		$E$	
		$\epsilon$	$a$
$S$	$\epsilon$	1	0
	$a$	0	1
	$b$	0	0
	$bb$	1	0
$S \bullet A$	$aa$	1	0
	$ab$	0	0
	$ba$	0	0
	$bba$	0	1
	$bbb$	0	0

$M_2/\delta$	$a$	$b$
$q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_2$
$q_2$	$q_2$	$q_0$



# Outline

## 1 FSM Learning

- FSM Learning Overview
- Angluin's Algorithm
- Example

## 2 Hidden Markov Model

- A Brief Overview

## 3 Markov Decision Process

- Introduction
- Utility Function, Policy
- Value Iteration
- Policy Iteration
- Conclusions

$L^*$  Algorithm - Example I <sup>[Ang87]</sup>

## Example 1

The unknown regular automaton accepts the set of all strings over  $\{a, b\}$  with an even number of  $a$ 's and an even number of  $b$ 's.

The initial observation table,  $S = E = \{\epsilon\}$

$T_1$		$E$
		$\epsilon$
$S$	$\epsilon$	1
$S \cdot A$	$a$	0
	$b$	0

- The observation table  $T_1$  is consistent, but not closed, since  $\text{row}(a)$  is distinct from  $\text{row}(\epsilon)$ .
- $L^*$  chooses to move the string  $a$  to the set  $S$  and then queries the strings  $aa$  and  $ab$  to construct the observation table  $T_2$ .



# $L^*$ Algorithm - Example II <sup>[Ang87]</sup>

## Example 2

The unknown regular automaton accepts the set of all strings over  $\{a, b\}$  with an even number of  $a$ 's and an even number of  $b$ 's.

$$S = \{\epsilon, a\}, E = \{\epsilon\}$$

$T_2$		$E$
		$\epsilon$
$S$	$\epsilon$	1
	$a$	0
$S \bullet A$	$b$	0
	$aa$	1
	$ab$	0

$M_1/\delta$	$a$	$b$
$q_0$	$q_1$	$q_1$
$q_1$	$q_0$	$q_1$

- The observation table  $T_2$  is consistent and closed.
- $L^*$  makes a conjecture of the acceptor  $M_1$ .
- The initial state of  $M_1$  is  $q_0$  and the final state is also  $q_0$ .
- The teacher selects a counterexample  $bb$  (rejected by  $M_1$ ).

# $L^*$ Algorithm - Example III [Ang87]

$$S = \{\epsilon, a, b, bb\}, E = \{\epsilon\}$$

$T_3$		$E$
		$\epsilon$
$S$	$\epsilon$	1
	$a$	0
	$b$	0
	$bb$	1
$S \bullet A$	$aa$	1
	$ab$	0
	$ba$	0
	$bba$	0
	$bbb$	0

- The observation table  $T_3$  is closed, but not consistent, since  $\text{row}(a) = \text{row}(b)$  but  $\text{row}(aa) \neq \text{row}(ba)$ .
- $L^*$  adds the string  $a$  to  $E$  and queries the strings  $aaa$ ,  $aba$ ,  $baa$ ,  $bbaa$ , and  $bbba$  to construct the table  $T_4$ .



# $L^*$ Algorithm - Example IV [Ang87]

$$S = \{\epsilon, a, b, bb\}, E = \{\epsilon, a\}$$

$T_4$		$E$	
		$\epsilon$	$a$
$S$	$\epsilon$	1	0
	$a$	0	1
	$b$	0	0
	$bb$	1	0
$S \bullet A$	$aa$	1	0
	$ab$	0	0
	$ba$	0	0
	$bb a$	0	1
	$bb b$	0	0

$M_2/\delta$	$a$	$b$
$q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_2$
$q_2$	$q_2$	$q_0$

- The observation table  $T_2$  is consistent and closed.
- $L^*$  makes a conjecture of the acceptor  $M_2$ .
- The initial state of  $M_2$  is  $q_0$  and the final state is also  $q_0$ .
- The teacher selects a counterexample  $abb$  (accepted by  $M_1$ , but not in  $U$ ).

