01 Modelling: Markov Processes versus Classical Planning, STRIPS & PDDL

Table of contents

- Markov Processes
- Markov Reward Processes
- Bellman Equation
- Modelling Languages: STRIPS & Planning Domain Definition Language (PDDL)

Markov Processes

Introduction to Markov Processes

A Markov process formally describes an environment for planning and learning

- Where the environment is *fully observable*
- I.e. The current state completely characterises the process (the Markov condition)

Markov Property (Revisited)

"The future is independent of the past given the present"

Definition: Markov Property

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1} \mid S_0, \ldots, S_t]$$

The state captures all relevant information from the history

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

A state transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{matrix} \\ from \end{matrix} \begin{matrix} \overset{to}{} \\ \begin{bmatrix} \mathcal{P}_{00} & \cdots & \mathcal{P}_{0n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n0} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \end{matrix}$$

where each row of the matrix sums to 1.

Markov Processes

A Markov process is a *memoryless* random process, i.e. a sequence of random states $S_0, S_1, \ldots$ with the Markov property.

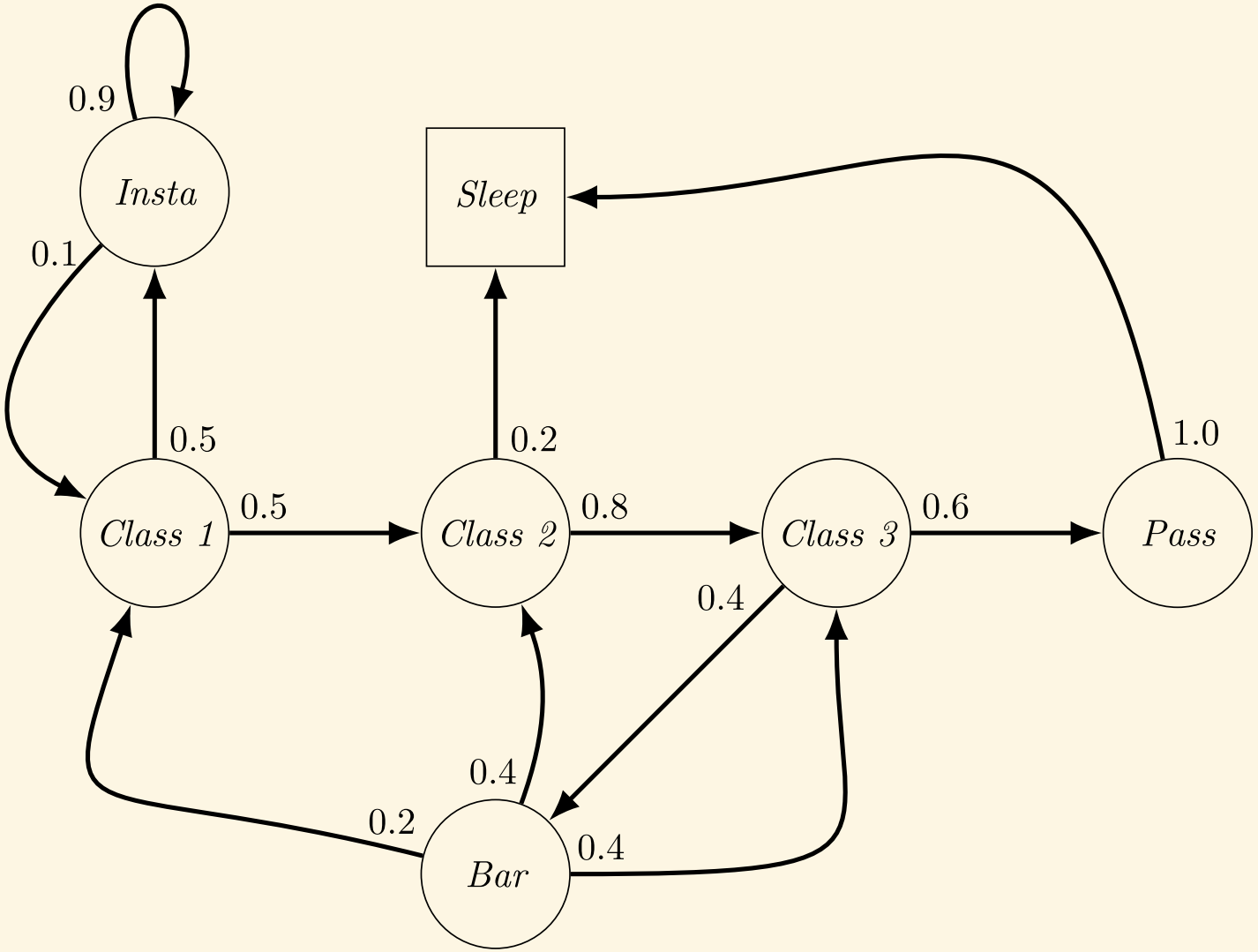
Definition: Markov Process (or Markov Chain)

A *Markov Process* (or *Markov Chain*) is a tuple $< \mathcal{S}, \mathcal{P} >$

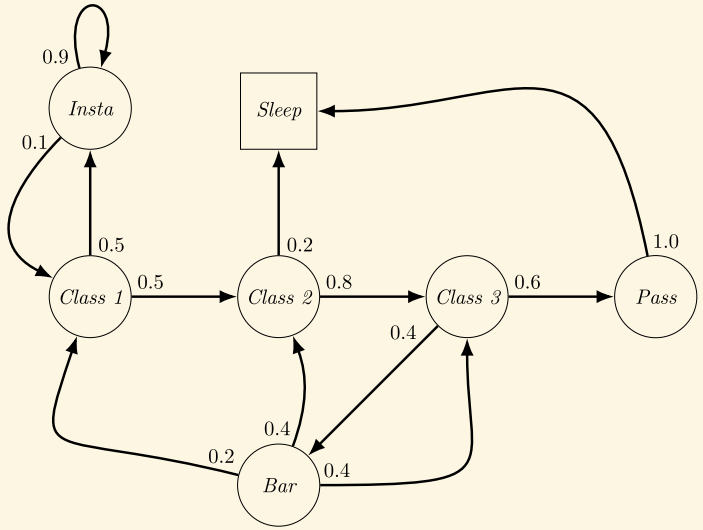
- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

Example: Student Markov Chain



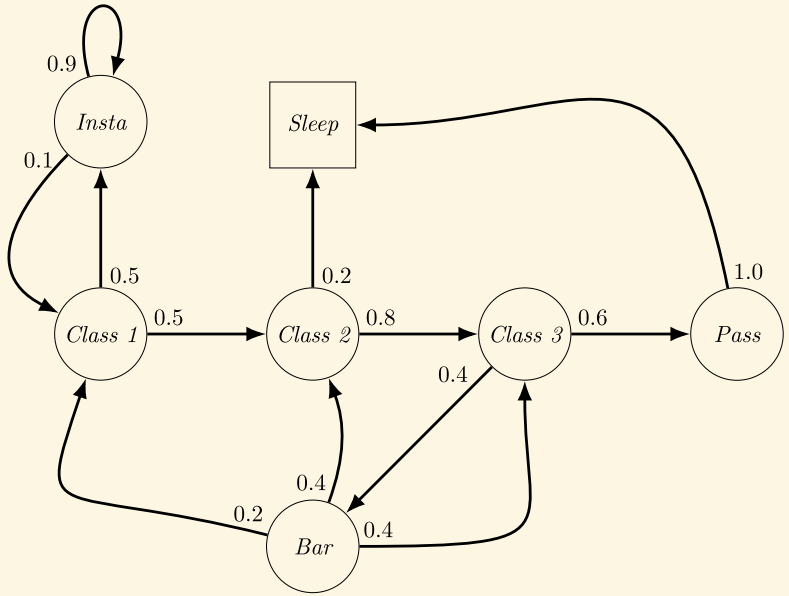
Example: Student Markov Chain Episodes



Sample episodes for Student Markov Chain (starting from $S_0 = C1$): $S_0; S_1; \ldots, S_T$

