5. Learning

5.1 What is learning?

- Herbert Simon:

  "Any change in the system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." (Simon, 1963)

- Rob Callan:

  "A process that allows an agent to adapt its performance through instruction or experience." (Callan, 2003)

5.2 Why learning?

- Accumulate knowledge
- Improve performance
- Gain efficiency: problems need not to be solved over and over again from scratch.
- Adapt to changes in the environment

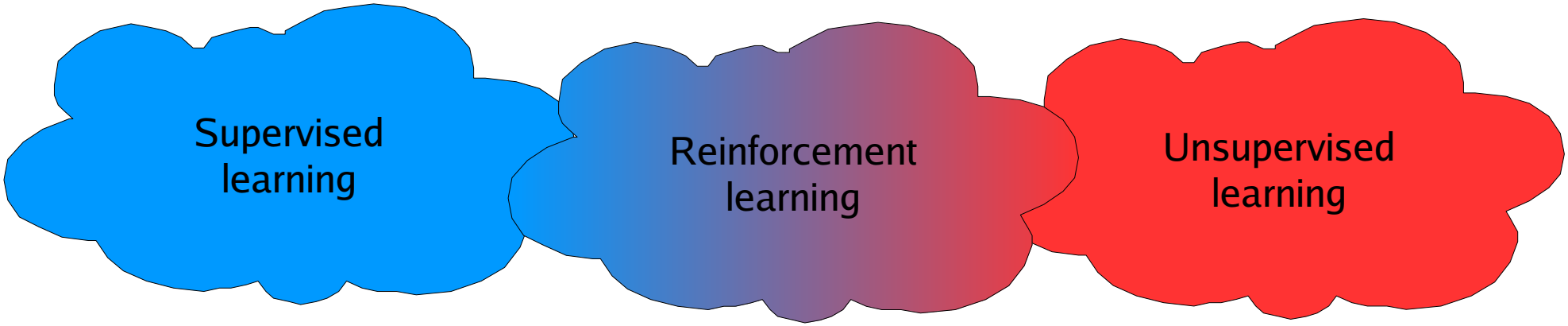
5.3 What is required?

- In order to learn:
  - Aptitude to change
  - Capability of telling relevant from irrelevant, useful from useless, important from unimportant
- In order to apply learned
  - Ability to generalize
  - Ability to recognize similar or analogical situations

5.4 Machine learning & AI

- To understand human cognition
- To make machines serve humans
- Types of learning task:
  - Classification learning
    - Recognition of hand-written characters
    - Fraud and fault detection
  - Learning sequence of actions
    - Playing games
    - Navigation
  - Learning optimal decisions
  - Learning a regression function
    - Association between one variable and other variables
- Symbol-based vs. connectionist learning

5.5 Types of learning



- Learning to know
  - * Concept formation
  - * Classification learning

Machine learning

Feedback : Correct output
Teacher : yes

- Learning to behave
  - * Strategy learning

Feedback : Reward or punishment
Teacher : No

- Self-organization
  - *Pattern recognition

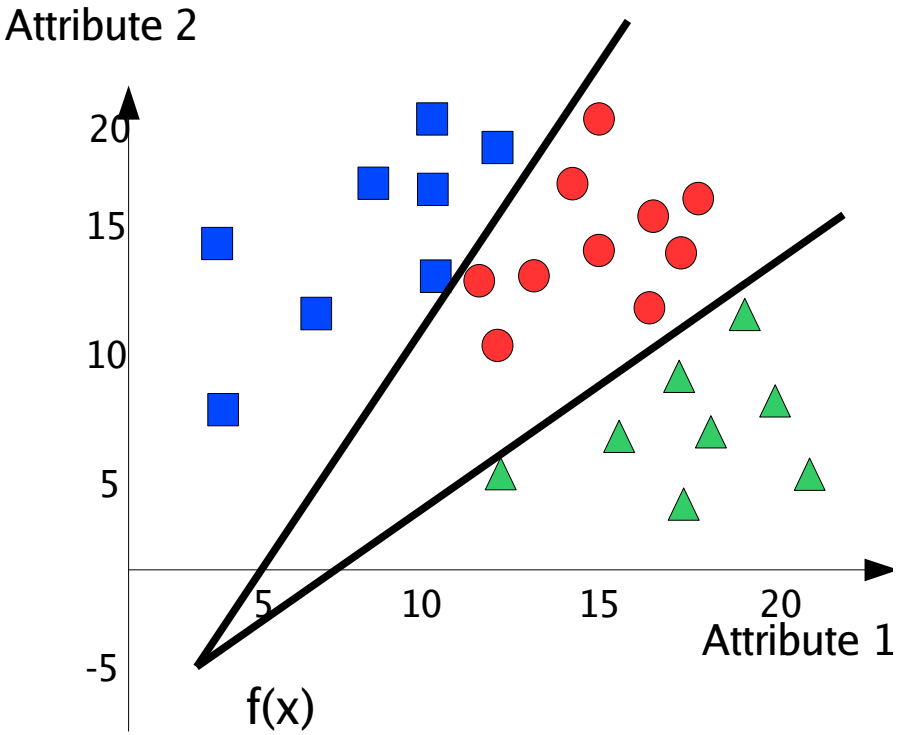
Neural network learning

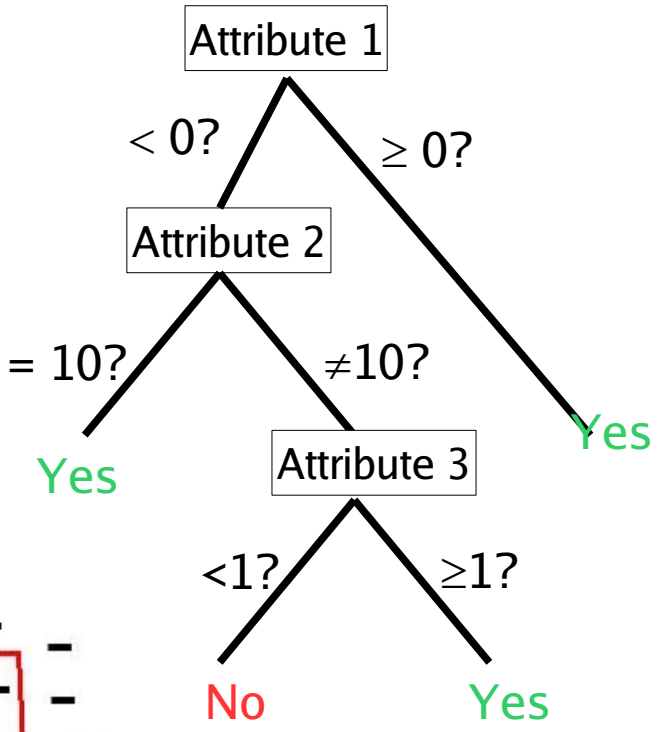
Feedback : None
Teacher : No

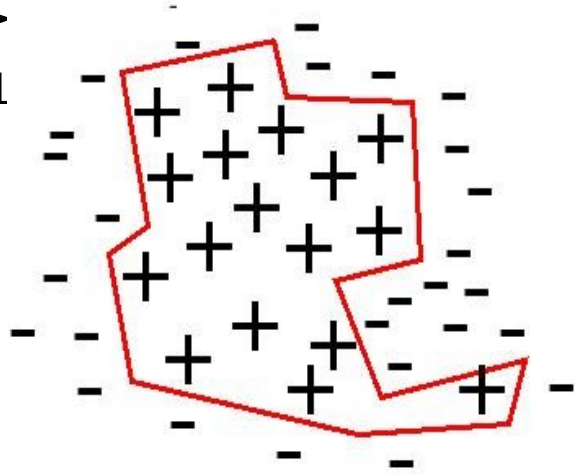
5.6 Elements of learning

- *Training examples* $X = x_1, ..., x_n$ (also called data)
- Each example x_i is a m-dimensional vector $<x_{i,1}, x_{i,2}, ... x_{i,m}>$ of attribute (or feature) values.
- *Target function* $f(x_i) = y_i$ is the goal of learning; given the input data (example) x_i, f returns the correct output, for instance the correct classification of the example x_i.
- *Hypothesis* $h(x_i)$ tries to approximate the target function f.

5.7 Discrete target functions







5.8 Supervised learning

- Simplest form of learning from experience = rote learning
  - Store each piece of training data with its correct classification.
  - When a new item is classified, look for it in the memory: if it is found, return the class, otherwise fail.
- Inductive inference = Learning general rules from specific examples. For instance,
  - Classification learning
  - Concept learning

- The training examples are of the form (x_1,y_1), (x_2,y_2), ... , (x_n,y_n), where y_i is the target value (e.g., the correct classification of x_i).
- Requires an external supervisor.
- Training:
  - The learning algorithm is given both the example and the target
- Goal:
  - To form a *hypothesis* of the relationship between the attributes and the target value.
- Testing
  - The learning algorithm is given the example without the target, and it returns the hypothesis.