# $L^*$ Algorithm - Example V [Ang87]

$T_5$		$E$	
		$\epsilon$	$a$
$S$	$\epsilon$	1	0
	$a$	0	1
	$b$	0	0
	$bb$	1	0
	$ab$	0	0
	$abb$	0	1
$S \bullet A$	$aa$	1	0
	$ba$	0	0
	$bba$	0	1
	$bbb$	0	0
	$aba$	0	0
	$abba$	1	0
	$abbb$	0	0

$$S = \{\epsilon, a, b, bb, ab, abb\}$$

$$E = \{\epsilon, a\}$$

- The observation table  $T_5$  is closed but not consistent since  $\text{row}(b) = \text{row}(ab)$  but  $\text{row}(bb) \neq \text{row}(abb)$ .
- $L^*$  adds the string  $b$  to  $E$  and queries the strings  $aab, bab, bbab, bbbb, abab, abbab,$  and  $abbbb$  to construct the table  $T_6$ .





# $L^*$ Algorithm - Example VI [Ang87]

$T_6$		$E$		
		$\epsilon$	$a$	$b$
$S$	$\epsilon$	1	0	0
	$a$	0	1	0
	$b$	0	0	1
	$bb$	1	0	0
	$ab$	0	0	0
	$abb$	0	1	0
$S \bullet A$	$aa$	1	0	0
	$ba$	0	0	0
	$bba$	0	1	0
	$bbb$	0	0	1
	$aba$	0	0	1
	$abba$	1	0	0
	$abbb$	0	0	0

$$S = \{\epsilon, a, b, bb, ab, abb\}$$

$$E = \{\epsilon, a, b\}$$

$M_3/\delta$	$a$	$b$
$q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_3$
$q_2$	$q_3$	$q_0$
$q_3$	$q_2$	$q_1$

- The observation table  $T_2$  is consistent and closed.
- $L^*$  makes a conjecture of the acceptor  $M_2$ .
- The initial state of  $M_3$  is  $q_0$  and the final state is also  $q_0$ .
- The teacher replies to this conjecture with yes.
  - $M_3$  is a correct acceptor for the language  $U$ .

# $L^*$ Algorithm Performance

- The example:
  - # MQ: 25
  - # EQ: 3
- Real protocols

Protocol	States	Letters	MQ	EQ
Abp-lossy	3	3	22	2
Buff3	9	3	202	5
Dekker-2	2	3	7	1
Sched2	13	6	691	7
VMnew	11	4	513	7

- Synthetic data

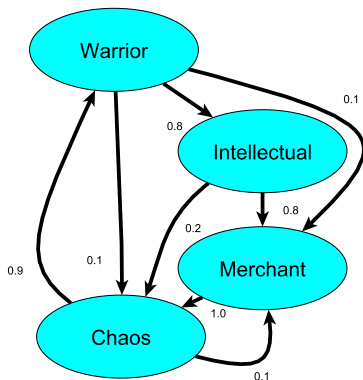
States	Letters	MQ	EQ
100	25	40000	15

- At present up to 1000 states.

# Outline

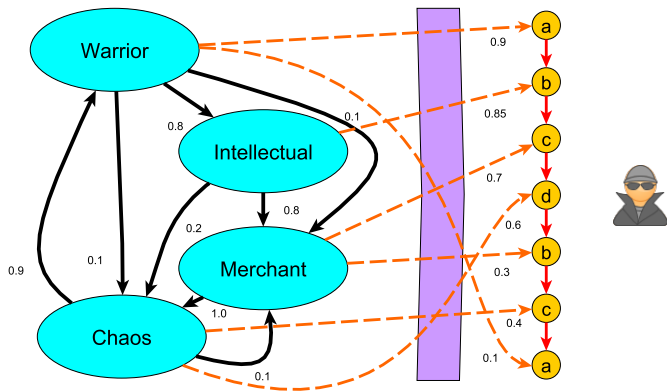
- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - Utility Function, Policy
  - Value Iteration
  - Policy Iteration
  - Conclusions