- C1 C2 C3 Pass Sleep
- C1 In In C1 C2 C3 Pass Sleep
- C1 C2 C3 Bar C2 C3 Pass Sleep
- C1 In In C1 C2 C3 Bar C1 In In In C1 C2 C3 Bar C2 C3 Pass Sleep

Example: Student Markov Chain Transition Matrix



$$\mathcal{P} = \begin{bmatrix} & C1 & C2 & C3 & Pass & Bar & In & Sleep \\ C1 & & 0.5 & & & & 0.5 & \\ C2 & & & 0.8 & & & & 0.2 \\ C3 & & & & 0.6 & 0.4 & & \\ Pass & & & & & & & 1.0 \\ Bar & 0.2 & 0.4 & 0.4 & & & & \\ In & 0.1 & & & & & 0.9 & \\ Sleep & & & & & & & 1 \end{bmatrix}$$

Notation for Probability

You will see some subtle differences in how classical planning and reinforcement learning handles the notation for probability.

The following link in the subject's Ed Discussion forum clarifies the notation used in this subject:

https://edstem.org/au/courses/32360/lessons/102661/slides/705762
(follow the link from Canvas LMS for permission to view lesson).

Markov Processes?

What problems (how many) can be formalised as Markov processes?

Almost all problems can be formalised as Markov processes

- Any non-Markov process can be turned into a Markov process if you define the *state* richly enough—at the extreme the 'state' can be the entire history up to now
- This is not practical in all scenarios but surprisingly useful when appropriate state representation is *learned* (i.e. the language of states)
- MRPs can handle *infinite* states (using *function approximation* which we will learn about in a later modules)
- MRPs can handle *memory*, by encoding memory in Markov states

Markov Reward Processes

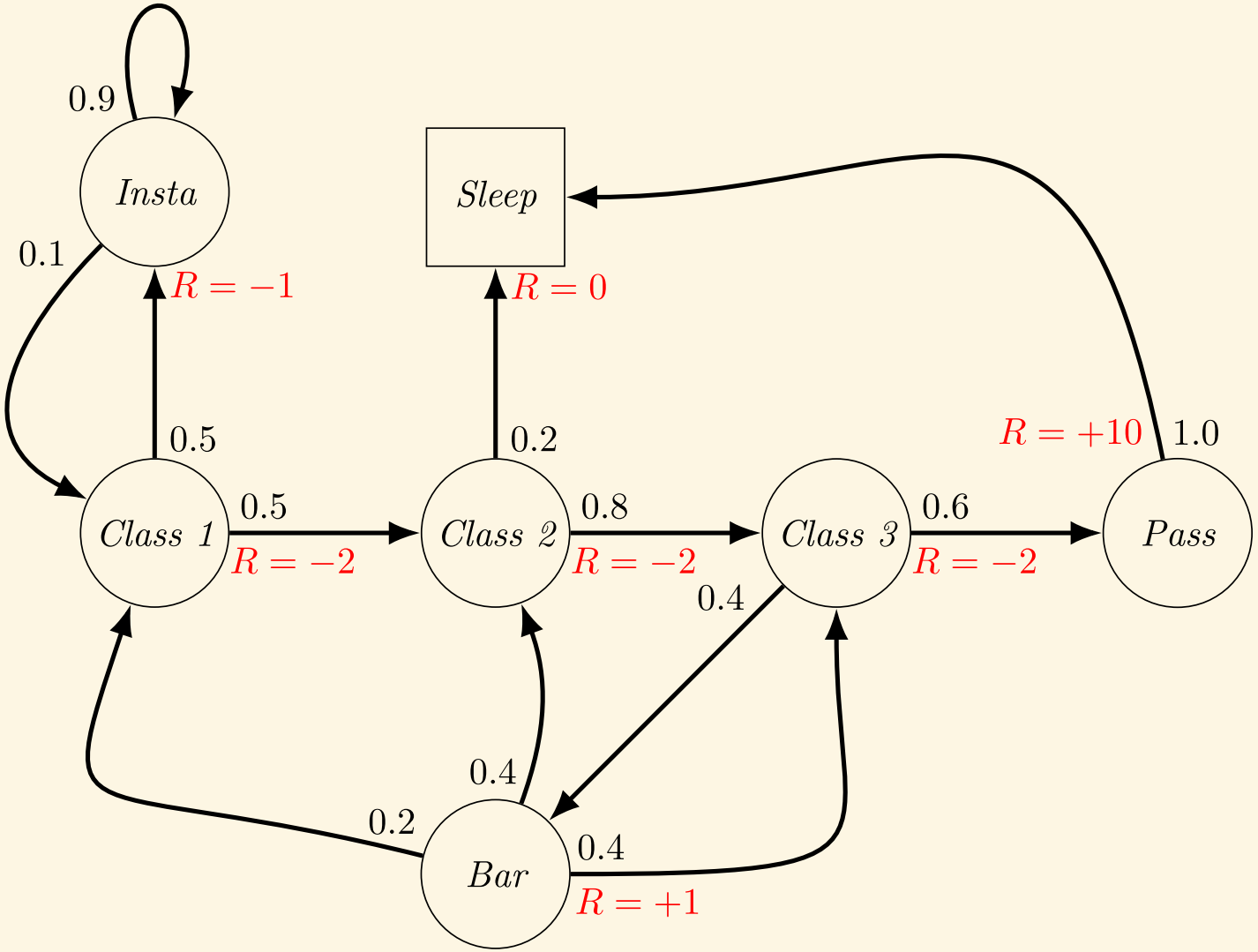
A Markov reward process is a Markov chain with *values*.

Definition: Markov Reward Process

A Markov Reward Process is a tuple $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a finite set of *states*
- $\mathcal{P}$ is a state transition *probability* matrix, $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}(s) = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a *discount* factor, $\gamma \in [0, 1]$

Example: Student Markov Reward Process (MRP)



Return

Definition: Return

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The *value* of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Why discount?

Why are most Markov reward/decision processes discounted?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward

Why discount (continued)?

- It *is* sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$),
  - e.g. especially if we can guarantee that all sequences *terminate*.

Value Function

The value function $v(s)$ gives the long-term value of state s

Definition: State Value Function

The state value function $v(s)$ of an MRP is the *expected return* starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Over how many episodes is expected return computed?

Expected return is over all possible episodes

Example: Student MRP Returns

Sample returns for Student MRP episodes:

Starting from $S_0 = C1$ with $\gamma = \frac{1}{2}$

$$G_0 = R_1 + \gamma R_2 + \ldots + \gamma^{T-1} R_T$$

C1 C2 C3 Pass Sleep

C1 In In C1 C2 Pass Sleep

C1 C2 C3 Bar C2 C3 Pass Sleep

C1 In In C1 C2 C3 Bar C1 . . .

