Computer Lab 1
Probabilistic Model Checking

Nok Wongpiromsarn
UT Austin/Iowa State

Richard M. Murray
Caltech

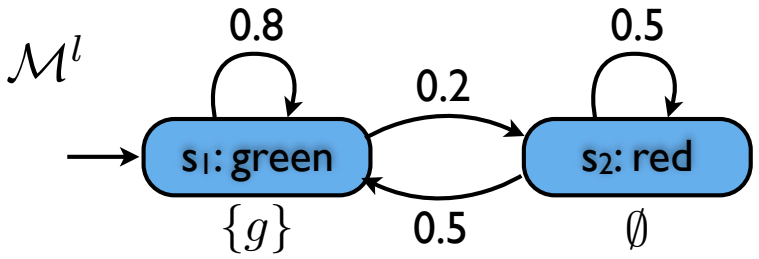
EECI-IGSC, 10 March 2020

Outline

- Probabilistic model checking
- Probabilistic synthesis
- TuLiP interface to stormpy

Prism Models: Discrete-Time Markov Chain (DTMC)

$$\mathcal{M} = (S, \mathbf{P}, \iota_{init}, AP, L)$$



light.pm

Model type →

States S, ι_{init} →

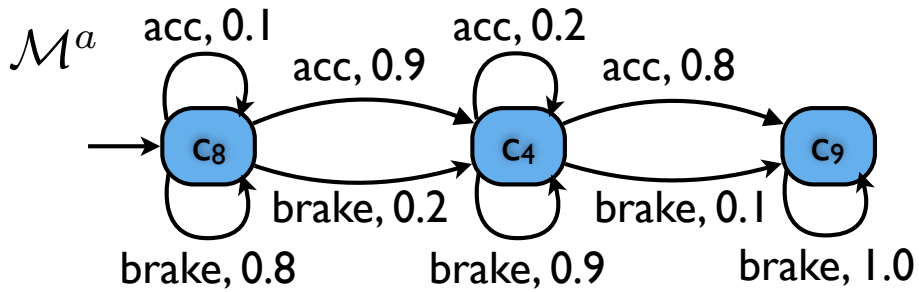
Transitions $\mathbf{P}$ →

Labels AP, L →

```
dtmc

module light

// state
s : [0..1] init 0;

[] s=0 -> 0.2 : (s'=1) + 0.8 : (s'=0);
[] s=1 -> 0.5 : (s'=0) + 0.5 : (s'=1);

endmodule

label "green" = s=0;
label "red" = s=1;
```

Prism Models: Markov Decision Process (MDP)

$$\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$$



ma.nm

Model type →

States S, ι_{init} →

Transitions $Act, \mathbf{P}$ →

Labels AP, L →

```
mdp

module autonomous_vehicle

    // state
    s : [4..9] init 8;

    [acc]    s=8 -> 0.1 : (s'=8) + 0.9 : (s'=4);
    [brake]  s=8 -> 0.8 : (s'=8) + 0.2 : (s'=4);
    [acc]    s=4 -> 0.2 : (s'=4) + 0.8 : (s'=9);
    [brake]  s=4 -> 0.9 : (s'=4) + 0.1 : (s'=9);
    [brake]  s=9 -> 1 :   (s'=9);

endmodule

label "a8" = s=8;
label "a4" = s=4;
label "a9" = s=9;
```

Property Specification: The P Operator

PCTL Property:

P bound [pathprop]

Quantitative Property:

P = ? [pathprop]

Nondeterministic Models:

Pmin = ? [pathprop]

Pmax = ? [pathprop]

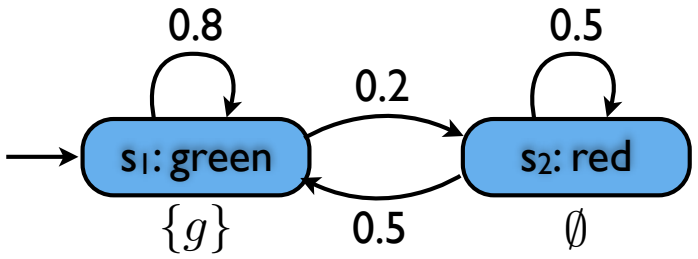
Property Specification: Path Properties

A single *temporal operator*

- **X**: next
- **U**: until
- **F**: eventually
- **G**: always
- **W**: week until
- **R**: release

