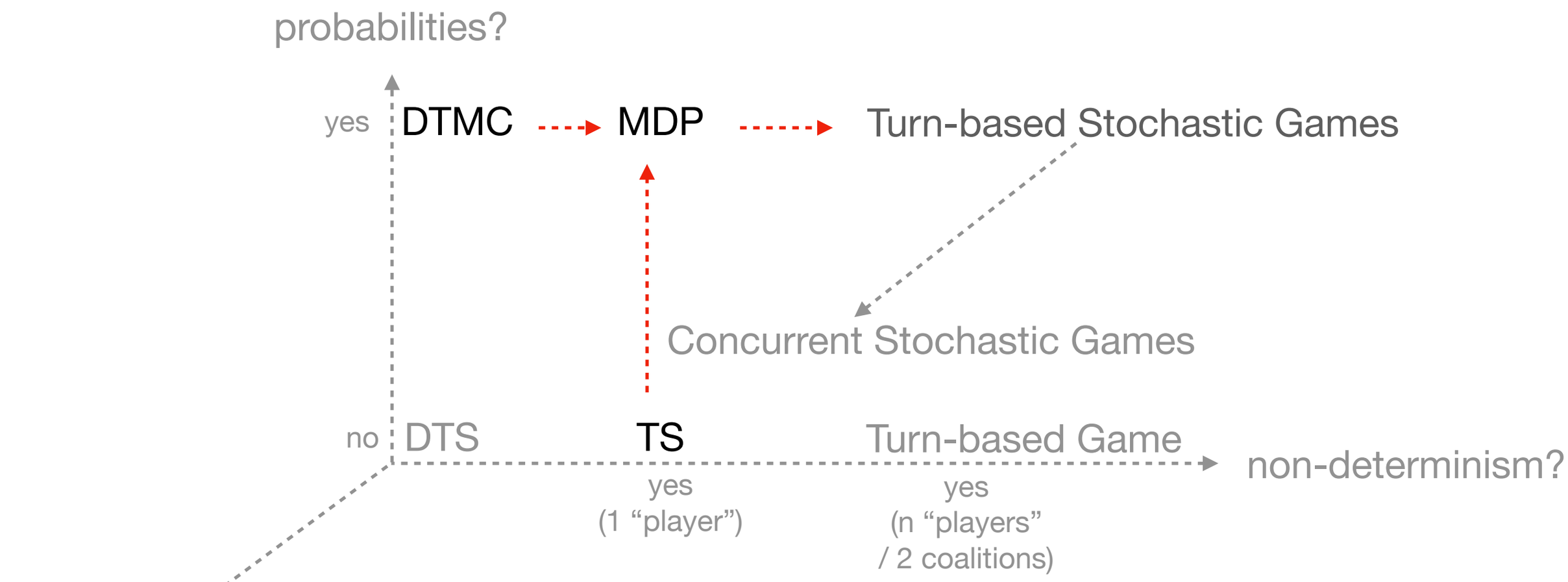
02246 - Model Checking

$$M \models \Phi ?$$

Lecture 10 - Markov Decision Processes

Technical University of Denmark

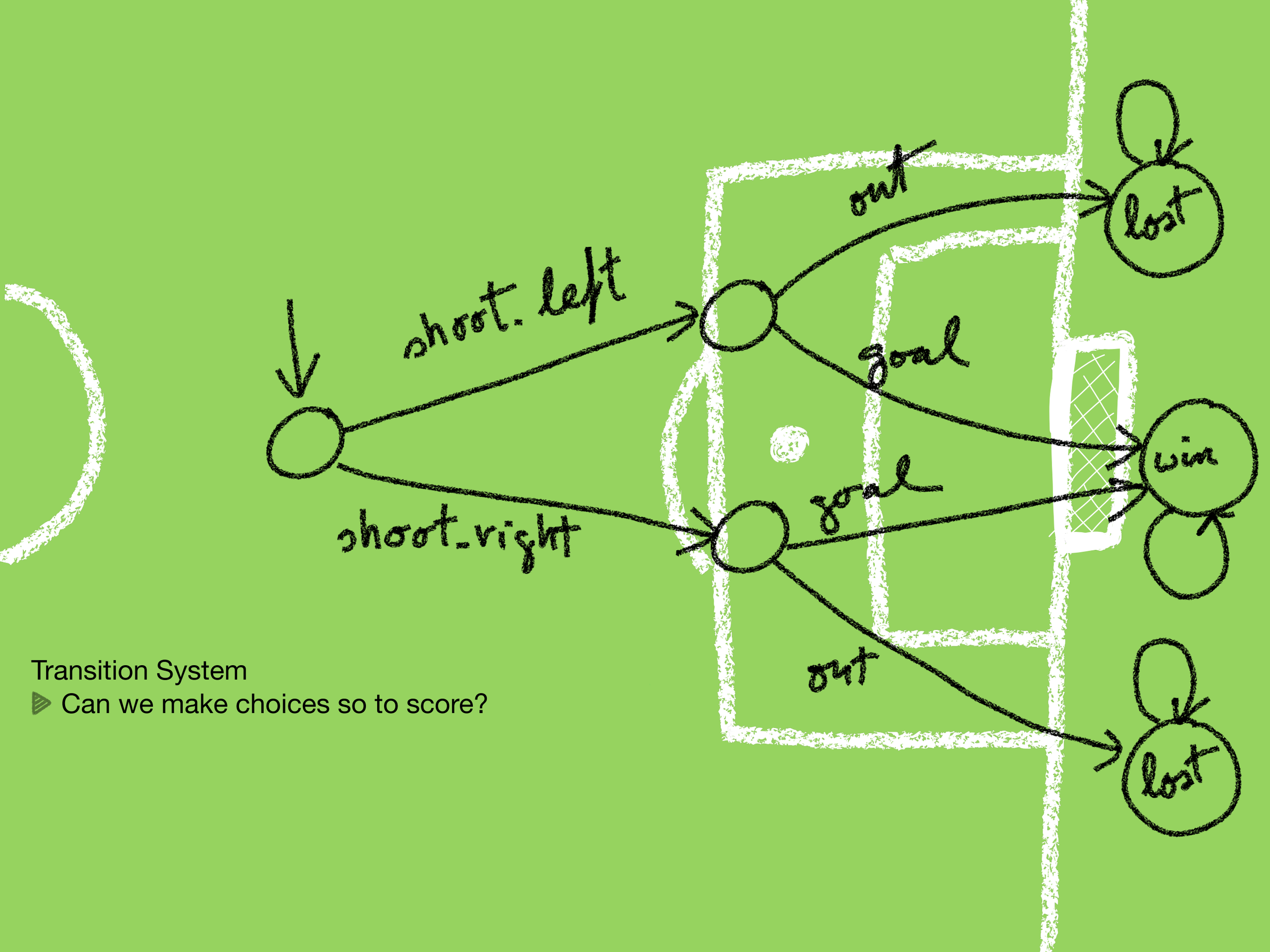
Some discrete-time models seen in the course… and beyond



Other dimensions:

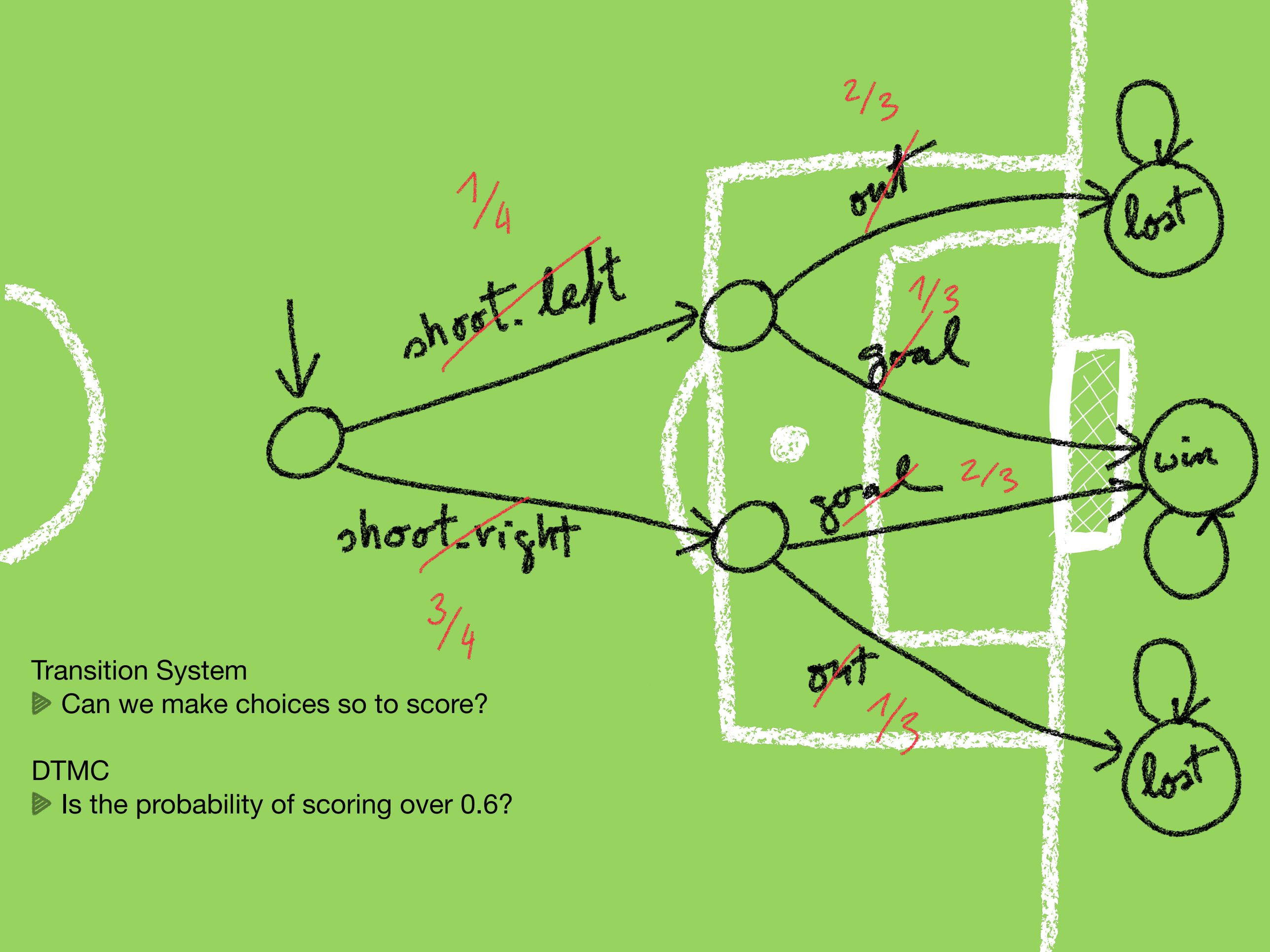
- games (turn/concurrent, collaborative/competitive)
- continuous-time (e.g.stochastic action durations)
- real-time (clocks, timers)
- multi-objective rewards/costs
- observability (partial, stochastic)
- …

Plus… logics to describe models and/or their properties (PL, FOL, CTL, LTL, CTL*, PCTL, rPCTL*, …)



Transition System

- Can we make choices so to score?

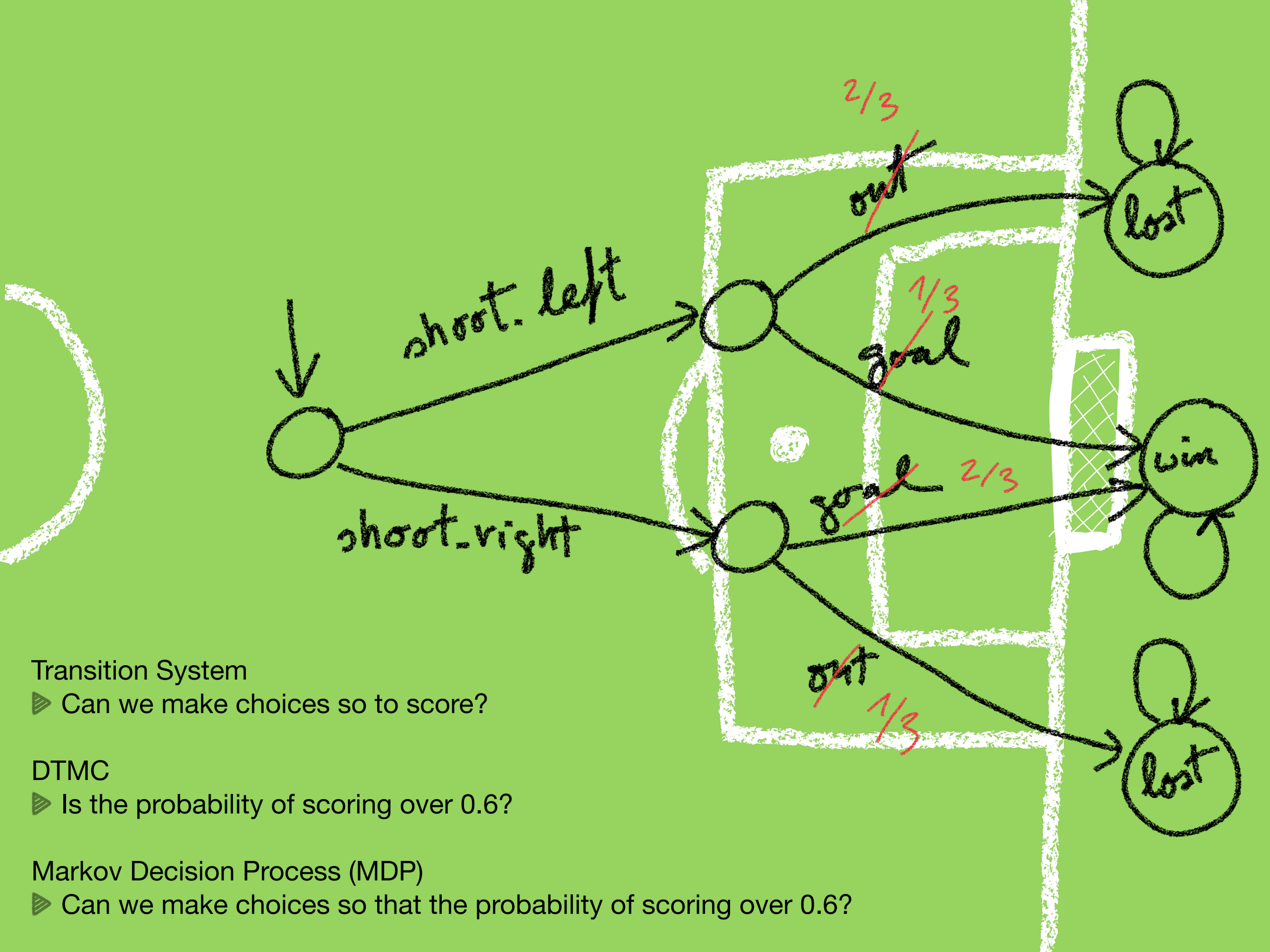


Transition System

- Can we make choices so to score?

DTMC

- Is the probability of scoring over 0.6?



Transition System

- Can we make choices so to score?

DTMC

- Is the probability of scoring over 0.6?

Markov Decision Process (MDP)

- Can we make choices so that the probability of scoring over 0.6?

Key points of this lecture

We can have the best of TSs and DTMCs: TS + DTMC = MDP

Definition of Markov Decision Process

Definition of paths and strategies in MDPs

Computation of max and min probabilities for reachability properties, the basis of model checking MDPs

MDPs are 1-player games!