# Hidden Markov Model (HMM) - Overview



- 1 Many observation sequences  $\rightarrow$  FSM model learning
  - Iterative Baum-Welch algorithm <sup>[BP66]</sup> - Expectation-Maximization (EM)
- 2 FSM Model + an observation sequence  $\rightarrow$  the probability of the state sequence
  - The Viterbi algorithm
- 3 FSM Model + a sequence part  $\rightarrow$  the most probable states

# Hidden Markov Model (HMM) - Overview



- 1 Many observation sequences  $\rightarrow$  FSM model learning
  - Iterative Baum-Welch algorithm <sup>[BP66]</sup> - Expectation-Maximization (EM)
- 2 FSM Model + an observation sequence  $\rightarrow$  the probability of the state sequence
  - The Viterbi algorithm
- 3 FSM Model + a sequence part  $\rightarrow$  the most probable states

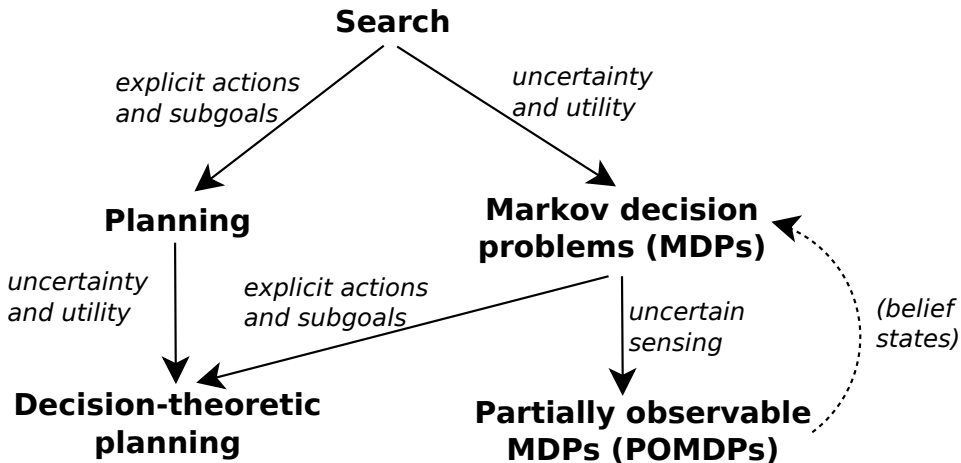
# Outline

- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - **Introduction**
  - Utility Function, Policy
  - Value Iteration
  - Policy Iteration
  - Conclusions

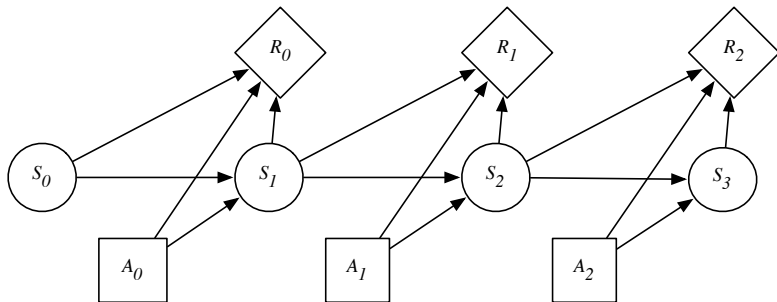
# Sequential Decisions <sup>[RN10]</sup>

- Achieving agent's objectives often requires **multiple steps**.
- A rational agent does not make a multi-step decision and carry it out without considering **revising** it based on future information.
  - Subsequent actions can depend on what is observed
  - What is observed depends on previous actions
- Agent wants to **maximize reward** accumulated along its course of action
- What should the agent do if environment is **non-deterministic**?
  - Classical planning will not work
  - Focus on state sequences instead of action sequences

# Sequential Decision Problems <sup>[Jak10]</sup>





Markov Decision Process <sup>[PM10]</sup>

# Markov Decision Process <sup>[PM10]</sup>

## Definition (Markov Decision Process)

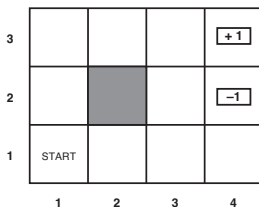
A **Markov Decision Process (MDP)** is a 5-tuple  $\langle S, A, T, R, s_0 \rangle$  where

