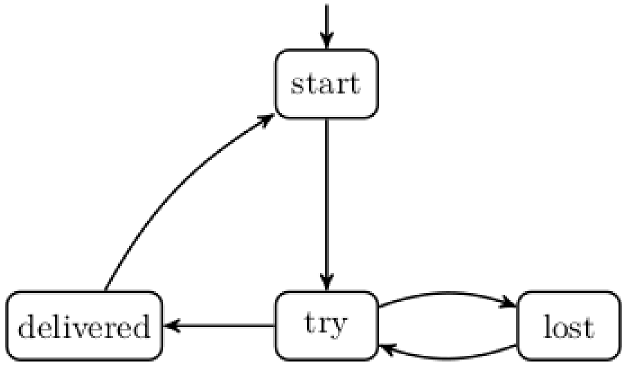
Probabilistic Model Checking

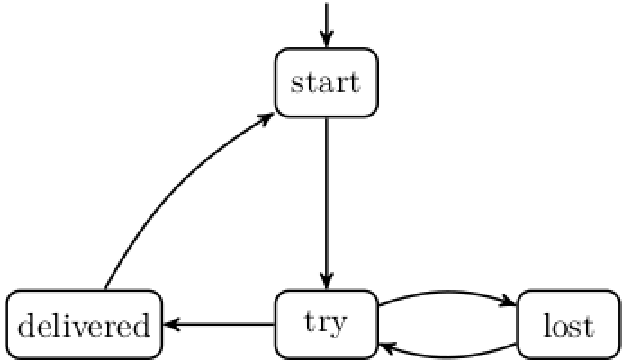
Stefan Pranger

03. 06. 2024

Communication Protocol



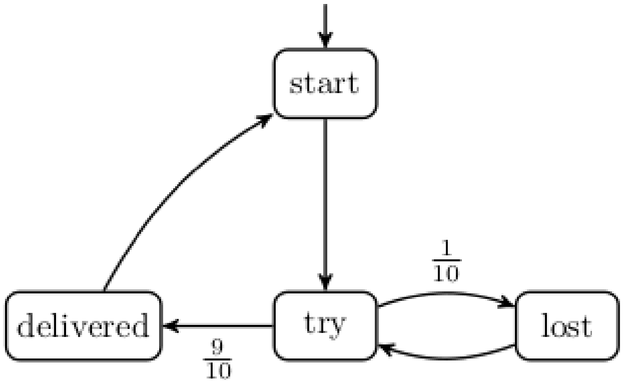
Communication Protocol



But $\mathcal{M}, start \models \exists \mathbf{G}\ \neg delivered$?

or $\mathcal{M}, start \models \forall \mathbf{F}\ delivered$?

Communication Protocol



But $\mathcal{M}, \mathit{start} \models \exists\mathbf{G}\ \neg \mathit{delivered}$?

or $\mathcal{M}, \mathit{start} \models \forall\mathbf{F}\ \mathit{delivered}$?

Does not make sense with probabilities! → We *need* new descriptions for properties.

We have different models.

Markov Chains

Markov Chain $\mathcal{M} = (S, \mathbb{P}, s_0, AP, L)$

- S a set of states and initial state s_0,
- $\mathbb{P} : S \times S \to [0,1]$, s.t.

$$\sum_{s' \in S} \mathbb{P}(s, s') = 1 \; \forall s \in S$$

- AP set of atomic propositions and $L : S \to 2^{AP}$ a labelling function.

What properties are we interested in?

What properties are we interested in?

- What is the probability to eventually send the message (within n steps)?

What properties are we interested in?

- What is the probability to eventually send the message (within n steps)?
- What is the probability to reach the destination without every running into an unsafe area?

What properties are we interested in?

- What is the probability to eventually send the message (within n steps)?
- What is the probability to reach the destination without every running into an unsafe area?
- What is the probability to send 6 messages successfully and only failing a maximum amount of 15 times?

But first... How do we describe models?

But first... How do we describe models?

- Describe states through variables:
  - $x \in [0, 20], y \in [0, 20], velocity \in [0, 1], \ldots$

But first... How do we describe models?

- Describe states through variables:
  - $x \in [0, 20], y \in [0, 20], velocity \in [0, 1], \dots$
  - $processor\_one\_idle, processor\_two\_idle, \dots$

But first... How do we describe models?

- Describe states through variables:
  - $x \in [0, 20], y \in [0, 20], velocity \in [0, 1], \ldots$
  - $processor\_one\_idle, processor\_two\_idle, \ldots$
  - $agent\_is\_on\_slippery, \ldots$
  - ...

But first... How do we describe models?

- Describe states through variables:
  - $x \in [0, 20], y \in [0, 20], velocity \in [0, 1], \ldots$
  - $processor\_one\_idle, processor\_two\_idle, \ldots$
  - $agent\_is\_on\_slippery, \ldots$
  - ...
- For each possible state we describe the possible variable updates:
  - If $x > 10$ & $y < 10$ & $agent\_is\_on\_slippery$ then the agent moves to one of its adjacent cells each with probability $1/4$.
  - If $processor\_one\_idle$ & $processor\_two\_idle$ then the process will be processed by processor one or two.

But first... How do we describe models?