Reading material

Section 10.6 of “Principles of Model Checking”

Markov Decision Process

A Markov Decision Process (MDP) is a tuple

$$(S, P, \iota, AP, L, Act)$$

such that:

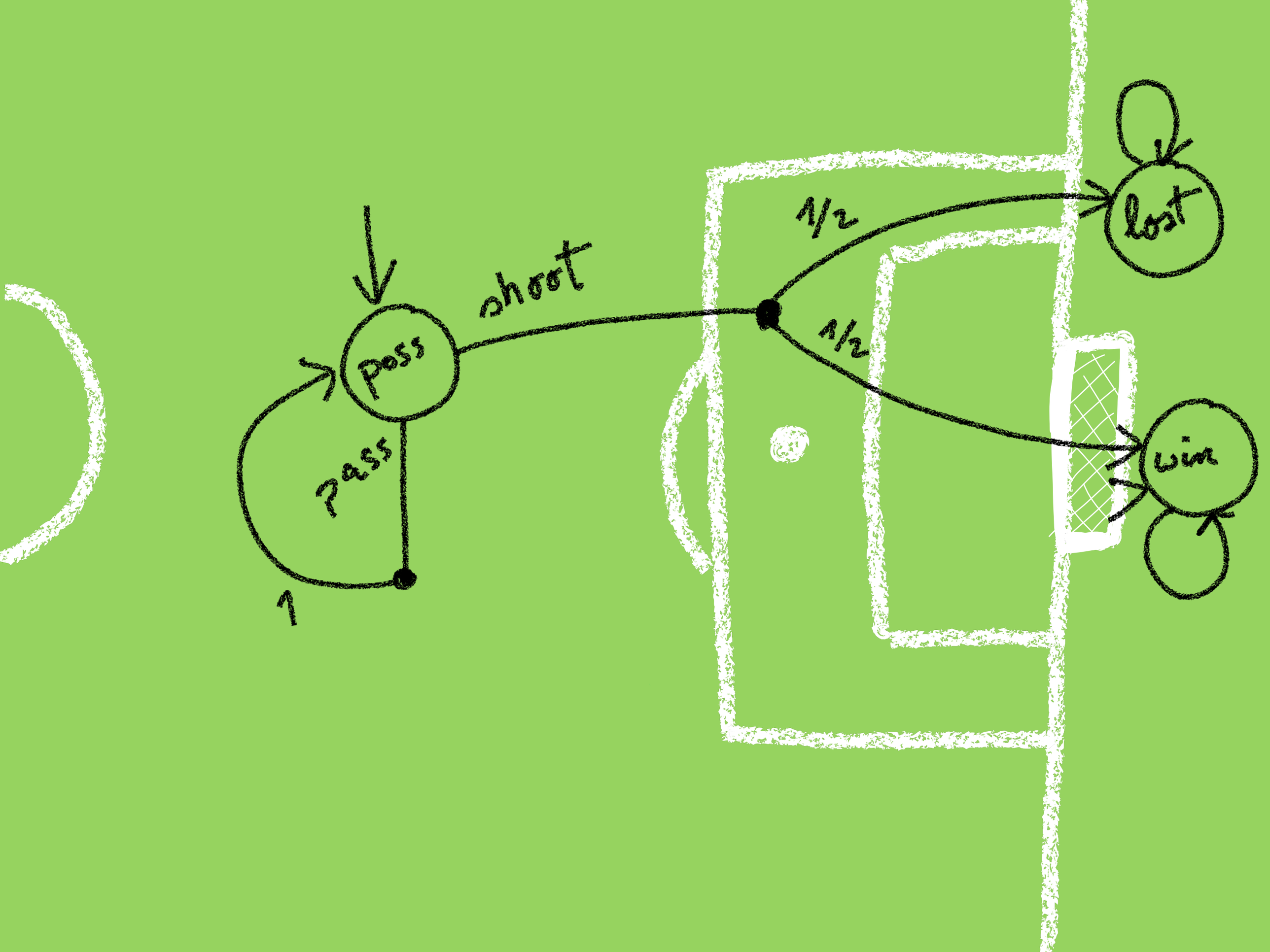
- S is a finite, a non-empty set of states
- $P : S \to Act \to S \to [0,1]$ is a probabilistic transition function, i.e. such that for all states s and actions a it holds

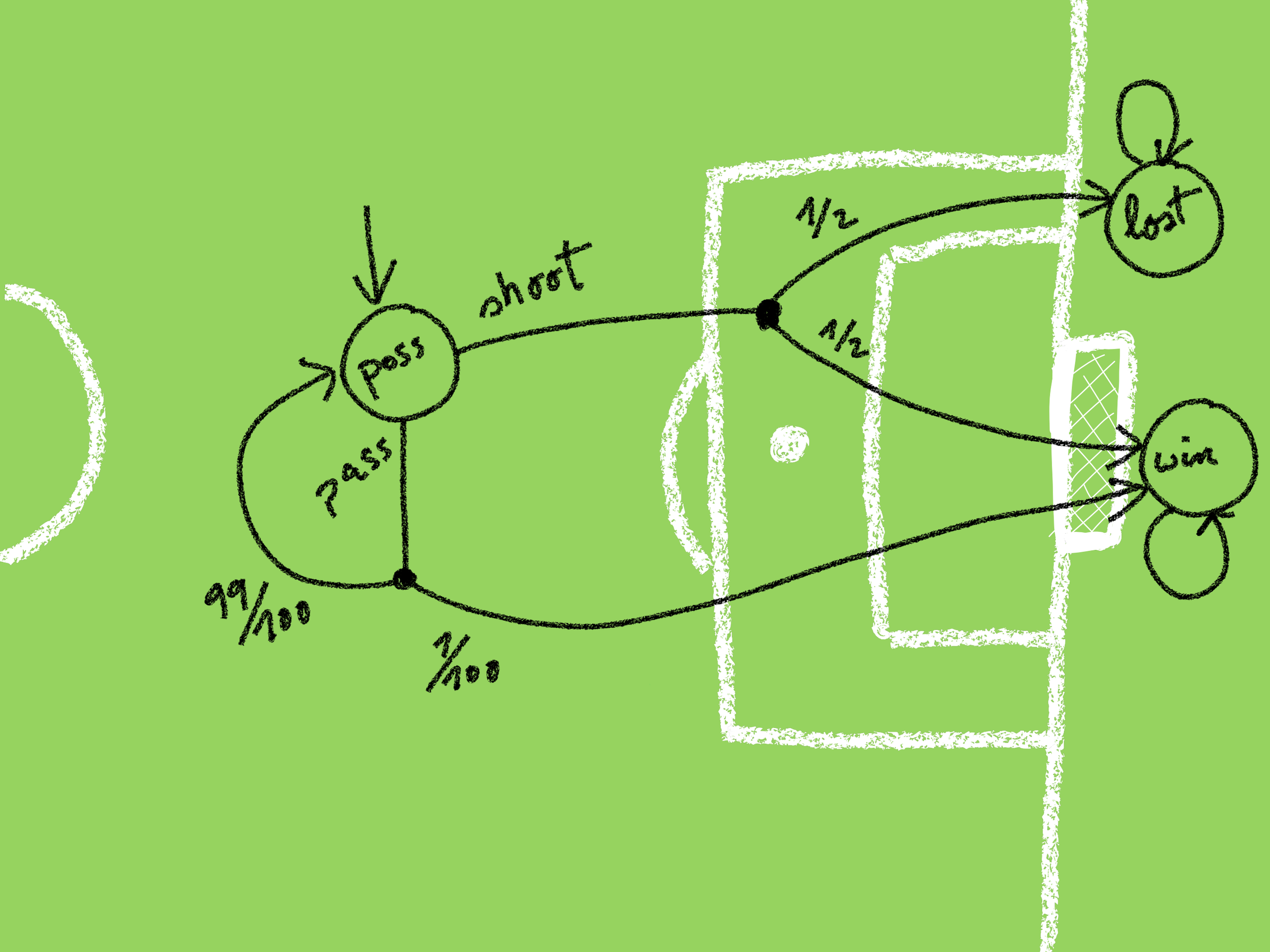
$$\sum_{s' \in S} P(s, a, s') = 1$$

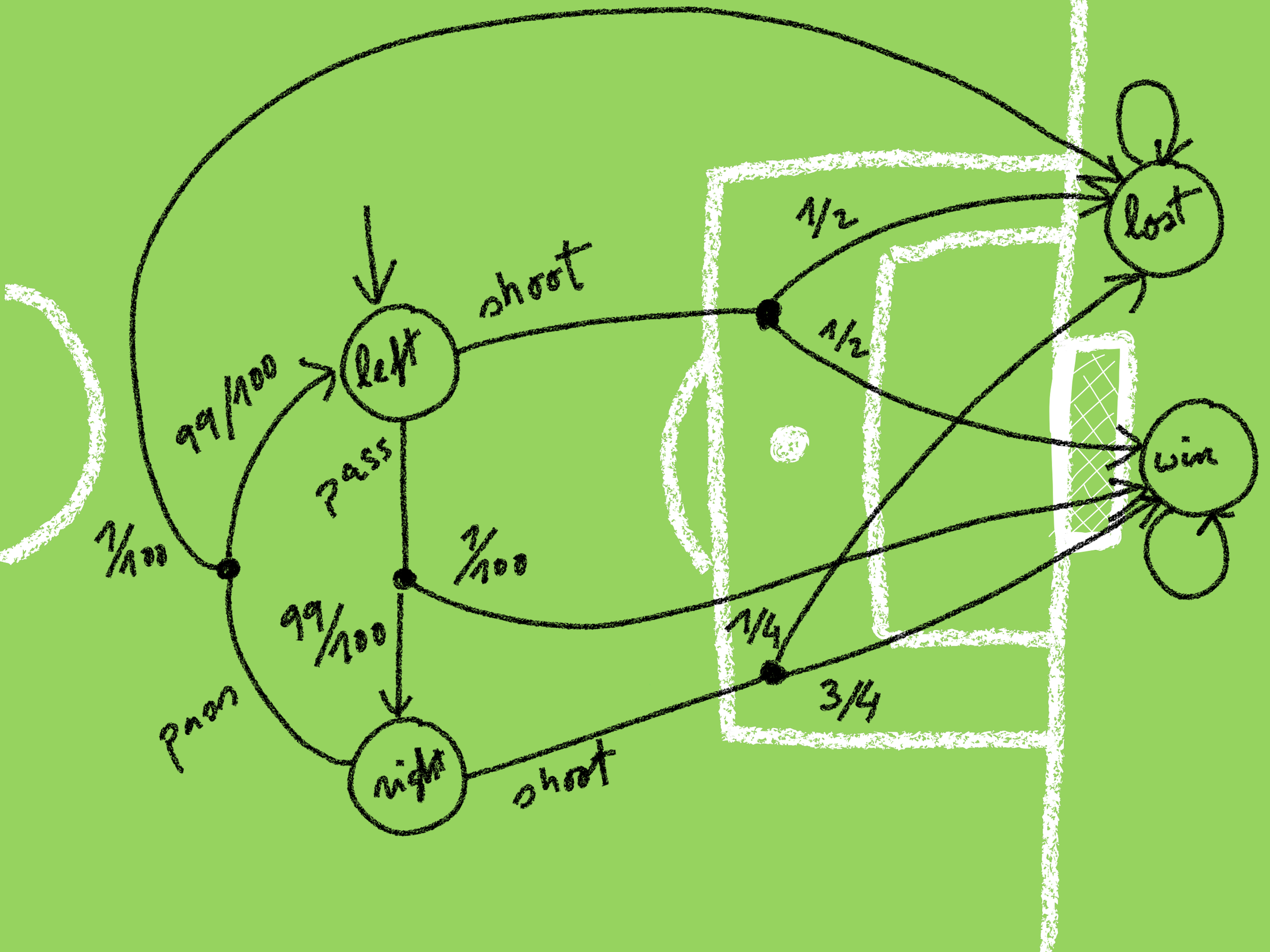
- $\iota : S \to [0,1]$ is an initial distribution, i.e. such that

$$\sum_{s \in S} \iota(s) = 1$$

- AP is a set of atomic propositions
- L is a labelling function
- Act is a set of action labels


pass
shoot
pass
1
1/2
1/2
lost
win


pass
shoot
pass
99/100
1/100
1/2
1/2
lost
win


left
pass
right
shoot
pass
shoot
99/100
1/100
99/100
1/100
1/2
1/2
1/4
3/4
lost
win

Paths in MDPs

A path in an MDP is a sequence of states and actions

$$s_0, a_0, s_1, a_1, \ldots$$

such that transitions can actually be followed

$$P(s_i, a_i, s_{i+1}) > 0$$

As usual we can define the set of all paths

$$Paths_s = \{s_0, a_0, s_1, a_1, \ldots \mid s = s_0 \wedge \forall i \in \mathbb{N} \,.\, P(s_i, a_i, s_{i+1}) > 0\}$$

A path mixes non-deterministic and probabilistic choices, how to measure probabilities of set of paths?

Strategies in MDPs

A memory-less strategy is a mapping of states into actions

$$\sigma : S \to Act$$

Every strategy induces a DTMC!

$$(S, P^\sigma, \iota, AP, L) \quad \text{where} \quad P^\sigma(s, s') = P(s, \sigma(s), s')$$

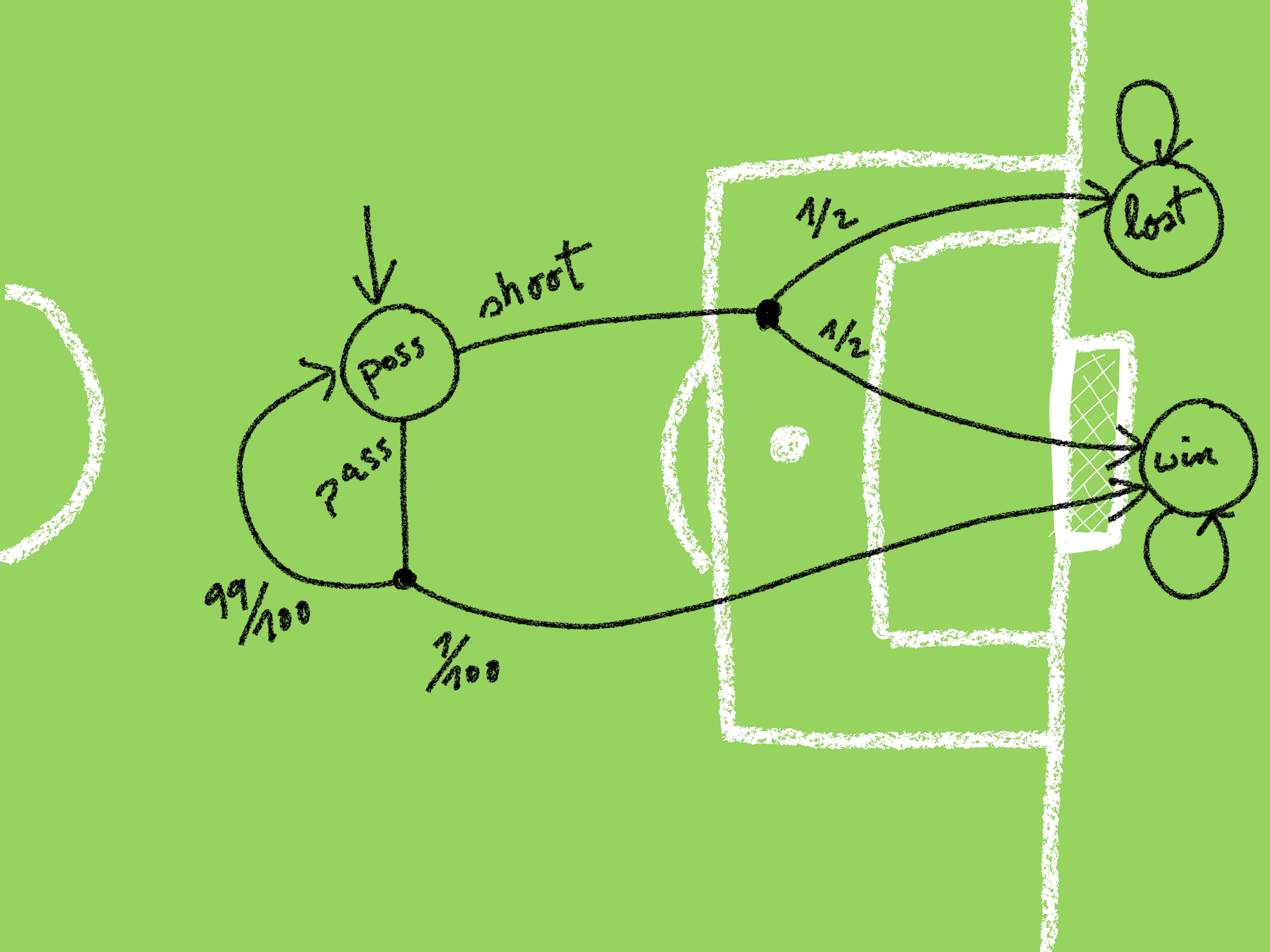
We can then re-use al lot of the DTMC machinery (probability measures for paths, PCTL model checking, etc) for various purposes.

