COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Senem Velipasalar

Lecture #22
April 24, 2003

1 Angluin's Algorithm

1.1 From Last Time

$Q =$ states
$q_0 =$ start state
$\delta : Q \times \{a, b\} \rightarrow Q$ (transition function)
$v : Q \rightarrow \{0, 1\}$ (accept/reject)
qs : state reached by executing s from q
$M(s) = v(q_o s)$
$n = |Q|$

•Maintain set T(initially $T = \{\lambda\}$)
•Use T and membership queries to maintain tree and to build $\hat{M}$
• Make equivalence query $\hat{M}$
• If $M = \hat{M}$ done
• else use counterexample to add to T.

1.2 How to grow T in order to find the FSM

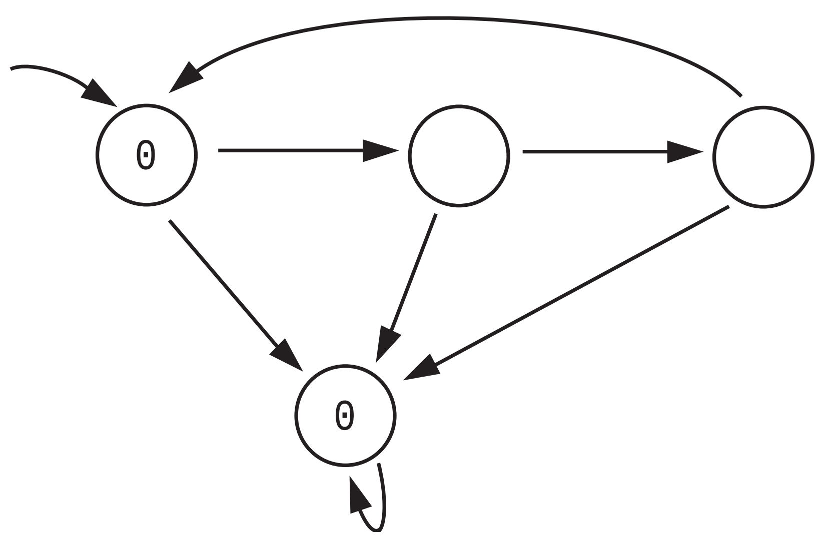

Figure 1: FSM M which we want to find

Let's run through the algorithm by using Fig. 1. Initially $T = \{\lambda\}$.

Fig. 2 shows how $\hat{M}$, in Fig. 3, is built by starting with $T = \{\lambda\}$.

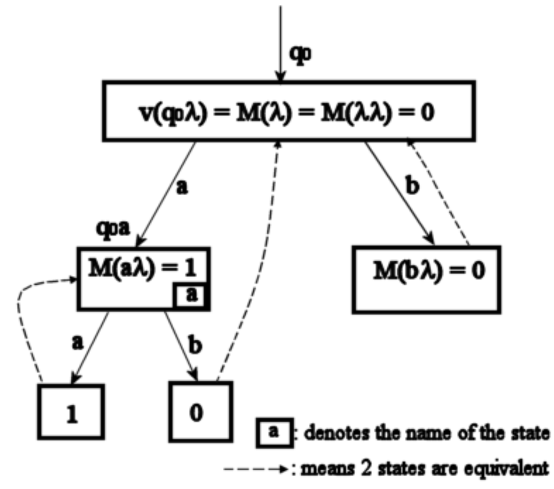

Figure 2: The tree shows how $\hat{M}$ is built by starting with $T = \{\lambda\}$.

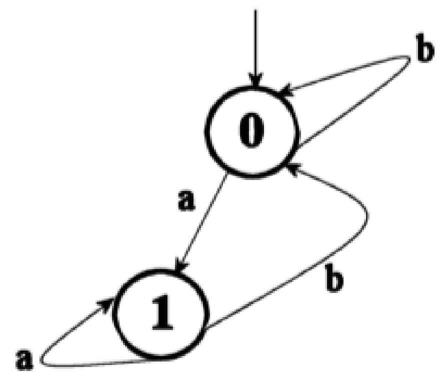

Figure 3: $\hat{M}$ obtained by starting with $T = \{\lambda\}$.

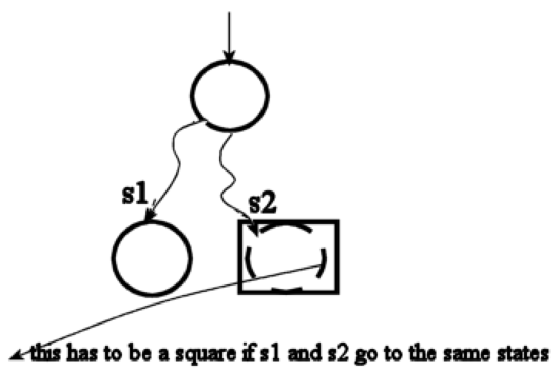

Figure 4: Shows that number of internal nodes cannot be larger than number of states

Number of internal nodes will never be larger than n. This can be seen from figure 4.