In In In C1 C2 C3 Bar C2 C3 Pass Sleep

Showing just first four samples...

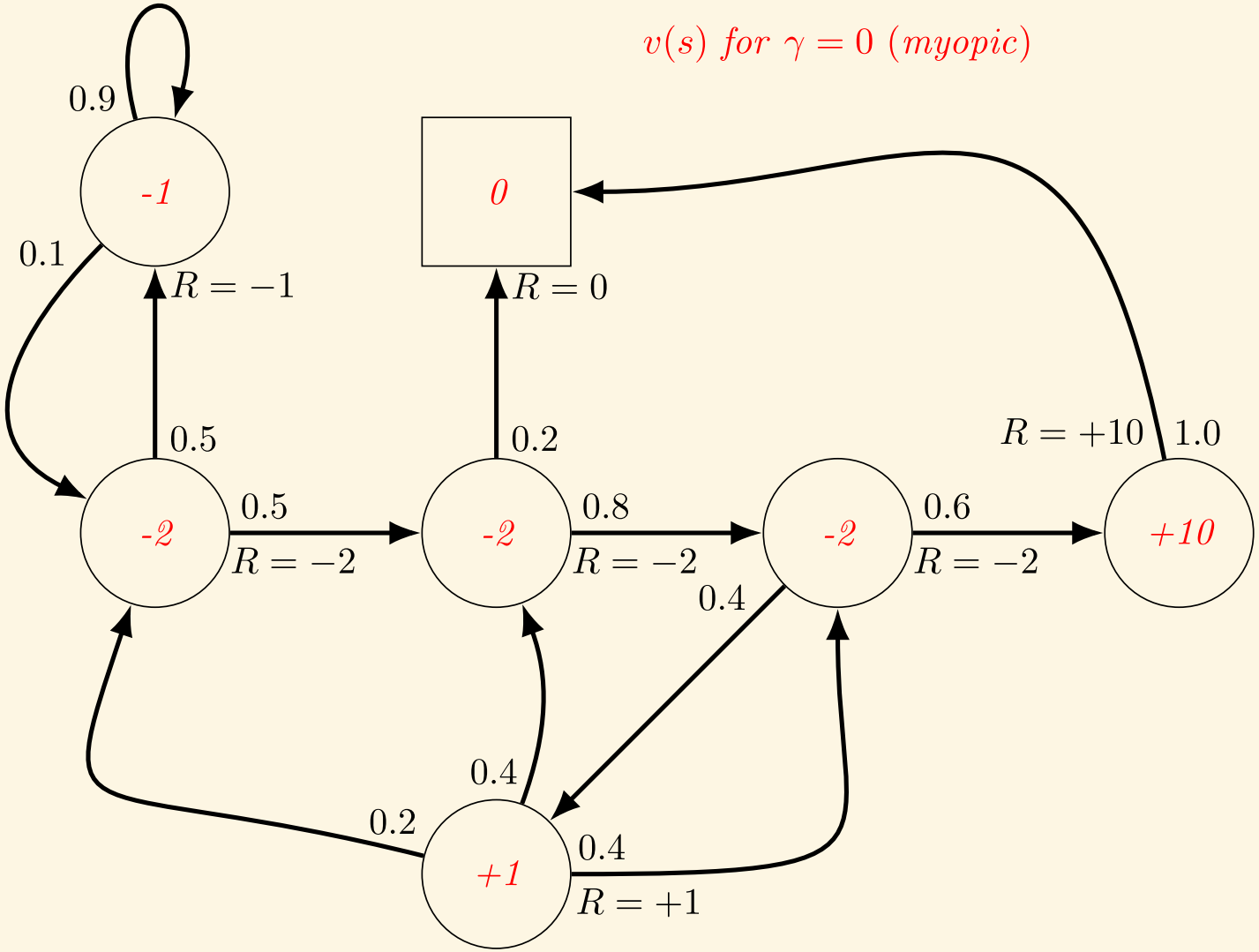
$$v_0 = -2 - 2 \cdot \tfrac{1}{2} - 2 \cdot \tfrac{1}{4} + 10 \cdot \tfrac{1}{8} = -2.25$$

$$v_0 = -2 - 1 \cdot \tfrac{1}{2} - 1 \cdot \tfrac{1}{4} - 2 \cdot \tfrac{1}{8} - 2 \cdot \tfrac{1}{16} = -3.125$$

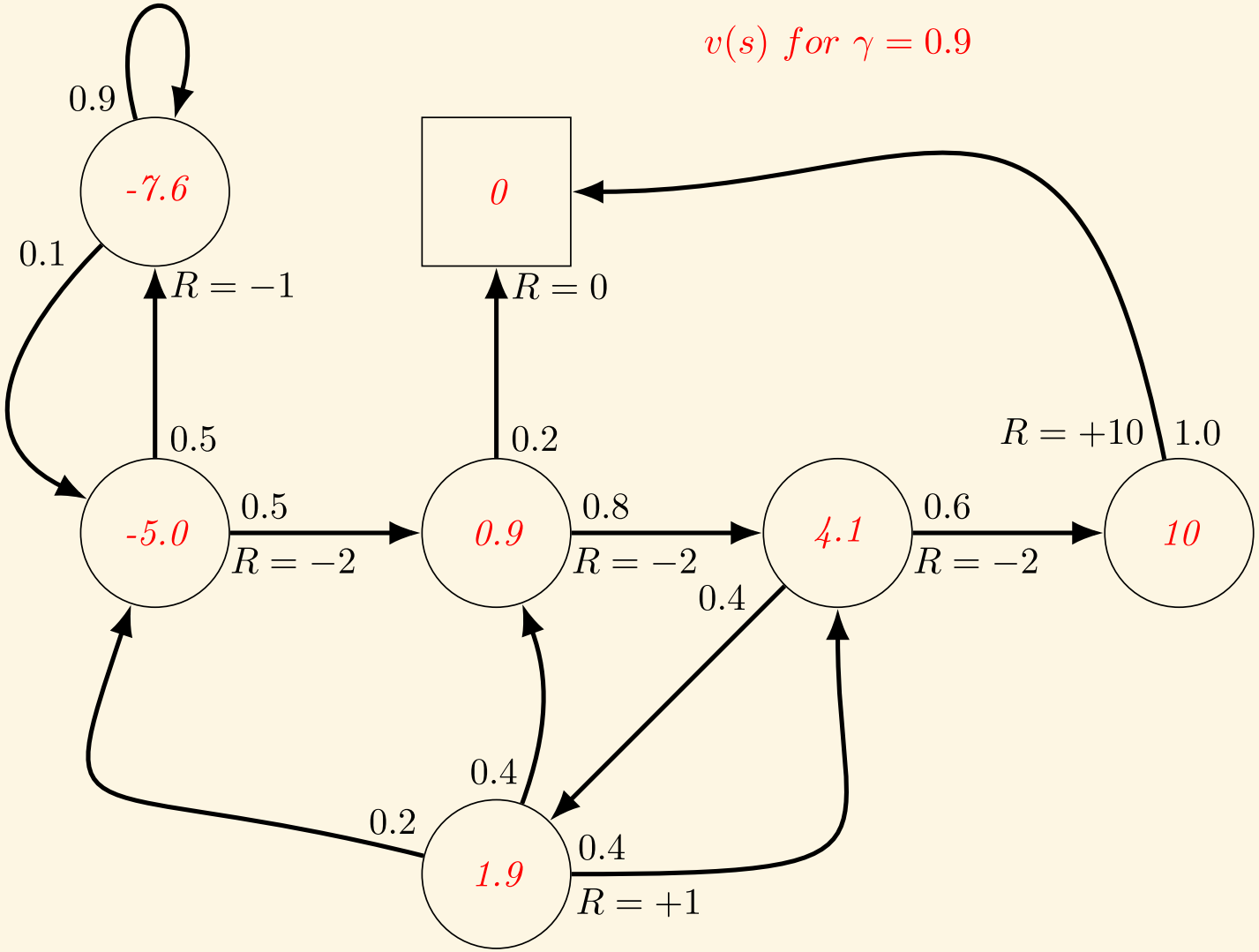
$$v_0 = -2 - 2 \cdot \tfrac{1}{2} - 2 \cdot \tfrac{1}{4} + 1 \cdot \tfrac{1}{8} - 2 \cdot \tfrac{1}{16} \ldots = -3.41$$