For example, we can define the set of paths induced by a strategy

$$Paths_s^\sigma = \{s_0, a_0, s_1, a_1, \ldots \mid s = s_0 \wedge \forall i \in \mathbb{N} \,.\, P(s_i, a_i, s_{i+1}) > 0 \wedge a_i = \sigma(s_i)\}$$

And measure probabilities subsets of it

NOTE: strategies can be history-based but we focus on memory less here for simplicity since they are enough for our purpose here, i.e. measuring max and min probabilities.


pass
shoot
pass
99/100
1/100
1/2
1/2
lost
win

$\Gamma_0 = \{ \text{pass} \mapsto \text{pass}, \ \text{lost} \mapsto \text{cry}, \ \text{win} \mapsto \text{celeb} \}$

cry

pass

shoot

1/2

1/2

lost

win

celeb

99/100

1/100

$\Gamma_1 = \{ \text{pass} \mapsto \text{shoot}, \ \text{lost} \mapsto \ldots, \ \text{win} \mapsto \ldots \}$

Measuring probabilities of paths

In an MDP it does not make sense to ask for the probability of arbitrary paths

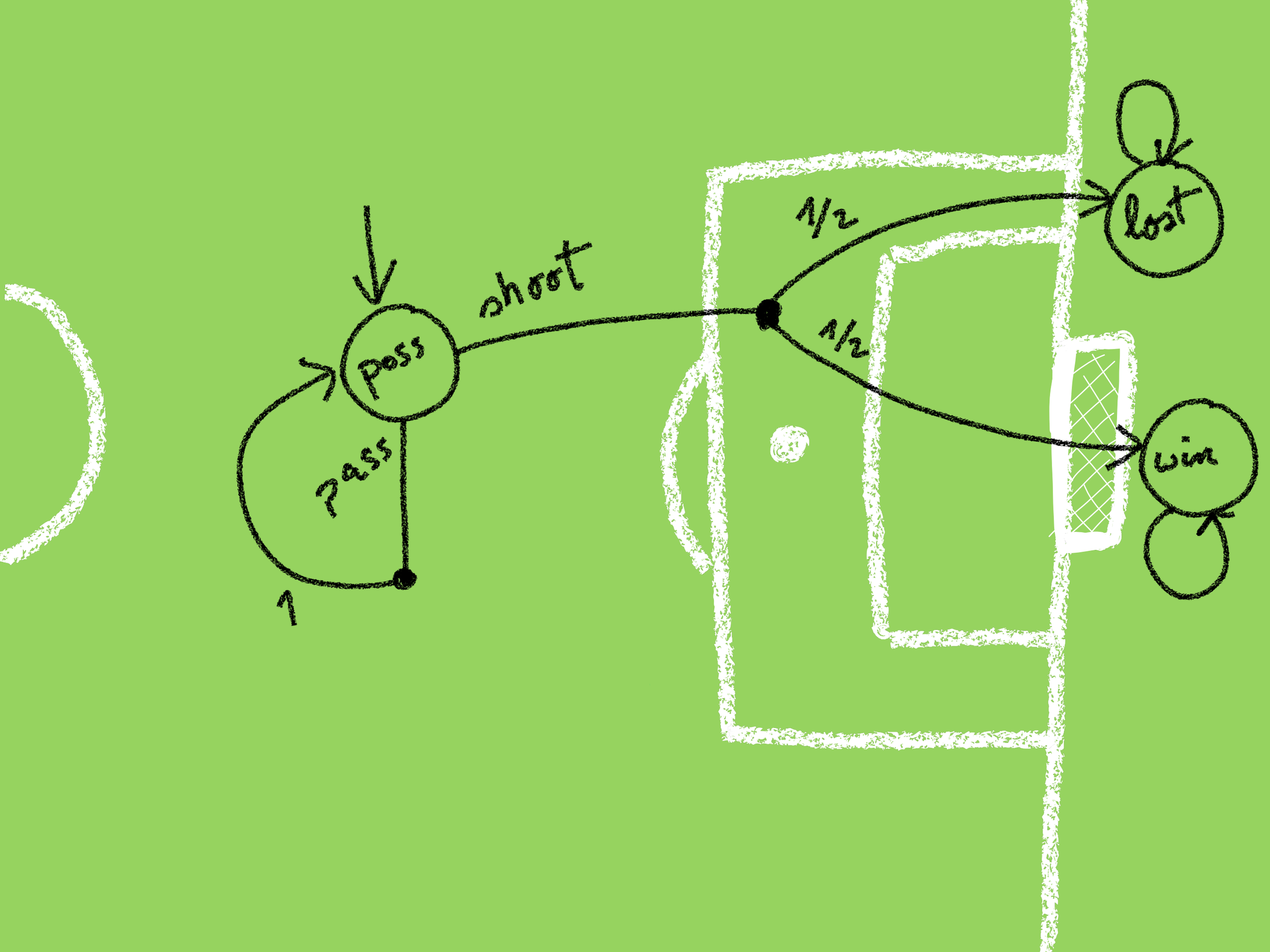
$$Pr_s(\{\pi \in Paths(s) \mid \pi \vDash \psi\})?$$

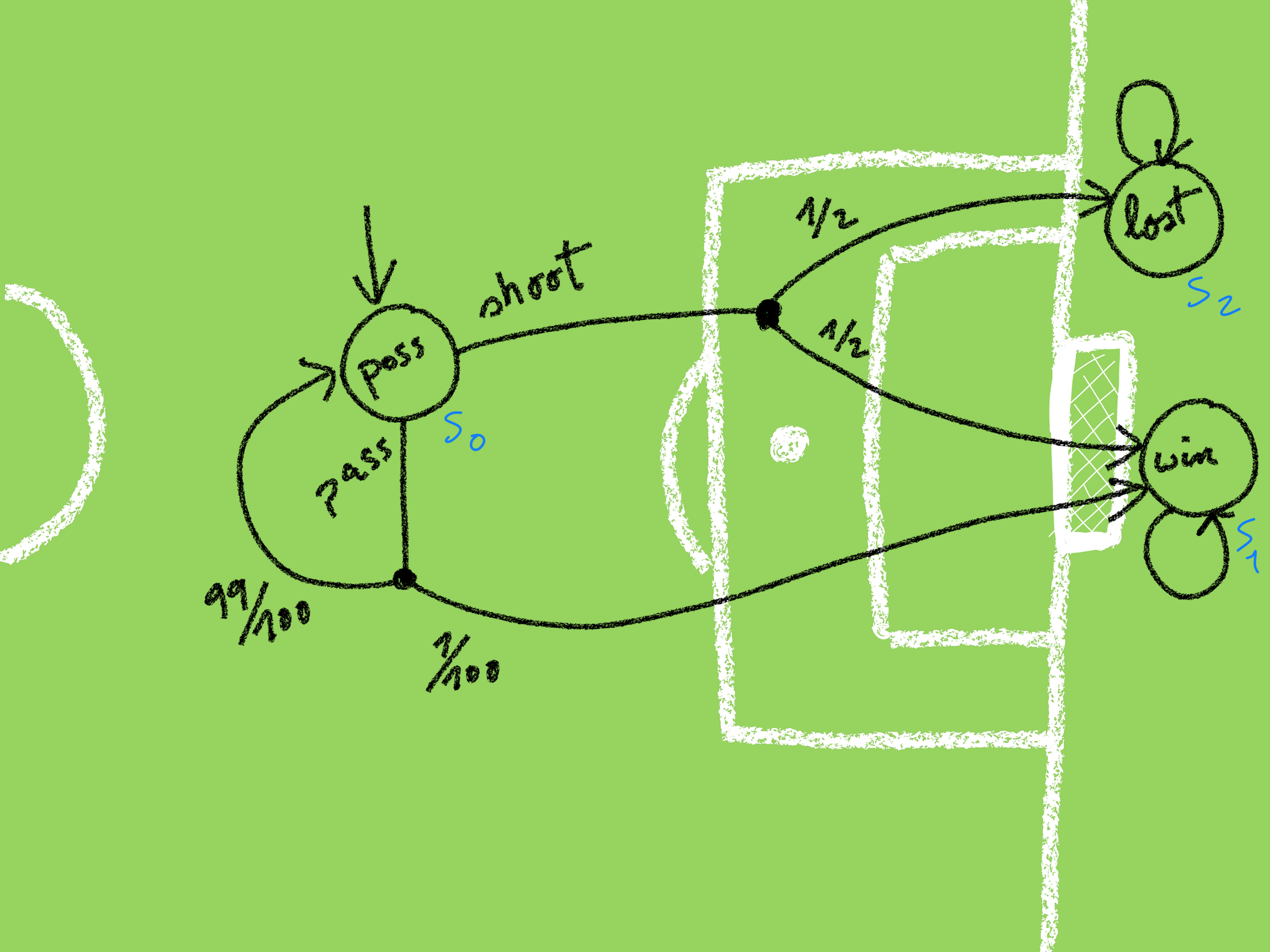
since we can't measure probabilities in presence of non-determinism.

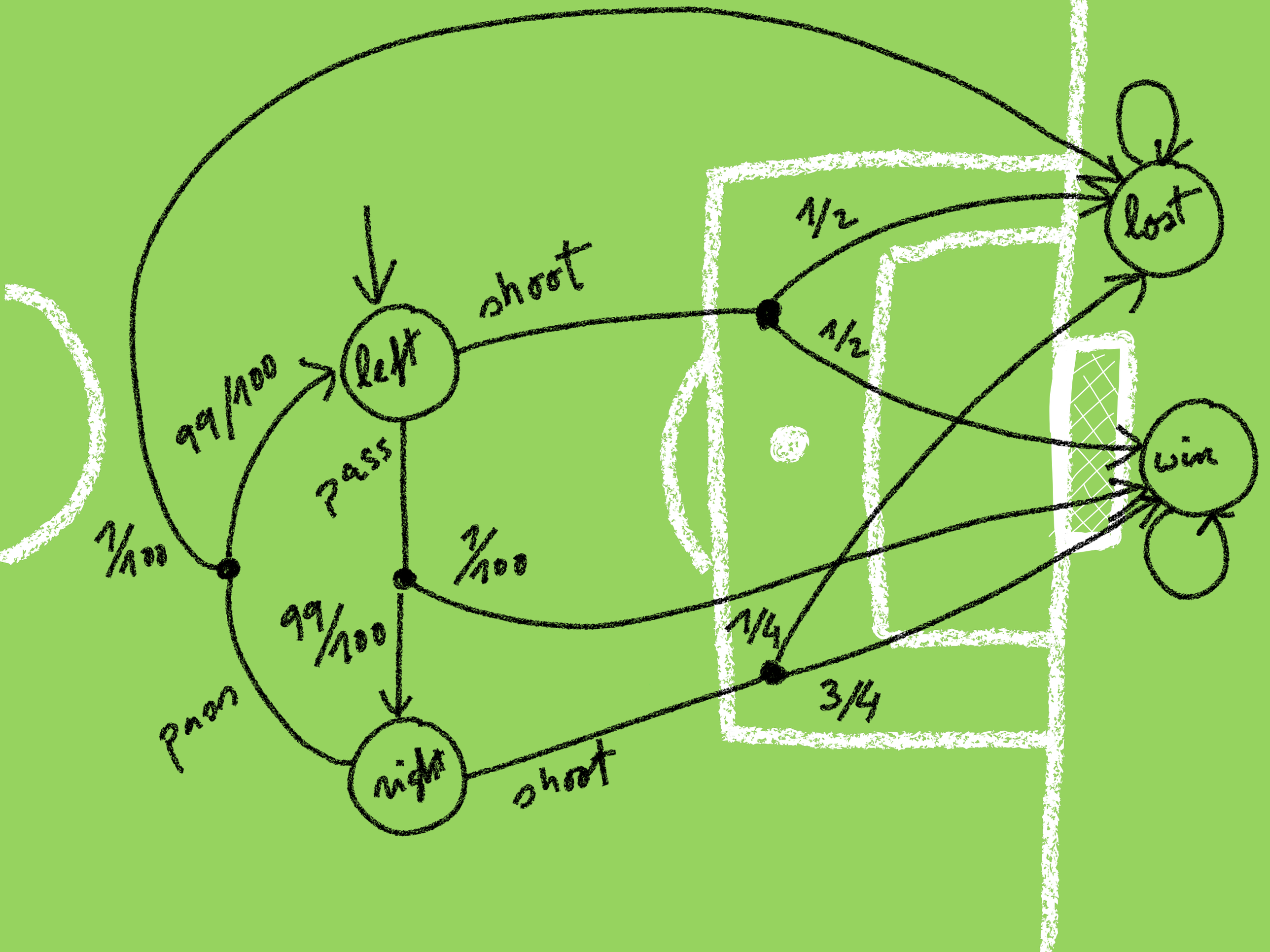
But we can ask about the best/worst possible strategies

$$min_{\sigma \in (S \to Act)}\{Pr_s(\{\pi \in Paths^{\sigma}(s) \mid \pi \vDash \psi\})\}?$$

$$max_{\sigma \in (S \to Act)}\{Pr_s(\{\pi \in Paths^{\sigma}(s) \mid \pi \vDash \psi\})\}?$$


pass
shoot
pass
1
1/2
1/2
lost
win


pass
shoot
1/2
1/2
lost
win
pass
99/100
1/100
S_0
S_2
S_1


left
pass
right
shoot
pass
shoot
99/100
1/100
99/100
1/100
1/2
1/2
1/4
3/4
lost
win

Probabilistic Reachability

How to compute min/max probabilities? There are too many strategies!

namely… $|Act|^{|S|}$

Fortunately, we can optimise locally, as we shall see.

As for DTMCs, we just need to solve a system of equations :)

NOTE: the space of possible memory-dependent strategies is even infinite!
It is thus good that we can restrict to memory-less strategies.