Logical operator

- !: negation
- &: conjunction
- |: disjunction



P = ? [F ("green")]

```
dtmc

module light

// state
s : [0..1] init 0;

[] s=0 -> 0.2 : (s'=1) + 0.8 : (s'=0);
[] s=1 -> 0.5 : (s'=0) + 0.5 : (s'=1);

endmodule

label "green" = s=0;
label "red" = s=1;
```

TuLiP's models

- TuLiP includes many types of models: *KripkeStructure*, *WeightedKripkeStructure*, *MarkovChain*, *MarkovDecisionProcess*, *FiniteTransitionSystem*, etc.
- The main difference between different types of models is the allowable state and transition attributes. For example, *KripkeStructure* only allows *"ap"* attribute on a state. *WeightedKripkeStructure* additionally allows *"cost"* attribute on a transition.
- Given a *model*, useful operations are
  - *model.states*: An iterable object that represents all the states
  - Given a *state* in *model.states*, *model.states[state]["ap"]* is the set of atomic propositions at *state*
  - *model.transitions.find()*: Return all the transitions, each is a tuple *(source_state, target_state, transition_attr)* where *source_state* and *target_state* are in *model.states* and *transition_attr* is a dictionary, e.g., {"cost" : 0} in the case where *model* is *WeightedKripkeStructure*

Composition of models

- *composed* = **synchronous_parallel(** *models* **)**

  Return the synchronous parallel composition (i.e., tensor product in graph theory) of *models*

- mc = **apply_policy(** *mdp, policy* **)**

  Return the MC induced by applying *policy* on MarkovDecisionProcess *mdp*

TuLiP's stormy interface

- *model* = **to_tulip_transys(** *path* **)**

  Parse the *path* file and return the MarkovChain or MarkovDecisionProcess *model*

- **to_prism_file(** *model, path* **)**

  Export MarkovChain or MarkovDecisionProcess *model* to a prism file

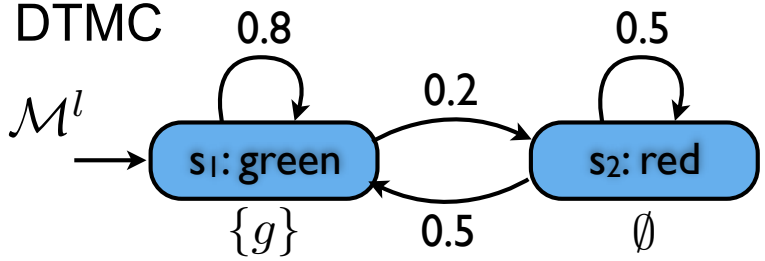
- result = **model_checking(** *model, formula, path, extract_policy* **)**

  Model check *model* against *formula and* export intermediate prism file to *path*

  - If *extract_policy* = *False*, then for each *state* in *model.states*, *result[state]* is the probability of satisfying the formula starting at *state*.
  - If *extract_policy* = *True*, then *result* = *(prob, policy)* where for each *state* in *model.states*, *prob[state]* is the probability of satisfying the formula starting at *state* and *policy[state]* is the action to be applied at *state.*

Example 1: traffic light (analysis)

System: DTMC



Specification:

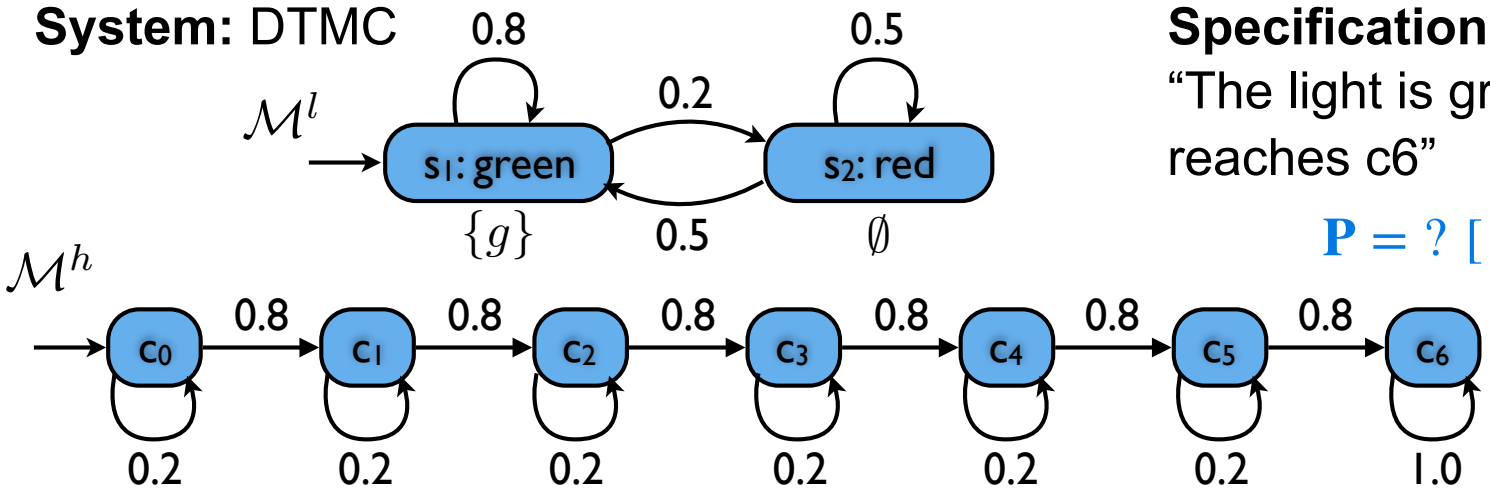
"The light is eventually green"

$$\mathbf{P} = ?\ [\ \mathbf{F}\ (\text{"green"})\ ]$$

`analysis_light.ipynb`

Example 2: traffic light + vehicle (analysis)

System: DTMC

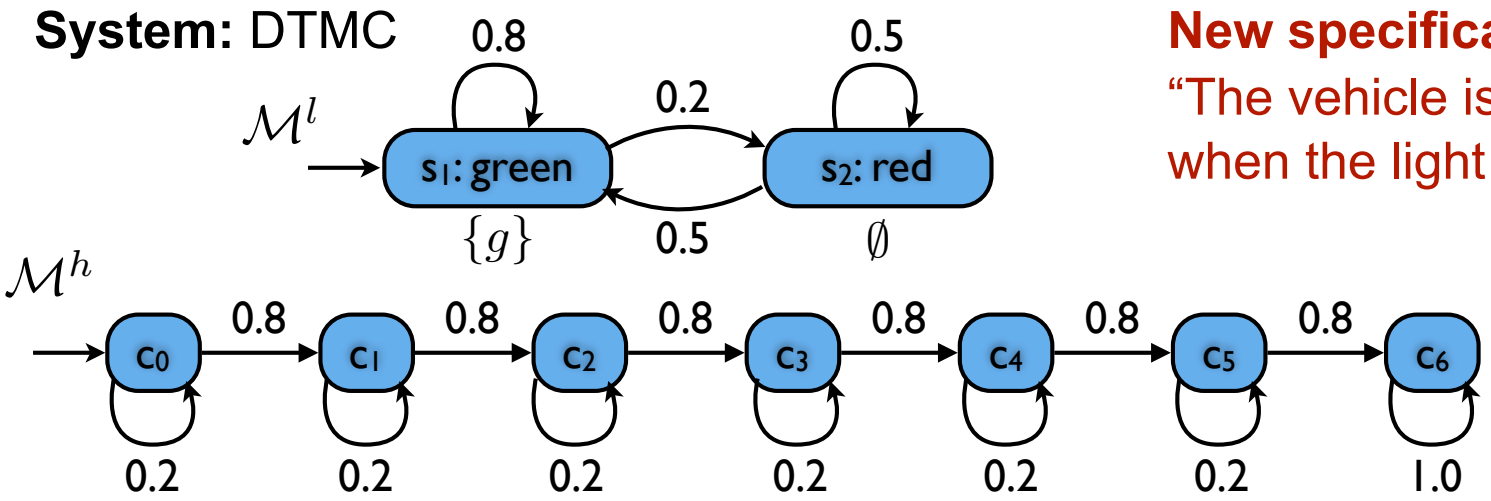


Specification:
"The light is green until the vehicle reaches c6"

P = ? ["green" **U** "h6"]

`analysis_compose.ipynb`

Exercise 1: traffic lights + vehicle



New specification:
"The vehicle is never at c4 or c5 when the light is red"

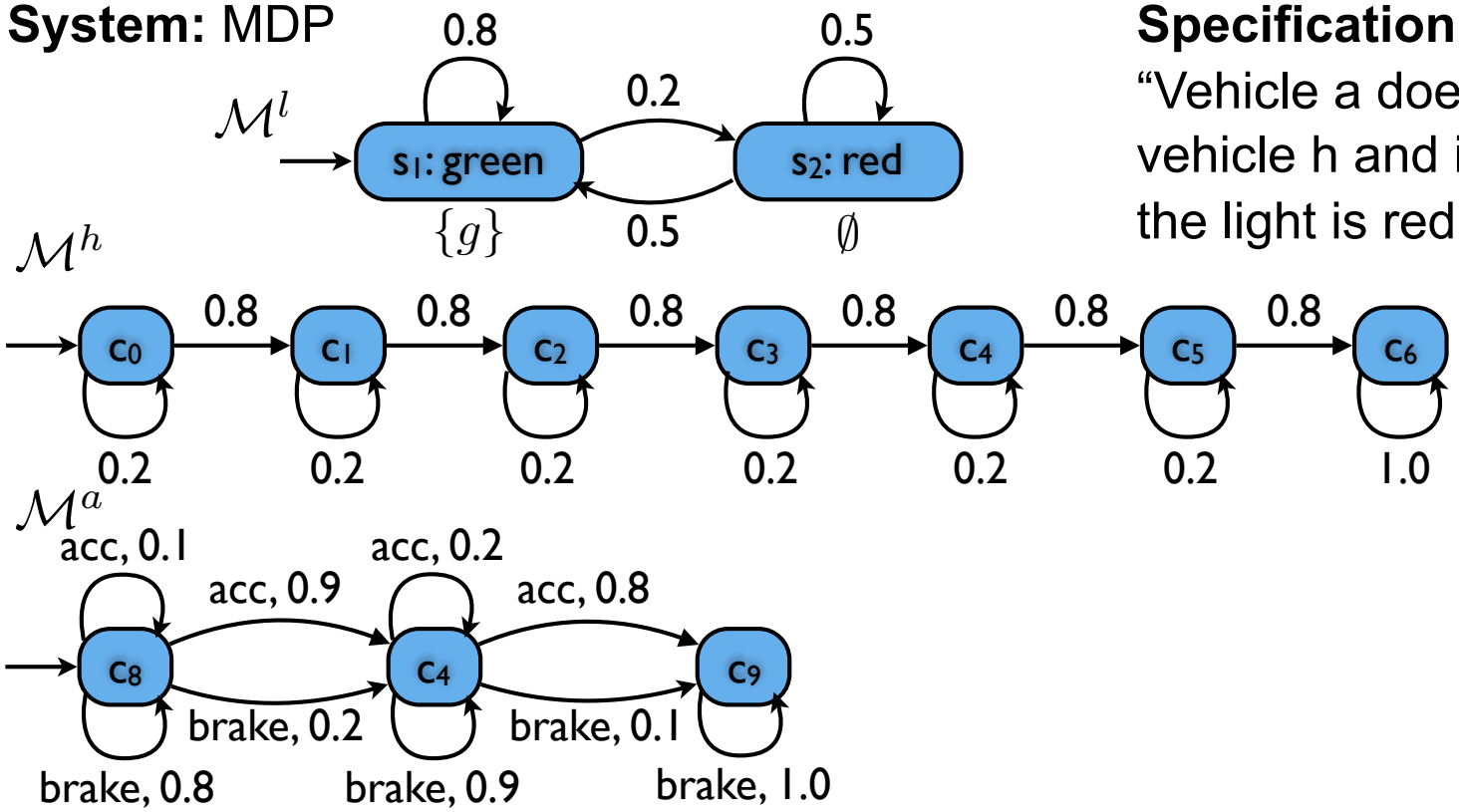
Ex: Modify

`c1-probabilistic/examples/analysis_compose.ipynb`

to compute the probability that the parallel composition of traffic light and vehicle satisfies the new specification.