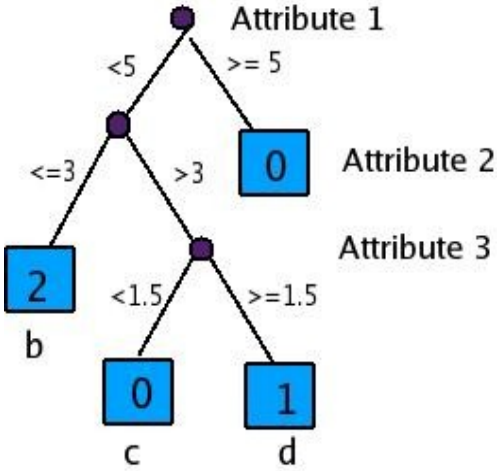
5.9 Classification learning

IRIS DATABASE (Fisher, 1936):

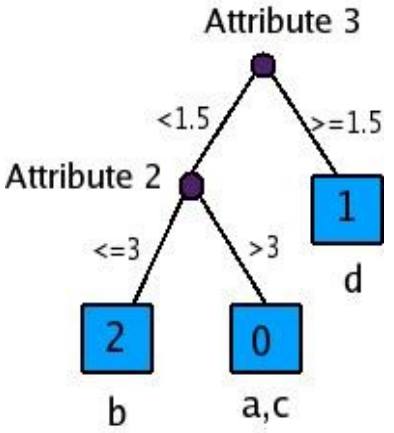
- Predictive attributes:
  1. Sepal width (cm)
  2. Petal length (cm)
  3. Sepal length (cm)
  4. Petal width (cm)
- Predicted attribute class:
  - Iris Setosa: Class 0
  - Iris Versicolour: Class 1
  - Iris Virginica: Class 2

(petal="teräl ehti", sepal="verhol ehti")

First attempt:



Second attempt:



- 150 examples, 50 in each class:

Sample:						
	a	5.1	3.5	1.4	0.2	0
	b	4.9	3.0	1.4	0.2	2
	c	4.7	3.2	1.3	0.2	0
	d	4.6	3.1	1.5	0.2	1

5.10 Concept learning

- Boolean target indicates whether the example belongs to the concept or not.
- Attributes:

1. Speed	(slow,medium,fast)
2. Weather	(wind,rain,snow,sun)
3. Distance to car ahead	(10m,20m,30m,40m,50m)
4. Units of alcohol drunk by driver	(0,1,2,3,4,5)
5. Time of day	(dawn,day,dusk,night)
6. Temperature	(cold,warm,hot)

- Concept: safe driving conditions
- Training data consists of positive examples:
  - <slow,wind,30m,0,dusk,cold,yes>
  - <slow,rain,20m,0,dusk,warm,yes>
  - <slow,snow,30m,0,dawn,cold,yes>

- Start with the most restrictive hypothesis:

  <Ø,Ø,Ø,Ø,Ø,Ø,yes>

- Generalize with attribute values that match the training samples:
  - After first example:

    <slow,wind,30m,0,dusk,cold,yes>
  - After the second:

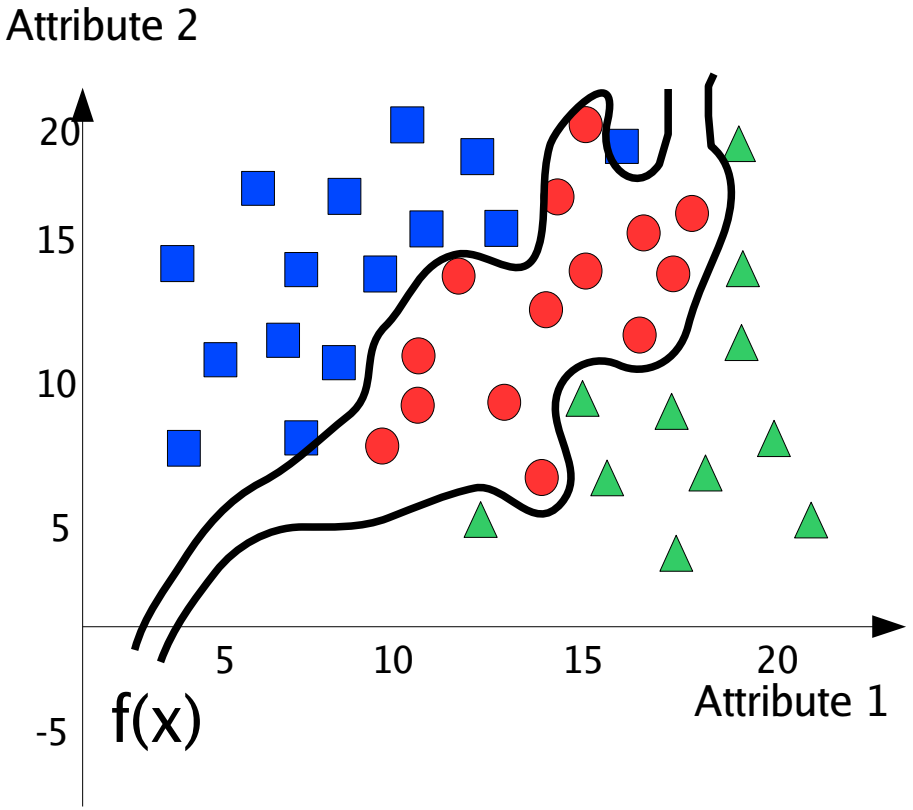
    <slow,?,?,0,dusk,?,yes>
  - And third:

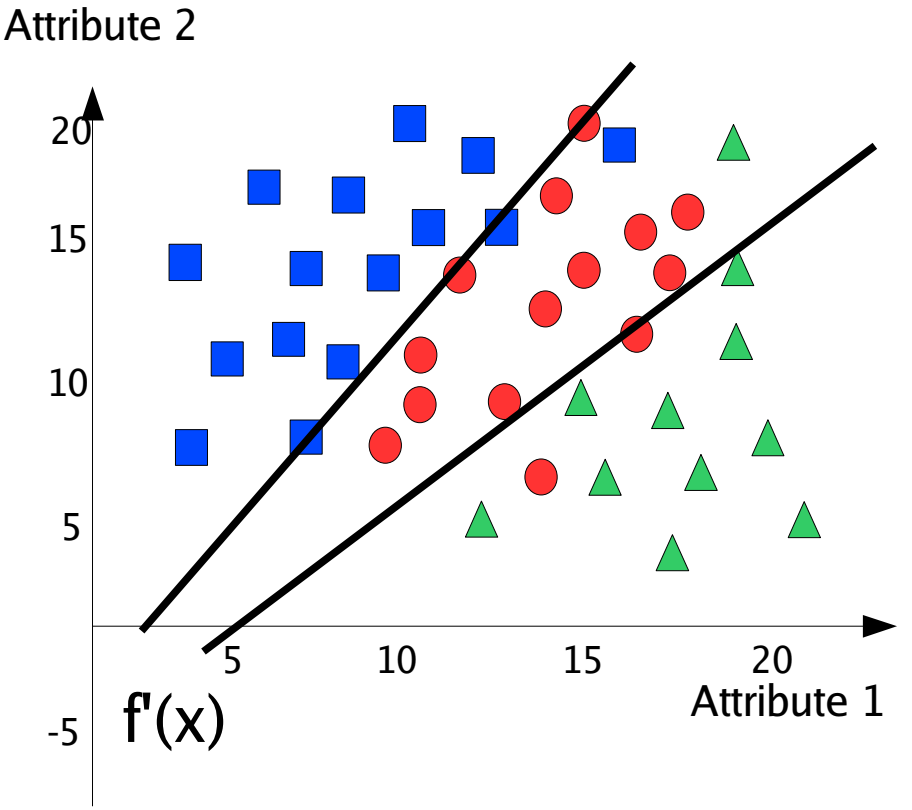
    <slow,?,?,0,?,?.yes>
- If also negative examples used, start also with the most general hypothesis.

5.11 Inductive bias and generalization

- Training data is usually samples from the larger population of all possible data.
- *Inductive bias* = restrictions imposed by the learning method.
- In order to have an unbiased learner we need all possible hypothesis that can exist $\rightarrow$ the solution would be the complete set of training data.
- Consequently, such a learner won't be able to *generalize* to new instances of the problem.
- Ockham's razor: "entiti es should not be multiplied beyond necessity," = one should prefer simplest explanation, i.e., one with fewest assumptions.

5.12 Overfitting





- There is always noise in the training data:
  - Random variation
  - Errors in measurements
  - Missing datapoints
- Sample size vs. hypothesis complexity

5.13 Explanation-based learning

- De Jong and Mooney (1986)
- EBL uses explicitly represented domain theory to construct explanations of training samples.
- *A target concept*
  - High-level description what is to be learned.
  - Find an effective definition for the concept.
- *A training sample*
  - Observed in the world
  - instance of the target.
- *A domain theory*
  - set of rules and facts used to explain how the training sample is an instance of the target.
- *Operationality criteria*
  - means of describing the form of concept definitions

5.14 EBL: Example

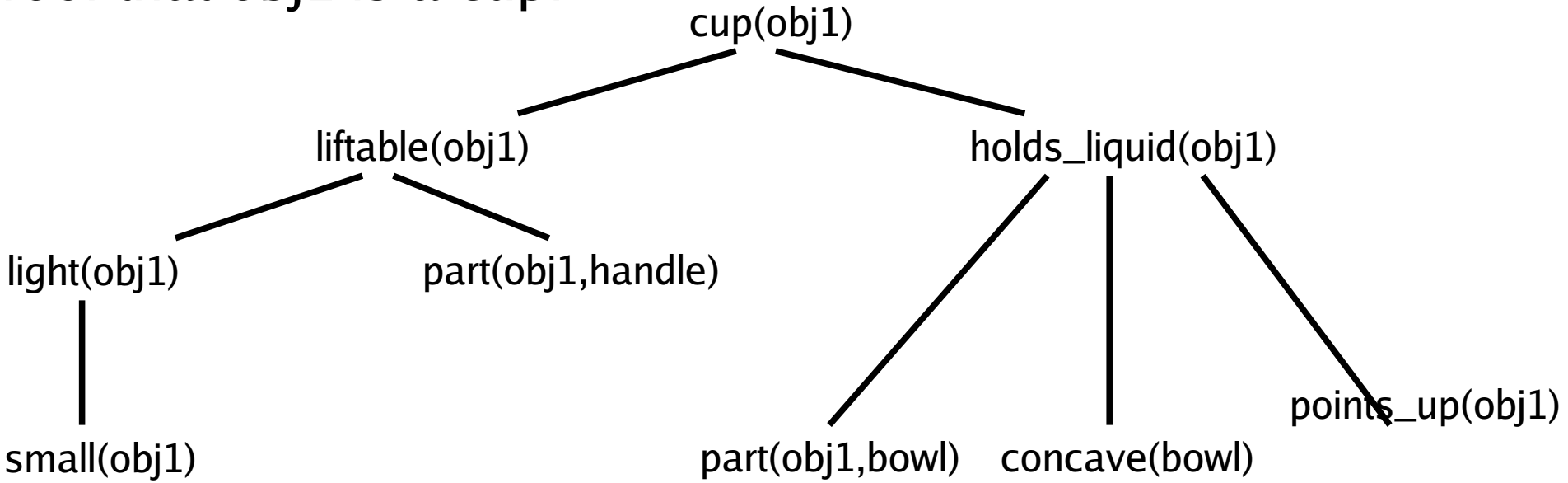
- Target: premise(X) $\rightarrow$ cup(X)
- Domain theory:

  liftable(X) $\wedge$ holds_liquid(X) $\rightarrow$ cup(X)
  part(Z,W) $\wedge$ concave(W) $\wedge$ points_up(W) $\rightarrow$ holds_liquid(W)
  light(Y) $\wedge$ part(Y, handle) $\rightarrow$ liftable(Y)
  small(A) $\rightarrow$ light(A)
  made_of(A, feathers) $\rightarrow$ light(A)

- Training samples:

cup(obj1)	small(obj1)	part(obj1,handle)
owns(bob,obj1)	part(obj1,bottom)	part(obj1,bowl)
points_up(bowl)	concave(bowl)	color(obj1,red)

Proof that obj1 is a cup:



- To construct a generalized proof tree:
  - Substitute variables for constants that are part of the training sample
  - Retain constants that are part of the domain theory.
- What does ELB learn?
  - Rules that are in a deductive closure of its theory.
  - Speed-up learning: focus on relevant aspects of the problem domain.
  - Makes implicit information in rules explicit.
- How to reason with imperfect or unsound theories?