Computing $Pmin_s(\Diamond B)$

It is as simple as solving the following system of equations

$$Pmin_s(\Diamond B) = 1 \qquad \text{if } s \in B$$

$$Pmin_s(\Diamond B) = 0 \qquad \textit{for some strategy } s \models \neg \exists \Diamond B$$

$$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\} \qquad \text{otherwise}$$

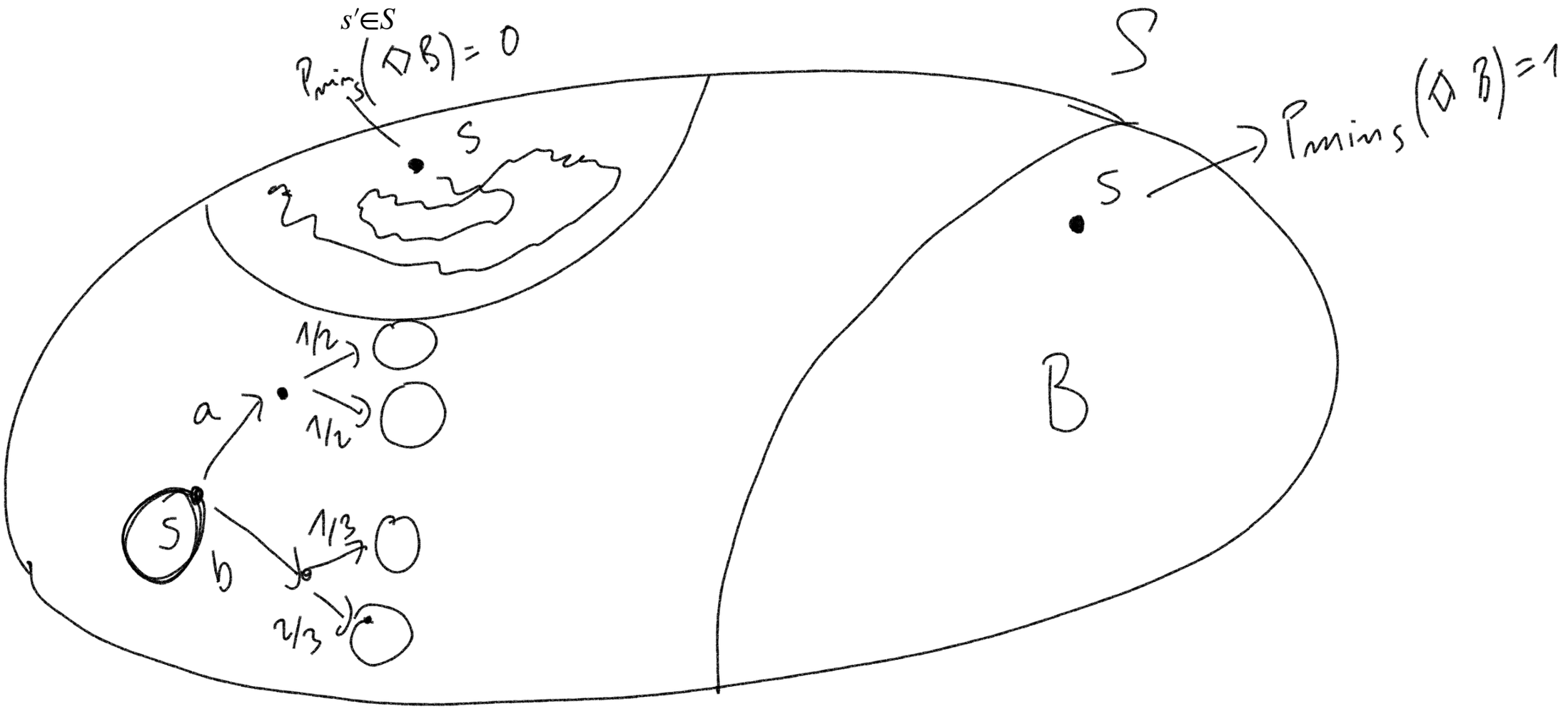
Computing $Pmin_s(\Diamond B)$

It is as simple as solving the following system of equations

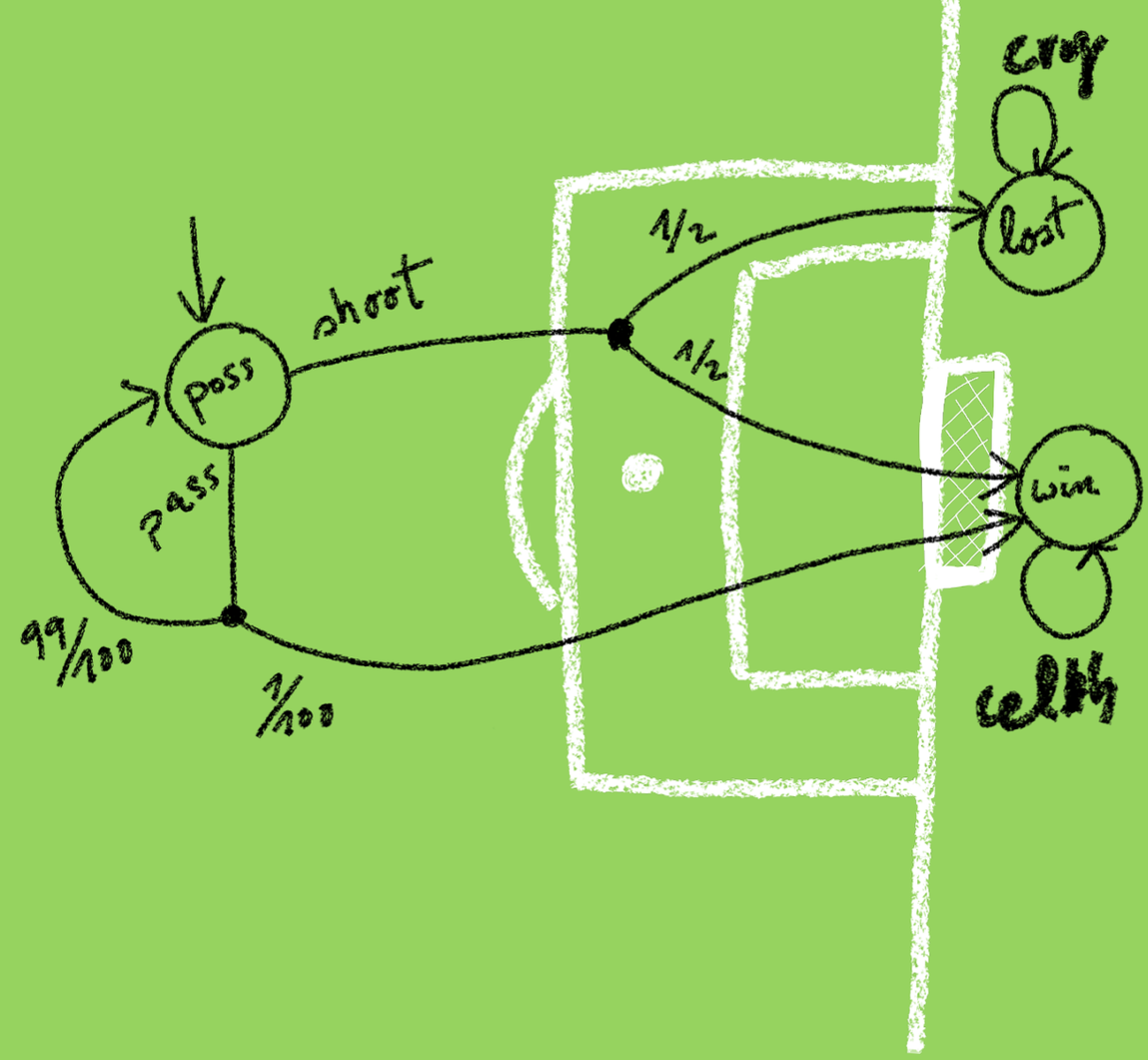
$Pmin_s(\Diamond B) = 1$ if $s \in B$

$Pmin_s(\Diamond B) = 0$ *for some strategy* $s \models \neg \exists \Diamond B$

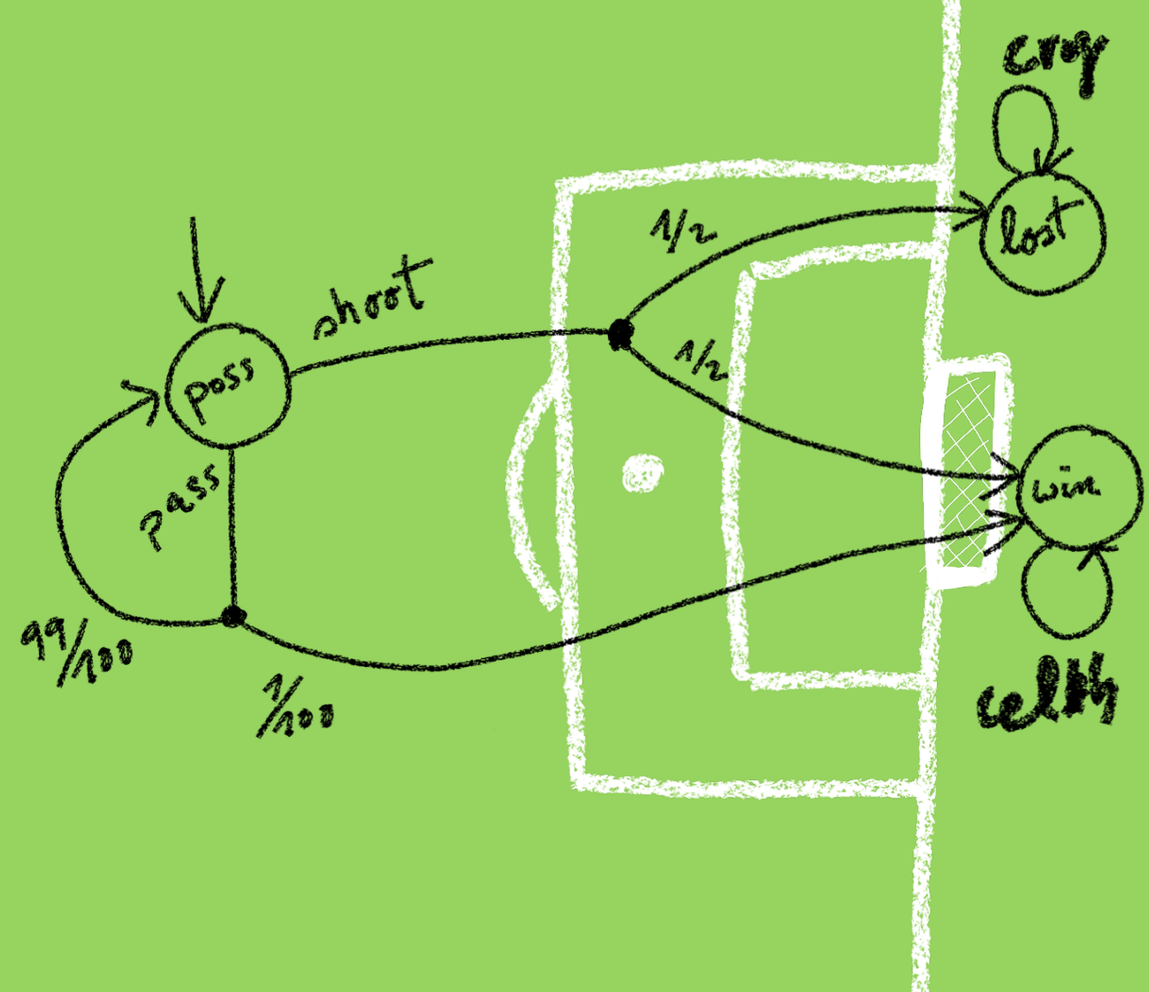
$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\}$ otherwise



$$\Pi_S = P_{\min S}(\text{◇ win})$$



$$\pi_s = P_{min_s}(0\ \text{win})$$



$$\pi_{lost} = 0$$

$$\pi_{win} = 1$$

$$\pi_{pass} = \min\left\{ \frac{1/2 \cdot 0 + 1/2 \cdot 1}{shoot},\ \frac{1/100 \cdot 1 + 99/100 \cdot \pi_{pass}}{pass} \right\}$$

$$\sigma(pass) = shoot$$

$$\downarrow$$

$$1/2$$

$$\pi_{pass} = 99/100\ \pi_{pass} + 1/100$$

$$\downarrow$$

$$\pi_{pass} = 1$$

Computing $Pmax_s(\Diamond B)$

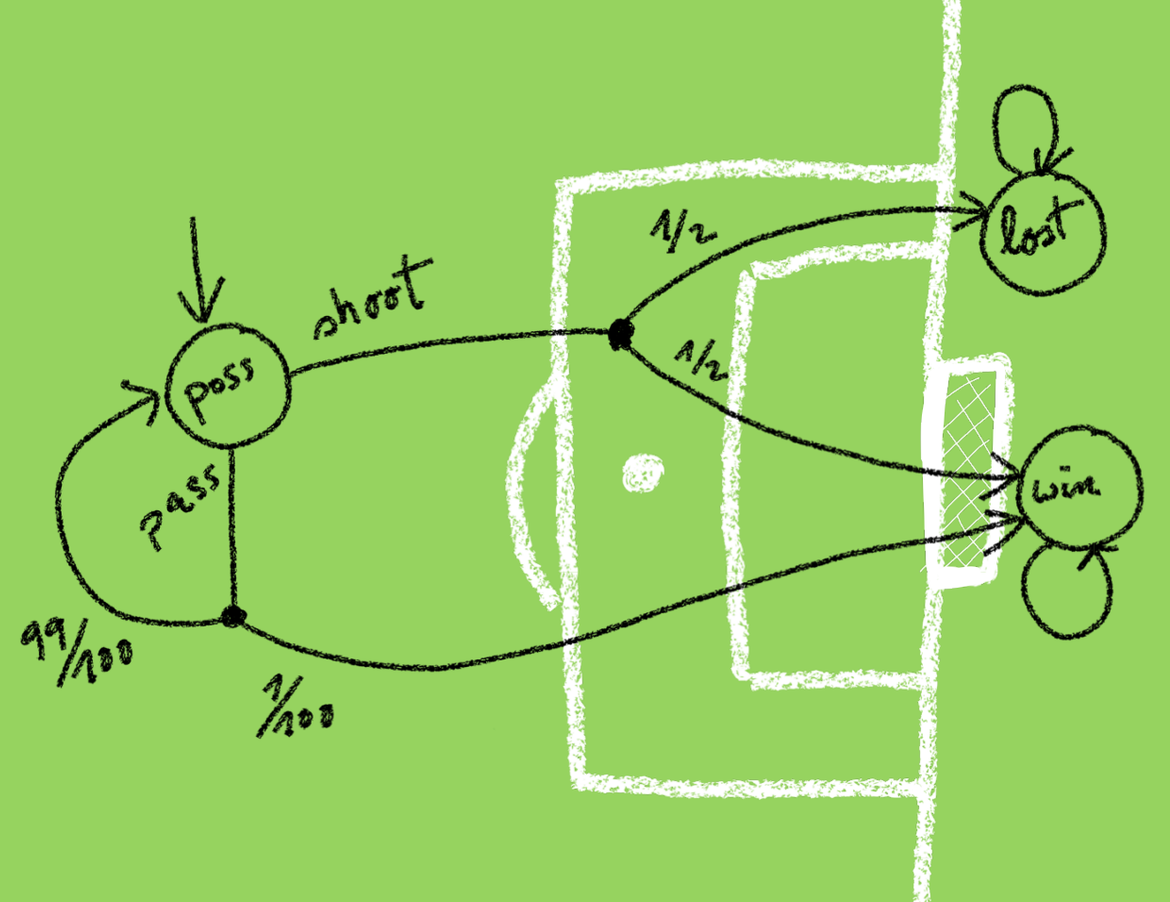
It is as simple as solving the following system of equations

$$Pmax_s(\Diamond B) = 1 \qquad \text{if } s \in B$$

$$Pmax_s(\Diamond B) = 0 \qquad \textit{for all strategies } s \models \neg \exists \Diamond B$$

$$Pmax_s(\Diamond B) = max_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmax_{s'}(\Diamond B)\} \qquad \text{otherwise}$$