- $S$  is a set of **states**
  - $A$  is a set of **actions**
  - $T(S, A, S')$  is the **transition model**
  - $R(S)$  is the **reward function**
  - $s_0$  is the **initial state**
- 
- Transitions are **Markovian**

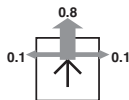
$$P(S_n|A, S_{n-1}) = P(S_n|A, S_{n-1}, S_{n-2}, \dots, S_0) = T(S_{n-1}, A, S_n)$$



# Example: Simple Grid World <sup>[RN10]</sup>



(a)



(b)

## Simple 4x3 environment

- States  $S = \{(i, j) | 1 \leq i \leq 4 \wedge 1 \leq j \leq 3\}$
- Actions  $A = \{up, down, left, right\}$
- Reward function

$$R(s) = \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

- Transition model  $T((i, j), a, (i', j'))$  given by (b)

# Outline

- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - **Utility Function, Policy**
  - Value Iteration
  - Policy Iteration
  - Conclusions

# Utility Function <sup>[RN10, Jak10]</sup>

- **Utility function** captures agent's preferences
  - In sequential decision-making, utility is a function over **sequences** of states
- Utility function accumulates rewards:
  - **Additive** rewards (special case):

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

- **Discounted** rewards

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

where  $\gamma \in [0, 1]$  is the **discount factor**

- Discounted rewards for  $\gamma < 1$  **finite** even for infinite horizons (see next slide)
- No other way of assigning utilities to state sequences is possible assuming stationary preferences between state sequences



# Policy <sup>[RN10, Jak10]</sup>

- A **stationary policy** is a function

$$\pi : S \rightarrow A$$

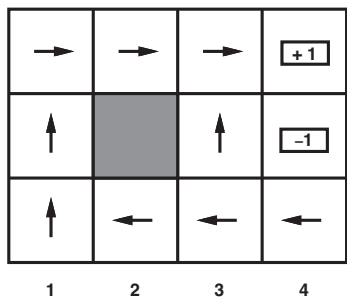
- **Optimal policy** is a function maximizing expected utility

$$\pi^* = \arg \max_{\pi} E[U([s_0, s_1, s_2, \dots]) | \pi]$$

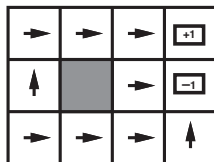
- For an MDP with stationary dynamics and rewards with infinite horizon, there **always exists** an optimal stationary policy
  - no benefit to randomize even if environment is random



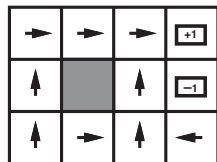
# Example: Optimal Policies in the Grid World [RN10, Jak10]



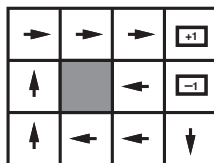
(a)



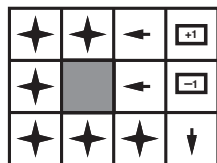
$$R(s) < -1.6284$$



$$-0.4278 < R(s) < -0.0850$$



$$-0.0221 < R(s) < 0$$



$$R(s) > 0$$

(b)

- (a) Optimal policy for state penalty  $R(s) = -0.04$
- (b) Dependence on penalty



# Decision-making Horizon <sup>[RN10, Jak10]</sup>

- A **finite horizon** means that there is a **finite deadline**  $N$  after which nothing matters (the game is over)
  - $\forall k \geq 1 \quad U_h([s_0, s_1, \dots, s_{N+k}]) = U_h([s_0, s_1, \dots, s_N])$
  - The optimal policy is **non-stationary**, i.e., it could change over time as the deadline approaches.
- An **infinite horizon** means that there is no deadline
  - The optimal policy is **stationary**  $\Leftrightarrow$  there is no reason to behave differently in the same state at different times
  - Easier than the finite horizon case
- **terminate / absorbing states** – agents stay there forever receiving zero reward at each step





# Outline

- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - Utility Function, Policy
  - **Value Iteration**
  - Policy Iteration
  - Conclusions