$$v_0 = -2 - 1 \cdot \tfrac{1}{2} - 1 \cdot \tfrac{1}{4} - 2 \cdot \tfrac{1}{8} - 2 \cdot \tfrac{1}{16} \ldots = -3.20$$

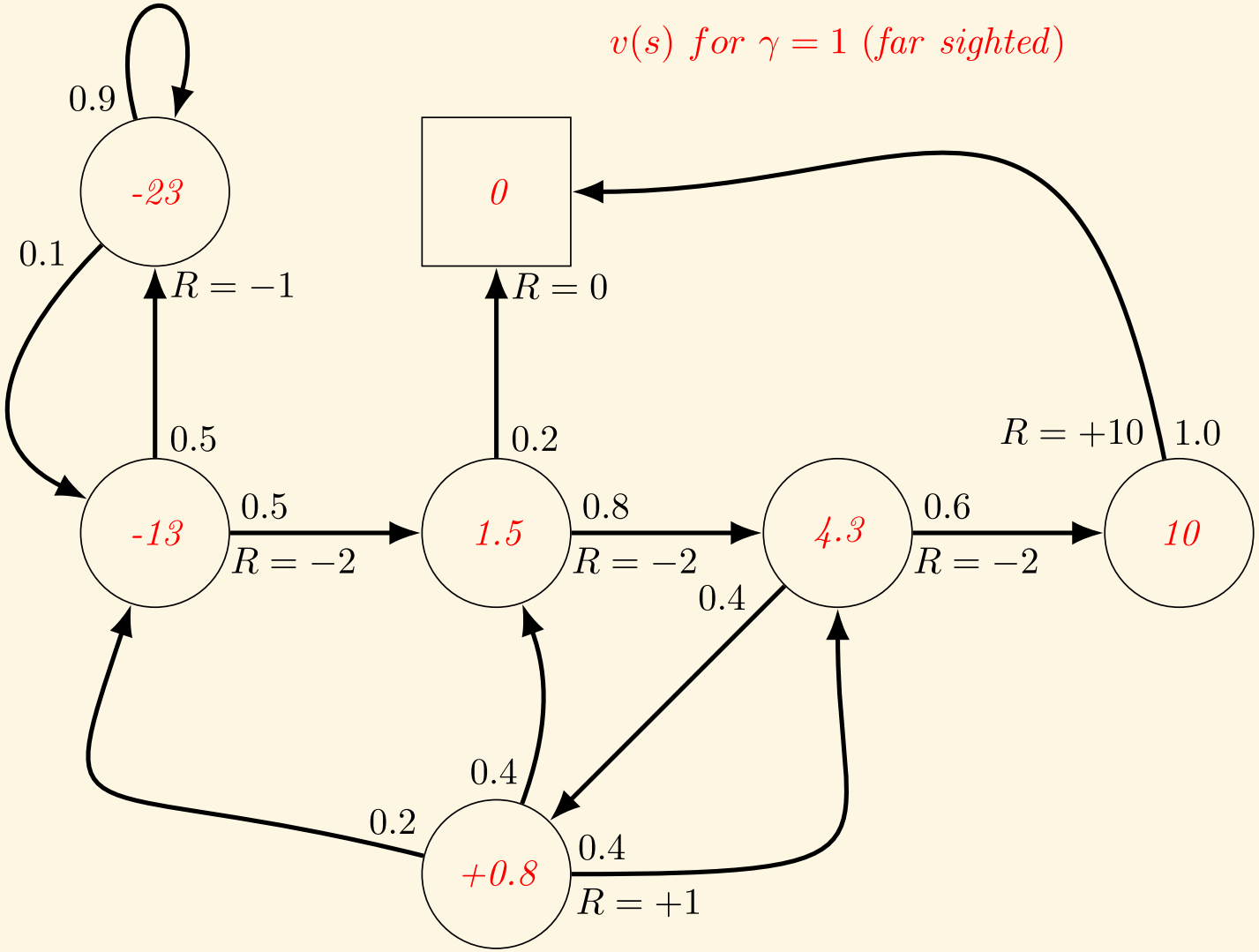
Example: State-Value Function for Student MRP (1)



Example: State-Value Function for Student MRP (2)



Example: State-Value Function for Student MRP (3)



Bellman Equation

Bellman Equation for MRPs

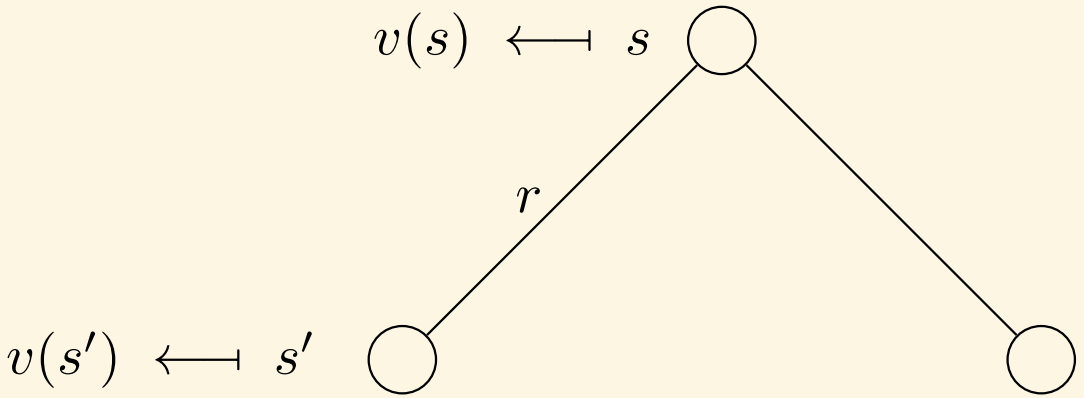
The value function can be decomposed into two parts:

- the immediate reward, R_{t+1}, and
- the discounted value of the successor state, $\gamma v(S_{t+1})$, by the *law of iterated expectations*

$$
\begin{aligned}
v(s) &= \mathbb{E}[\, G_t \mid S_t = s \,] \\
&= \mathbb{E}\big[\, R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \,\big|\, S_t = s \,\big] \\
&= \mathbb{E}[\, R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \dots\right) \mid S_t = s \,] \\
&= \mathbb{E}[\, R_{t+1} + \gamma G_{t+1} \mid S_t = s \,] \\
&= \mathbb{E}[\, R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s \,]
\end{aligned}
$$