$$\pi_s = P_{\cancel{min}\ max\ s}(\Diamond\, win)$$



$$\pi_{lost} = 0$$

$$\pi_{win} = 1$$

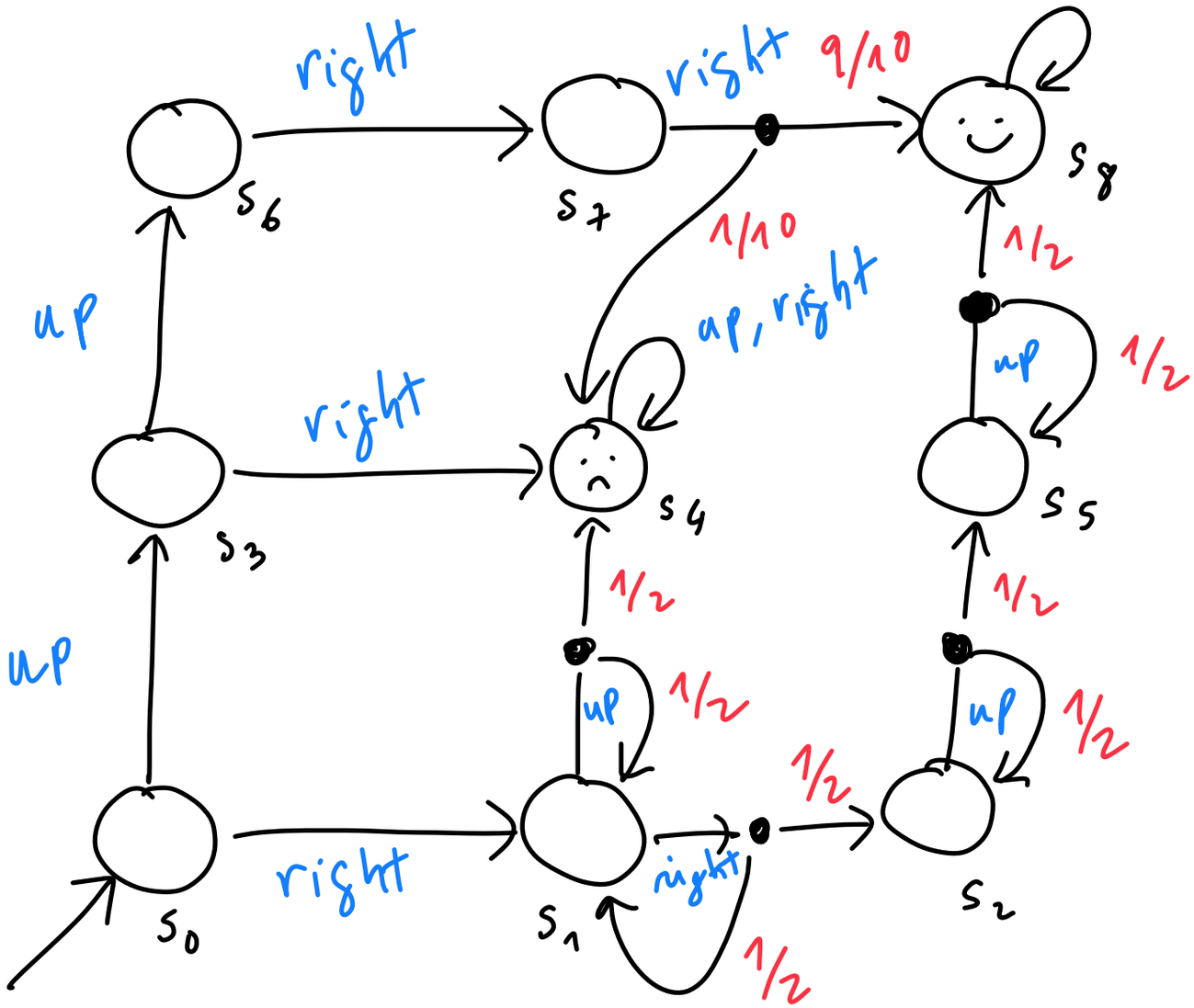
$$\pi_{pass} = \cancel{min}\ max \left\{ \begin{array}{l} \underbrace{1/2 \cdot 0 + 1/2 \cdot 1}_{shoot} \\ \\ \underbrace{1/100 \cdot 1 + 99/100 \cdot \pi_{pass}}_{pass} \end{array} \right\}$$

$\sigma(pass) = pass$

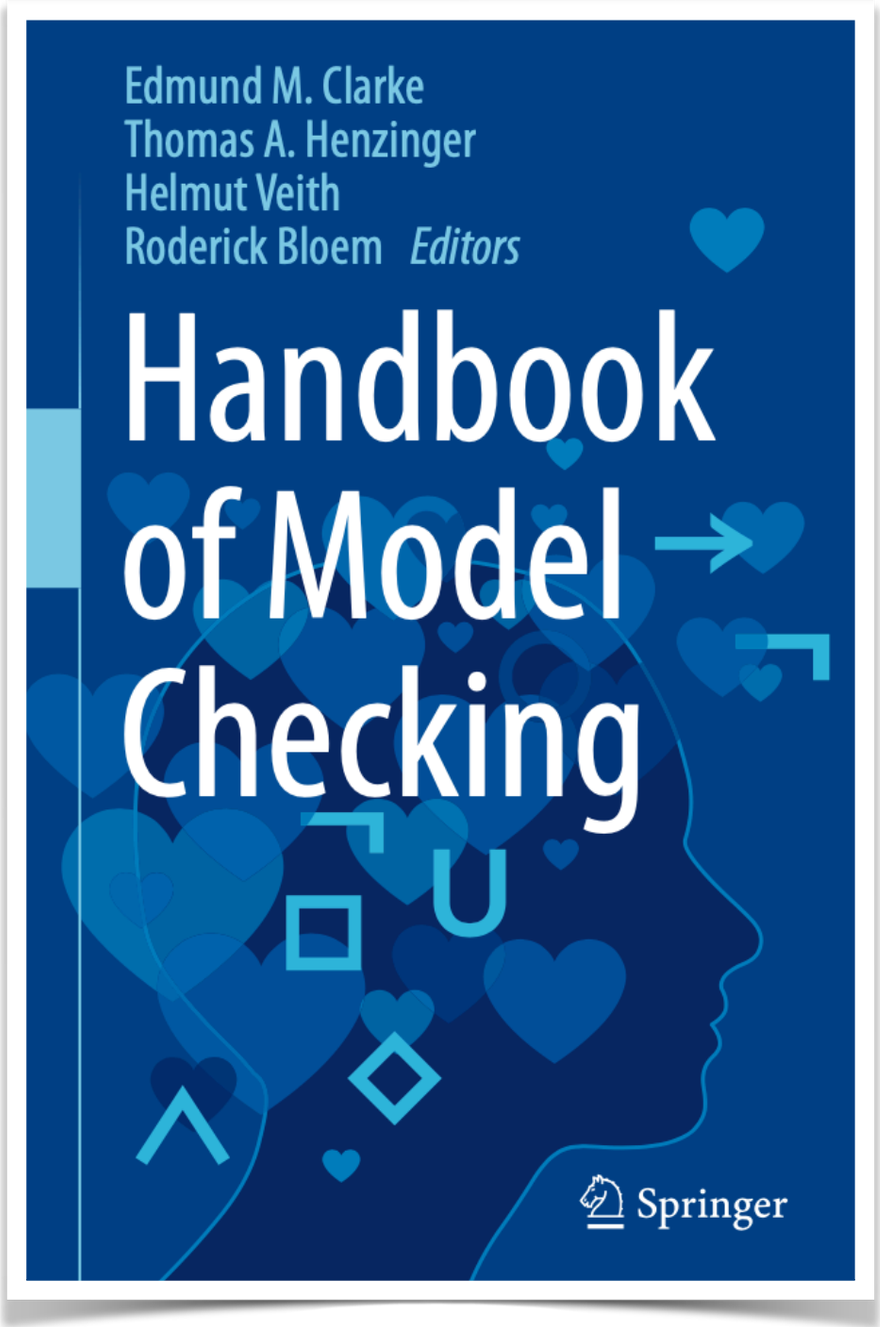
$$\pi_{pass} = 99/100\ \pi_{pass} + 1/100$$

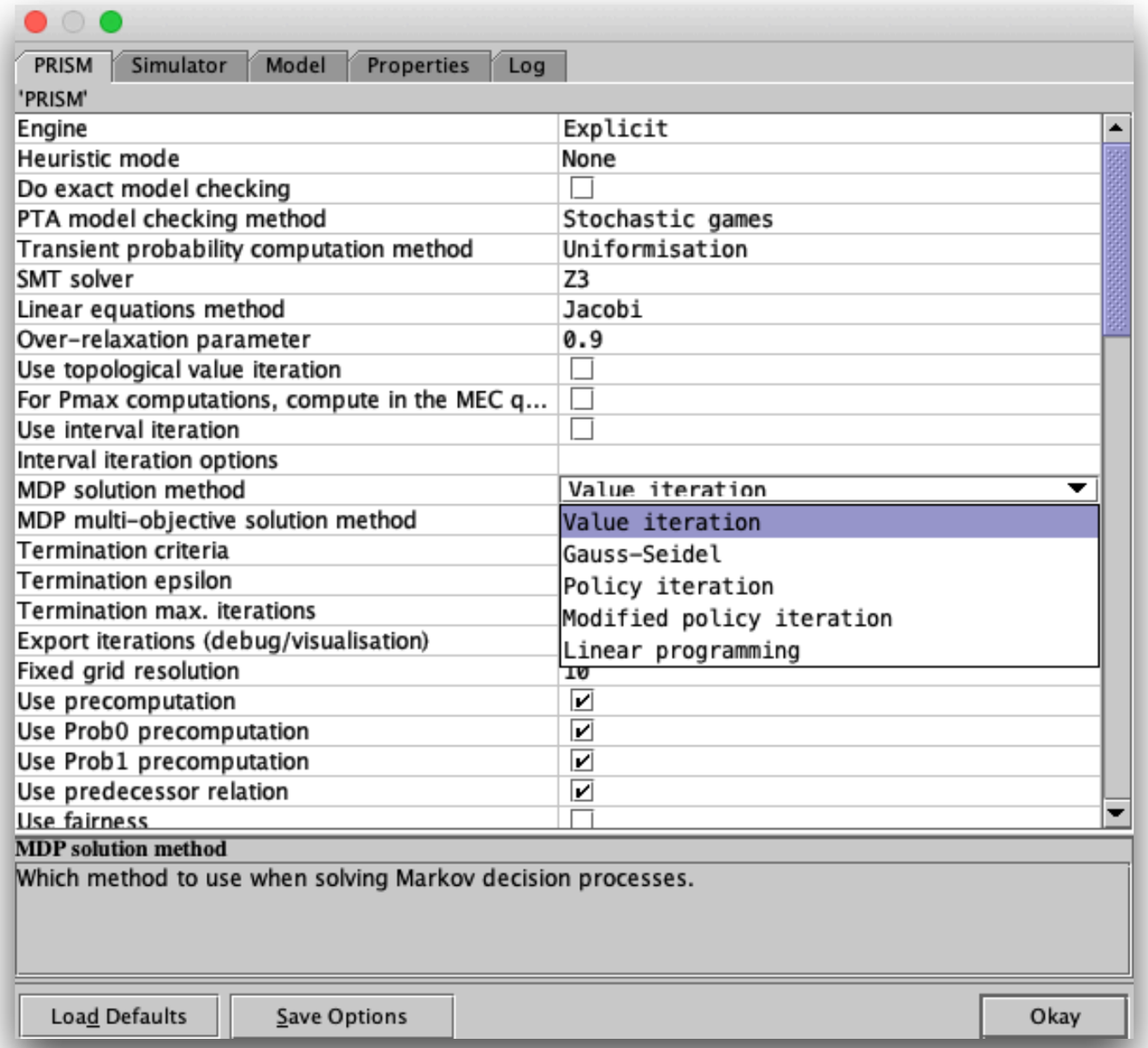
$$\pi_{pass} = 1$$

Let's try another example



Algorithms for computing $Pmin_s(\Diamond B)$ (and the rest of PCTL)





https://findit.dtu.dk/en/catalog/5b449dcf5010df0108547c8d

Computing $Pmin_s(\Diamond B)$ as a linear program

Recall our goal

$$Pmin_s(\Diamond B) = 1 \qquad \text{if } s \in B$$

$$Pmin_s(\Diamond B) = 0 \qquad \textit{for some strategy } s \models \neg \exists \Diamond B$$

$$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\} \qquad \text{otherwise}$$

Solving this can be done by solving the following linear program:

We introduce variables $x_s = Pmin_s(\Diamond B)$

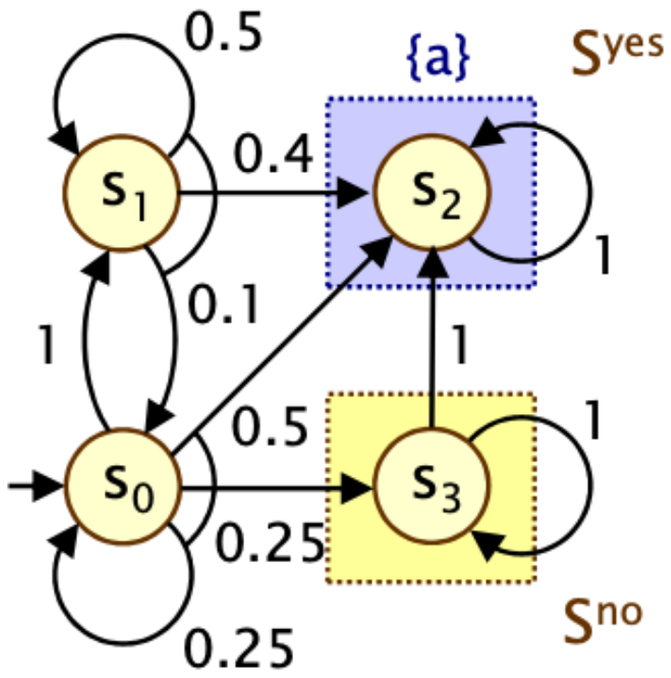
Introduce trivial equalities for the simple cases (see above)

$$x_s = 0 \quad \text{or} \quad x_s = 1$$

For the rest of the states introduce an inequality like this for every action a:

$$x_s \leq \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot x_{s'}$$

Solve equations that maximize $\sum_{s \in S} x_s$



Let $x_i = Pr_{s_i}^{min}(F\ a)$

S^{yes}: $x_2=1$, S^{no}: $x_3=0$

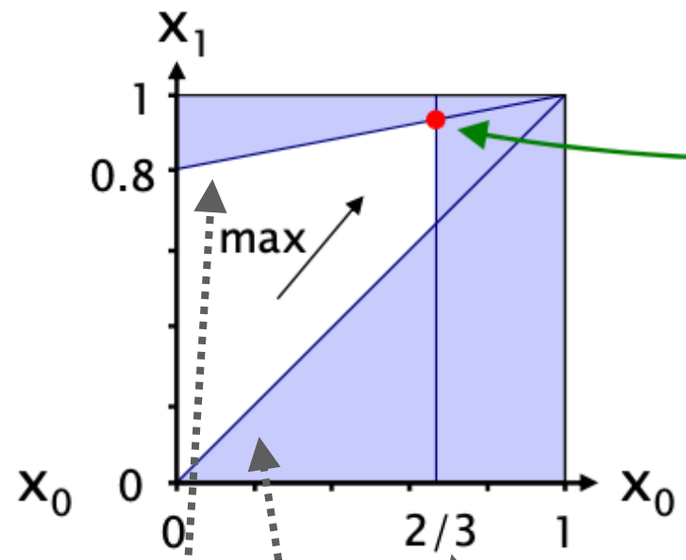
For $S^? = \{x_0, x_1\}$:

Maximise x_0+x_1 subject to constraints:

- $x_0 \le x_1$
- $x_0 \le 0.25 \cdot x_0 + 0.5$
- $x_1 \le 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

simplify

- $x_0 \le x_1$
- $x_0 \le 2/3$
- $x_1 \le 0.2 \cdot x_0 + 0.8$



Solution:

(x_0, x_1)

=

(2/3, 14/15)

[taken from Marta Kwiatowka's lecture at GSSI 2015 - http://www.prismmodelchecker.org/courses/gran-sasso15/]

Linear programming in Z3

- $x_0 \leq x_1$
- $x_0 \leq 2/3$
- $x_1 \leq 0.2 \cdot x_0 + 0.8$

```
# Equations for solving example
solver = Optimize()
solver.add(
  x0 <= x1 ,
  x0 <= Q(2,3) ,
  x1 <= Q(2,10)*x0 + Q(8,10)
)
```