# Solving MDPs <sup>[RN10, Jak10]</sup>

- How do we find the optimum policy  $\pi^*$ ?
- Two basic techniques:
  - 1 **value iteration** – compute utility  $U(s)$  for each state and use it for selecting best action
  - 2 **policy iteration** – represent policy explicitly and update it in parallel to the utility function



# Utility of State <sup>[RN10, Jak10]</sup>

- Utility of a state under a given policy  $\pi$ :

$$U^\pi(s) = E \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s \right]$$

- **True utility**  $U(s)$  of a state is the utility assuming optimum policy  $\pi^*$

$$U(s) := U^{\pi^*}(s)$$

- Reward  $R(s)$  is “**short-term**” reward for being in  $s$ ;  
utility  $U(s)$  is a “**long-term**” **total** reward from  $s$  onwards
- Selecting the optimum action according to the MEU (Maximum Expected Utility) principle

$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') U(s')$$



# Bellman Equation <sup>[RN10, Jak10]</sup>

- Definition of utility of states leads to a simple relationship among utilities of neighboring states
- The utility of a state is the immediate reward for the state plus the expected discounted utility of the next state, assuming the agent chooses the optimal action

## Definition (Bellman equation (1957))

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad \forall s \in S$$

- One equation per state  $\Rightarrow n$  **non-linear** equations for  $n$  unknowns
  - The solution is **unique**



# Iterative Solution <sup>[RN10, Jak10]</sup>

- Analytical solution is not possible  $\Rightarrow$  **iterative approach**

## Definition (Bellman update)

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \quad \forall s \in S$$

- Dynamic programming: given an estimate of the  $k$ -step lookahead value function, determine the  $k + 1$ -step lookahead utility function.
- If applied infinitely often, **guaranteed to reach an equilibrium** and the **final utility values are the solutions** to the Bellman equations
- Value iteration **propagates information** through the state space by means of local updates.



# Value Iteration Algorithm [RN10, Jak10]

**Input:**  $mdp$ , a MDP with states  $S$ , transition model  $T$ , reward function  $R$ , discount  $\gamma$

**Input:**  $\epsilon$ , the maximum error allowed in the utility of a state

**Local variables:**  $U, U'$ , vectors of utilities for states in  $S$ , initially zero

**Local variables:**  $\delta$ , the maximum change in the utility of any state in an iteration

**repeat**

$U \leftarrow U'; \delta \leftarrow 0$  ;

**foreach** state  $s \in S$  **do**

$U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s')U[s']$  ;

**if**  $|U'[s] - U[s]| > \delta$  **then**

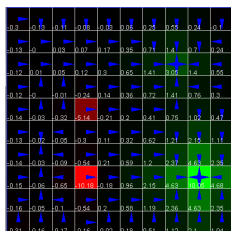
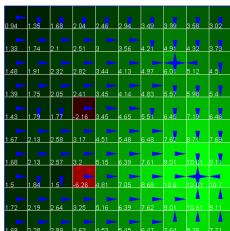
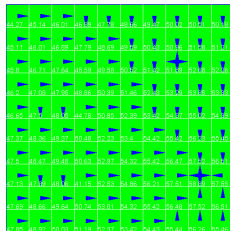
$\delta \leftarrow |U'[s] - U[s]|$ ;

**end**

**end**

**until**  $\delta < \epsilon(1 - \gamma)/\gamma$ ;

**return**  $U$

Value Iteration Example <sup>[RN10, PM10, Jak10]</sup>(a)  $\gamma = 0.6$ (b)  $\gamma = 0.9$ (c)  $\gamma = 0.99$ 

- 4 movement actions; 0.7 chance of moving in the desired direction, 0.1 in the others
- $R = -1$  for bumping into walls; four special rewarding states
  - +10 (at position (9,8); 9 across and 8 down),
  - one worth +3 (at position (8,3)),
  - one worth -5 (at position (4,5)) and
  - one -10 (at position (4,8))



# Outline

- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - Utility Function, Policy
  - Value Iteration
  - **Policy Iteration**
  - Conclusions



# Policy Iteration <sup>[RN10, Jak10]</sup>

- Search for optimal policy and utility values simultaneously
- Alternates between two steps:
  - 1 **policy evaluation** – recalculates values of states  $U_i = U^{\pi_i}$  given the current policy  $\pi_i$
  - 2 **policy improvement/iteration** – calculates a new MEU policy  $\pi_{i+1}$  using one-step look-ahead based on  $U_i$
- Terminates when the policy improvement step yields no change in the utilities.