Bellman Equation for MRPs (look ahead)

$$v(s) = \mathbb{E}[R_{t+1} + \gamma\, v(S_{t+1}) \mid S_t = s]$$



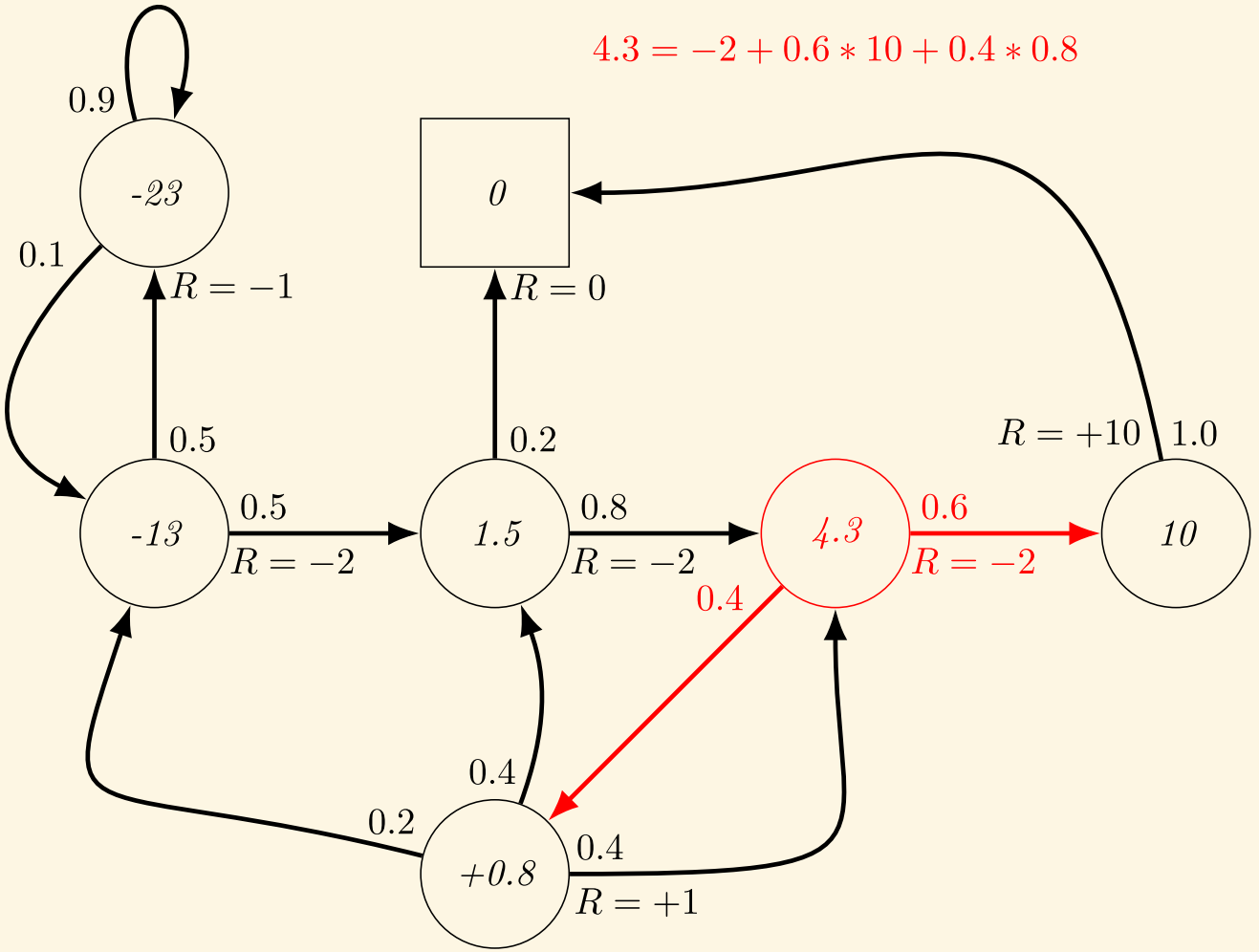
$$v(s) \;=\; \mathcal{R}_s \;+\; \gamma \sum_{s' \in \mathcal{S}} P_{ss'} v(s')$$

We integrate over probability by computing each $v(s')$ in backup diagrams - *add up reward for each branch and backup to root*

Example: Bellman Equation for Student MRP

What does the Bellman Equation look like for the Student MRP?

Example: Computing Bellman Equation via Backup Diagrams



(with no discounting)

Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using vectors and matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(0) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_0 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{00} & \cdots & \mathcal{P}_{0n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n0} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(0) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

Since the Bellman equation is a *linear* equation of matrices and vectors

- It can be solved directly for *evaluating* rewards, assuming matrix is small enough to invert
- Note that this won't be true when we move to Markov decision processes where we wish to *optimise* rewards

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$(I - \gamma \mathcal{P}) v = \mathcal{R}$$

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

Note that the computational complexity of inverting this matrix is $O(n^3)$ for n states, hence:

- Direct solution only possible for *small* MRPs

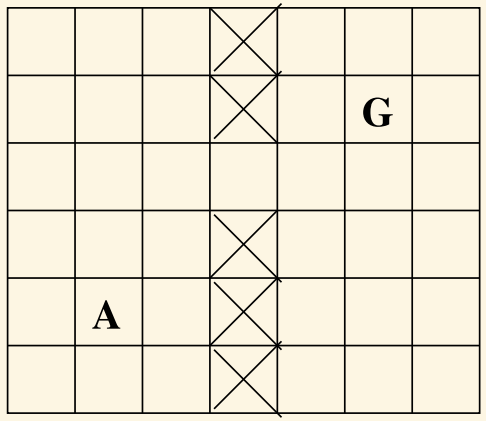
We will explore iterative solution methods for doing this more efficiently for large MRPs in future modules, including

- Dynamic programming
- Monte-Carlo evaluation
- Temporal-Difference learning

Modelling Languages: STRIPS & Planning Domain Definition Language (PDDL)

Consider Example Problem:

Imagine that agent *A* must reach *G*, moving one cell at a time in *known* map



- If the actions are stochastic and the locations (states) are observable, the problem is a *Markov Decision Process (MDP)*
- If the actions are deterministic and only the initial location is known, the problem is a *Classical Planning Problem*

Different combinations of uncertainty and feedback lead to two problems: two models

STRIPS & Planning Domain Definition Language (PDDL)

Classical planning is typically specified *symbolically* via

- a state-transition model (states, actions, initial state, goal),
- STRIPS-style action semantics (how action descriptions change states via preconditions and effects), and
- PDDL as the standard syntax for writing the domain + problem that, under the chosen semantics, induces the planning model.

State-Transition Model

Definition: State-Transition Model

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function $s' = f(a, s)$ for $a \in A(s)$
- positive action costs $c(a, s)$

- A solution is a sequence of applicable actions that maps s_0 into S_G, and it is *optimal* if it minimizes *sum of action costs* (e.g., # of steps)
- Different *models* and *controllers* obtained by relaxing assumptions in *blue* ...