5.15 Reinforcement learning

- Not characterized by learning method, but by a learning problem (Sutton & Barto, 1998)
- Evaluates actions taken instead of giving the correct one.
- Defining characteristics:
  - Interaction with the environment to achieve one's goals
  - Perceive the state of environment, and act to change the state
  - Learning from own experience
  - Uncertainty involved
- Two fundamental issues:
  - Trial and error learning
    - Exploration: try actions never tried before
    - Exploitation: retry actions that have been beneficial before
  - Delayed reward
    - Credit assignment problem

5.16 Framework

- *State* — whatever information is available for the learner
- *Policy* — mapping from perceived state to action available in that state.
- *Reward* function ~~—~~map ping from state-action pair to a single number
  - Immediate desirability of a state
  - Goal is to maximize the total reward in the long run.
  - Cannot be changed by the learner; considered part of the environment
- *Value* function ~~—~~the total amount of reward accumulated starting from a certain state
  - Long-term desirability of a state
  - Value estimation the most central issue
- *Model* of the environment
  - Used for planning
  - Learner's control defines the learner-environment boundary .

5.17 Key concepts 1

- At each time point $t = 0,1,...(,T)$ learner perceives the state $s_t \in S$ (set of possible states), selects an action $a_t \in A(s_t)$ (all actions available in s_t), receives a numerical reward $r_{t+1} \in$ R, and moves to state s_{t+1}.
- Policy $\pi_t(s,a)$ maps state s to the probability of choosing action a at time t in that state.
- Learning method determines how policy changes with experience.

5.18 Key concepts 2

- Sequence of rewards received after time t is denoted $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$
- Tasks that repeat time steps from 1 through T are called episodic.
- The learner wants to maximize the *expectation of the total return:* $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$
- Sometimes $T = \infty$.
- Then the learner wants to maximize the expectation of the *discounted return*:

  $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad 0 \leq \gamma < 1,$

  where γ is a discount rate.

5.19 Value functions

- How good it is to be in state s, or to perform action a in state s.
  - "How good" is defined as expected rewards in the future.
- *State-value* function for policy π, i.e., expected discounted return starting in state s, and follow policy π thereafter:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$$

- *Action-value* function for policy π, i.e., expected (discounted) return from taking action a in state s, and following policy π thereafter:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\Sigma_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}$$

5.20 Optimal policy

- A policy π is better than or equal to another policy π', if its expected return is better than or equal for all states, i.e., $\pi \geq \pi'$, iff $V^{\pi}(s) \geq V^{\pi'}(s)$, for all $s \in S$.
- There is always at least one policy that is better than or equal to the others, but there can be several such policies = they are called o*ptimal policies* π^*.
- In order to compute the optimal policy, learner needs to assess the expected rewards for any state and the projected accumulating rewards thereafter.
- Optimal policies share optimal state-value and action value functions:

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s,\ a_t = a\}$$

  - This gives the expected return for taking action a in state s and thereafter following the optimal policy.

5.21 Open problems

- How to find an optimal policy?
  - Compute all possible policies and simulate a large number of episodes $\rightarrow$ average the returns.
  - Too time consuming
  - Other methods for finding optimal policy:
    - Value iterations and policy iteration —assume that transition probabilities are known.
    - Q-learning
- Credit assignment problem
  - Temporal difference (TD) learning

5.22 Q-learning

- Probability model — transition probabilities between states ~~is~~ learned exploring the environment.
- Maintains Q-values for each state-action pair $Q_n(a,s)$, where n is the iteration.

  Initialize all $Q(a,s) \leftarrow 0$

  While exploring

    Select(*) action a in current state s, and move to state s'

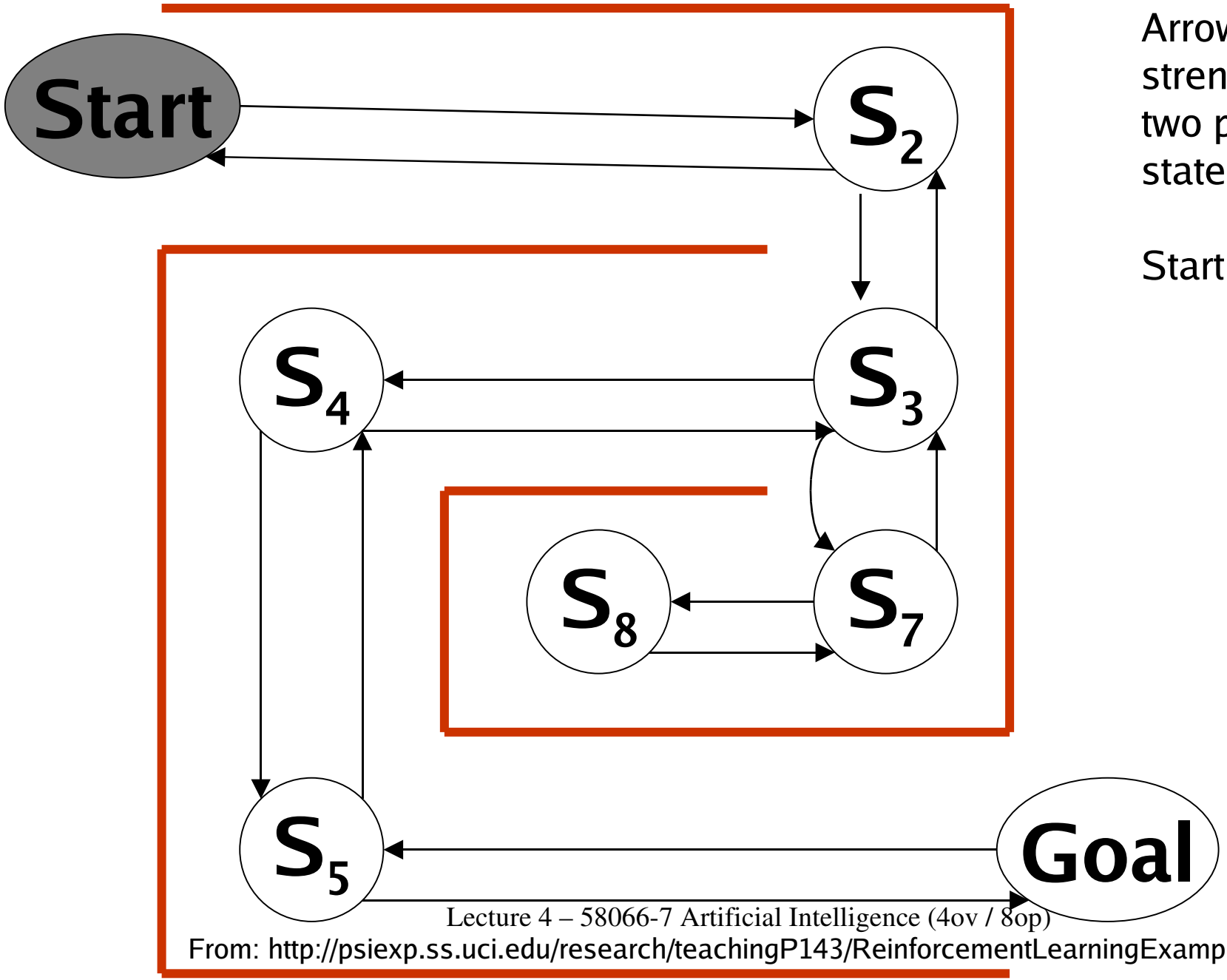
    Update Q-value in state s:

$$Q_n(a,s) \leftarrow (1 - \alpha_n)\, Q_{n-1}(a,s) + \alpha_n\, [R_{ss'} + \arg\max_{a'} Q_{n-1}(a',s')],$$

  where $\alpha_n = 1/[1 + \text{number\_of\_visits}(a, s, s')]$

(*) action selection can be done randomly, by largest Q-value, or stochastically proportional to the Q-value.