- Describe states through variables:
  - $x \in [0, 20], y \in [0, 20], velocity \in [0, 1], \ldots$
  - $processor\_one\_idle, processor\_two\_idle, \ldots$
  - $agent\_is\_on\_slippery, \ldots$
  - ...
- For each possible state we describe the possible variable updates:
  - If $x > 10$ & $y < 10$ & $agent\_is\_on\_slippery$ then the agent moves to one of its adjacent cells each with probability $1/4$.
  - If $processor\_one\_idle$ & $processor\_two\_idle$ then the process will be processed by processor one or two.
  - If $processor\_one\_idle$ & $processor\_two\_idle$ then we can **decide** to use processor one or two.

The PRISM Modelling Language

- Modules: Group associated behaviour

```
module processor1 ... endmodule
module processor2 ... endmodule
```

The PRISM Modelling Language

- Modules: Group associated behaviour

```
module processor1 ... endmodule
module processor2 ... endmodule
```

- Variables (Constants) : Either bool or integer (or double):

```
x : [0..2] init 0;
b : bool init false;
global temperature : [0..100] init 32;
const double pi = 3.14;
```

The PRISM Modelling Language

- Modules: Group associated behaviour

```
module processor1 ... endmodule
module processor2 ... endmodule
```

- Variables (Constants) : Either bool or integer (or double):

```
x : [0..2] init 0;
b : bool init false;
global temperature : [0..100] init 32;
const double pi = 3.14;
```

  - Updating variables of a module is restricted to each module, e.g. private access.

The PRISM Modelling Language

- Modules: Group associated behaviour

```
module processor1 ... endmodule
module processor2 ... endmodule
```

- Variables (Constants) : Either bool or integer (or double):

```
x : [0..2] init 0;
b : bool init false;
global temperature : [0..100] init 32;
const double pi = 3.14;
```

  - Updating variables of a module is restricted to each module, e.g. private access.

- Commands:

```
[] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
[moveNorth] x<height -> 0.9: (x'=x+1) + 0.1: true;
```

The PRISM Modelling Language

- Modules: Group associated behaviour

```
module processor1 ... endmodule
module processor2 ... endmodule
```

- Variables (Constants) : Either bool or integer (or double):

```
x : [0..2] init 0;
b : bool init false;
global temperature : [0..100] init 32;
const double pi = 3.14;
```

  - Updating variables of a module is restricted to each module, e.g. private access.

- Commands:

```
[] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
[moveNorth] x<height -> 0.9: (x'=x+1) + 0.1: true;
```

  - We use it to describe the set of possible states and transitions between them.

The PRISM Modelling Language

- Formulas and Labels:

```
formula num_tokens = q1+q2+q3+q+q5;
formula crash = x1=x2 & y1=y2;
label "crashed" = crash
//[moveNorth] !crash & ... -> ...;
```

The PRISM Modelling Language

- Formulas and Labels:

```
formula num_tokens = q1+q2+q3+q+q5;
formula crash = x1=x2 & y1=y2;
label "crashed" = crash
//[moveNorth] !crash & ... -> ...;
```

- Turn-based behaviour:

```
[] move=0 & ... -> ... & (move'=1);
[] move=1 & ... -> ... & (move'=2);
etc.
```

The PRISM Modelling Language

- Formulas and Labels:

```
formula num_tokens = q1+q2+q3+q+q5;
formula crash = x1=x2 & y1=y2;
label "crashed" = crash
//[moveNorth] !crash & ... -> ...;
```

- Turn-based behaviour:

```
[] move=0 & ... -> ... & (move'=1);
[] move=1 & ... -> ... & (move'=2);
etc.
```

- Rewards:

```
rewards
x>0 & x<10 : 2*x;
x=10 : 100;
[a] true : x;
[b] true : 2*x;
endrewards
```

The PRISM Modelling Language

- Modelling language allows to design models in a code-like style
- Code de-duplication with formulas and labels

The PRISM Modelling Language

- Modelling language allows to design models in a code-like style
- Code de-duplication with formulas and labels

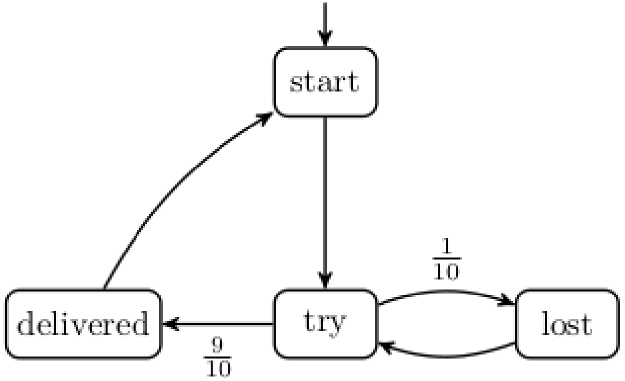
Other concepts include:

- Module Renaming

```
module Proc2 = Proc1 [ idle2=idle1, ... ] endmodule
```

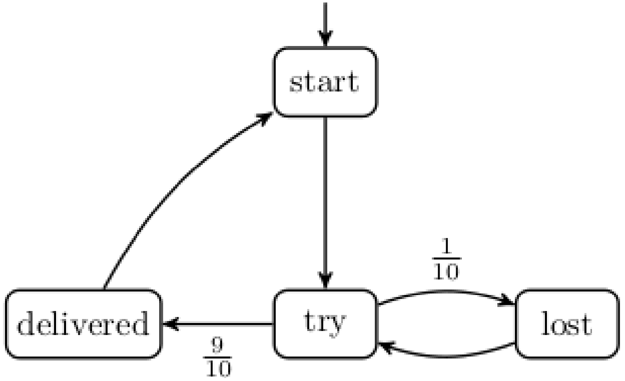
- Synchronization between modules
- Partially Observable Models
- Continuous-time Models
- Process Algebra Operators