Now we will use a counterexample to grow T. **Counterexample**: *baaa* (In Fig. 1 you get 0, but in $\hat{M}$, Fig. 3 you get 1)

$[s]$ = state reached in $\hat{M}$ on string s (name of the reached state)(**Note:** Name of a state is the string that brings you to that state from the initial state.)
$[aba] = a$
$[aab] = \lambda$
$\hat{M}(s) = \hat{v}(\hat{q}_o s) = v(q_o[s]) = M([s]) = \hat{M}([s])$

where, $\hat{M}(s)$ is the output of the machine when s is executed. Here, $\hat{M}(s) = \hat{M}([s])$ because both s and $[s]$ reach the same state in $\hat{M}$.

Fact: $[[s]w] = [sw]$. This can be seen from Fig. 5.

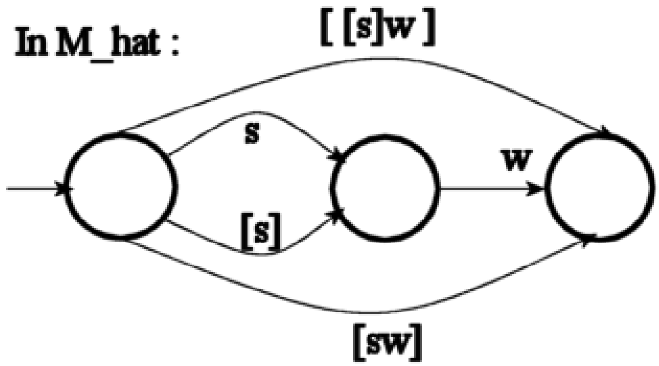

Figure 5: $[[s]w] = [sw]$

We are trying to figure out what to add to $T = \{\lambda\}$. Suppose we can find:
s: internal node
$e \in \{a, b\}$
t: that distinguishes se from $[se]$, i.e. $M(set) \neq M([se]t)$
Claim: If we add t to T, se will become an internal node, i.e., the number of internal nodes will increase. Because $M(set) \neq M([se]t)$This can be seen from Fig. 6.

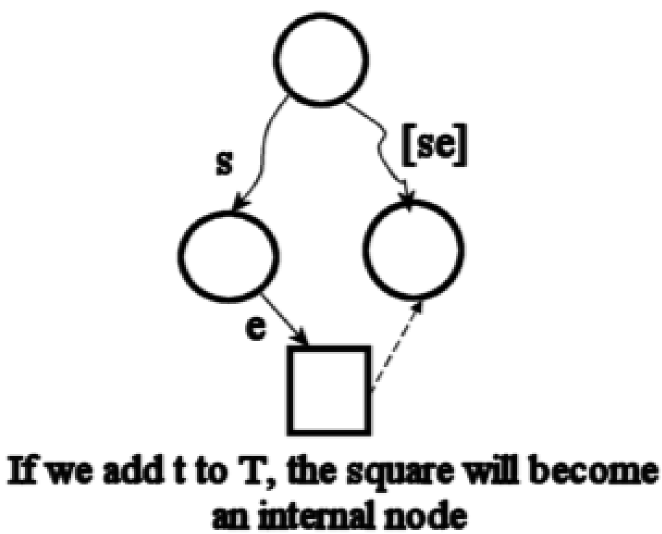

Figure 6: If we add t to T the number of internal nodes will increase

Now, we will use the counterexample $baaa$ to figure out what we should add to T.

$M([\lambda]baaa) = M(baaa) = 0$
$\hat{M}(baaa) \;=\; M([baaa]) = 1$

So, we got a 0 and a 1. Now we will fill in between step by step, and find two consecutive outputs that are different.

$$\begin{aligned} M([\lambda]baaa) &= 0 \\ M(\lambda aaa) = M([b]aaa) &= 0 \\ M(aaa) = M([ba]aa) &= 0 \\ M(aa) = M([baa]a) &= 1 \\ M([baaa]) &= 1 \end{aligned}$$

$M([ba]aa) \neq M([baa]a)$, so we can set:
$s = [ba]$
$e = a$, and
$t = a$
Let's double check to see that t distinguishes se from $[se]$.
$se = [ba]a$
$[se] = [[ba]a] = [baa]$ (from the above fact)
So, $M(set) = M([ba]aa) \neq M([baa]a) = M([se]t)$.

Number of equivalence queries$\leq n$ (because every time we use an equivalence query, we increase the number of internal nodes by 1 and the total number of internal nodes $\leq n$).

Number of membership queries $\leq O(n^2 + rn)$ where r is the length of the longest counterexample.