Example 3: manually-defined policy

System: MDP

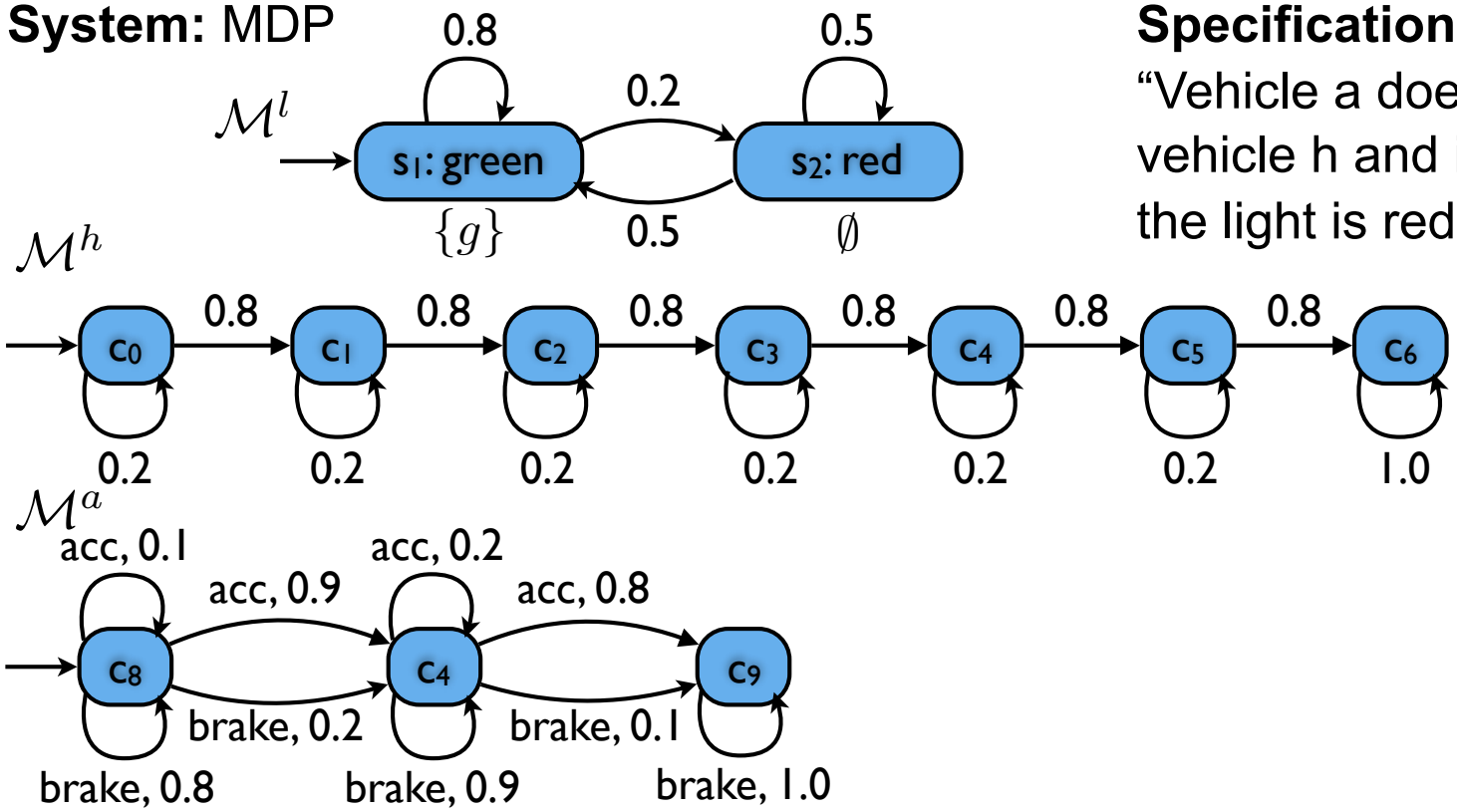


Specification:
"Vehicle a does not collide with vehicle h and is not at c8 or c4 when the light is red until it reaches c6"

`analysis_policy.ipynb`

Exercise 2: manually-defined policy

System: MDP



Specification:
"Vehicle a does not collide with vehicle h and is not at c8 or c4 when the light is red until it reaches c6"