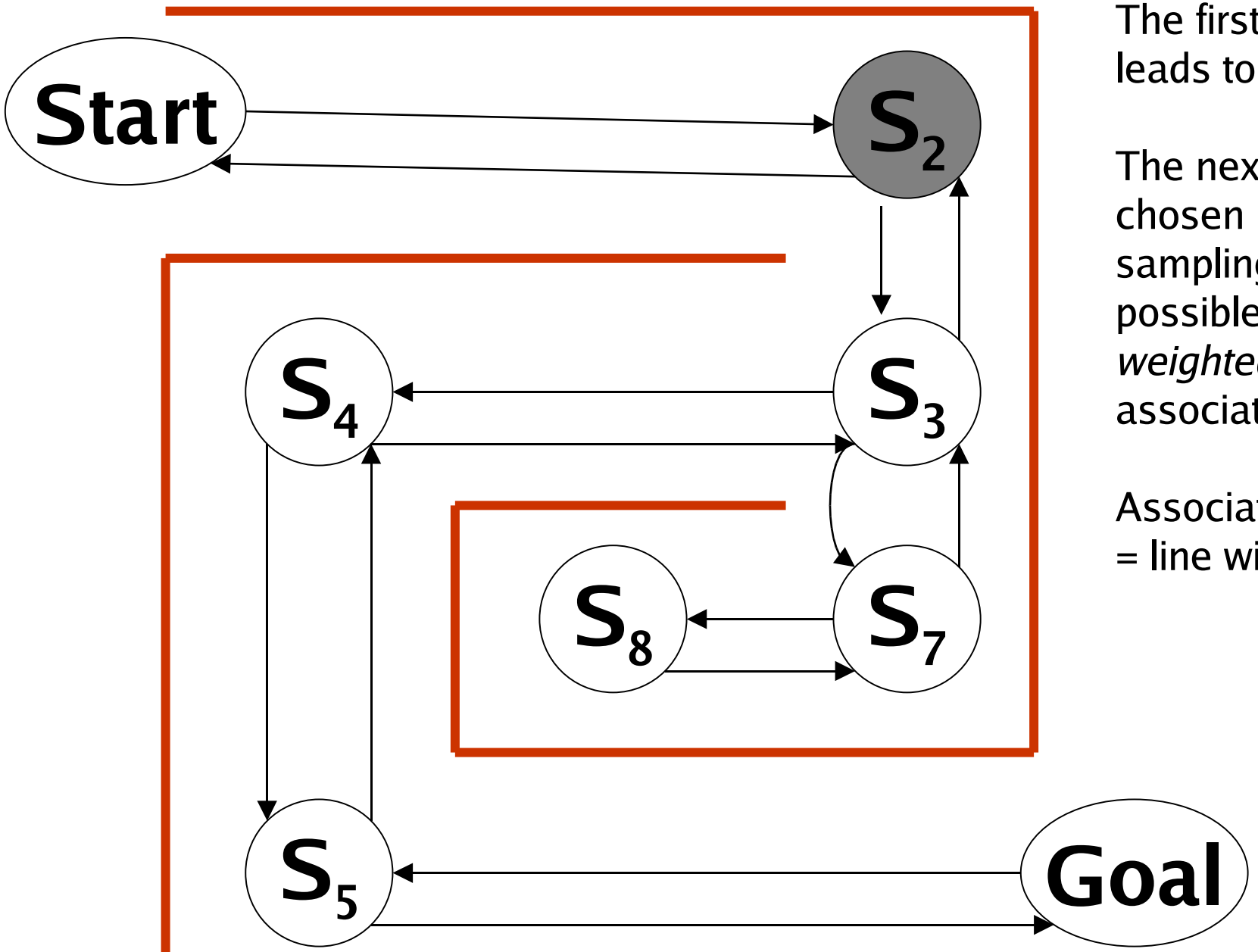
5.23 Reinforcement learning example



Arrows indicate strength between two problem states.

Start maze …

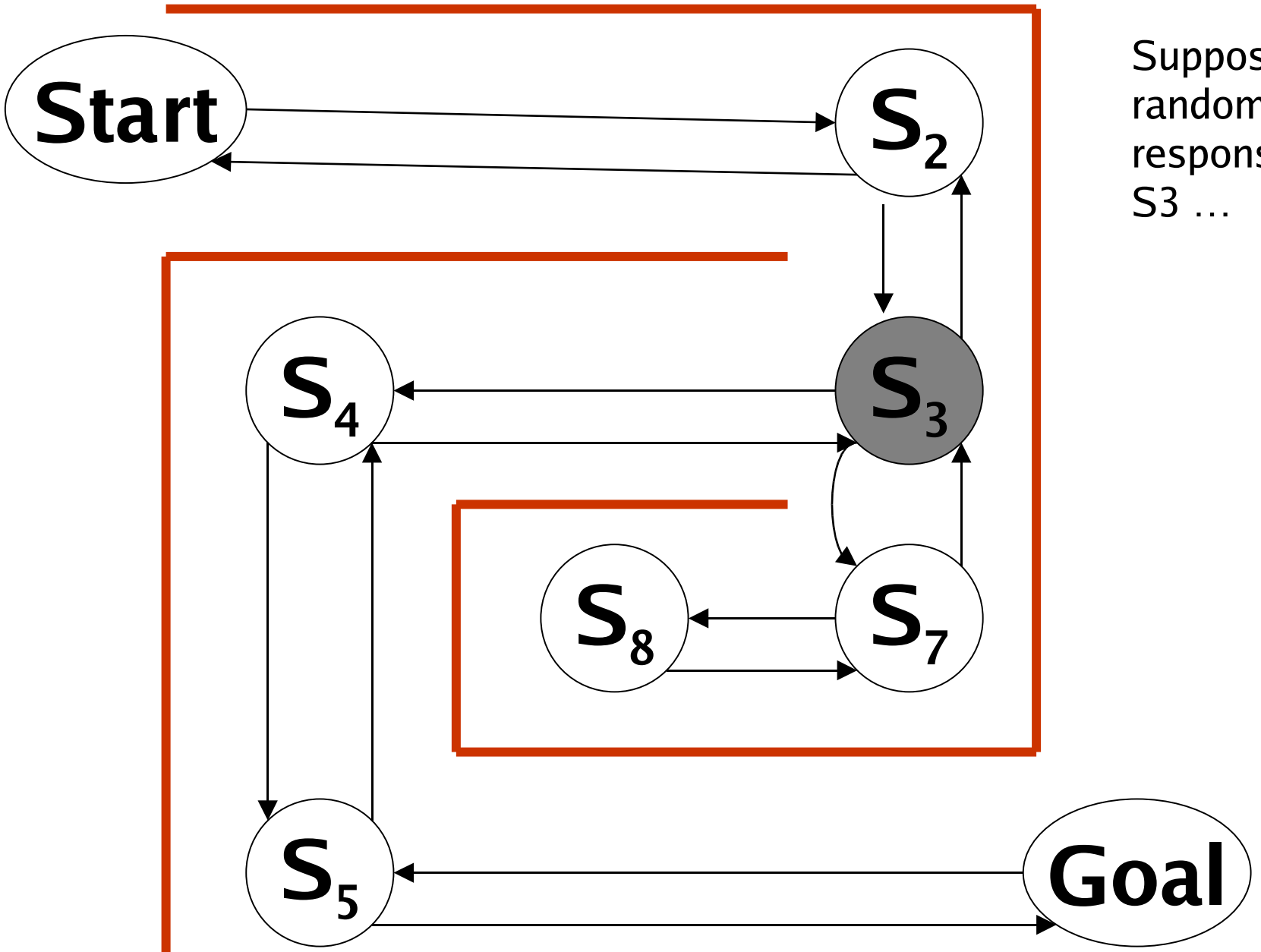
From: http://psiexp.ss.uci.edu/research/teachingP143/ReinforcementLearningExample.ppt



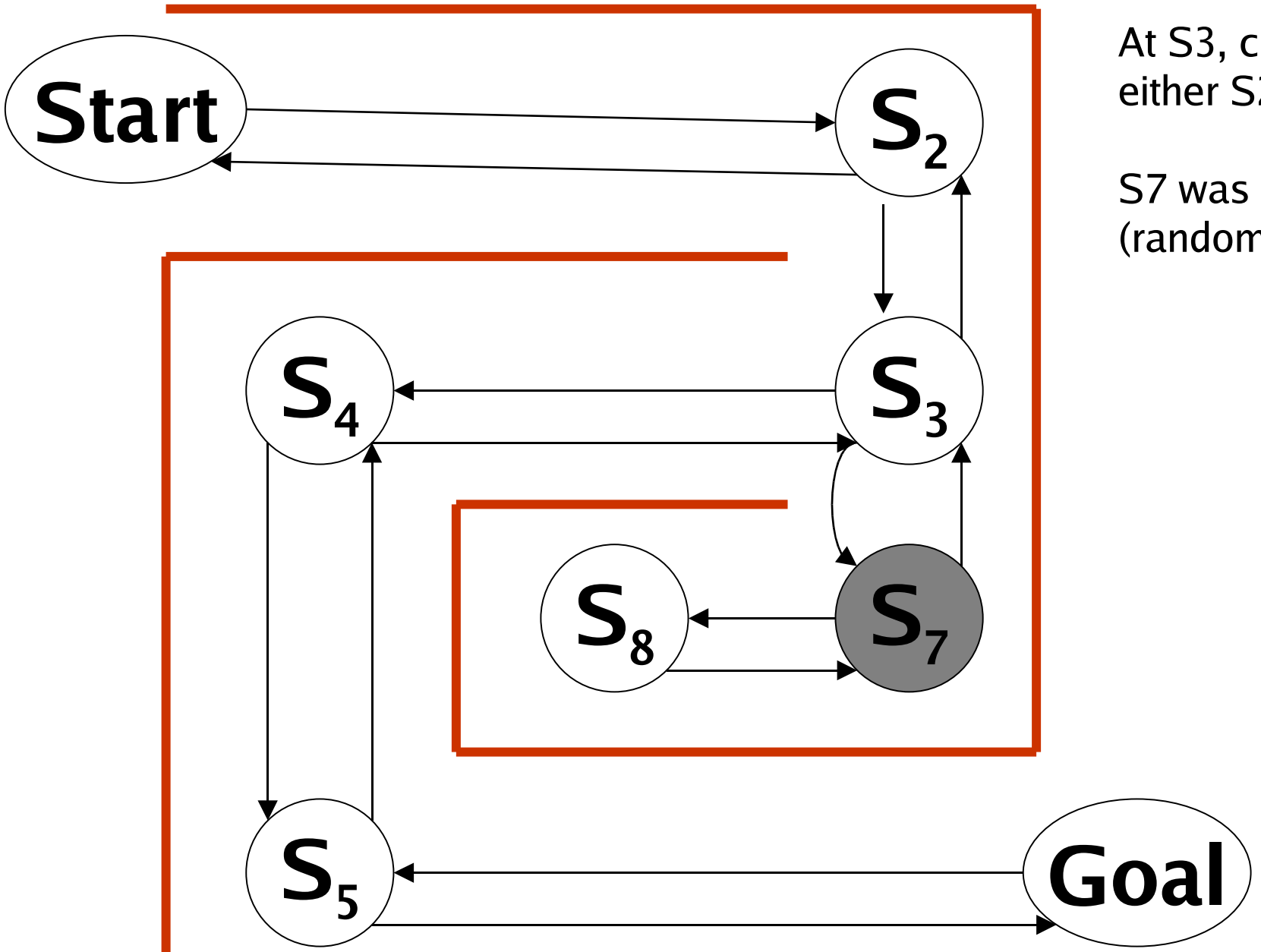
The first response leads to S2 …

The next state is chosen by randomly sampling from the possible next states *weighted* by their associative strength.