Computing $Pmin_s(\Diamond B)$ with value iteration

Recall our goal

can be enlarged

$Pmin_s(\Diamond B) = 1$ if $s \in B$

$Pmin_s(\Diamond B) = 0$ *for some strategy* $s \models \neg \exists \Diamond B$

$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\}$ otherwise

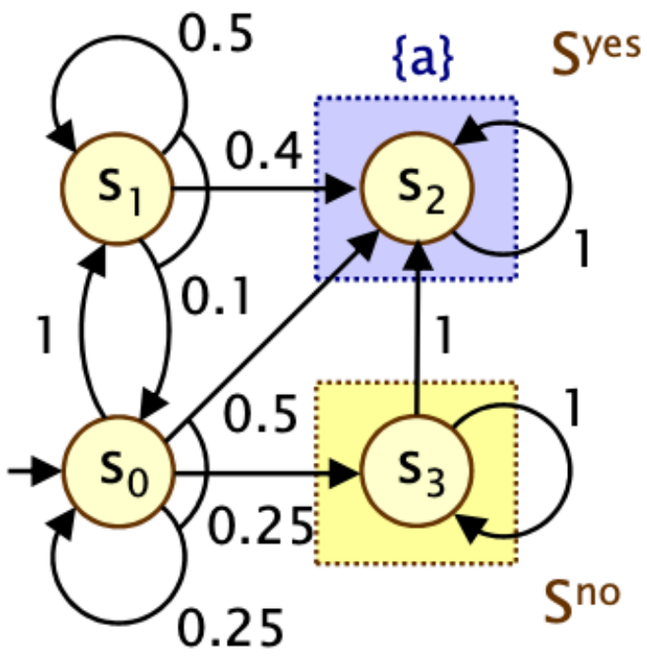
We introduce variables x_s^n

Rationale: $\lim_{n \to \infty} x_s^n = Pmin_s(\Diamond B)$

Start with $x_s^0 = 1$ when s in B (as above). For the rest of states: $x_s^0 = 0$

Successively compute $x_s^{n+1} = \min_{a \in Act} \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot x_{s'}^n$ can be simplified

Until $\max_{s \in S} |x_s^{n+1} - x_s^n| < \varepsilon$ for some tolerance factor $\varepsilon > 0$



Compute: $Pr_{s_i}^{min}(F\ a)$

$S^{yes} = \{x_2\}$, $S^{no} = \{x_3\}$, $S^{?} = \{x_0, x_1\}$

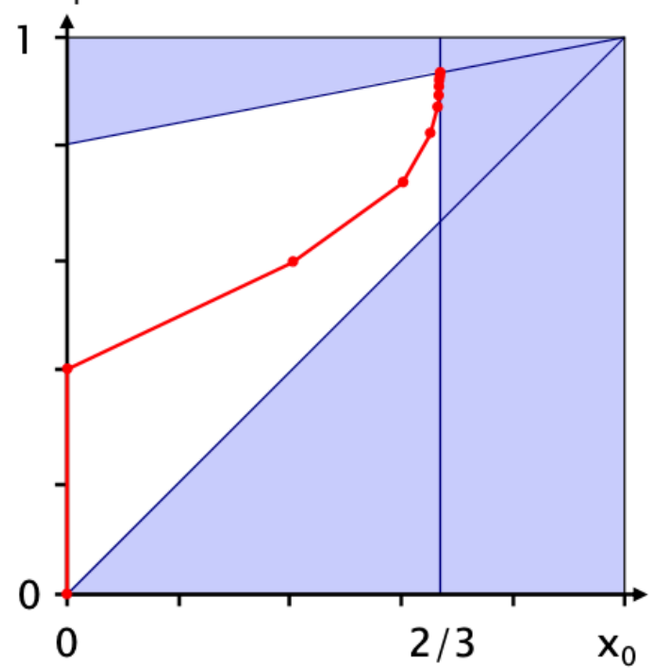
$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$

n=0: $[\ 0, 0, 1, 0\ ]$

n=1: $[\ \min(0, 0.25 \cdot 0 + 0.5),\ 0.1 \cdot 0 + 0.5 \cdot 0 + 0.4, 1, 0\ ]$
$= [\ 0, 0.4, 1, 0\ ]$

n=2: $[\ \min(0.4, 0.25 \cdot 0 + 0.5),\ 0.1 \cdot 0 + 0.5 \cdot 0.4 + 0.4, 1, 0\ ]$
$= [\ 0.4, 0.6, 1, 0\ ]$

n=3: ...



$[\ x_0^{(n)}, x_1^{(n)}, x_2^{(n)}, x_3^{(n)}\ ]$

n=0: $[\ 0.000000, 0.000000, 1, 0\ ]$

n=1: $[\ 0.000000, 0.400000, 1, 0\ ]$

n=2: $[\ 0.400000, 0.600000, 1, 0\ ]$

n=3: $[\ 0.600000, 0.740000, 1, 0\ ]$

n=4: $[\ 0.650000, 0.830000, 1, 0\ ]$

n=5: $[\ 0.662500, 0.880000, 1, 0\ ]$

n=6: $[\ 0.665625, 0.906250, 1, 0\ ]$

n=7: $[\ 0.666406, 0.919688, 1, 0\ ]$

n=8: $[\ 0.666602, 0.926484, 1, 0\ ]$

n=9: $[\ 0.666650, 0.929902, 1, 0\ ]$

...

n=20: $[\ 0.666667, 0.933332, 1, 0\ ]$

n=21: $[\ 0.666667, 0.933332, 1, 0\ ]$

$\approx [\ 2/3, 14/15, 1, 0\ ]$

[taken from Marta Kwiatowka's lecture at GSSI 2015 - http://www.prismmodelchecker.org/courses/gran-sasso15/]

Computing $Pmin_s(\Diamond B)$ with policy/strategy iteration

Recall our goal

can be enlarged

$Pmin_s(\Diamond B) = 1$ if $s \in B$

$Pmin_s(\Diamond B) = 0$ *for some strategy* $s \models \neg \exists \Diamond B$

$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\}$ otherwise

Take any strategy σ

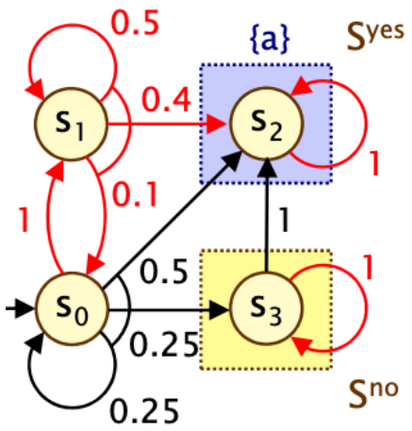
Try to improve the strategy by changing just one choice per state (all or one at a time).

I.e. if the below condition holds, then update the strategy by setting σ(s) = a

$$Pr^{\sigma}_s(\Diamond B) > \min_{a \in Act} \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pr^{\sigma}_{s'}(\Diamond B)$$

can be simplified

Termination criteria based on tolerance or fixpoint reached (no change in strategy)



Arbitrary strategy σ:

Compute: $\underline{Pr}^{\sigma}(F\ a)$

Let $x_i = Pr_{s_i}^{\sigma}(F\ a)$

$x_2=1$, $x_3=0$ and:

- $x_0 = x_1$
- $x_1 = 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4$

Solution:

$\underline{Pr}^{\sigma}(F\ a) = [\ 1, 1, 1, 0\ ]$

Refine σ in state s_0:

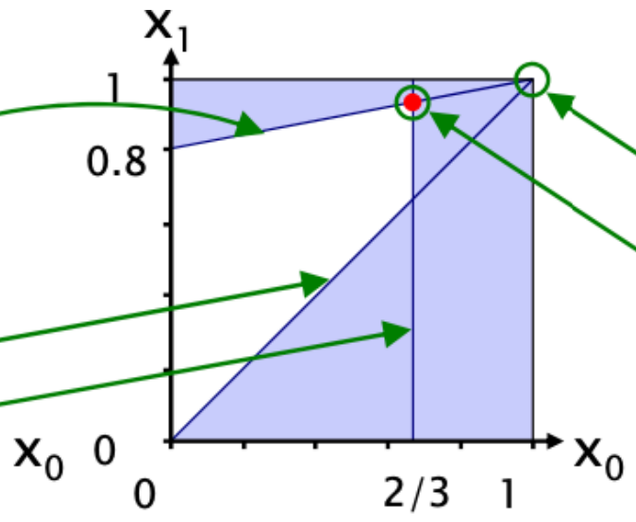
min{1(1), 0.5(1)+0.25(0)+0.25(1)}

= min{1, 0.75} = 0.75

$x_1 = 0.2 \cdot x_0 + 0.8$

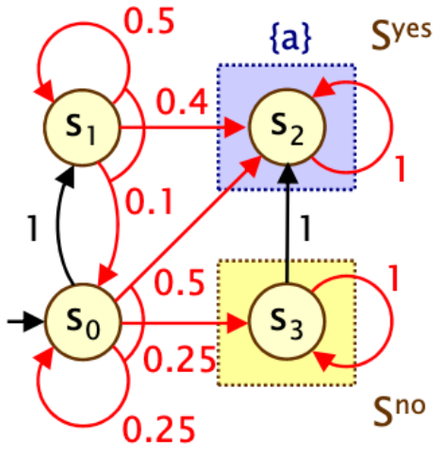
$x_0 = x_1$

$x_0 = 2/3$



σ

σ'



[taken from Marta Kwiatowka's lecture at GSSI 2015 - http://www.prismmodelchecker.org/courses/gran-sasso15/]

Synthesis of optimal strategies

Not only we can compute min/max probabilities, we can also get the corresponding strategies!

Just optimise locally, choose an action a that minimises/maximizes the probability.

$\sigma(s) = a$, where a is any action in A

$$Pmin_s(\Diamond B) = 1 \qquad \text{if } s \in B$$

$$Pmin_s(\Diamond B) = 0 \qquad \textit{for some strategy } s \models \neg \exists \Diamond B$$

$$Pmin_s(\Diamond B) = min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B)\} \qquad \text{otherwise}$$

$\sigma(s) = \arg\min(\ldots)$

Synthesis of optimal strategies via OMT

As an OMT

$$Pmin_s(\Diamond B) = 1 \qquad \text{if } s \in B$$

$$Pmin_s(\Diamond B) = 0 \qquad \textit{for some strategy } s \models \neg \exists \diamond B$$

$$Pmin_s(\Diamond B) = \sum_{a \in Act} y_{s,a} \cdot \sum_{s' \in S} \mathbf{P}(s,a,s') \cdot Pmin_{s'}(\Diamond B) \qquad \text{otherwise}$$

Where $y_{s,a}$ are indicator variables for choosing action a in state s:

$$\bigwedge_{s \in A, a \in Act} y_{s,a} = 1 \vee y_{s,a} = 0$$ either action a is chosen (1) or not (0)

$$\bigwedge_{s \in A} \sum_{a \in Act} y_{s,a} = 1$$ exactly one action can be chosen at each state

Solve equations that minimise $\sum_{s \in S} Pmin_s(\Diamond B)$

Strategy synthesis via OMT

```
# Strategy synthesis via OMT
solver2 = Optimize()
solver2.add(
  x0 == y_0_0 * x1 + y_0_1 * Q(2,3) ,
  x1 == Q(2,10)*x0 + Q(8,10) ,
  Or(y_0_0 == 0 , y_0_0 == 1),
  Or(y_0_1 == 0 , y_0_1 == 1),
  y_0_0 + y_0_1 == 1
)
solver2.minimize(x0+x1)
```

Do we need richer strategies?

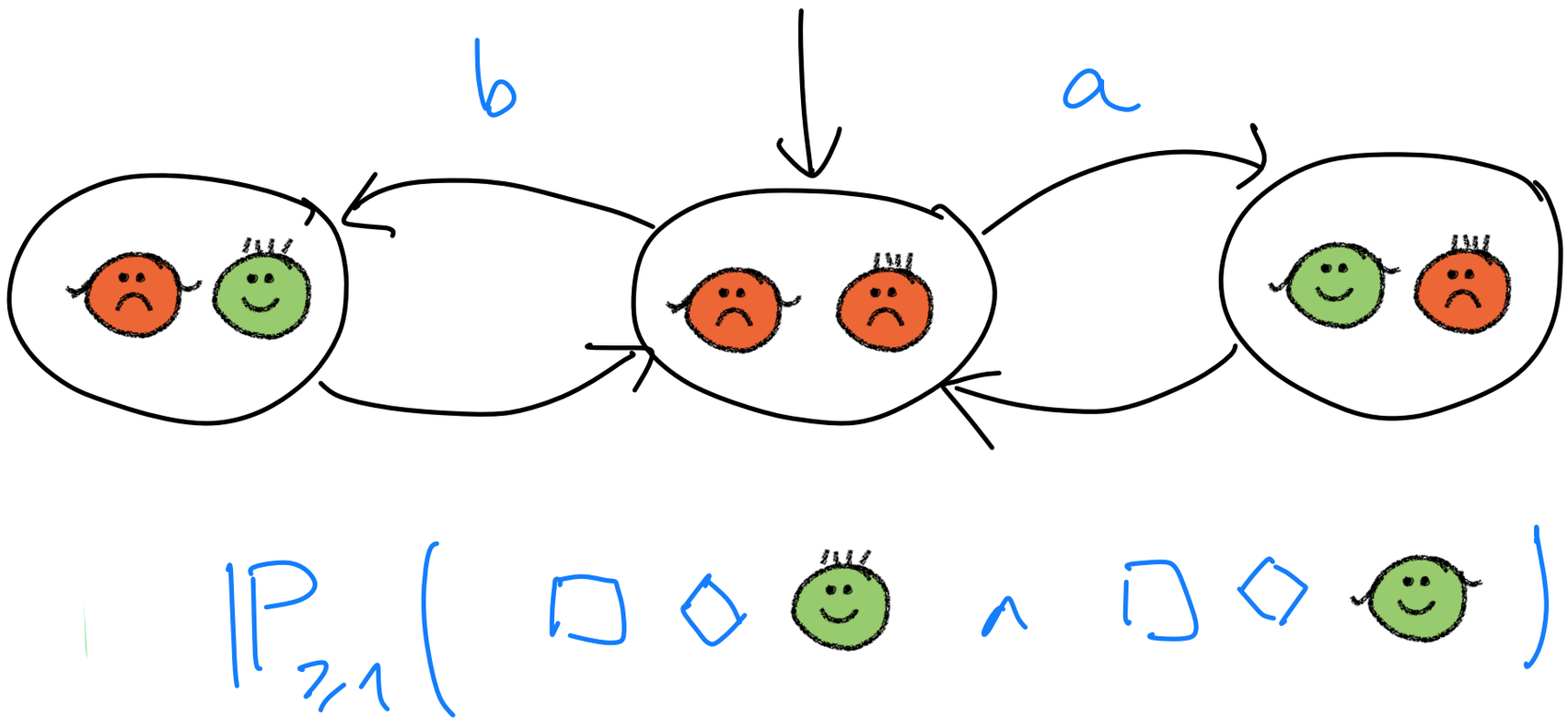
Memory-less, deterministic schedulers are enough for PCTL.

Are there cases where they are not enough?

Do we need richer strategies?

Memory-less, deterministic schedulers are enough for PCTL.

Are there cases where they are not enough?



No memory-less, deterministic strategy would satisfy the property.

What about using memory-dependent or randomised strategies?

PCTL for MDPs

Pretty much like PCTL for DTMCs.

Same syntax and a different semantics for the P operator

Semantics for DTMCs

$$s \models \mathbb{P}_J(\psi) \quad \text{iff} \quad Pr(\{\pi \in Paths(s) \mid \pi \models \psi\}) \in J$$

Semantics for MPDs

$$s \models \mathbb{P}_J(\psi) \quad \text{iff} \quad Pr(\{\pi \in Paths^{\sigma}(s) \mid \pi \models \psi\}) \in J \quad \text{for all strategies } \sigma$$

How do we determine this?