A Language for Classical Planning: STRIPS

Definition: STRIPS Problem

A **problem** in STRIPS is a tuple $P = \langle F, O, I, G \rangle$:

- F stands for set of all facts (atoms) (boolean variables)
- O stands for set of all operators (actions)
- $I \subseteq F$ stands for initial situation
- $G \subseteq F$ stands for goal situation

Operators $o \in O$ represented by:

- the Add list $Add(o) \subseteq F$
- the Delete list $Del(o) \subseteq F$
- the Precondition list $Pre(o) \subseteq F$

From Language to Models (STRIPS Semantics)

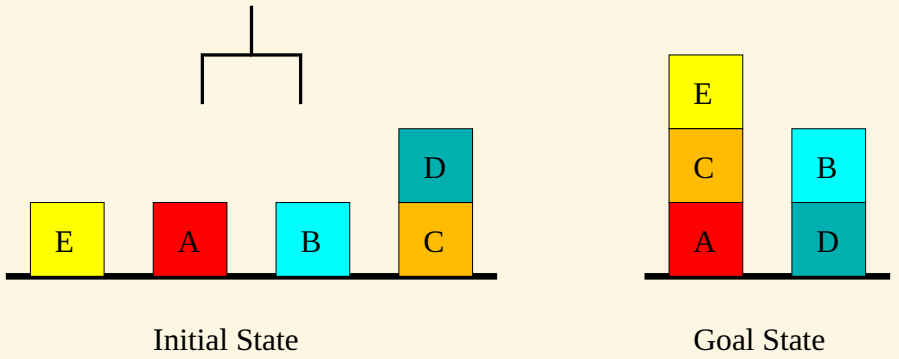
Definition: STRIPS Semantics

A STRIPS problem $P = \langle F, O, I, G \rangle$ determines a state model $\mathcal{S}(P)$ where

- the states $s \in S$ are collections of facts (atoms) from F, where $S = 2^F$
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are operators in O such that $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

- An optimal solution of P is an optimal solution of the state model $\mathcal{S}(P)$
- Language extensions often convenient: negation, conditional effects, non-boolean variables, etc.

Example: The Blocksworld



- Propositions: $on(x, y), onTable(x), clear(x), holding(x), armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \ldots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- Goal: $\{on(E, C), on(C, A), on(B, D)\}$.
- Actions: $stack(x, y), unstack(x, y), putdown(x), pickup(x)$.
- $stack(x, y)$?

$pre : \{holding(x), clear(y)\}$
$add : \{on(x, y), armEmpty(), clear(x)\} \quad del : \{holding(x), clear(y)\}$.

PDDL Quick Facts

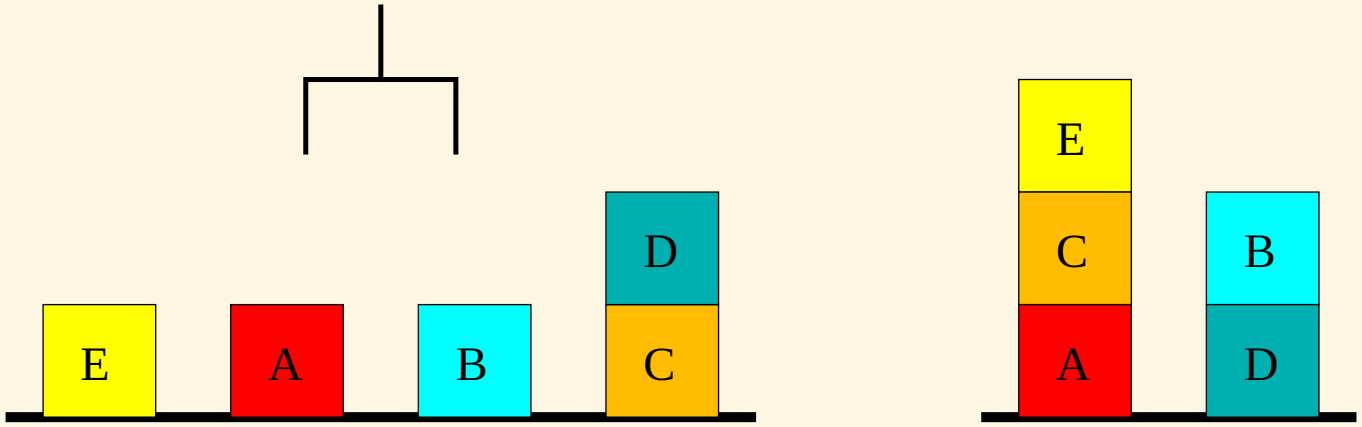
PDDL is not a propositional language:

- Representation is lifted, using object variables which are instantiated from a finite set of objects (similar to predicate logic which quantifies over *variables*).
- Action schemas are parameterized by objects.
- Predicates are to be instantiated with objects.

A PDDL planning task comes in two pieces:

- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each domain typically has only *one* file.

The Blocksworld in PDDL: Domain File

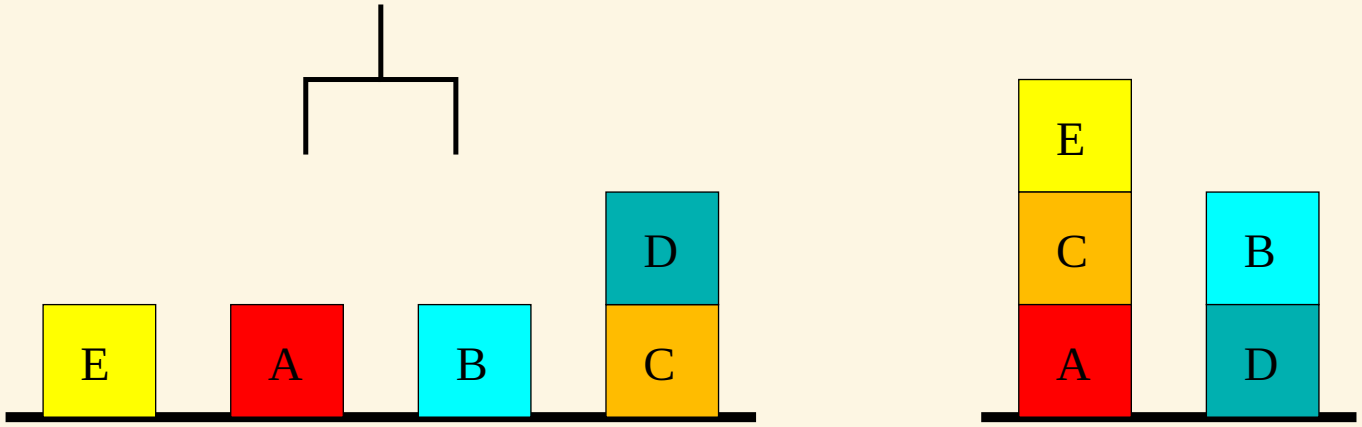


Initial State

Goal State

```
(define (domain blocksworld)
(:predicates  (clear ?x) (holding ?x) (on ?x ?y)
              (on-table ?x) (arm-empty))
(:action stack
  :parameters  (?x ?y)
  :precondition (and (clear ?y) (holding ?x))
  :effect (and (arm-empty) (on ?x ?y)
               (not (clear ?y)) (not (holding ?x)))
)
```