Associative strength = line width

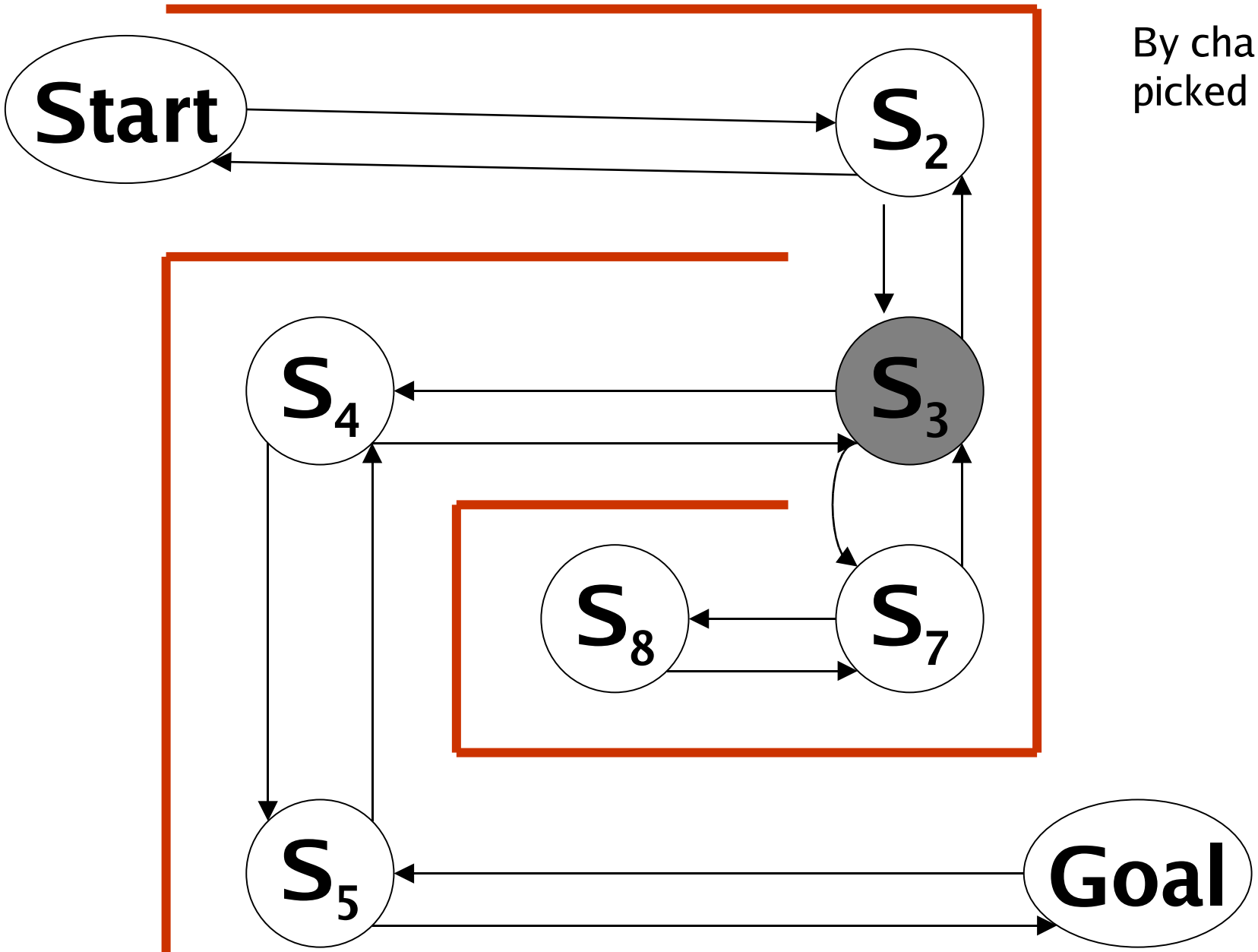


Suppose the randomly sampled response leads to S3 …

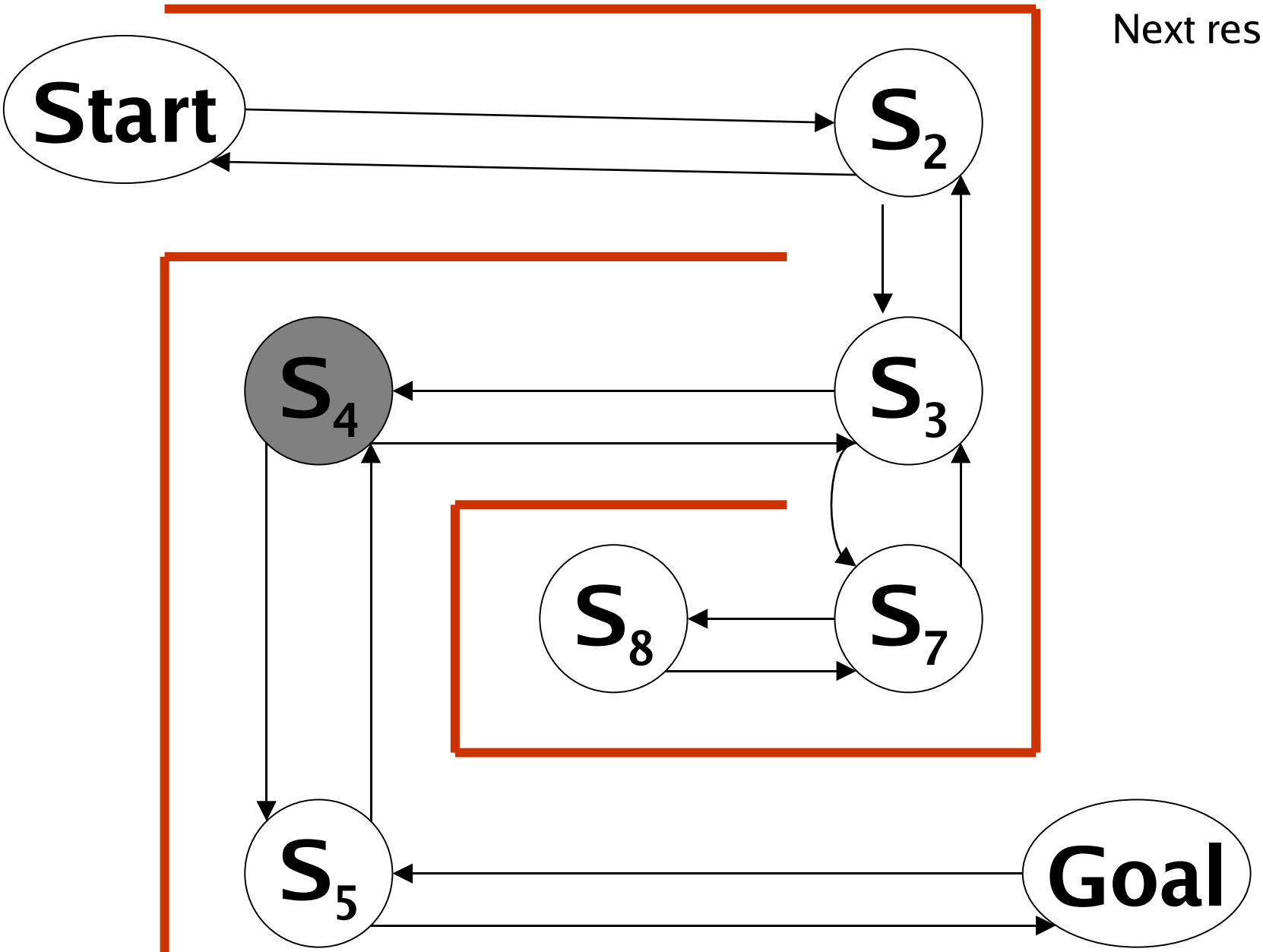


At S3, choices lead to either S2, S4, or S7.

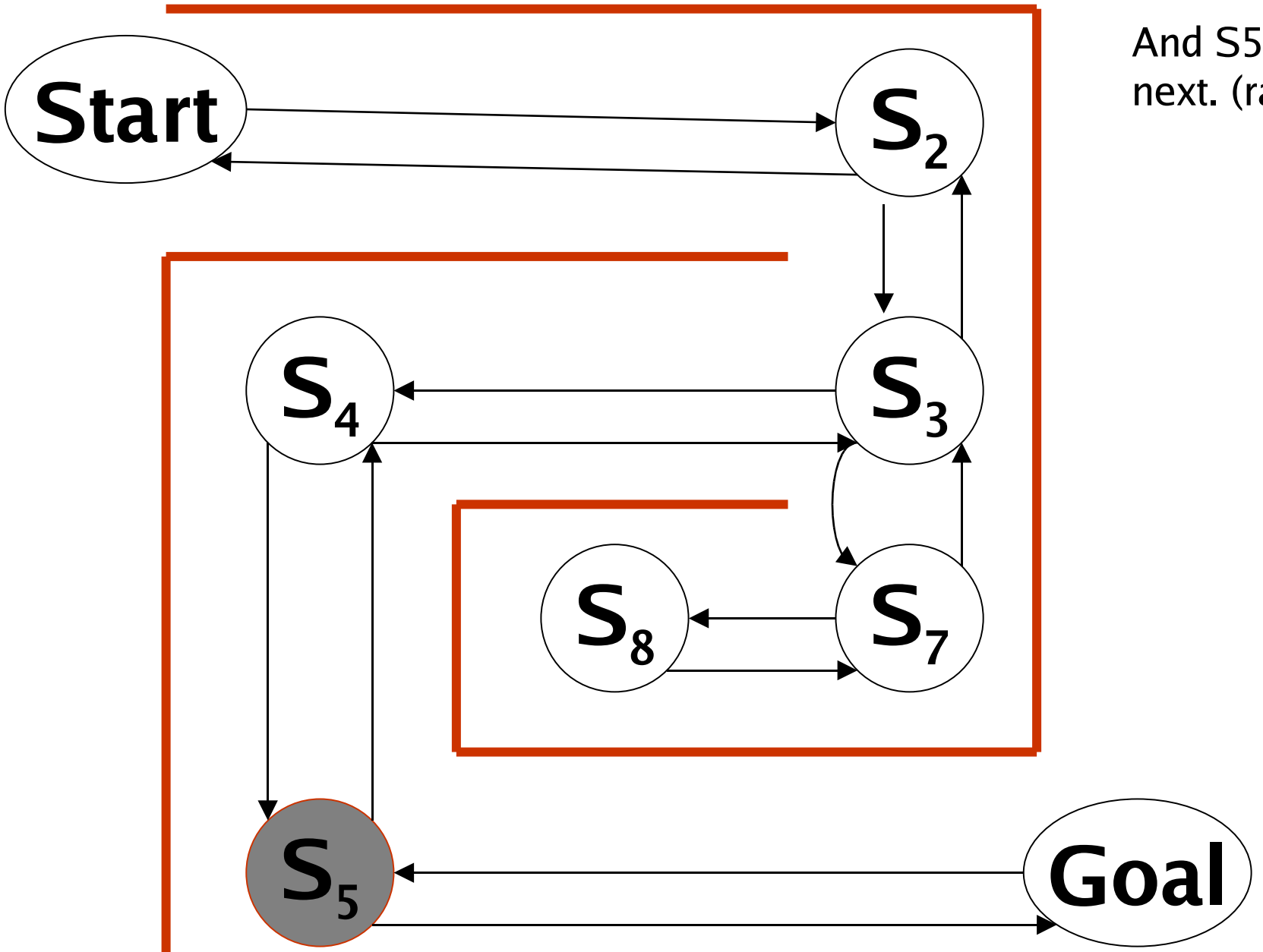
S7 was picked. (randomly)