Communication Protocol



```
dtmc

...
```

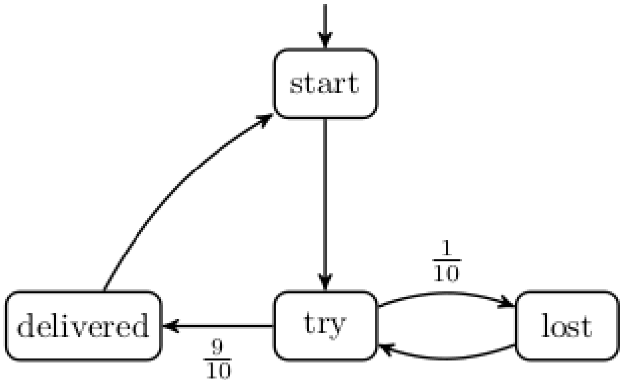
Communication Protocol



```
dtmc

...
```

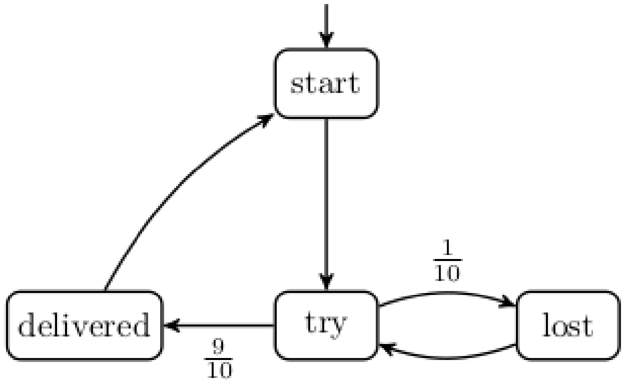
Live Coding!

Communication Protocol



```
dtmc

label "success" = delivered=1;
label "lost" = lost=1;

module msg_delivery
    start: [0..1] init 1;
    try: [0..1] init 0;
    lost: [0..1] init 0;
    delivered: [0..1] init 0;


    [] start=1      -> 1: (start'=0) & (try'=1);
    [] try=1        -> 0.1: (try'=0) & (lost'=1) +
                       0.9: (try'=0) & (delivered'=1);
    [] lost=1       -> 1: (lost'=0) & (try'=1);
    [] delivered=1  -> 1: (delivered'=0) & (start'=1);

endmodule
```

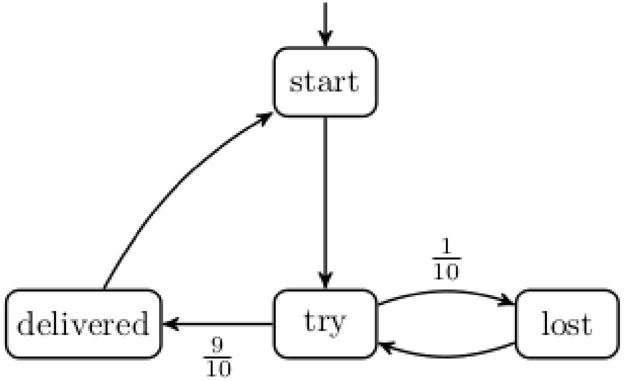
Communication Protocol with Counting



```
dtmc

label "success" = delivered=1;
label "lost" = lost=1;

...

module msg_delivery









endmodule
```

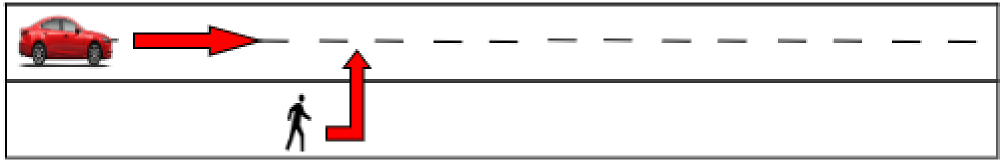
Live Coding!

Communication Protocol with Counting



```
dtmc

label "success" = delivered=1;
label "lost" = lost=1;

const int MAX_COUNT;

module msg_delivery
    start: [0..1] init 1;
    try: [0..1] init 0;
    lost: [0..1] init 0;
    delivered: [0..1] init 0;
    delivered_count: [0..MAX_COUNT] init 0;
    lost_count: [0..MAX_COUNT] init 0;


    [] start=1      -> 1: (start'=0) & (try'=1);
    [] try=1        -> 0.1: (try'=0) & (lost'=1) +
                       0.9: (try'=0) & (delivered'=1);
    [] lost=1       & lost_count<MAX_COUNT       -> 1: (lost'=0) & (try'=1) & (lost_count'=lost_count+1);
    [] delivered=1  & delivered_count<MAX_COUNT  -> 1: (delivered'=0) &
                                                       (start'=1) &
                                                       (delivered_count'=delivered_count+1) &
                                                       (lost_count'=0);

    [] lost=1       & lost_count=MAX_COUNT       -> 1: (lost'=0) & (try'=1) & (lost_count'=lost_count);
    [] delivered=1  & delivered_count=MAX_COUNT  -> 1: (delivered'=0) &
                                                       (start'=1) &
                                                       (delivered_count'=delivered_count) &
                                                       (lost_count'=0);

endmodule
```