The Blocksworld in PDDL: Problem File



Initial State

Goal State

```
1 (define (problem bw-abcde)
2   (:domain blocksworld)
3   (:objects a b c d e)
4   (:init (on-table a) (clear a)
5          (on-table b) (clear b)
6          (on-table e) (clear e)
7          (on-table c) (on d c) (clear d)
8          (arm-empty))
9   (:goal (and (on e c) (on c a) (on b d))))
```

Example: Logistics in STRIPS PDDL

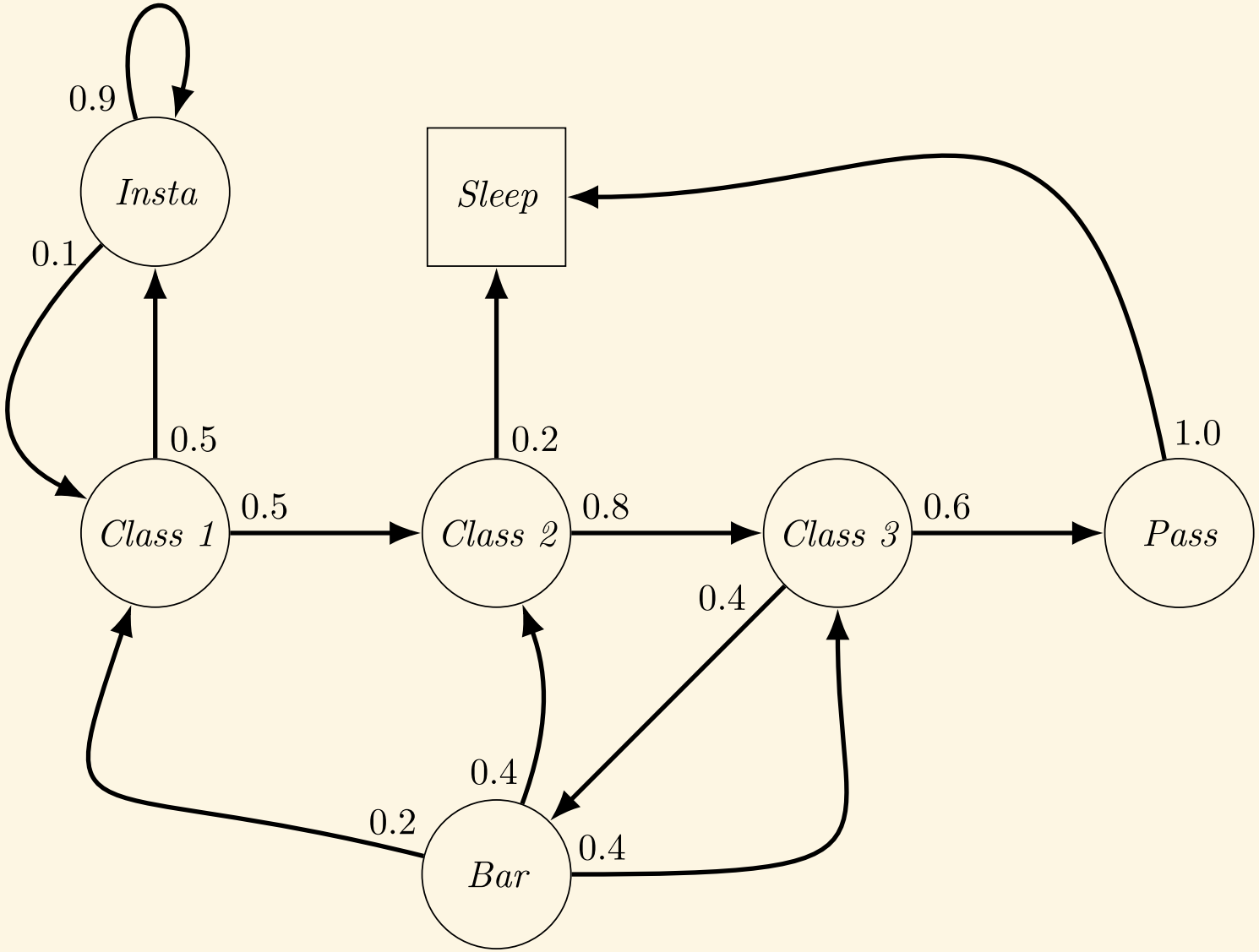
```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location  truck airplane - vehicle  vehicle packet - th
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - locatio
  (:action load
    :parameters (?x - packet ?y - vehicle ?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location ?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at ?x
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  airplan
  (:init (at packet1 office1) (at packet2 office3) ...)
```

When is a problem MRP or STRIPS/PDDL?

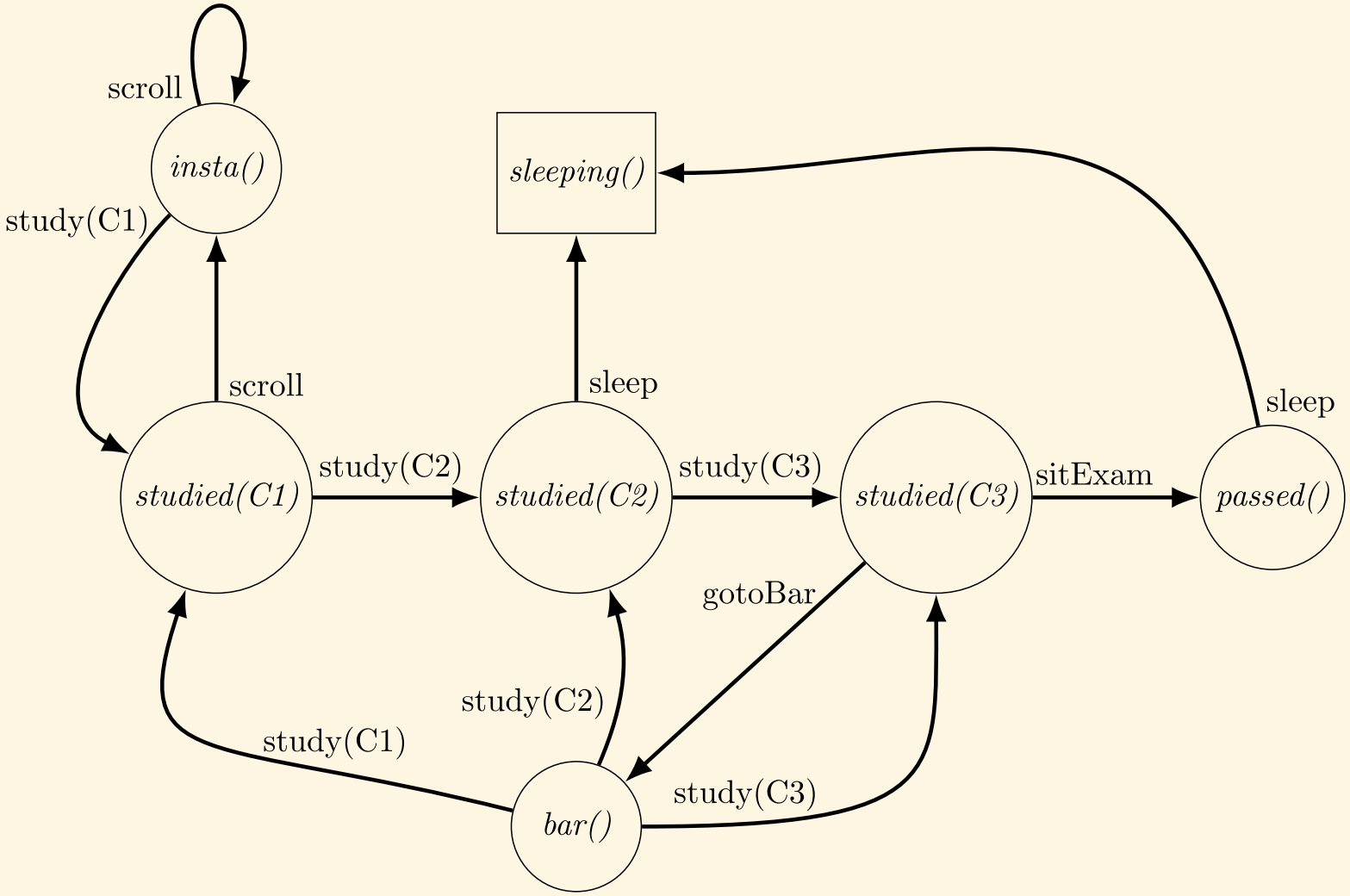
Depends whether we know the propositions, states and/or the probabilities…