2 Reinforcement Learning

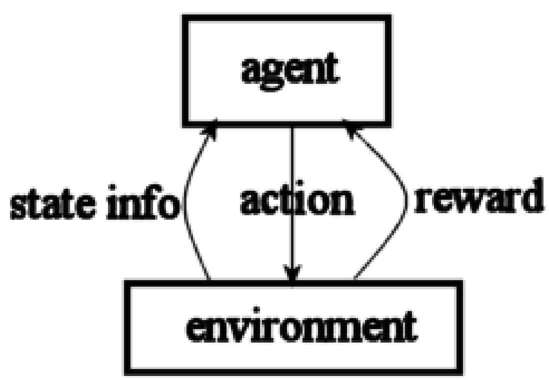

Figure 7: Summary of reinforcement learning

Agent decides on an action based on the state. We want to find a way to choose the actions to maximize the reward.
Policy: A mapping from states to actions.

$$s_0 \xrightarrow[r_1]{a_0} s_1 \xrightarrow[r_2]{a_1} s_2 \xrightarrow[r_3]{a_2} \ldots \tag{1}$$

Two things to consider:
- Delayed reward
- The tradeoff between exploration and exploitation. (Should you try different things or always go with the way you know?)

Rewards and states are random variables.

We will assume that $Pr[s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \cdots] = Pr[s_{t+1}|s_t, a_t]$ and also that r_i depends only on s_{i-1} and a_{i-1}. These assumptions are called the "Markov property".

Example: In Fig. 8, when you choose to go up, you may fall down with probability 0.01, or reach next level with probability 0.99. When you fall down, you get a negative reward depending on how high you were before you fell down. When you reach the top level you get a reward of 100.

In a Markov Decision Process (MDP), we have:
S: States,
A: Actions,
distribution defining $Pr[s_{t+1} = s'|s_t = s, a_t = a] = Pr[s'|s, a]$, and
distribution defining (rewards): $Pr[r_{t+1}|s_t, a_t]$
We will be more interested in $E[r_{t+1}|s_t = s, a_t = a] = r(s, a)$.

Policy $\pi : S \to A$
Goal: Find the best π that maximizes the total reward.
How do we define "best"?: The value of a policy is the reward we get when we use that policy.

$$\text{the value of } \pi \text{ at state } s = V^\pi(s) = E_\pi[r_{t+1} + r_{t+2} + \cdots |s_t = s]$$
$$a_t = \pi(s_t)\text{: the action recommended by } \pi$$
$$R_t = r_{t+1} + r_{t+2} + \cdots \ : \text{return}$$

The rewards in the future will be made less valuable, in other words, we will discount the future rewards. So R_t will be:
$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$ where $\gamma \in [0, 1)$
Thus if γ is large, we are interested in longer term. If $|r_t| \leq K$, then

$$| R_t | \leq K + \gamma K + \gamma^2 K + \cdots = \tfrac{K}{1-\gamma}$$

The goal is: $\max_\pi V^\pi(s)$ for all s.

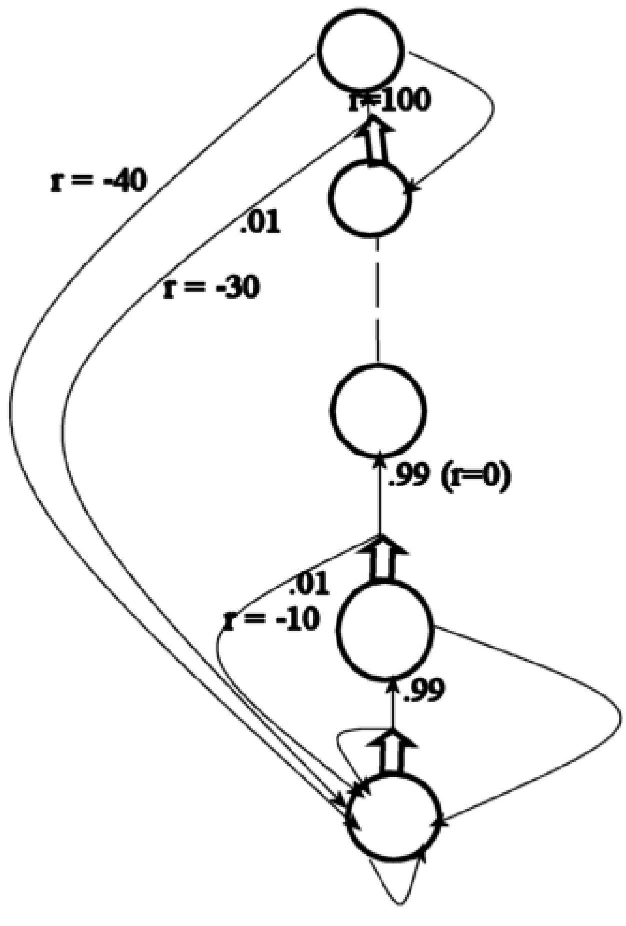

Figure 8: Markov Decision Process