Simulating Urban Environments

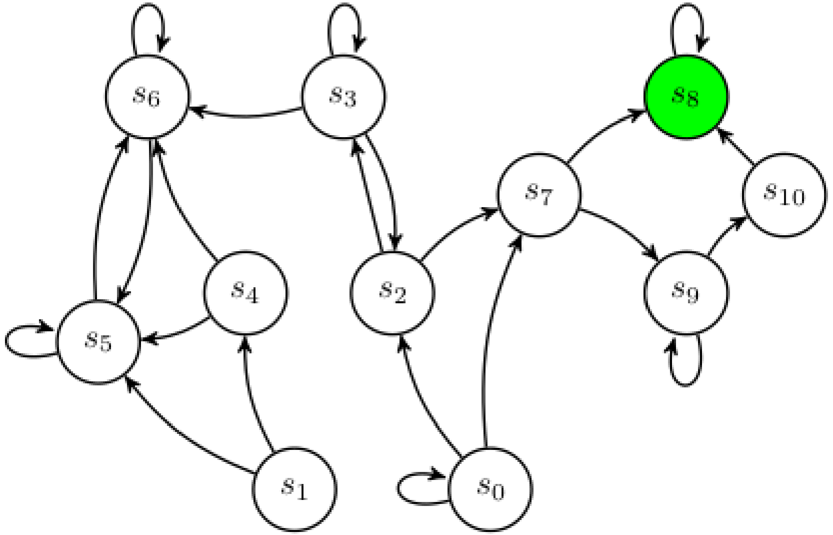


```
dtmc


...



module car
 // x and y coordinates, velocity







endmodule

module pedestrian
 // x and y coordinates, viewing direction in {left, right, north}







endmodule
```

Probabilistic Reachability

- We start with objectives similar to the ones discussed at the beginning of the semester:

What is the probability that our system reaches its goal state?



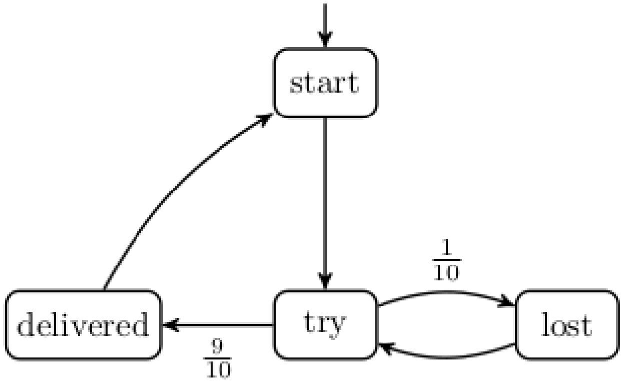
Before we talk about Algorithms...

Before we talk about Algorithms...

How can we represent a MC in code/memory?

Before we talk about Algorithms...

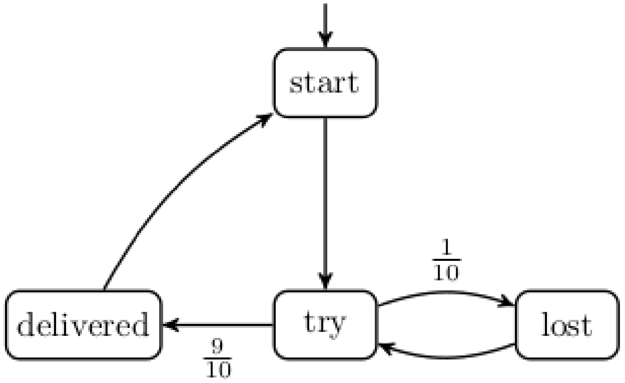
How can we represent a MC in code/memory?



$\longmapsto$

Before we talk about Algorithms...

How can we represent a MC in code/memory?



$$\longmapsto \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{10} & \frac{9}{10} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Model Checking with Markov Chains

- Explicit CTL model checking allows *qualitative* model checking.
  - $\mathcal{M}, \mathit{start} \models \exists \mathbf{G}\, \neg \mathit{delivered}$?

Model Checking with Markov Chains

- Explicit CTL model checking allows *qualitative* model checking.
  - $\mathcal{M}, \mathit{start} \models \exists\mathbf{G}\ \neg \mathit{delivered}$?

- We want to do *quantitative* model checking.
  - How *likely* is the system to fail?

$$Pr(\mathcal{M}, s \models \mathbf{F}\ s_{error})$$

  - Whats the *probability* of my message to arrive after infinitely many tries?

$$Pr(\mathcal{M}, s \models \mathbf{F}\ \text{delivered})$$

Paths

- A *path* $\pi = s_0 s_1 s_2 \ldots \in S^\omega$, s.t. $\mathbb{P}(s_i, s_{i+1}) > 0, \forall i \geq 0$
- $Paths(\mathcal{M})$ is the set of all paths in $\mathcal{M}$ and
- $Paths_{fin}(\mathcal{M})$ is the set of all finite path fragments in $\mathcal{M}$.