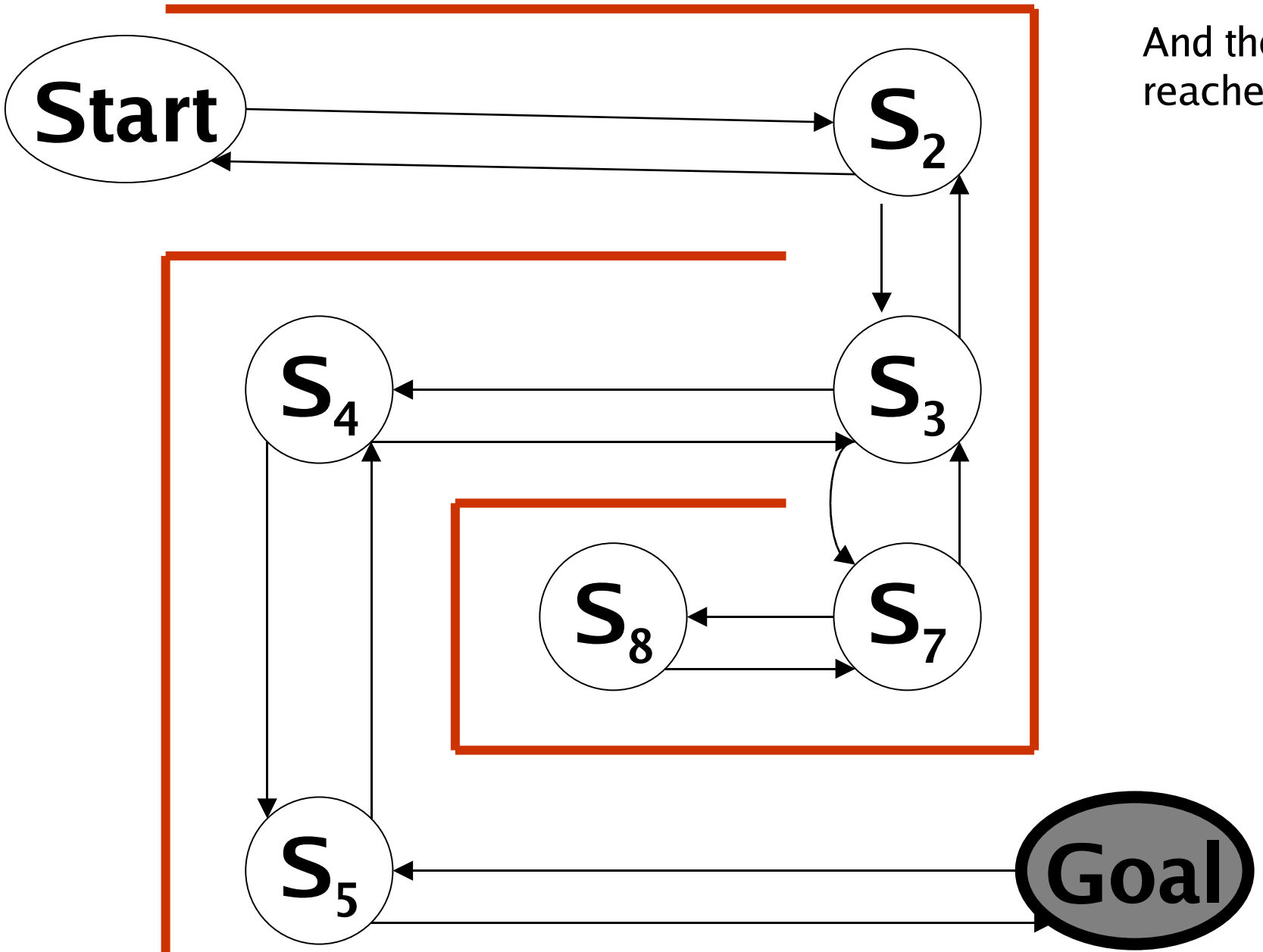
By chance, S3 was picked next…



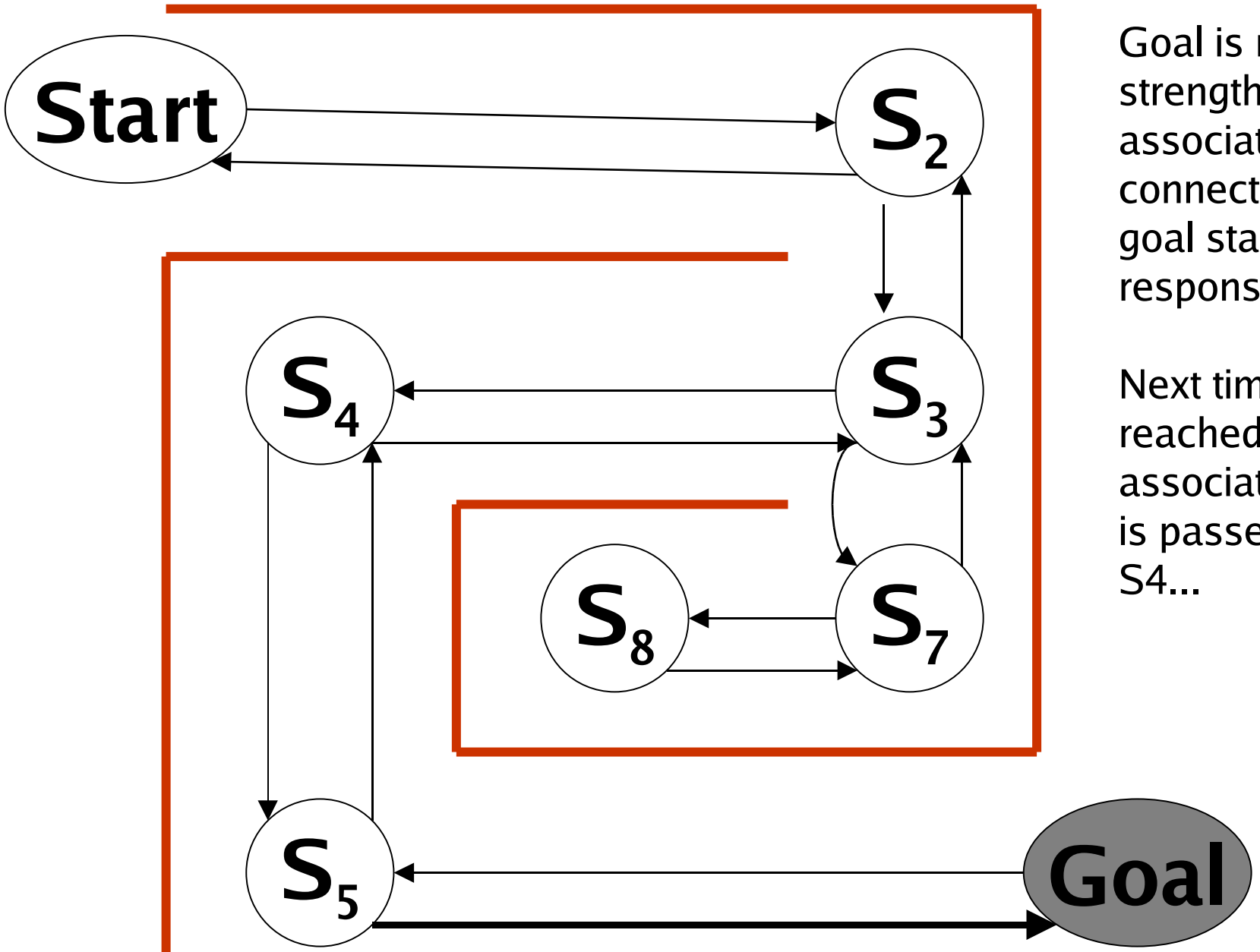
Next response is S4.