PCTL for MDPs

$s \models \mathbb{P}_J(\psi)$ iff $Pr(\{\pi \in Paths^{\sigma}(s) \mid \pi \models \psi\}) \in J$ for all strategies σ

IDEA: for intervals J of the form <p or <=p rely on $Pmax_s(\psi)$

$s \models \mathbb{P}_{\leq q}(\psi)$ iff …

for intervals J of the form >p or >=p rely on $Pmin_s(\psi)$

$s \models \mathbb{P}_{\geq r}(\psi)$ iff …

other intervals can be decomposed

$s \models \mathbb{P}_{[r..q]}(\psi)$ iff …

PCTL for MDPs

$s \models \mathbb{P}_J(\psi)$ iff $Pr(\{\pi \in Paths^{\sigma}(s) \mid \pi \models \psi\}) \in J$ for all strategies σ

IDEA: for intervals J of the form <p or <=p rely on $Pmax_s(\psi)$

$$s \models \mathbb{P}_{\leq q}(\psi) \quad \text{iff} \quad Pmax_s(\psi) \leq q$$

for intervals J of the form >p or >=p rely on $Pmin_s(\psi)$

$$s \models \mathbb{P}_{\geq r}(\psi) \quad \text{iff} \quad Pmin_s(\psi) \geq r$$

other intervals can be decomposed

$$s \models \mathbb{P}_{[r..q]}(\psi) \text{ iff } Pmin_s(\psi) \geq r \text{ and } Pmax_s(\psi) \leq q$$

Key points of this lecture

We can have the best of TSs and DTMCs: TS + DTMC = MDP

Definition of Markov Decision Process

Definition of paths and strategies in MDPs

Computation of max and min probabilities for reachability properties, the basis of model checking MDPs

MDPs are 1-player games…we can can have multiple players!

Turn-based Stochastic Games (TSGs) as one possible extension.

PRISM Games supports PCTL model checking for TSGs (and more).

(In the next lecture :)

BONUS: Expected rewards

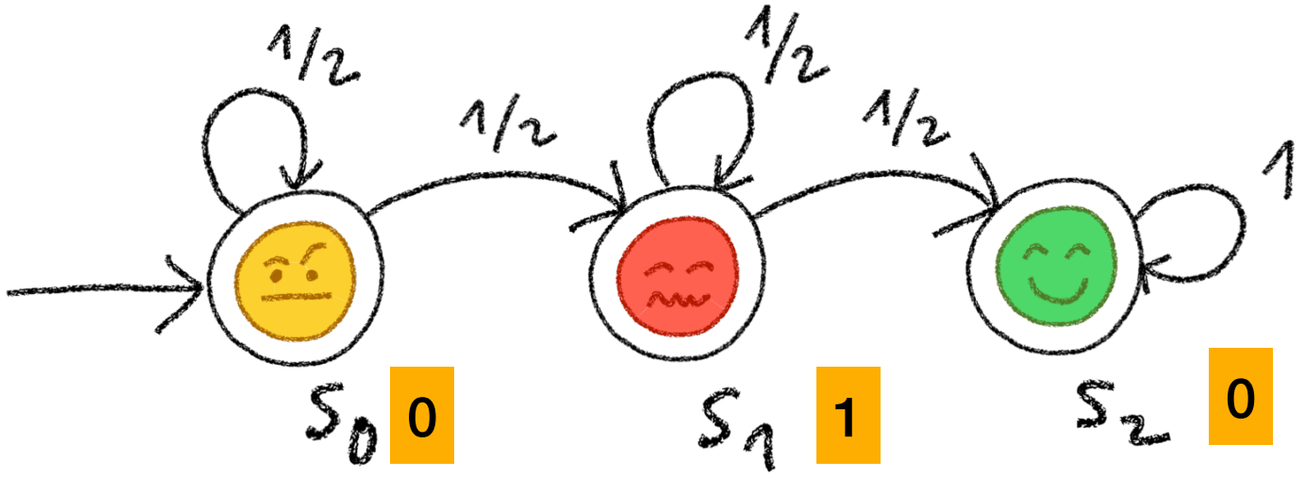
MDPs can be enriched with rewards/costs
(just like we saw for DTMCs)

DTMCs with rewards/costs

Definition 10.69. Markov Reward Model (MRM)

A *Markov reward model* (MRM) is a tuple $(\mathcal{M}, rew)$ with $\mathcal{M}$ a Markov chain with state space S and $rew : S \to \mathbb{N}$ a reward function that assigns to each state $s \in S$ a non-negative integer reward $rew(s)$. ■

Example with rew function counting “sick days”



i.e. rew(s0) = 0, rew(s1) = 1, rew(s2) = 0

Minimal Expected Reward $MinExpRew_s(\Diamond B)$

Recursive definition:

$$MinExpRew(s \models \Diamond B) = rew(s) \qquad \text{if } s \in B$$

$$MinExpRew(s \models \Diamond B) = +\infty \text{ or } 0 \qquad \text{if } s \models \neg\exists\Diamond B$$

$$MinExpRew(s \models \Diamond B) = rew(s) + min_{a \in Act}\{ \sum_{s' \in S} \mathbf{P}(s, a, s') \cdot MinExpRew(s' \models \Diamond B)\} \qquad \text{otherwise}$$

For the sake of simplicity, let's consider models where
and that all states $Pr_s(\Diamond B) = 1$

Expected reward $ExpRew_s(\Diamond B)$

Formally…

cumulative reward of path fragments until B,

the *cumulative reward* for a finite path $\widehat{\pi} = s_0\, s_1 \ldots s_n$ is defined by

$$rew(\widehat{\pi}) = rew(s_0) + rew(s_1) + \ldots + rew(s_{n-1}).$$

$$ExpRew(s \models \Diamond B) = \sum_{\widehat{\pi}} \mathbf{P}(\widehat{\pi}) \cdot rew(\widehat{\pi})$$

where $\widehat{\pi}$ ranges over all finite paths $s_0 \ldots, s_n$ with $s_n \in B$, $s_0 = s$ and $s_0, \ldots, s_{n-1} \notin B$.

Recursive definition:

$$ExpRew(s \models \Diamond B) = rew(s) \qquad \text{if } s \in B$$

$$ExpRew(s \models \Diamond B) = +\infty \text{ or } 0 \qquad \text{if } s \models \neg \exists \Diamond B$$

$$ExpRew(s \models \Diamond B) = rew(s) + \sum_{s' \in S} \mathbf{P}(s, s') \cdot ExpRew(s' \models \Diamond B) \qquad \text{otherwise}$$

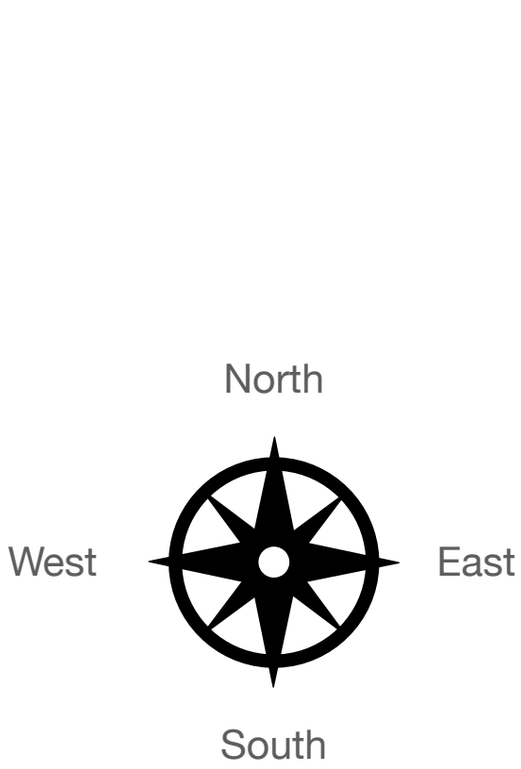
For the sake of simplicity, let's consider models where this does not happen and that all states $Pr_s(\Diamond B) = 1$

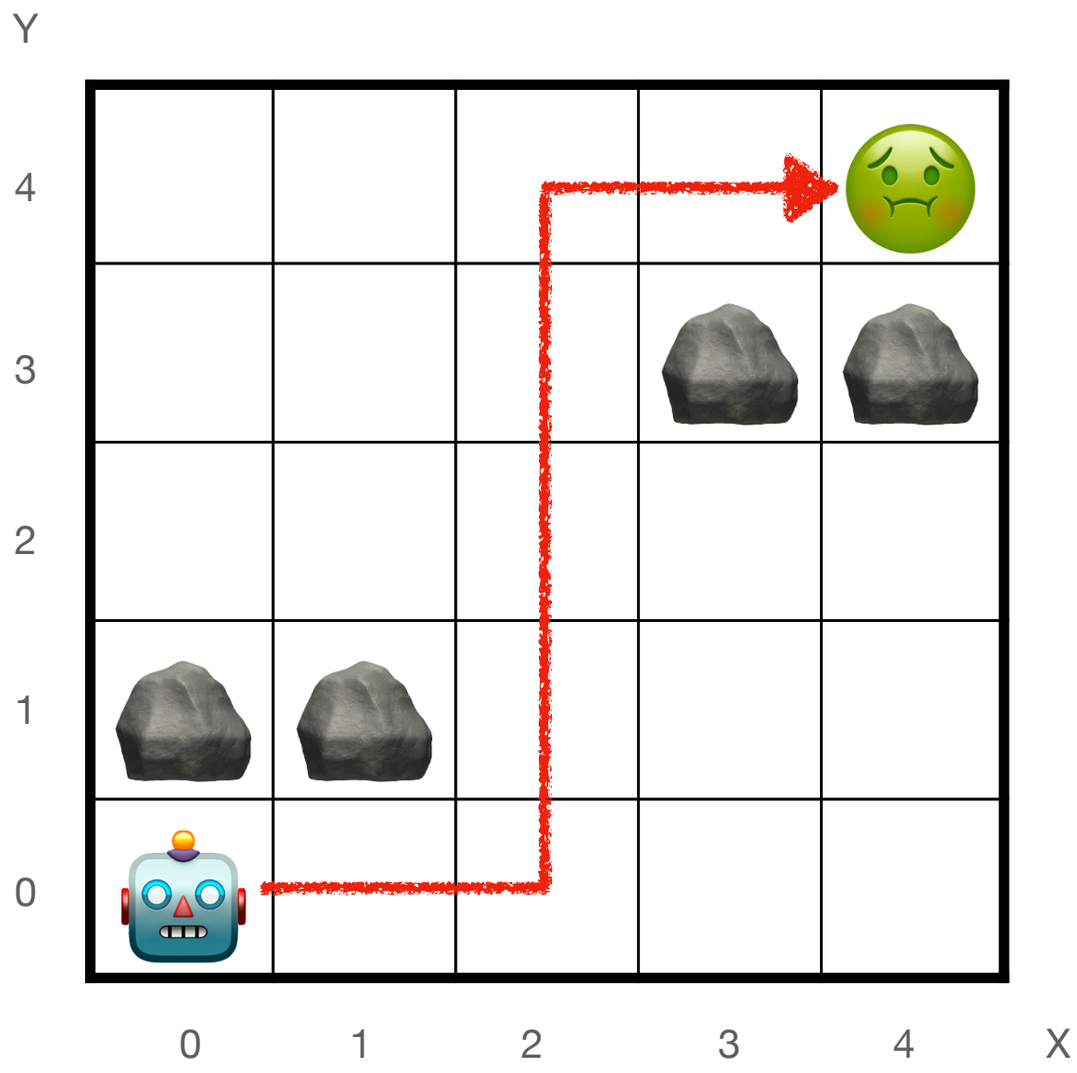
Robot planning

A robot is deployed on a maze and has to reach a goal location, where a poor victim is waiting to be rescued. The goal is to rescue the victim as fast as possible

The maze is modelled as an NxN grid. The robot can move in four possible locations (east, west, north and south). The maze is surround by walls and may contain obstacles.

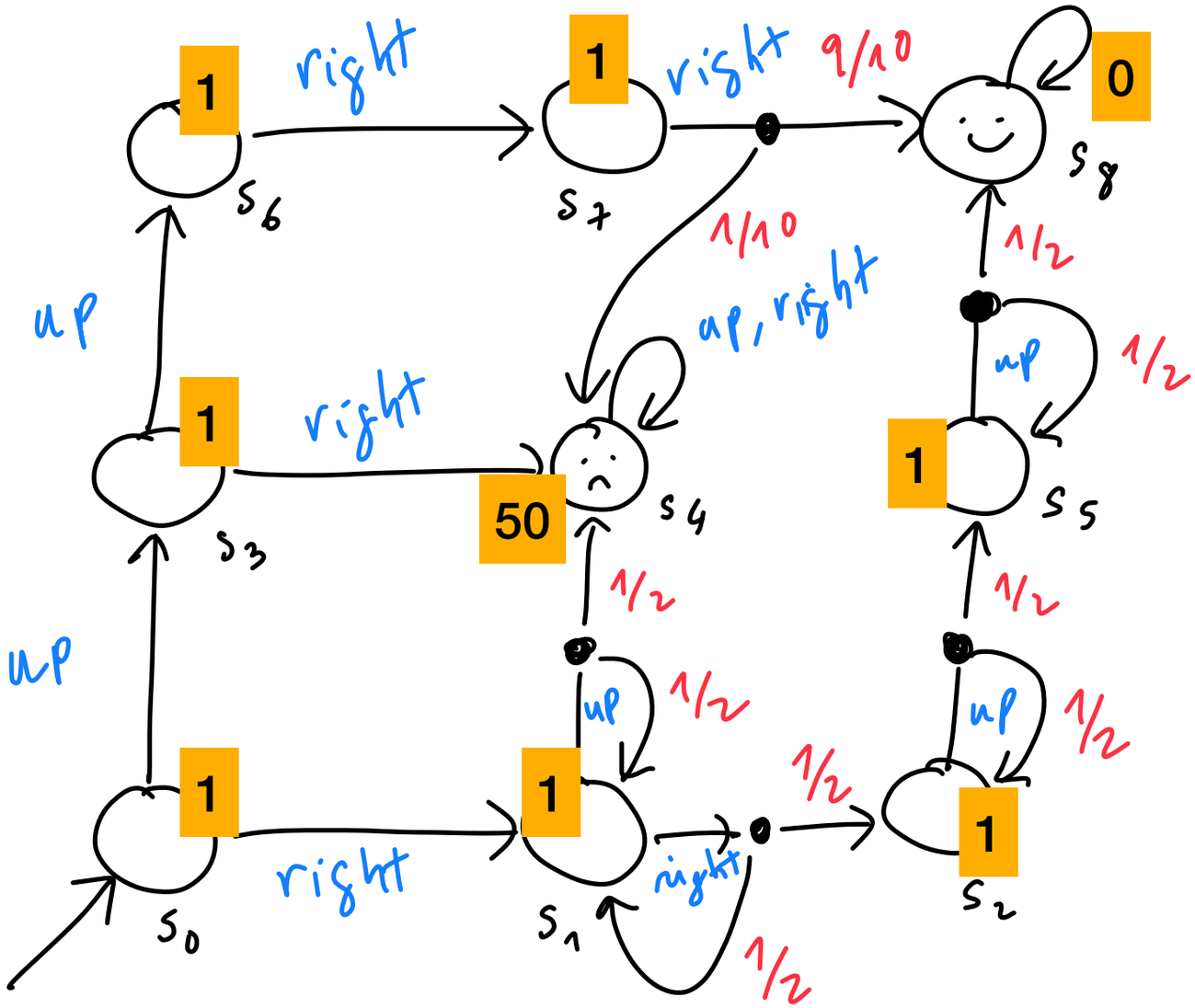
The floor is slippery (probabilistic behaviour).





What is the optimal strategy? What is the minimal expected number of “steps” to reach the victim?