Events and Paths

In order to talk about probabilities of certain paths we need to briefly touch probability spaces.

- Outcomes = $\{HH, HT, TH, TT\}$
- Events = $\{HH\}, \{HT\}, \{TH\}, \{TT\}$

We could, for example, be interested in the events where H is thrown first = $\{HH\}, \{HT\}$.

What is a possible outcome in a specific Markov Chain $\mathcal{M}$?

Events and Paths

In order to talk about probabilities of certain paths we need to briefly touch probability spaces.

- Outcomes = $\{HH, HT, TH, TT\}$
- Events = $\{HH\}, \{HT\}, \{TH\}, \{TT\}$

We could, for example, be interested in the events where H is thrown first = $\{HH\}, \{HT\}$.

What is a possible outcome in a specific Markov Chain $\mathcal{M}$?

$\rightarrow$ an infinite path $\pi \in Paths(\mathcal{M})$!

- Outcomes = $Paths(\mathcal{M})$
- Events *of interest* are $\hat{\pi}_1, \hat{\pi}_2, \ldots \in Paths_{fin}(\mathcal{M})$ *that satisfy our property*
- Formally we introduce the *cylinder set* of a prefix:

$$Cyl(\hat{\pi}_i) = \{\pi \in Paths(\mathcal{M}) \mid \hat{\pi}_i \in \text{pref}(\pi)\}$$

Events and Paths

What is a possible outcome in a specific Markov Chain $\mathcal{M}$?

→ an infinite path $\pi \in Paths(\mathcal{M})$!

- Outcomes = $Paths(\mathcal{M})$
- Events *of interest* are $\hat{\pi}_1, \hat{\pi}_2, \ldots \in Paths_{fin}(\mathcal{M})$ *that satisfy our property*
- Formally we introduce the *cylinder set* of a prefix:

$$Cyl(\hat{\pi}_i) = \{\pi \in Paths(\mathcal{M}) \mid \hat{\pi}_i \in \text{pref}(\pi)\}$$

- The probability of one event of interest is then:

$$Pr(Cyl(\hat{\pi}_i)) = Pr(Cyl(s_0 s_1 \ldots s_n)) = \prod_{0 \leq i < n} \mathbb{P}(s_i, s_{i+1})$$

Reachability Probabilities

Let $B \subseteq S$ be a set of states. We are interested in

$$Pr(\mathcal{M}, s_0 \models \mathbf{F} B).$$

Reachability Probabilities

Let $B \subseteq S$ be a set of states. We are interested in

$$Pr(\mathcal{M}, s_0 \models \mathbf{F}B).$$

We can characterize all path fragments π that satisfy $\mathbf{F}B$ with the set

$$\Pi_{\mathbf{F}B} = Paths_{fin}(\mathcal{M}) \cap (S \setminus B)^* B$$

All $\hat{\pi} \in \Pi_{\mathbf{F}B}$ are pairwise disjoint, hence:

$$Pr(\mathcal{M}, s_0 \models \mathbf{F}B) = \sum_{\hat{\pi} \in \Pi_{\mathbf{F}B}} Pr(Cyl(\hat{\pi}))$$

Computing $Pr(\mathcal{M}, s_0 \models C \; \mathbf{U} \; B)$

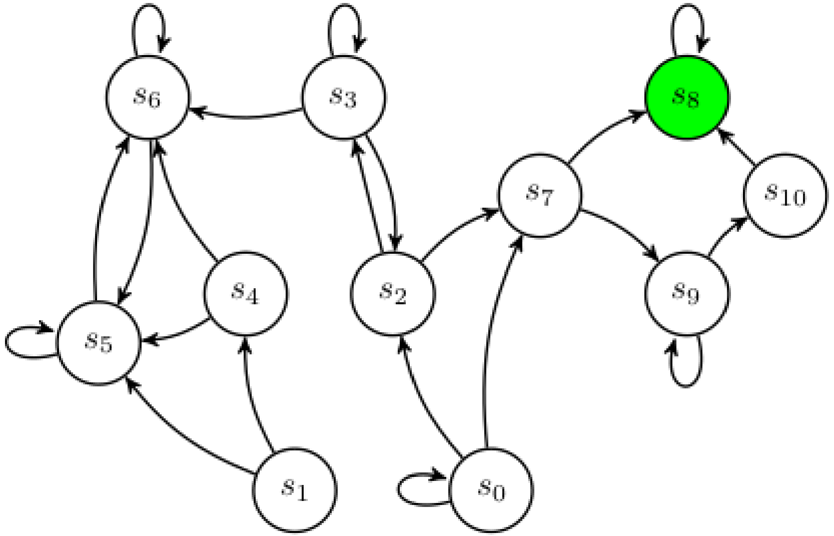
- We know that $\mathbf{F}B \equiv C \; \mathbf{U} \; B$, with $C = S$ or simply '$true \; \mathbf{U} \; B$'
    - Develop algorithm for arbitrary C

Computing $Pr(\mathcal{M}, s_0 \models C \ \mathbf{U} \ B)$

- We know that $\mathbf{F}B \equiv C \ \mathbf{U} \ B$, with $C = S$ or simply '$true \ \mathbf{U} \ B$'
  - Develop algorithm for arbitrary C

2-step algorithm:

1) Identify three disjoint subsets of S:
   - $S_{=1}$: The set of states with probability of 1 to reach B.
   - $S_{=0}$: The set of states with probability of 0 to reach B.
   - $S_?$: The set of states with probability $\in (0, 1)$ to reach B.