And S5 was chosen next. (randomly)



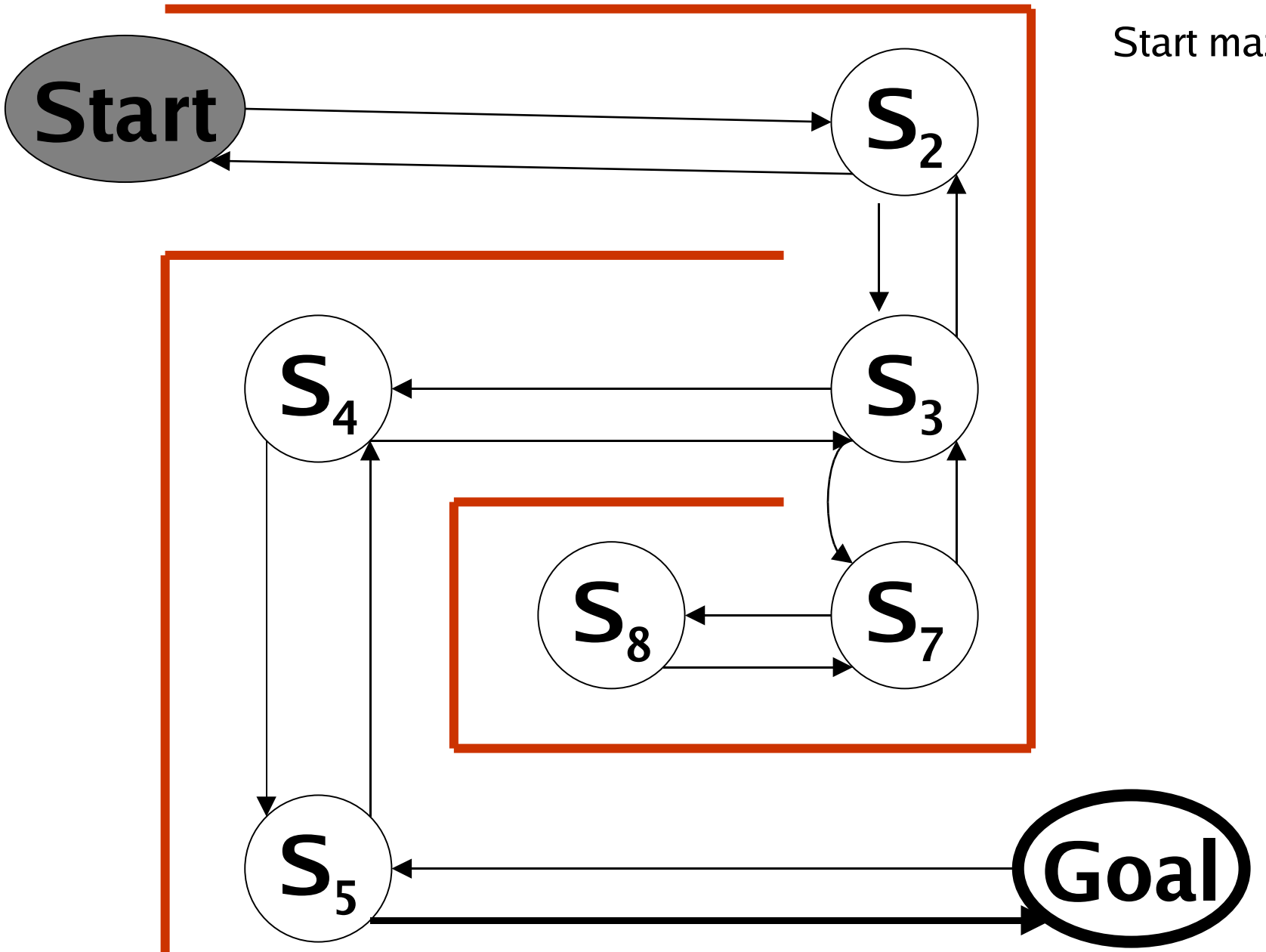
And the goal is reached …

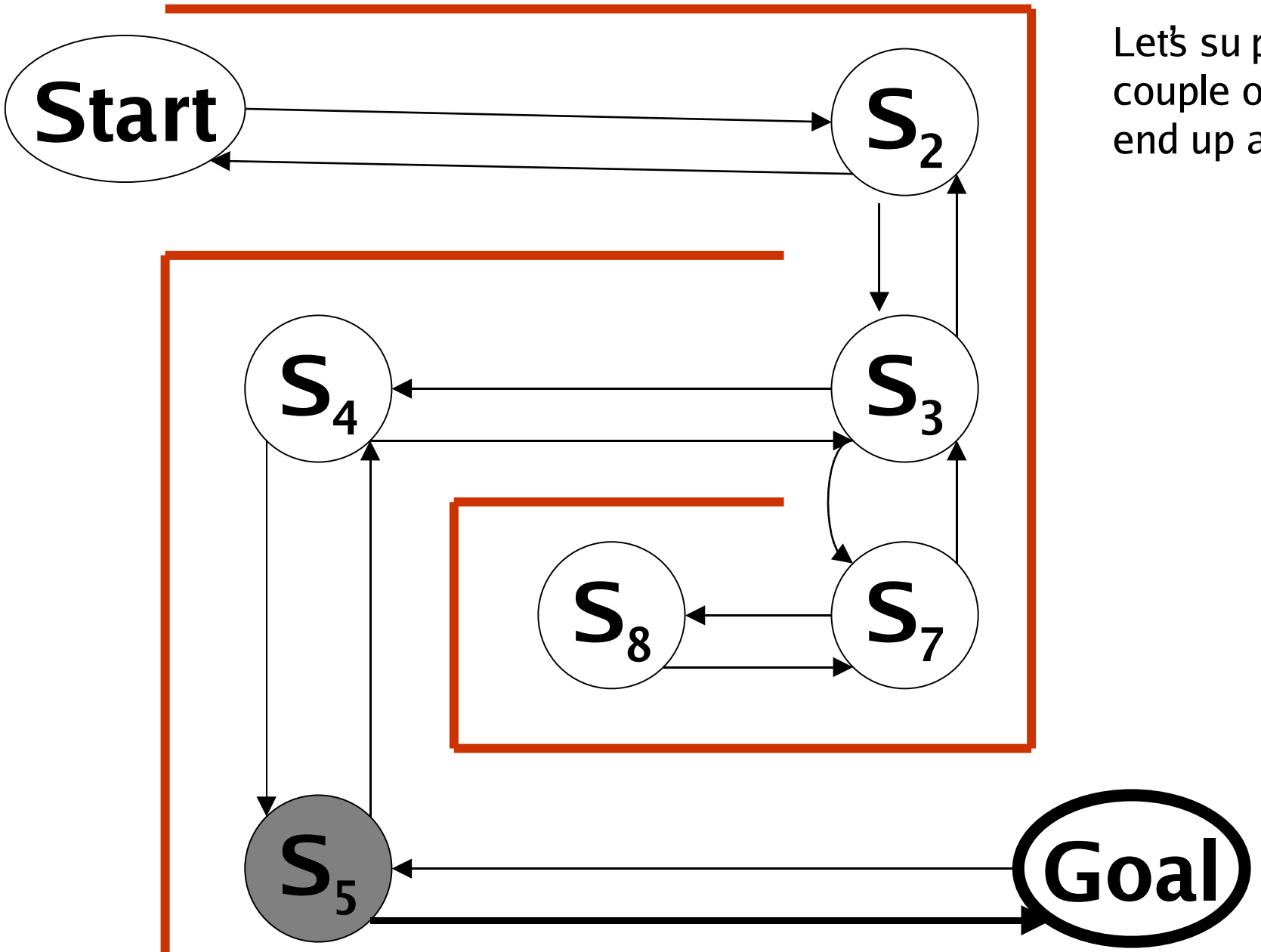


Goal is reached, strengthen the associative connection between goal state and last response.

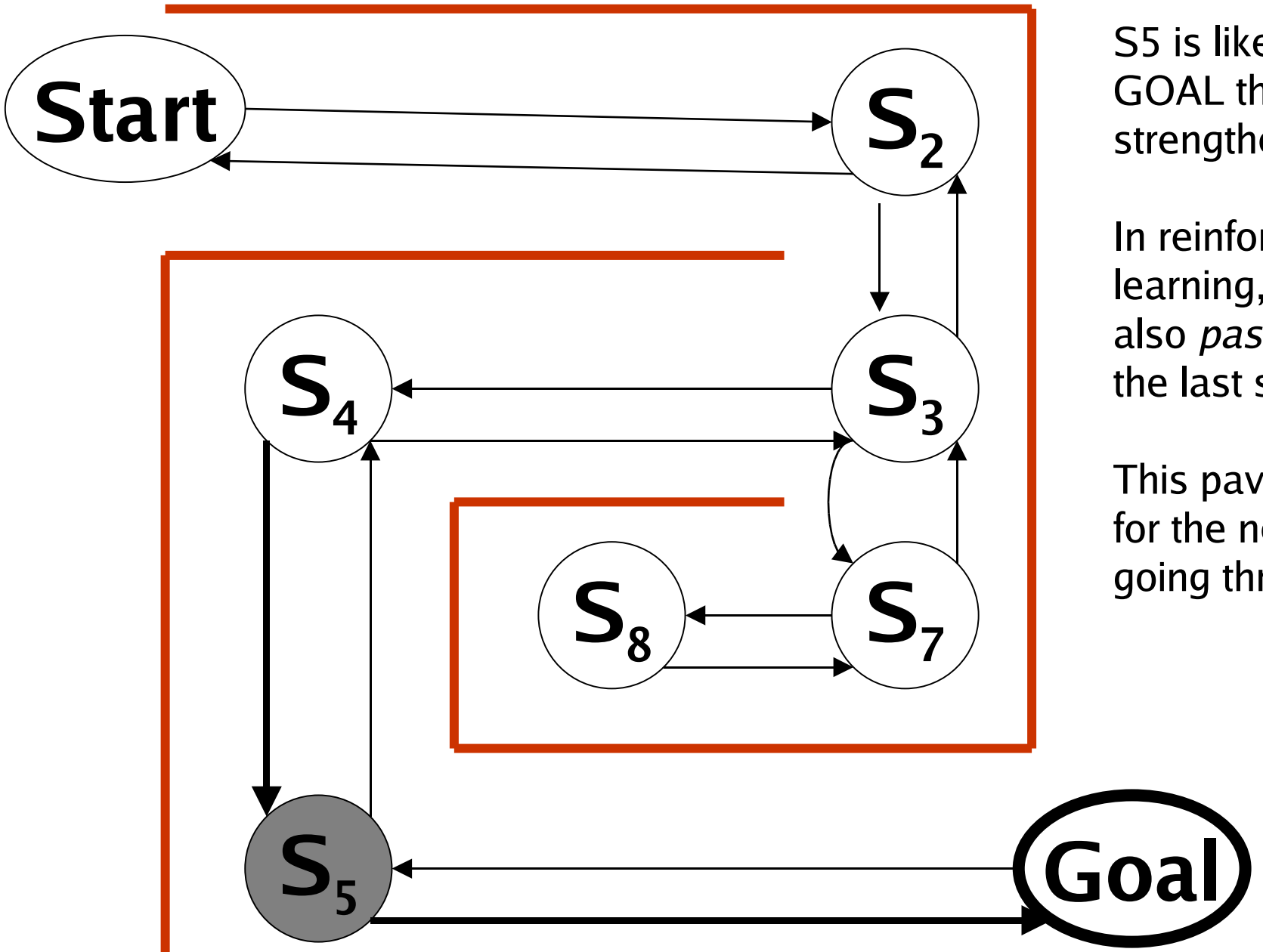
Next time S5 is reached, part of the associative strength is passed back to S4...