# Policy Iteration Algorithm <sup>[RN10, Jak10]</sup>

**Input:**  $mdp$ , a MDP with states  $S$ , transition model  $T$

**Local variables:**  $U$ , a vector of utilities for states in  $S$ , initially zero

**Local variables:**  $\pi$ , a policy vector indexed by state, initially random

**repeat**

$U \leftarrow \text{Policy-Evaluation}(\pi, U, mdp)$  ;

$unchanged? \leftarrow \text{true}$ ;

**foreach** state  $s \in S$  **do**

**if**  $\max_a \sum_{S'} T(s, a, s')U[s'] > \sum_{S'} T(s, \pi(s), s')U[s']$  **then**

$\pi(s) \leftarrow \arg \max_a \sum_{S'} T(s, a, s')U[s']$ ;

$unchanged? \leftarrow \text{false}$ ;

**end**

**end**

**until**  $unchanged?$ ;

**return**  $\pi$

# Policy Evaluation [RN10, Jak10]

- Simplified Bellman equations:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \quad \forall s \in S$$

- The equations are now **linear**  $\Rightarrow$  can be solved in  $O(n^3)$



# Modified Policy Iteration <sup>[RN10, Jak10]</sup>

- Policy iteration often converges in few iterations but each iteration is **expensive**
  - $\Leftarrow$  has to solve large systems of linear equations
- Main idea: use **iterative approximate** policy evaluation
  - **Simplified Bellman update:**

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s') \quad \forall s \in S$$

- Use a few steps of value iteration (with  $\pi$  fixed)
  - Start from the value function produced in the last iteration
- Often **converges much faster** than pure value iteration or policy iteration (combines the strength of both approaches)
- Enables much more general **asynchronous** algorithms
  - e.g. Prioritized sweeping



# Choosing the Right Technique <sup>[RN10, Jak10]</sup>

- Many actions?  $\Rightarrow$  policy iteration
- Already got a fair policy?  $\Rightarrow$  policy iteration
- Few actions, acyclic?  $\Rightarrow$  value iteration
- Modified policy iteration typically the best

# Outline

- 1 FSM Learning
  - FSM Learning Overview
  - Angluin's Algorithm
  - Example
- 2 Hidden Markov Model
  - A Brief Overview
- 3 Markov Decision Process
  - Introduction
  - Utility Function, Policy
  - Value Iteration
  - Policy Iteration
  - Conclusions

# Conclusions <sup>[RN10, Jak10]</sup>

- MDPs generalize deterministic state space search to **stochastic environments**
  - At the expense of computational complexity
- An **optimum policy** associates an optimal action with every state
- **Iterative techniques** used to calculate optimum policies
  - basic: value iteration and policy iteration
  - improved: modified policy iteration, asynchronous policy iteration
- Further issues
  - **large state spaces** – use state space approximation
  - **partial observability** (POMDPs) – need to consider information gathering; can be mapped to MDPs over continuous belief space

# Literatura I

- [Ang86] Dana Angluin. Learning regular sets from queries and counter-examples. Technical Report YALEU/DCS/TR-464, Yale University, Department of Computer Science, March 1986.
- [Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [BP66] Leonard E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.
- [Hon13] Marek Honzírek. Aktivní učení a automaty. Master's thesis, Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, ČVUT, Praha, 2013.
- [Jak10] Michal Jakob. A3M33UI decision theory essentials, lecture notes. <http://cw.felk.cvut.cz/doku.php/courses/a3m33ui/prednasky>, February 2010.
- [PM10] David Poole and Alan Mackworth. Artificial intelligence, foundations of computational agents. <http://artint.info/slides/slides.html>, 2010.
- [RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach*. Pre, third edition, 2010.
- [Sha08] Muzammil Muhammad Shahbaz. *Reverse Engineering Enhanced State Models of Black Box Software Components to support Integration Testing*. PhD thesis, Institut Polytechnique de Grenoble, 2008.