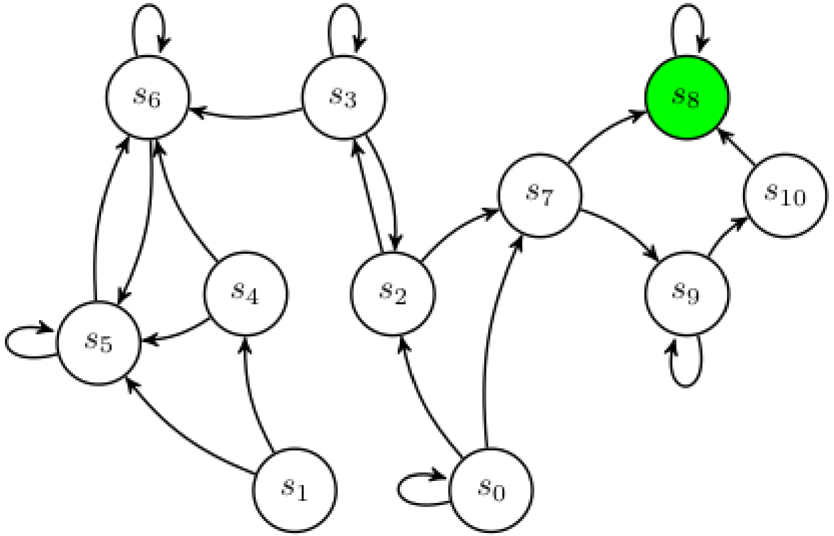
Computing $Pr(\mathcal{M}, s_0 \models C\ \mathbf{U}\ B)$

- We know that $\mathbf{F}B \equiv C\ \mathbf{U}\ B$, with $C = S$ or simply '$true\ \mathbf{U}\ B$'
  - Develop algorithm for arbitrary C

2-step algorithm:

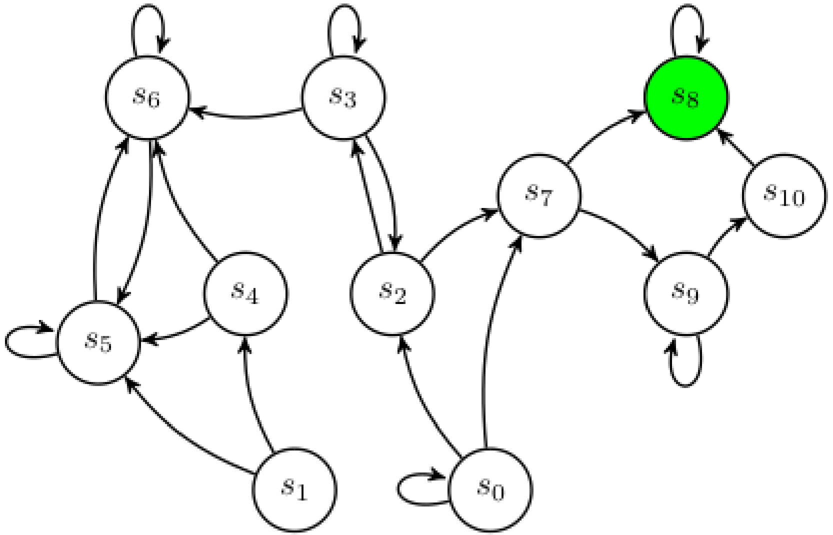
1) Identify three disjoint subsets of S:
   - $S_{=1}$: The set of states with probability of 1 to reach B.
   - $S_{=0}$: The set of states with probability of 0 to reach B.
   - $S_{?}$: The set of states with probability $\in (0, 1)$ to reach B.

2) Compute the probabilities for all $s \in S_?$.



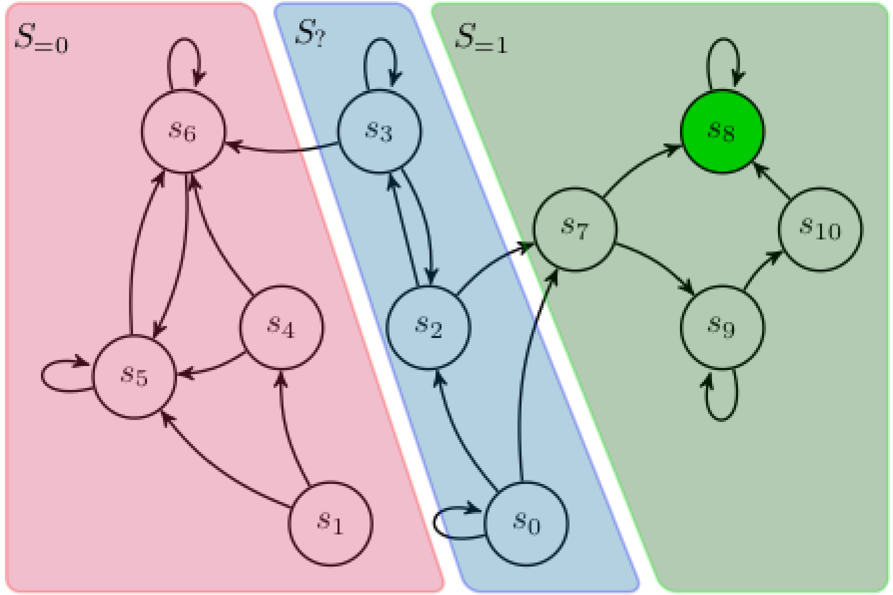
Computing $S_{=1}$ and $S_{=0}$

We can use DFS to compute these sets:



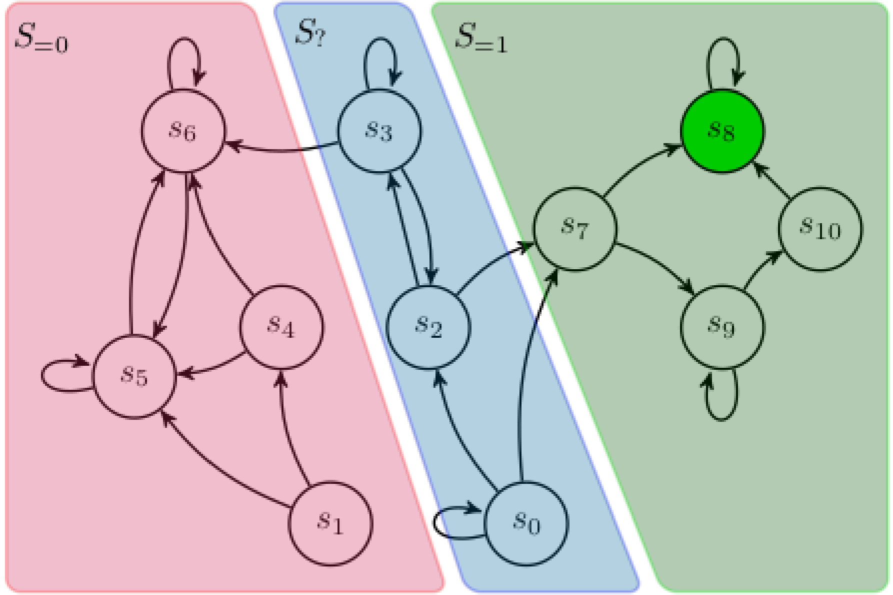
Computing $S_{=1}$ and $S_{=0}$

We can use DFS to compute these sets:



Computing $S_?$

We are left with computing the probabilities for $s \in S_?$



Computing $S_?$

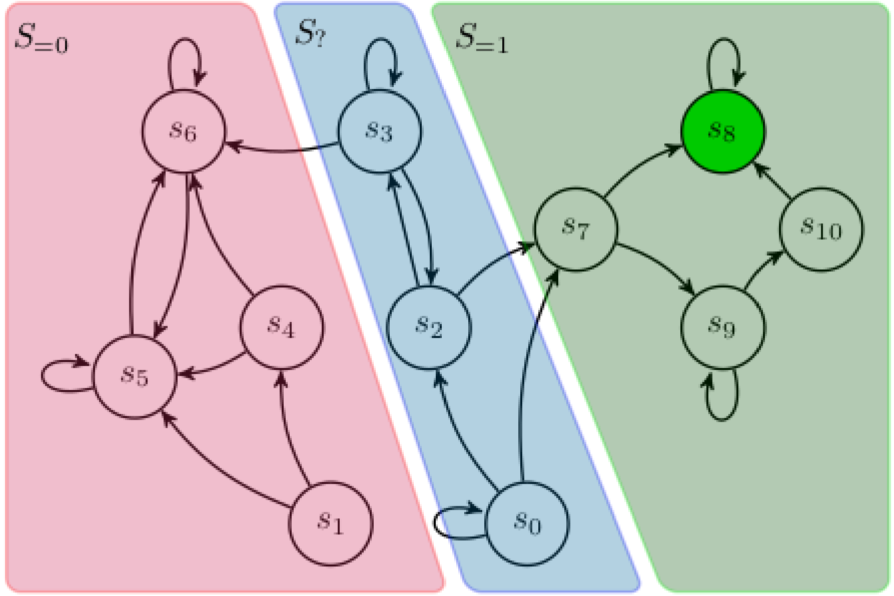
We are left with computing the probabilities for $s \in S_?$

Computing $S_?$

We are left with computing the probabilities for $s \in S_?$

- The probability to reach $S_{=1}$ in one step: $\sum_{u \in S_{=1}} \mathbb{P}(s,u)$
- and the probability to reach $S_{=1}$ via a path fragment $(s\ t\ \ldots\ u)$: $\sum_{t \in S_?} \mathbb{P}(s,t) \cdot x_t$
- Together

$$x_s = \sum_{t \in S_?} \mathbb{P}(s,t) \cdot x_t + \sum_{u \in S_{=1}} \mathbb{P}(s,u)$$



Computing $S_?$

Let us rewrite this into matrix notation:

- $A_? = (\mathbb{P}(s,t))_{s,t \in S_?}$
- $x = (x_s)_{s \in S_?}$
- $b = (\sum_{u \in S_{=1}} \mathbb{P}(s,u))_{s \in S_?}$

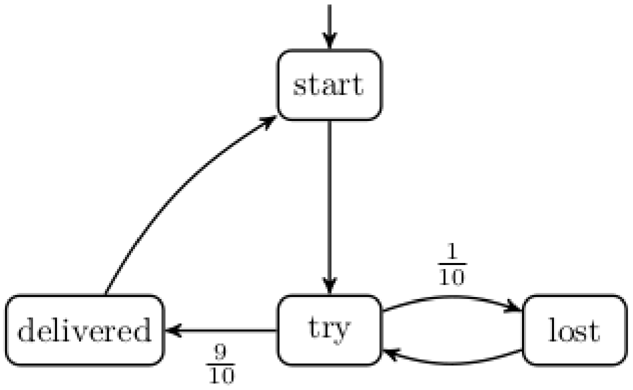
Computing $S_?$

Let us rewrite this into matrix notation:

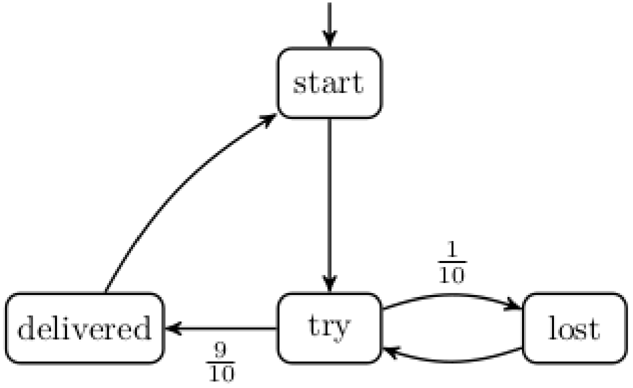
- $A_? = (\mathbb{P}(s,t))_{s,t \in S_?}$
- $x = (x_s)_{s \in S_?}$
- $b = (\sum_{u \in S_{=1}} \mathbb{P}(s,u))_{s \in S_?}$

$$x_s = \sum_{t \in S_?} \mathbb{P}(s,t) \cdot x_t + \sum_{u \in S_{=1}} \mathbb{P}(s,u) \rightsquigarrow x = A_? \cdot x + b = (I - A_?) \cdot x = b$$

Communication Protocol



Communication Protocol



$$\mathbf{A}_? = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & \frac{1}{10} \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ \frac{9}{10} \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -\frac{1}{10} \\ 0 & -1 & 1 \end{bmatrix} \cdot \mathbf{x} = \begin{pmatrix} 0 \\ \frac{9}{10} \\ 0 \end{pmatrix} \rightarrow \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Done

Transient State Probabilities

We will consider a slightly different algorithm:

$$\mathbf{A}^n = \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A} \cdot \cdots \cdot \mathbf{A}$$

contains the probability to be in state t after n steps in entry $\mathbf{A}^n(s,t)$.

We call

$$\Theta_n^{\mathcal{M}}(t) = \sum_{s \in S} \mathbf{A}^n(s,t)$$

the *transient state probability* for state t.

Transient State Probabilities

Let's consider $(\Theta_n^{\mathcal{M}}(t))_{s \in S}$, the vector of transient state probabilities for the nth step.

We can compute $Pr(\mathcal{M}, s_0 \models \mathbf{F}^{\leq n} B)$ in a modified Markov chain:

$$\mathcal{M}_B = (S, s_0, \mathbb{P}_B, AP, L)$$

where:

- $\mathbb{P}_B(s,t) = \mathbb{P}(s,t)$ if $s \notin B$
- $\mathbb{P}_B(s,s) = 1$ if $s \in B$
- $\mathbb{P}_B(s,t) = 0$ if $s \in B$ and $t \notin B$

i.e. all $s \in B$ become sinks and B cannot be left anymore.

Transient State Probabilities

- $\mathbb{P}_B(s,t) = \mathbb{P}(s,t)$ if $s \notin B$
- $\mathbb{P}_B(s,s) = 1$ if $s \in B$
- $\mathbb{P}_B(s,t) = 0$ if $s \in B$ and $t \notin B$

i.e. all $s \in B$ become sinks and B cannot be left anymore.

We then have

$$Pr(\mathcal{M}, s \models \mathbf{F}^{\leq n} B) = Pr(\mathcal{M}_B, s \models \mathbf{F}^{= n} B)$$

and therefore

$$Pr(\mathcal{M}, s \models \mathbf{F}^{\leq n} B) = \sum_{t \in B} \Theta_n^{\mathcal{M}_B}(t)$$

Computing $Pr(\mathcal{M}, s \models \mathbf{F}^{\leq n} B)$ via Transient State Probabilities

We have the following algorithm to compute $Pr(\mathcal{M}, s \models \mathbf{F}^{\leq n} B)$:

- $\Theta_0^{\mathcal{M}}(t) = \mathbf{e}_i$, i.e. the unit vector with 1 at the ith position and 0 else.
- For $k = 0$ up to $n - 1$: $\Theta_{k+1}^{\mathcal{M}}(t) = \mathbf{A} \cdot \Theta_k^{\mathcal{M}}(t)$
- $Pr(\mathcal{M}, s \models \mathbf{F}^{\leq n} B) = \sum_{t \in B} \Theta_n^{\mathcal{M}_B}(t)$