Start maze again…





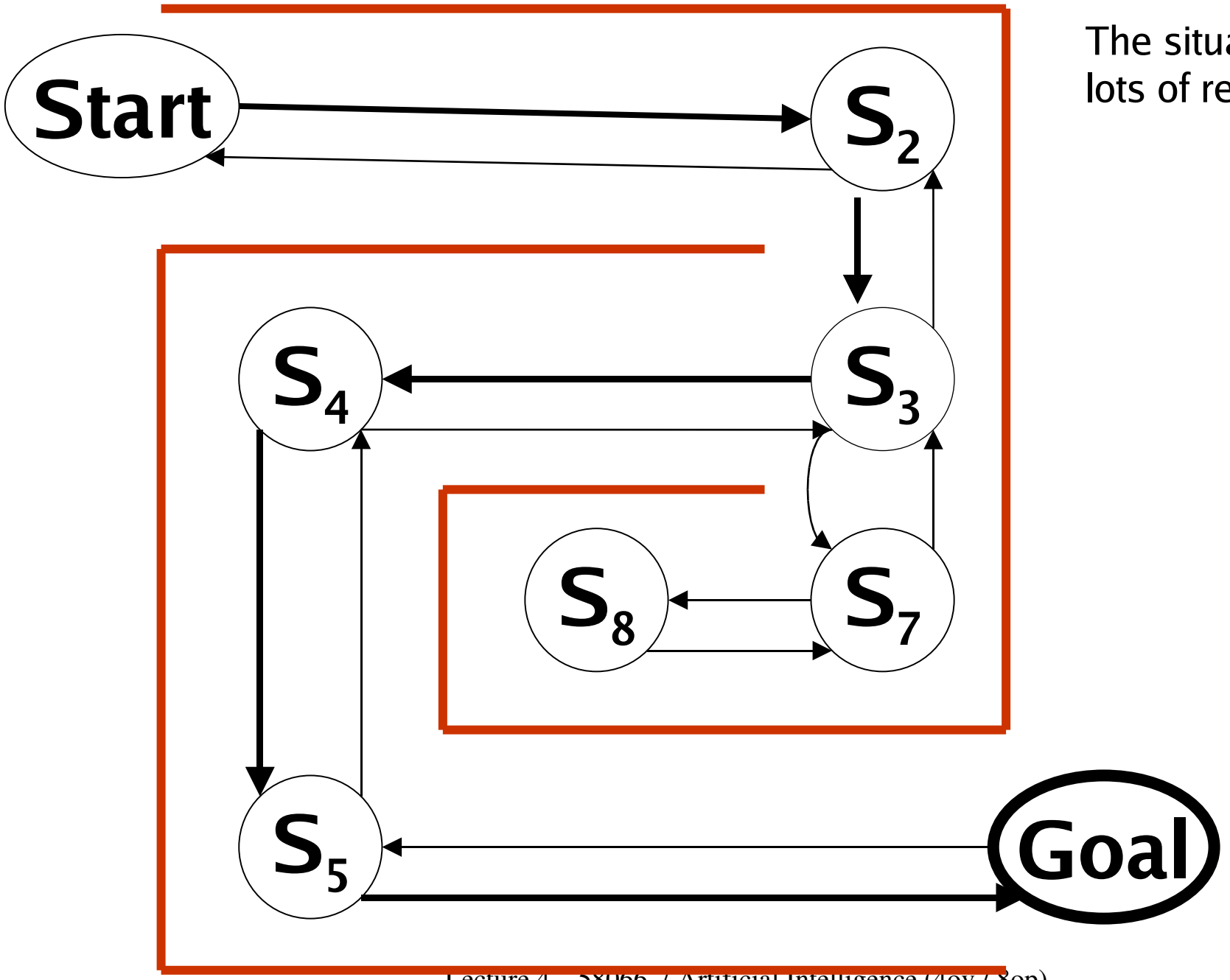
Let's suppose after a couple of moves, we end up at S5 again.



S5 is likely to lead to GOAL through strengthened route.

In reinforcement learning, strength is also *passed back* to the last state.

This paves the way for the next time going through maze.



The situation after lots of restarts …

5.24 Temporal credit assignment

- Delayed reinforcement learning paradigm
- Difficult problem
  - No teacher signal at each time step
  - Temporal delay of the reward or punishment signal $\rightarrow$ learner needs to assign a part of the blame or credit to each state and action that led to the final outcome.
- In general, the challenge of making a system learn only from the final outcome of its experience is appealing (Tesauro, 1992)

5.25 Temporal difference learning

- TD(λ) for adjusting weights in a connectionist network (Sutton, 1988):

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k,$$

where

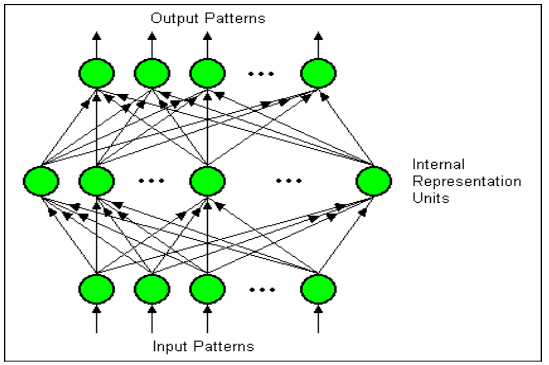


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

- α is the learning rate
- Y_t is the network's output pattern at time t
- w is the weight vector of the network
- $\nabla_w P_k$ is the gradient of the output with respect to weights
- λ heuristic parameter that controls the temporal credit assignment.

5.26 Backgammon

- Probabilistic game
  - Large number of states $\rightarrow$ table-lookup possible only in some simplified end-game situations.
  - High branching ratio $\rightarrow$ brute-force search not feasible.
- Neurogammon (Tesauro, 1989)
  - Was trained by backpropagation on data of recorded expert games (supervised learning!).
  - Strong intermediate level of play
- Heuristic evaluation functions from human experts + supervised learning has proved unsuccessful.

5.27 TD-gammon

- Gerald Tesauro's TD-Gammon (1995) learns by playing against itself.
- Originally designed to study
  - TD-learning in complex real-world domains.
  - capability of multi-layer network to learn complex non-linear functions.
- Some impressive games against top human players.
- Uses massive computing power for extensive off-line training of good evaluation function.
- Has given some insight to neural network learning.

5.28 TD-Gammon:Architecture

- Multi-layer feed-forward network
- Nodes output a weighted linear sum of inputs, followed by non-linear sigmoidal “squashing” function.
- Connections are parameterized with weights.
- Learning = adjust the weights so that the function approximated by the network more closely resembles the target.

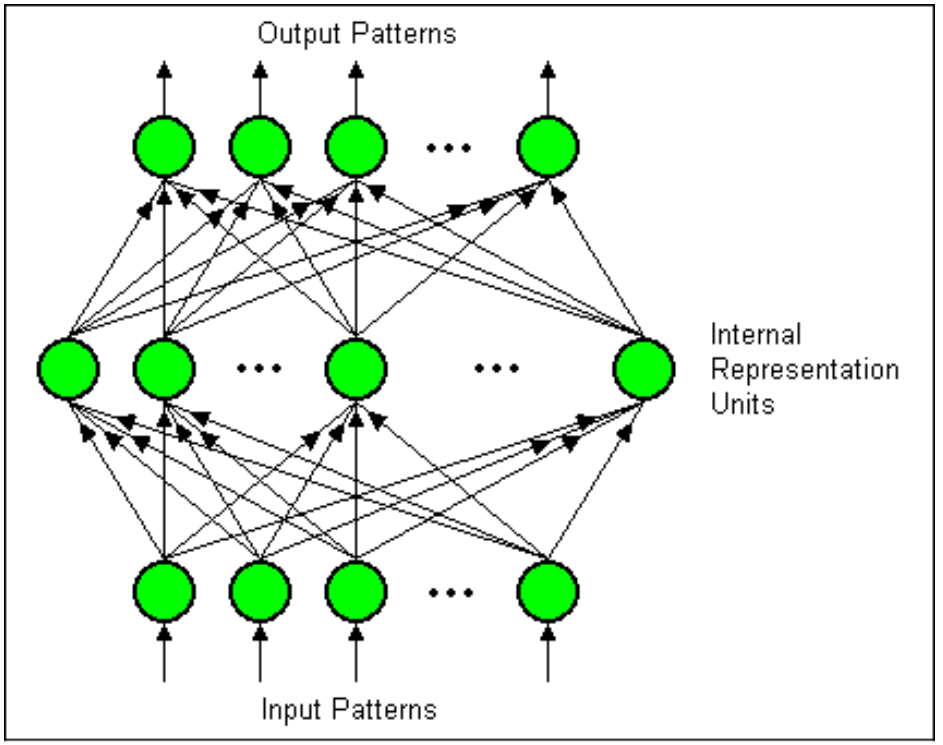


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

5.29 TD-gammon: Training

- Observe board positions from opening position to the terminal position (number of white and black checkers at each location)
- Positions are input vectors to the network: $x_1, x_2, \ldots, x_T$.
- Y_t is the network output pattern corresponding input vector x_t: four-component vector estimating expected outcome of the game: (black/white) x (win/gammon).
- Weights are adjusted using: $w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \Sigma_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$,
- At the end of the game the final reward signal r is used to assign the credit $(r - Y_T)$.
- The network selects the moves on both sides.

5.30 Results of learning

- After few thousand training games elementary tactics appear.
- After several tens of thousand of games more sophisticated concepts acquired.
- Promising finding: as the size of the network and training experience increases performance improves.
- Strong intermediate level of play after training with 200,000 games (40-hidden unit network).
- Interesting spatially organized patterns of weights.