- If we know the states and probabilities: MRP
- If we know the propositions (and details of STRIPS model): Classical Planning
- If we don't know the propositions and/or the probabilities, we can typically learn them (later modules)
- If we know propositions **and** probabilities: We can use *both* MRPs and STRIPS/PDDL (we will see integrated learning and planning in later modules)

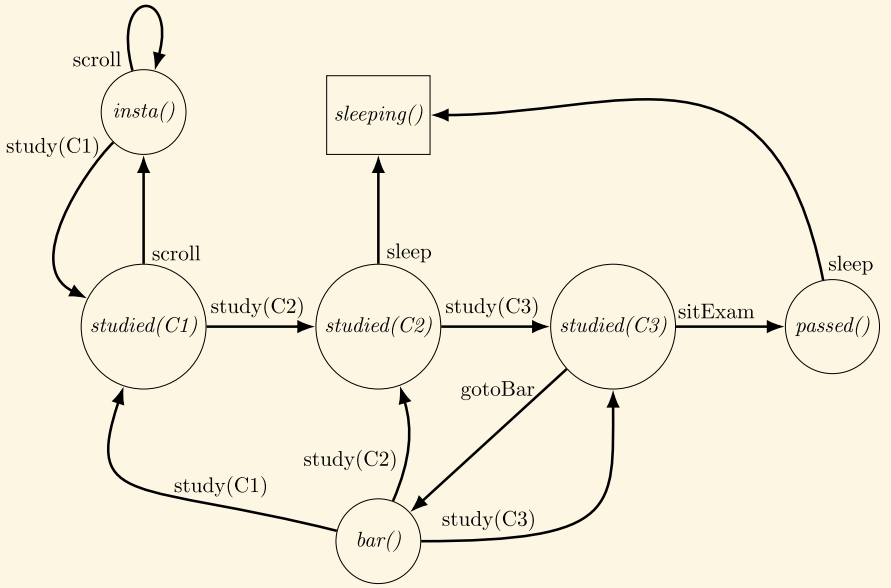
Example: STRIPS version of Student?



Example: STRIPS version of Student?

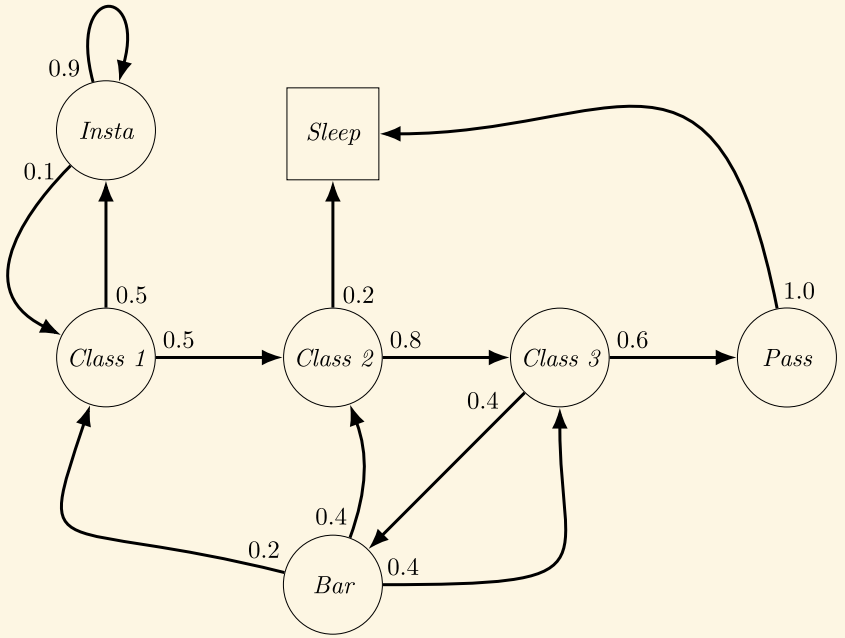


Example: STRIPS version of Student?



- Propositions: $studied(x), bar(), insta(), sleeping(), passed()$
- Initial state: $insta()$
- Goal: $passed(), sleeping()$
- Actions (operators): $sleep, study(x), scroll, study(x), gotoBar, sitExam$

Example: STRIPS version of Student?



- Propositions: ?
- Initial state: ?
- Goal: ?
- Actions (operators): ?

Difference between modelling as MRPs versus Classical Planning?

- **Classical search problems** require finding a path of actions leading from an initial state to a goal state.
- They assume a single-agent, fully-observable, deterministic, static environment.
- STRIPS is a simple but reasonably expressive language: Boolean facts + actions with precondition, add list, delete list.
- PDDL is the de-facto standard language for describing planning problems.

Difference between modelling as MRPs versus Classical Planning?

- **Markov processes** require transition probabilities from state to state.
- They do not assume single-agent - they include both agent and environment transitions, or event multi-agent transitions.
- They capture non-deterministic, dynamic environments, and partially obervability is easily included.
- Dynamics can either be learned (model-free) or provided (model-based).

Difference between MRPs & Classical Planning: Table

Markov reward process (MRP)	Classical Planning
probabilities	*preconditions, add list & delete list*
states	*facts (atoms) / propositions*
non-deterministic actions	*deterministic actions*
policies	*initial state & plans*
actions are stochastic (stochastic policies)	*actions are deterministic*
rewards	*goals*

Categorising Agents

For a categorisation of agents, see the Appendix.

Reading - Markov Processes & Markov Reward Processes

- *Reinforcement Learning, An Introduction* by Richard Sutton and Andrew Barto, Second Edition MIT Press (2020) (available for download at http://www.incompleteideas.net/book/RLbook2020.pdf)
  - See Chapter 3 Finite Markov Decision Processes

Reading - Classical Planning

- Planning textbooks at https://comp90054.github.io/reinforcement-learning/
- *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* Joerg Hoffmann (2011).
  - *Available at*: http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf
  - *Content*: Joerg's personal perspective on planning, for example excerpt from the abstract: "*What is it good for? Does it work? Is it interesting to do research in?*"

Additional Reading - Classical Planning

- *Introduction to STRIPS, from simple games to StarCraft:* http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/
- *Online/Offline editor to model in PDDL:* http://editor.planning.domains
- *VS Code PDDL extension:* https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl