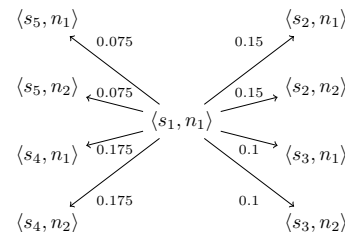
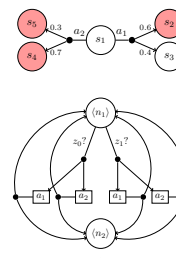
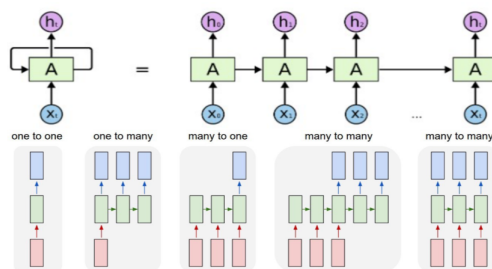
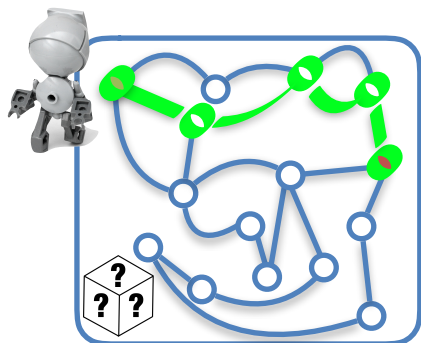


# Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks

Machine Learning and Formal Verification Join Forces?

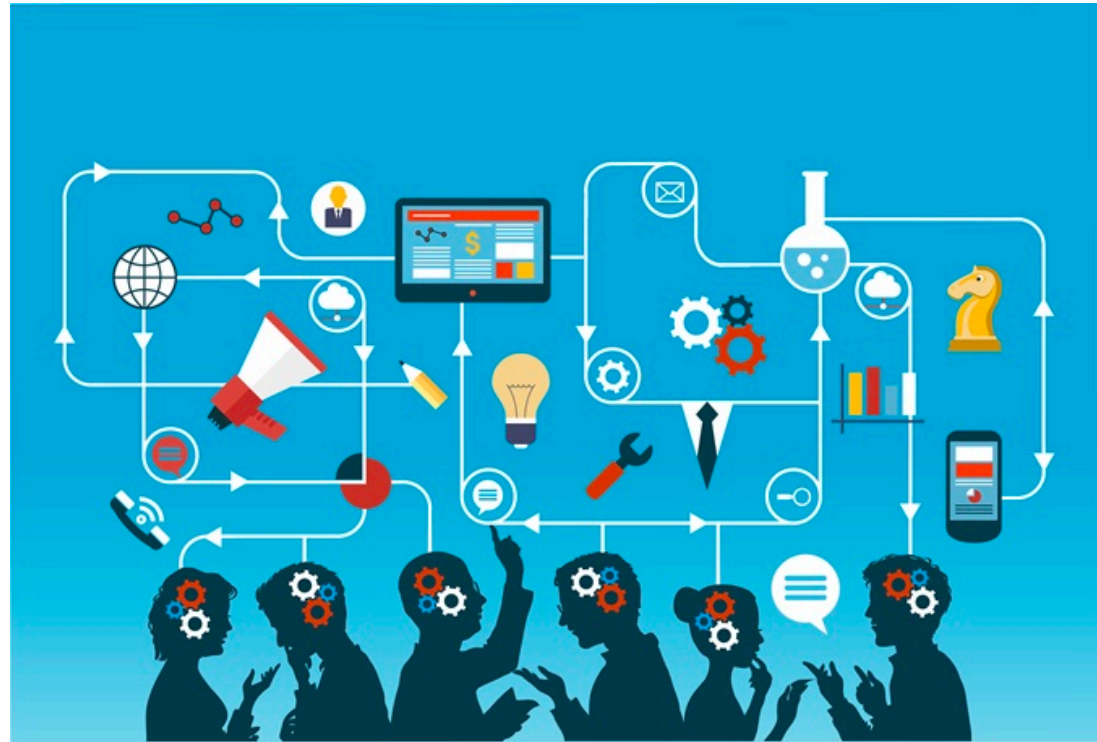


Nils Jansen

IJCAI, August 15, 2019

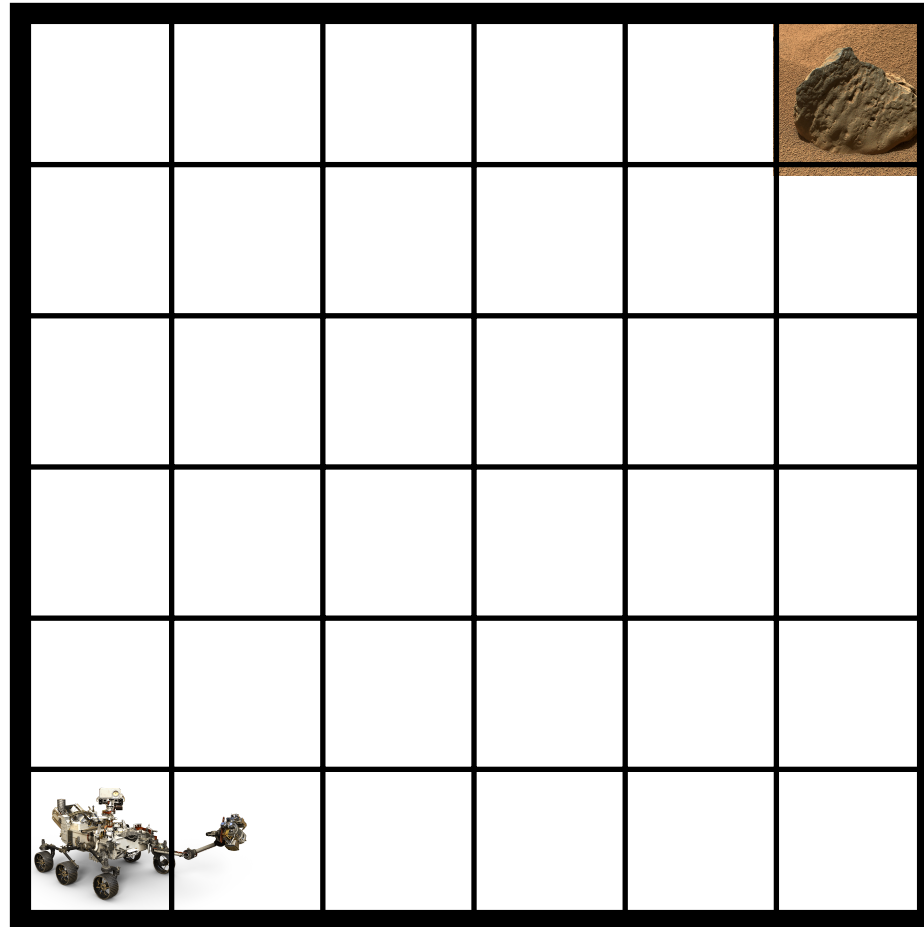
# joint work with:

Steve Carr, Alexandru Serban, Ralf Wimmer, **Ufuk Topcu**, Bernd Becker



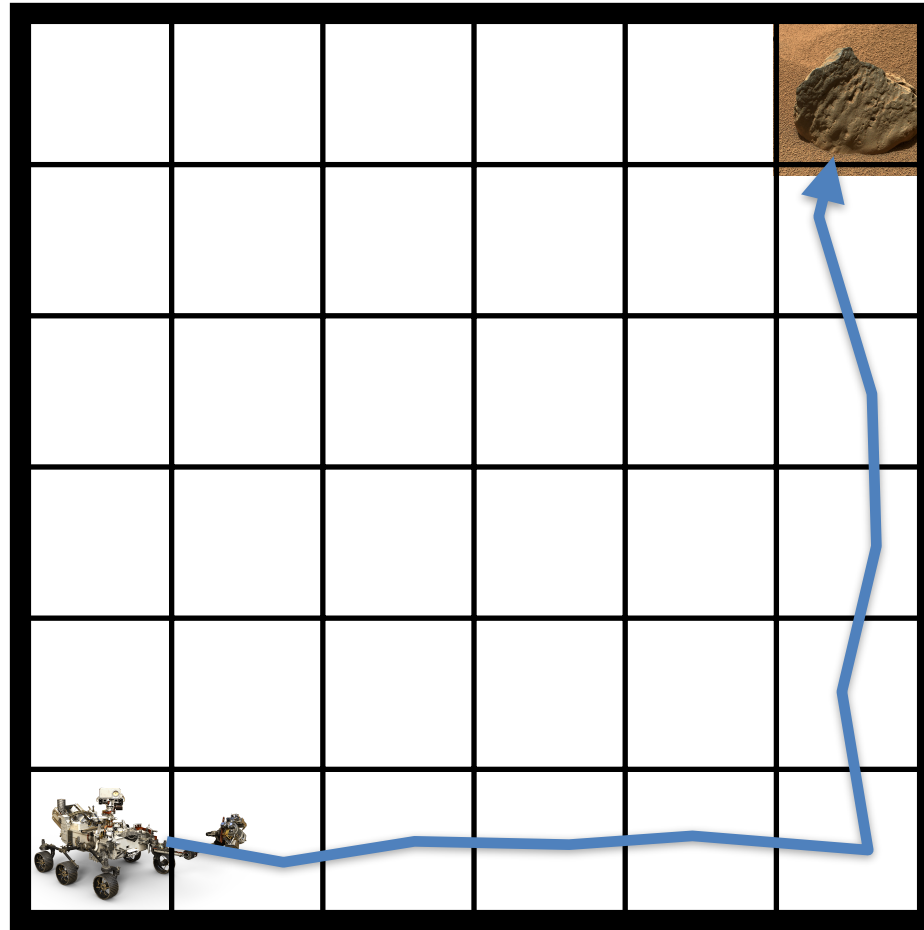
# Help the Robot

Find the best way to  
the **rock**



# Help the Robot

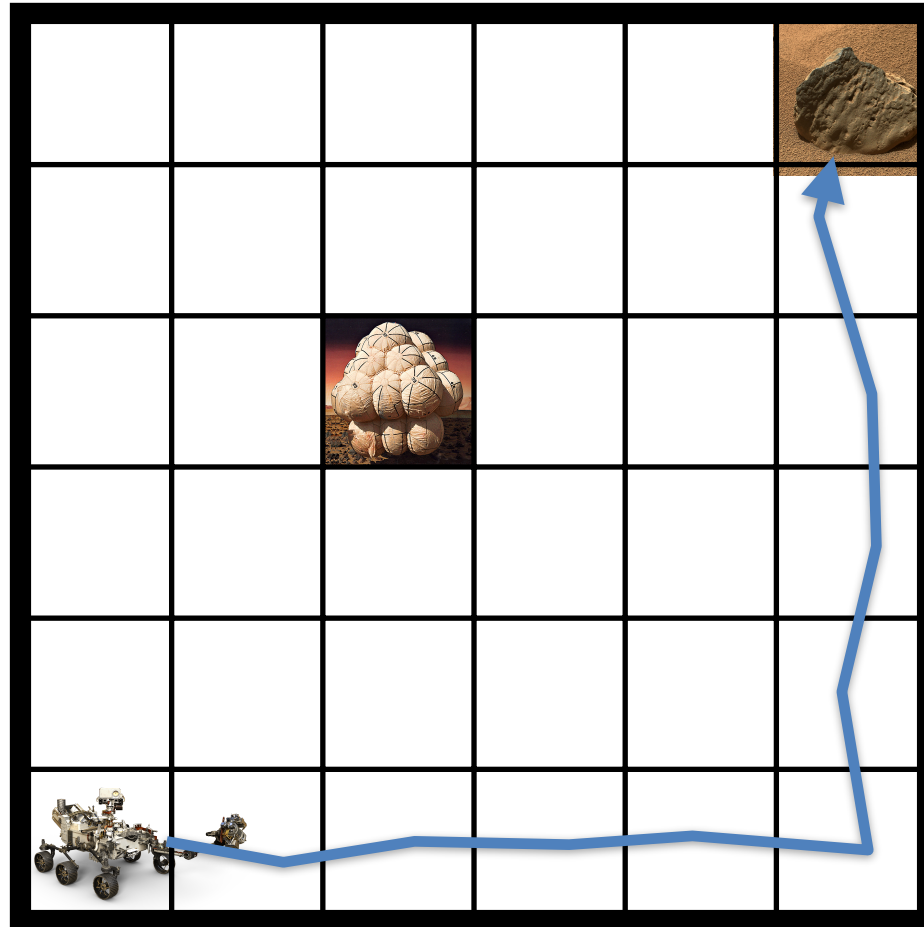
Find the best way to  
the **rock**



# Help the Robot

Find the best way to  
the **rock**

Visit the **parachute**  
on the way

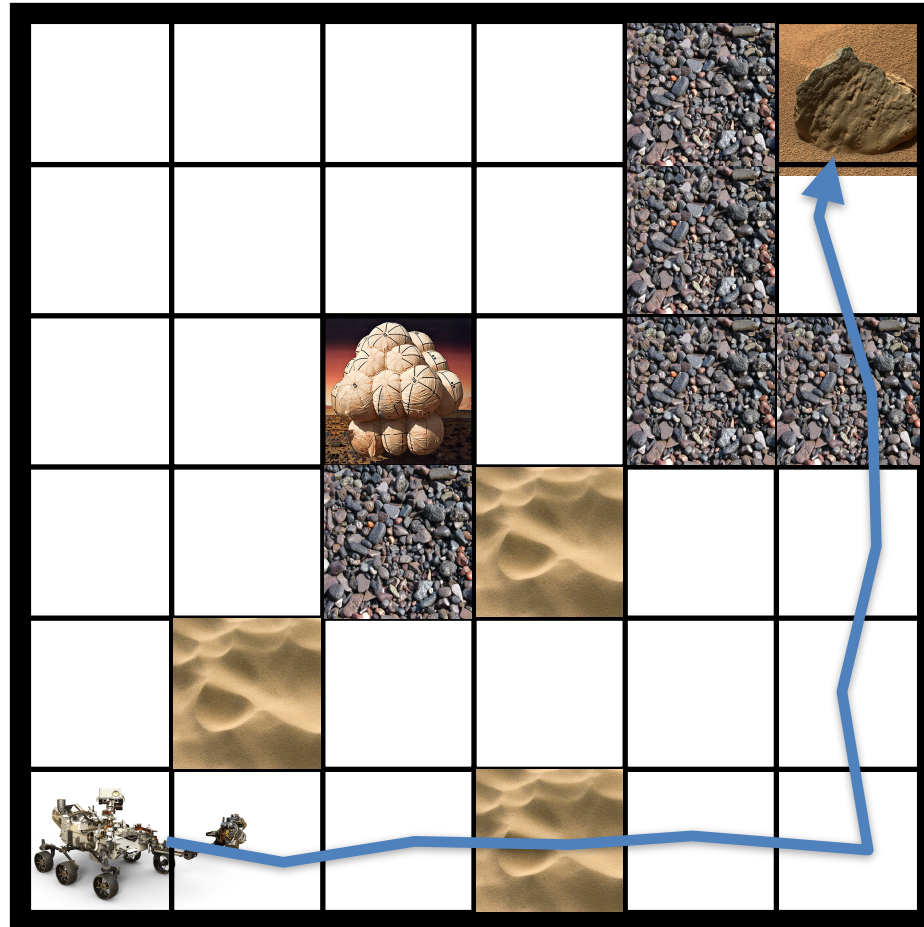


# Help the Robot

Find the best way to  
the **rock**

Visit the **parachute**  
on the way

Take **expensive**  
**surfaces** into  
account



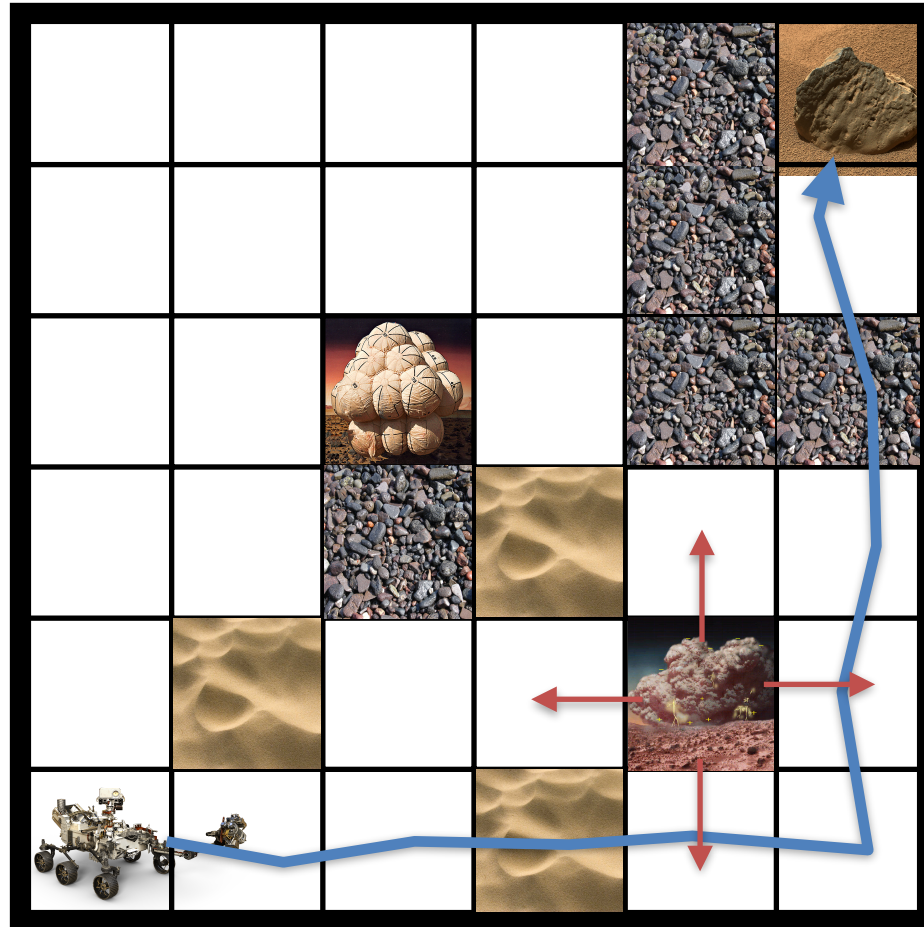
# Help the Robot

Find the best way to  
the **rock**

Visit the **parachute**  
on the way

Take **expensive**  
**surfaces** into  
account

Avoid randomly  
moving **dust storm**



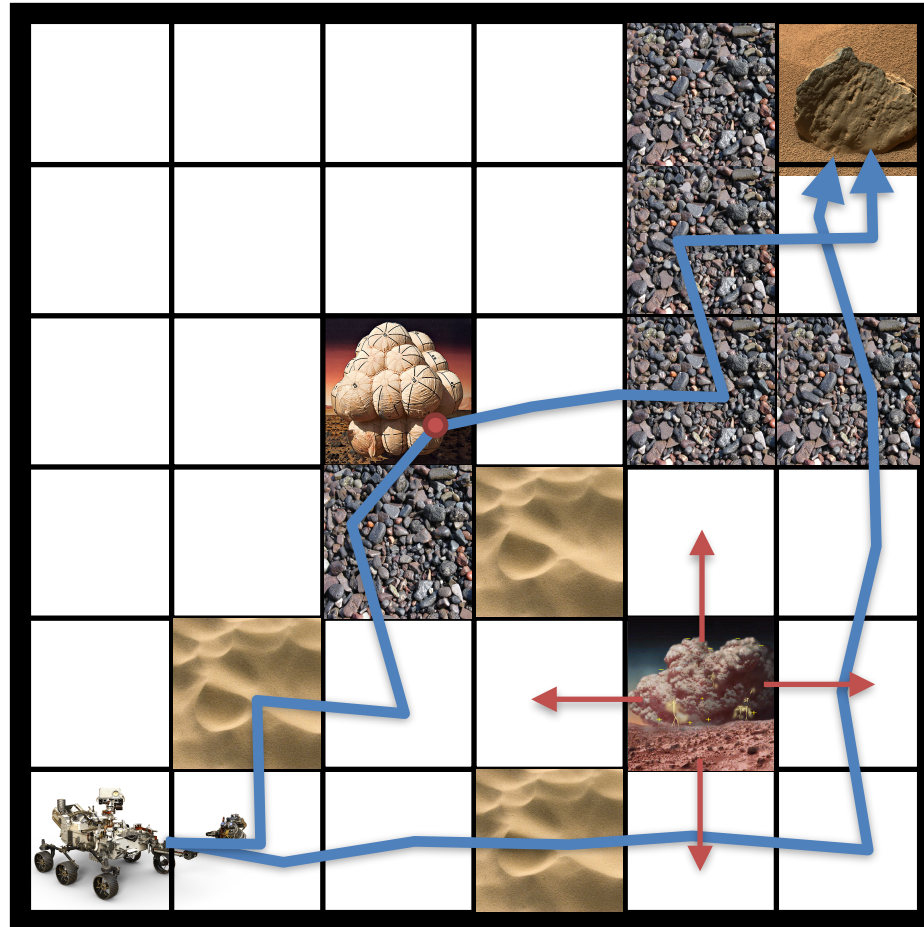
# Help the Robot

Find the best way to  
the **rock**

Visit the **parachute**  
on the way

Take **expensive**  
**surfaces** into  
account

Avoid randomly  
moving **dust storm**



Find safe  
and/or cost-  
optimal  
strategy to  
get to the  
airbag

# Help the Robot

Find the best way to the **rock**

Visit the **parachute** on the way

Take **expensive surfaces** into account

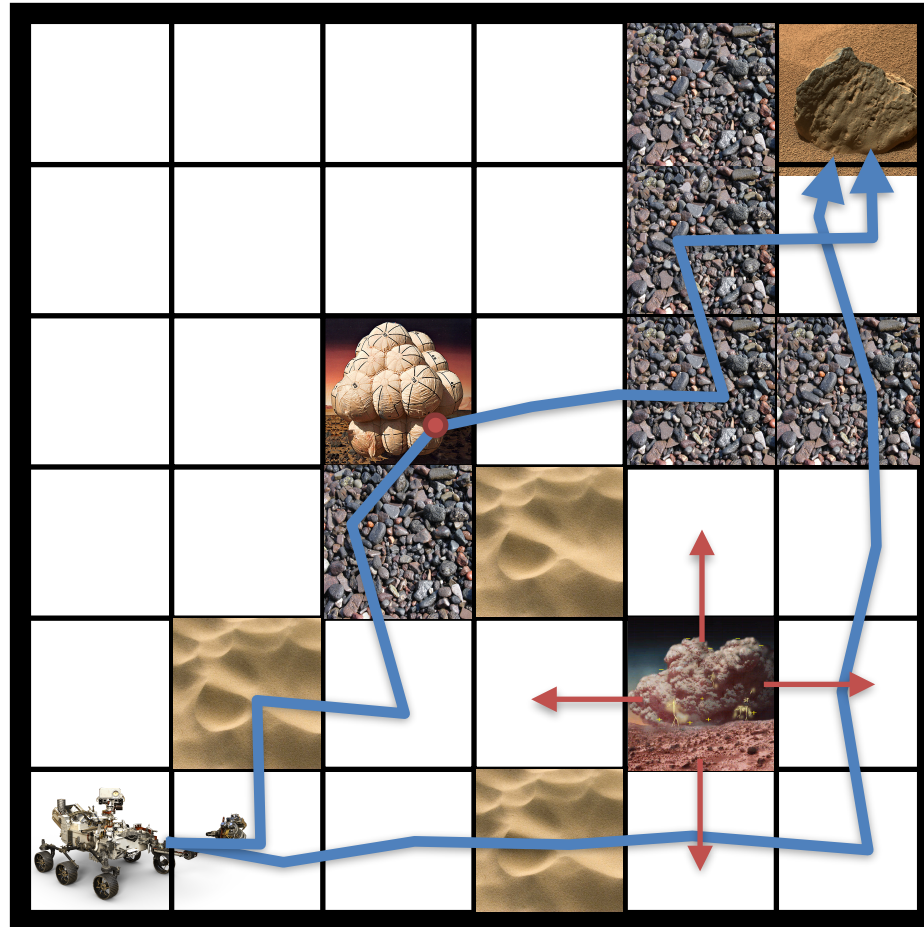
Avoid randomly moving **dust storm**

$$Pr_{max}(\neg s \ U (\Diamond(p \wedge \bigcirc \Diamond r)))$$

$$EC_{min}(\Diamond r)$$

temporal logic

expected cost



Find safe and/or cost-optimal strategy to get to the airbag

Underlying Model: Markov Decision Process

# Partial Observability

It's a well known fact that you must spin a USB **three times** before it will fit. From this, we can gather that a USB has three states:



Up position



Down position



Superposition

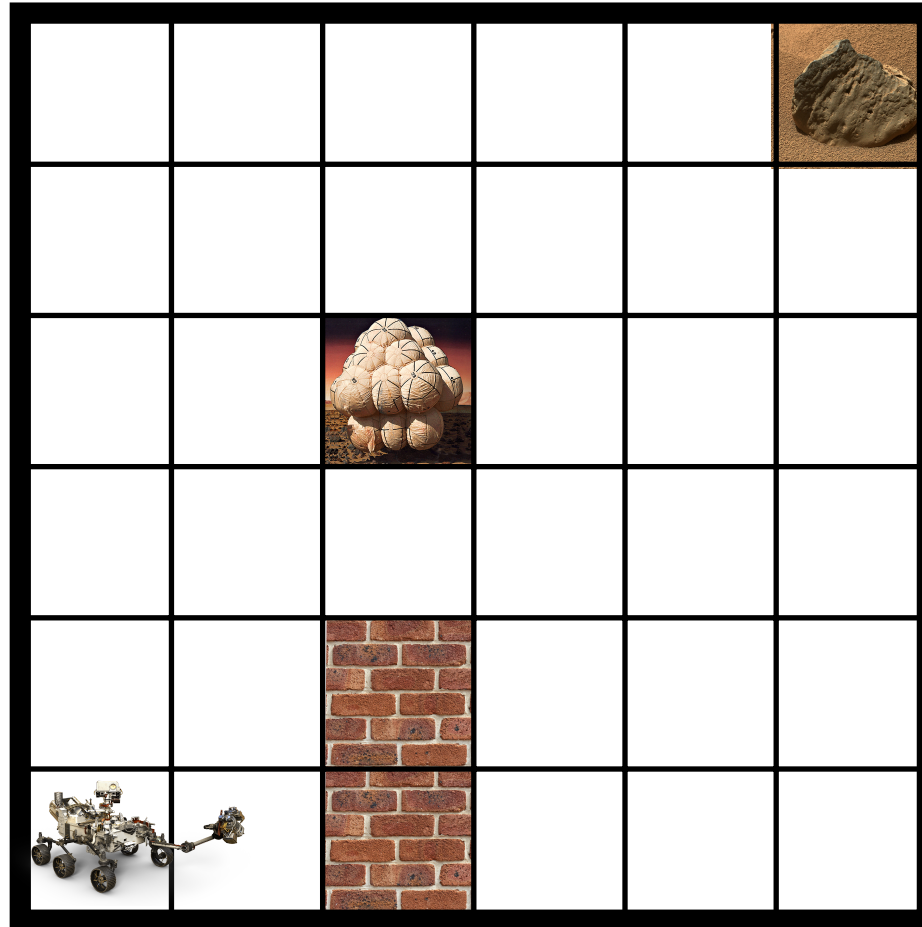


FUNCOOL.net

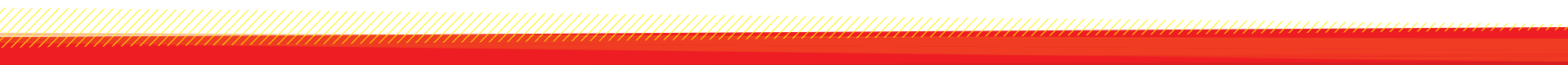
Until the USB is observed it will stay in the superposition. Therefore it will not fit until observed - except for in cases of USB tunnelling.

# Help the Robot with Partial Observability

Robot has restricted range of vision



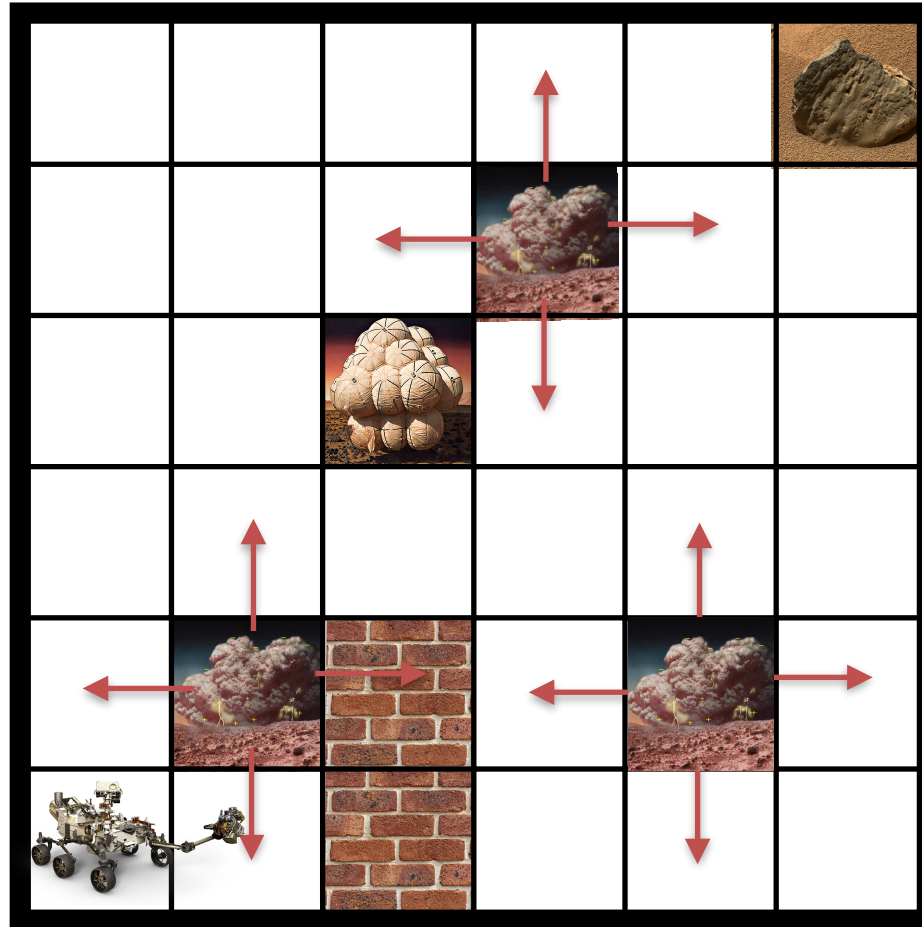
Find safe and/or cost-optimal strategy to get to the airbag



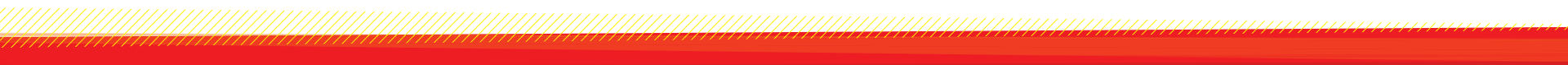
# Help the Robot with Partial Observability