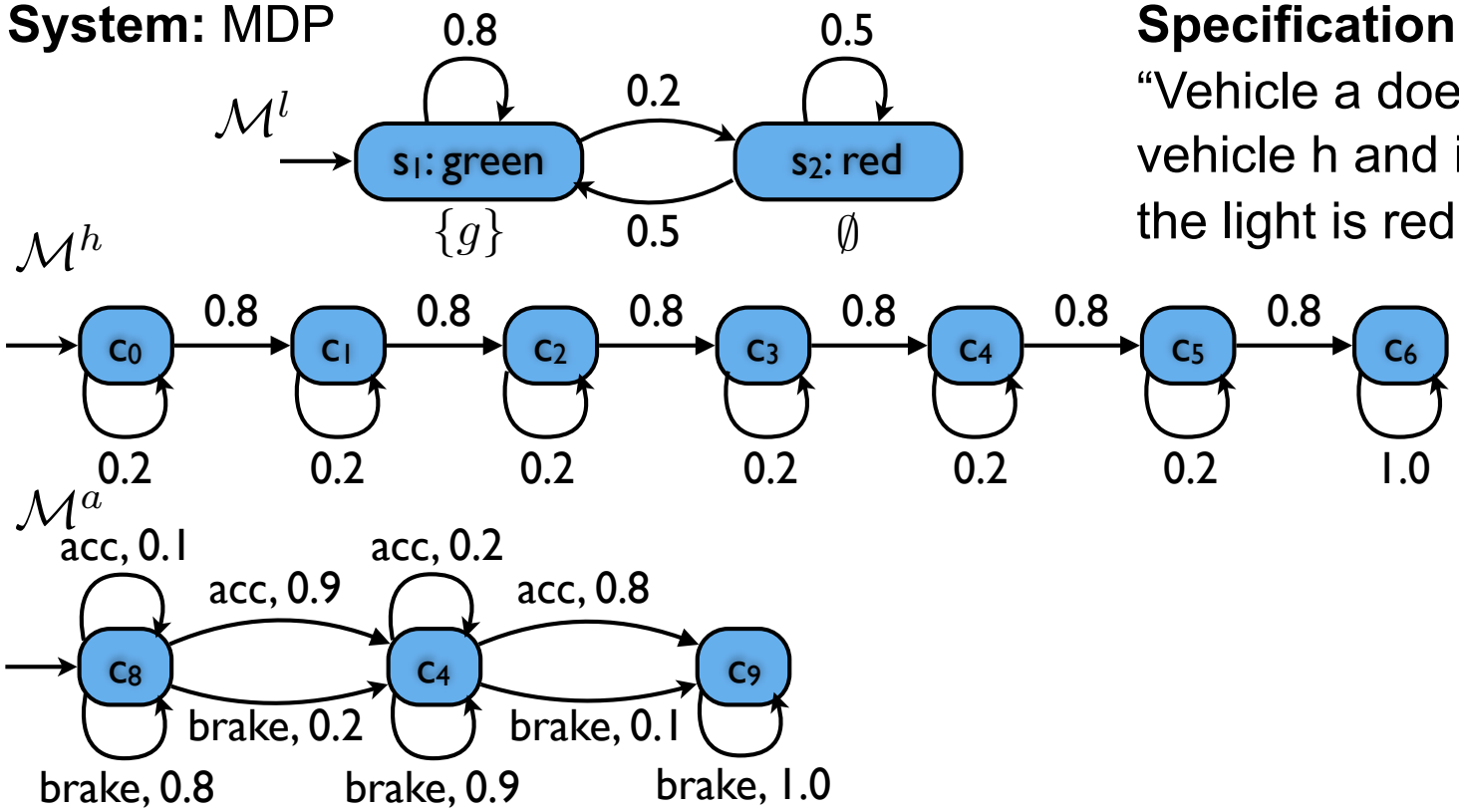
Ex: Modify

`c1-probabilistic/models/policy.json`

to maximize the probability of satisfying the spec at the initial state.