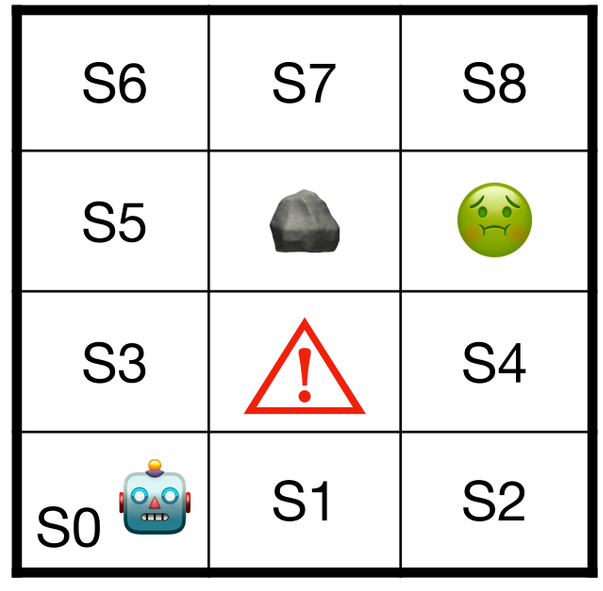
Example



Minimal cost/reward to end up in one of the faces? Optimal strategy for said minimal cost?

Exercises

Exercise 10.1



Our robot (🤖) wants to save a sick person (🤢), with highest probability.

The initial state is position s0.

Position ⚠ is a trap, if the robot enters the trap it cannot escape.

The robot can move a round a maze by moving up/down/left/right. Not all directions are always available:

- The walls (thick lines) and the rock (🪨) cannot be reached.
- The robot will never try to go back. For example in S3, the robot can try to go up or left, but not down.

The floor is slippery:

- If the robot tries to go in one direction, the probability success is 2/3.
- With probability 1/3 the movement fails:
  - If there are cells adjacent to the current one and different from the destination, the robot will end up in one of those adjacent cells with equal probability. For example, When moving right from s0, the robot will end in s1 with prob. 2/3 and in S3 with prob. 1/3
  - Otherwise, the robot stays in the same cell. For example, when moving right from s7, the robot ends with prob. 2/3 in S8 and stays in S7 with prob. 1/3.

Your task is to:

- Model the problem as an MDP and determine the strategy that maximises the probability to rescue the sick person.
- Write down the problem as a linear program and use a solver (e.g. Z3) to solve the problem.
- Use PRISM to solve the problem.
- Compare the solutions. Do they match?

Exercise 10.3

Model the Monty Hall problem as an MDP and determine the probability the best strategy (keep/change door) and its probability.

You can read about the Monty Hall problem here:

https://en.wikipedia.org/wiki/Monty_Hall_problem

Your task is to:

1. Model the problem as an MDP and determine the strategy that maximises the probability to win the car.
2. Write down the problem as a linear program and use a solver (e.g. Z3) to solve the problem.
3. Use PRISM to solve the problem.
4. Compare the results.

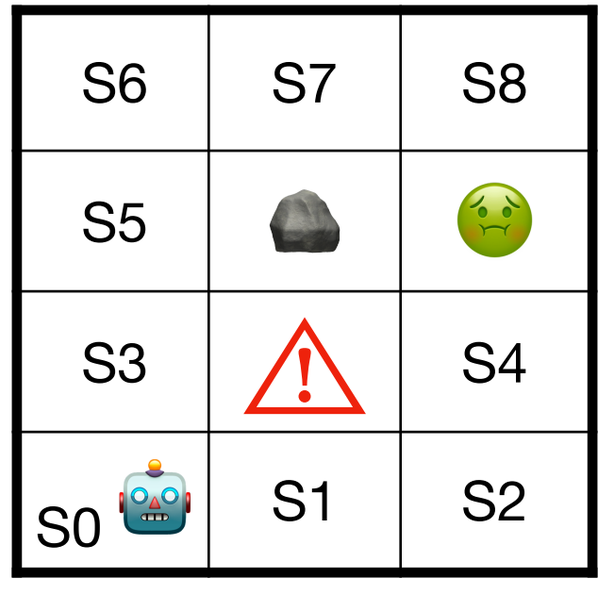
Exercise 10.4

Consider the double-or-nothing cookie game seen during the lecture.

Model it as an MDP in PRISM and in Z3. Try to reproduce the results we obtained in the lecture.

Solutions

Exercise 10.1



Our robot (🤖) wants to save a sick person (🤢), with highest probability.

The initial state is position s0.

Position ⚠ is a trap, if the robot enters the trap it cannot escape.

The robot can move a round a maze by moving up/down/left/right. Not all directions are always available:

- The walls (thick lines) and the rock (🪨) cannot be reached.
- The robot will never try to go back. For example in S3, the robot can try to go up or left, but not down.

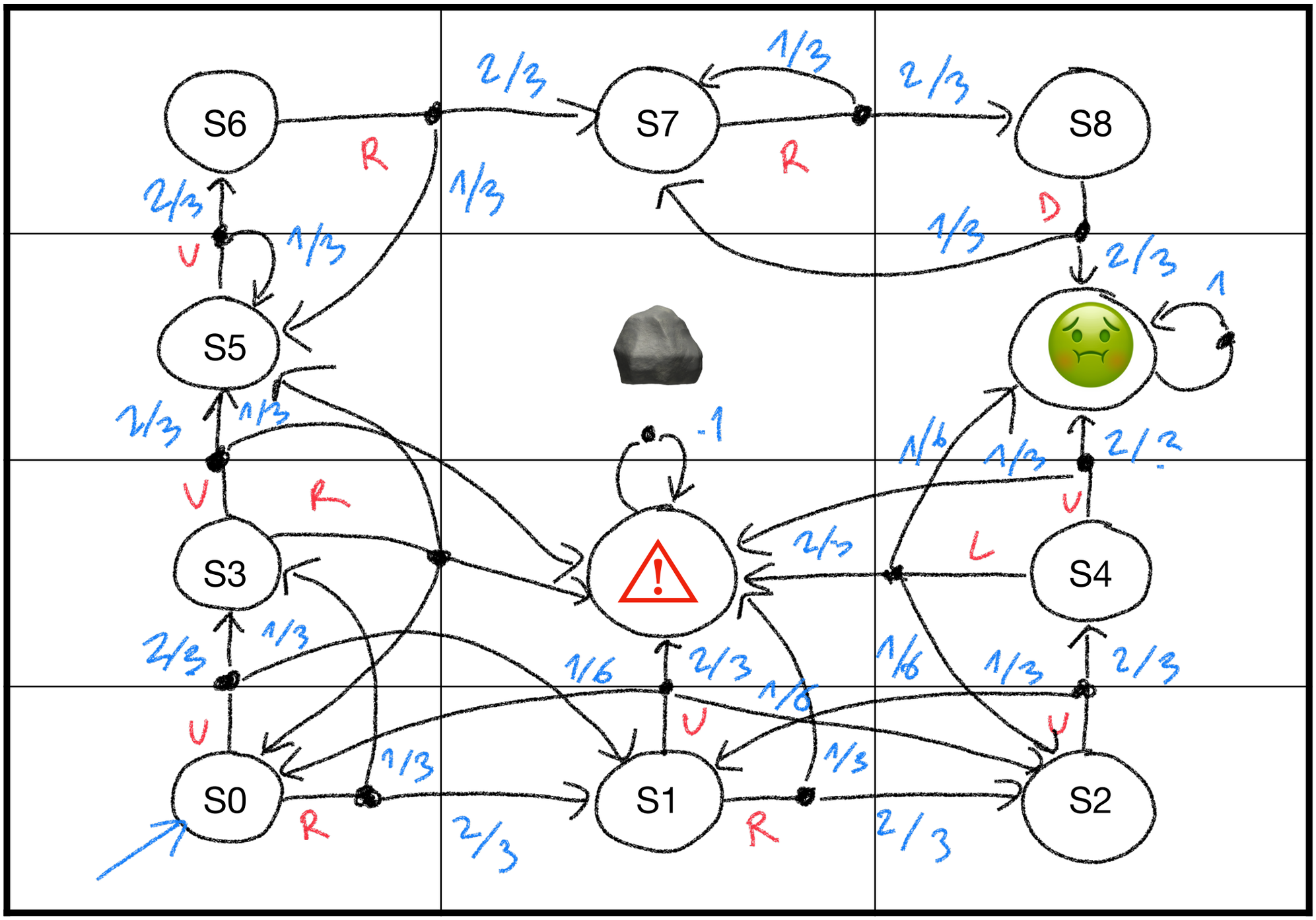
The floor is slippery:

- If the robot tries to go in one direction, the probability success is 2/3.
- With probability 1/3 the movement fails:
  - If there are cells adjacent to the current one and different from the destination, the robot will end up in one of those adjacent cells with equal probability. For example, When moving right from s0, the robot will end in s1 with prob. 2/3 and in S3 with prob. 1/3
  - Otherwise, the robot stays in the same cell. For example, when moving right from s7, the robot ends with prob. 2/3 in S8 and stays in S7 with prob. 1/3.

Your task is to:

1. Model the problem as an MDP and determine the strategy that maximises the probability to rescue the sick person.
2. Write down the problem as a linear program and use a solver (e.g. Z3) to solve the problem.
3. Use PRISM to solve the problem.
4. Compare the results.

Solution to Exercise 10.1



A sketch of how the MDP would be (omitting actions that take the robot back).

The problem as a linear program in Z3

```
# Equations for solving Exercise 10.1
robot = Optimize()
robot.add(
  x0 >= Q(2,3)*x3 + Q(1,3)*x1 ,
  x0 >= Q(2,3)*x1 + Q(1,3)*x3 ,
  x1 >= Q(2,3)*x2 + Q(1,3)*xTRAP,
  x1 >= Q(2,3)*xTRAP + Q(1,6)*x0 + Q(1,6)*x2 ,
  x2 >= Q(2,3)*x4 + Q(1,3)*x1 ,
  x3 >= Q(2,3)*x5 + Q(1,3)*xTRAP ,
  x3 >= Q(2,3)*xTRAP + Q(1,6)*x0 + Q(1,6)*x5 ,
  x4 >= Q(2,3)*xGOAL + Q(1,3)*xTRAP ,
  x4 >= Q(2,3)*xTRAP + Q(1,6)*x2 + Q(1,6)*xGOAL ,
  x5 >= Q(2,3)*x6 + Q(1,3)*x5 ,
  x6 >= Q(2,3)*x7 + Q(1,3)*x5 ,
  x7 >= Q(2,3)*x8 + Q(1,3)*x7 ,
  x8 >= Q(2,3)*xGOAL + Q(1,3)*x7 ,
  xTRAP == 0 ,
  xGOAL == 1
)

robot.minimize(x0+x1+x2+x3+x4+x5+x6+x7+x8+xGOAL+xTRAP)

solver = robot
if solver.check() == sat:
    m = solver.model()
    print("Hoorray! Here is a possible solution:")
    print(m)
```

The problem as an MDP in PRISM

```
mdp

const GOAL=9;
const TRAP=10;
const INITIAL;

module ROBOT

state : [0..10] init INITIAL;

[u] state=0 -> 2/3:(state'=3) + 1/3:(state'=1);
[r] state=0 -> 2/3:(state'=1) + 1/3:(state'=3);
[r] state=1 -> 2/3:(state'=2) + 1/3:(state'=TRAP);
[u] state=1 -> 2/3:(state'=TRAP) + 1/6:(state'=0) + 1/6:(state'=2);
[u] state=2 -> 2/3:(state'=4) + 1/3:(state'=1);
[u] state=3 -> 2/3:(state'=5) + 1/3:(state'=TRAP);
[r] state=3 -> 2/3:(state'=TRAP) + 1/6:(state'=0) + 1/6:(state'=5);
[u] state=4 -> 2/3:(state'=GOAL) + 1/3:(state'=TRAP);
[l] state=4 -> 2/3:(state'=TRAP) + 1/6:(state'=2) + 1/6:(state'=GOAL);
[u] state=5 -> 2/3:(state'=6) + 1/3:(state'=5);
[r] state=6 -> 2/3:(state'=7) + 1/3:(state'=5);
[r] state=7 -> 2/3:(state'=8) + 1/3:(state'=7);
[d] state=8 -> 2/3:(state'=GOAL) + 1/3:(state'=7);
[] state=GOAL -> 1:(state'=GOAL);
[] state=TRAP -> 1:(state'=TRAP);

endmodule
```

Both tools agree that the maximum probability of reaching the goal is 4/7.

The strategy is S0:u, S3:u, S5:u, S6:r, S7:r, S8:d (the rest of the states are irrelevant)

Exercise 10.2

Consider again the scenario of Exercise 10.1.

Use the notion of cost/reward to determine the minimal expected number of steps to reach the goal.

Use PRISM and Z3 to compute the result. Do the results agree? Do they make sense to you?

SOLUTION: Ask teacher.

Exercise 10.3

Model the Monty Hall problem as an MDP and determine the probability the best strategy (keep/change door) and its probability.

You can read about the Monty Hall problem here:

https://en.wikipedia.org/wiki/Monty_Hall_problem

Your task is to:

1. Model the problem as an MDP and determine the strategy that maximises the probability to win the car.
2. Write down the problem as a linear program and use a solver (e.g. Z3) to solve the problem.
3. Use PRISM to solve the problem.
4. Compare the results.

Solution to Exercise 10.3

The best strategy is to switch door. You will win with probability 2/3.

One possible solution (in PRISM)

```
mdp

module MontyHall

state : [0..5] init 0;      // steps of the game
switch : [0..1] init 0;     // strategy for the contestant
car : [0..3] init 0;        // door where the car hides
open1 : bool init false;    // status of the doors (false = closed, true = open)
open2 : bool init false;
open3 : bool init false;
chosenDoor : [0..3] init 0; // chosen door

// choose strategy
[keep]   (state=0) -> 1.0 : (switch'=0)&(state'=1);
[switch] (state=0) -> 1.0 : (switch'=1)&(state'=1);

// choose door probabilistially
[] (state =1) -> 1/3:(chosenDoor'=1)&(state'=2)
               + 1/3:(chosenDoor'=2)&(state'=2)
               + 1/3:(chosenDoor'=3)&(state'=2);

// randomize where the car hides
[] (state =2) -> 1/3:(car'=1)&(state'=3)
               + 1/3:(car'=2)&(state'=3)
               + 1/3:(car'=3)&(state'=3);

// Monty Hall opens a door (one of the ones hiding a goat)
// It does not really matter which one
[] (state=3) & (chosenDoor!=1) & (car!=1) -> 1.0:(open1'=true) & (state'=4);
[] (state=3) & (chosenDoor!=2) & (car!=2) -> 1.0:(open2'=true) & (state'=4);
[] (state=3) & (chosenDoor!=3) & (car!=3) -> 1.0:(open3'=true) & (state'=4);

// Player executes his/her strategy
[] (state=4)&(switch=1)&(chosenDoor=1)&(!open2) -> 1.0:(state'=5)&(chosenDoor'=2);
[] (state=4)&(switch=1)&(chosenDoor=1)&(!open3) -> 1.0:(state'=5)&(chosenDoor'=3);
[] (state=4)&(switch=1)&(chosenDoor=2)&(!open1) -> 1.0:(state'=5)&(chosenDoor'=1);
[] (state=4)&(switch=1)&(chosenDoor=2)&(!open3) -> 1.0:(state'=5)&(chosenDoor'=3);
[] (state=4)&(switch=1)&(chosenDoor=3)&(!open1) -> 1.0:(state'=5)&(chosenDoor'=1);
[] (state=4)&(switch=1)&(chosenDoor=3)&(!open2) -> 1.0:(state'=5)&(chosenDoor'=2);
[] (state=4)&(switch=0) -> 1.0:(state'=5);

[] (state=5) -> (state'=5);

endmodule
```

Exercise 10.4

Consider the double-or-nothing cookie game seen during the lecture.

Model it as an MDP in PRISM and in Z3. Try to reproduce the results we obtained in the lecture.

Solution: see board :)