Example 4: policy synthesis

System: MDP



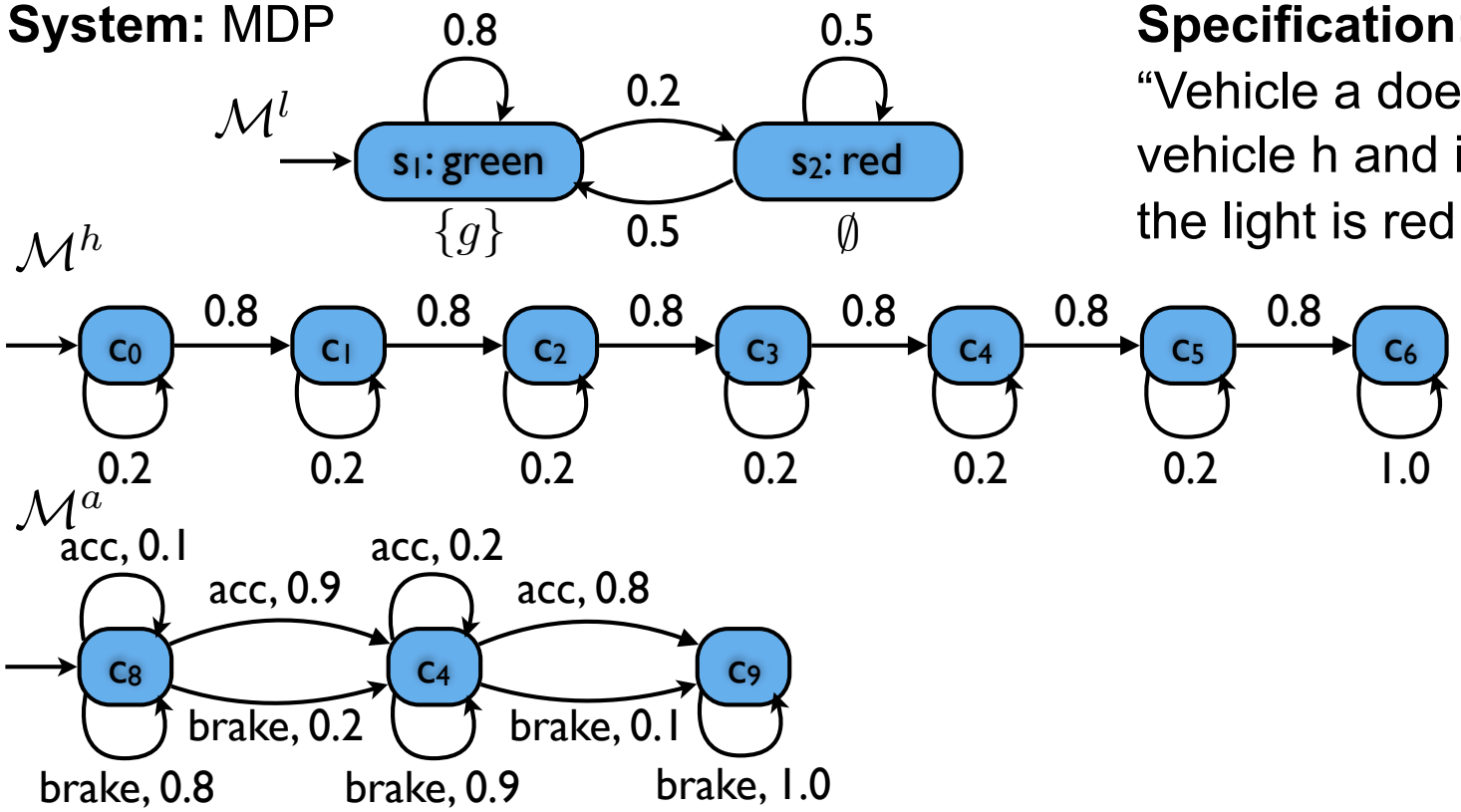
Specification:

"Vehicle a does not collide with vehicle h and is not at c8 or c4 when the light is red until it reaches c6"

`synthesis.ipynb`

Exercise 3: policy synthesis

System: MDP



Specification:
"Vehicle a does not collide with vehicle h and is not at c8 or c4 when the light is red until it reaches c6"

Ex:

- Examine the optimal policy. How is it different from your policy?
- Compute the MC induced by the optimal policy.
- What is the probability of satisfying the spec at the initial state with the optimal policy?