Robot has restricted range of vision

Storm is only **observable** when near



Find safe and/or cost-optimal strategy to get to the airbag

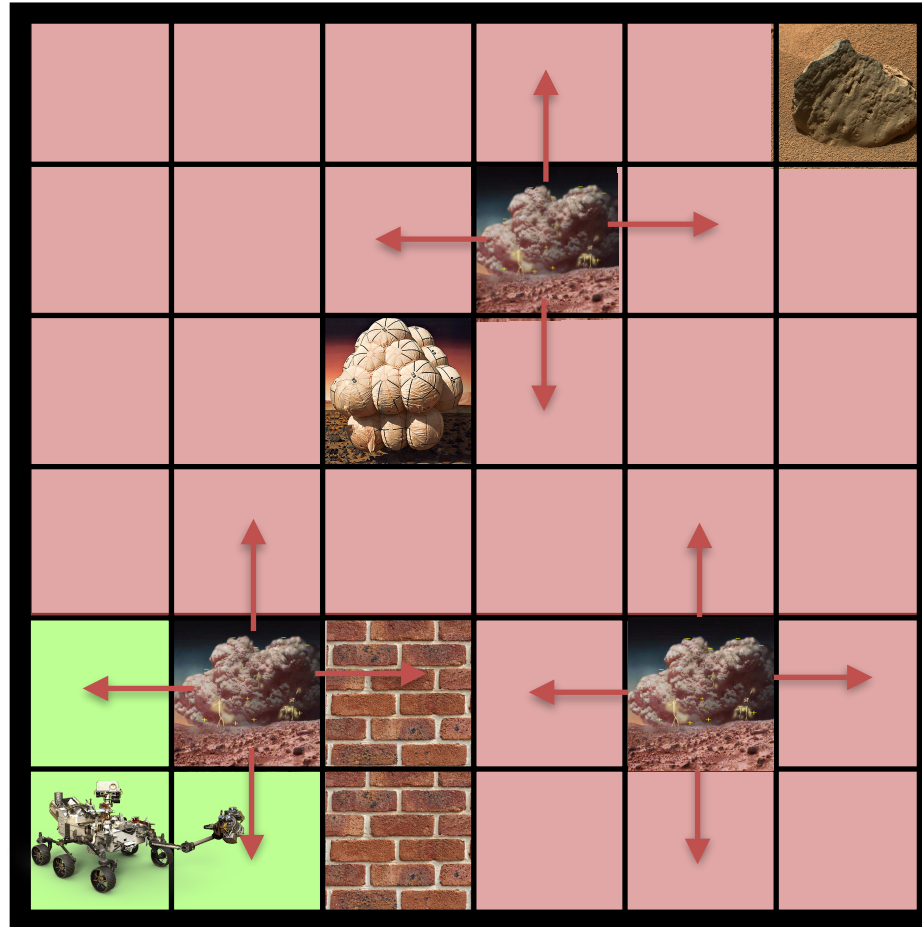


# Help the Robot with Partial Observability

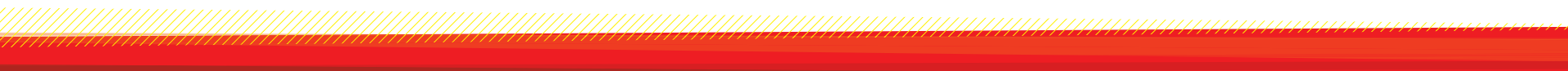
Robot has restricted range of vision

Storm is only **observable** when near

For robot, storm is either **near** or **far**



Find safe and/or cost-optimal strategy to get to the airbag

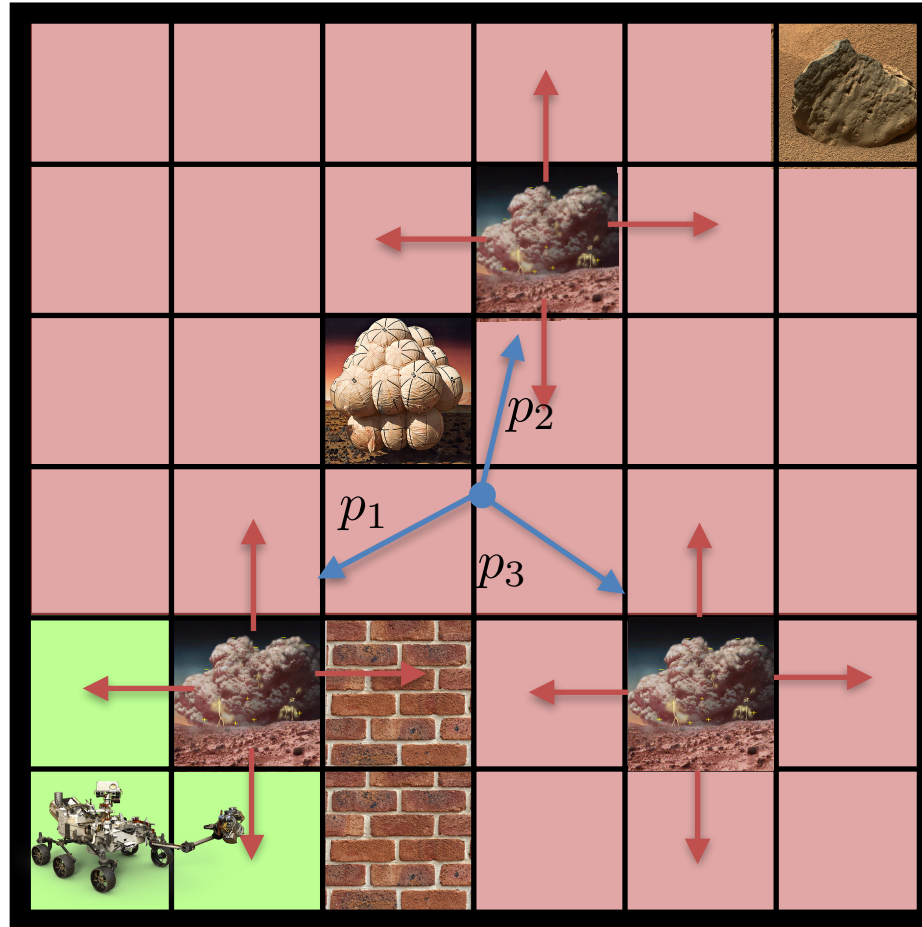


# Help the Robot with Partial Observability

Robot has restricted range of vision

Storm is only **observable** when near

For robot, storm is either **near** or **far**



Find safe and/or cost-optimal strategy to get to the airbag

Belief state: Likelihood of the actual position of the storm

infinite belief MDP

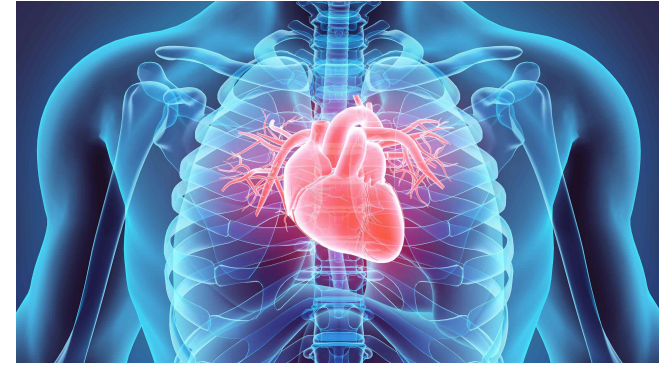
# POMDPs - Applications



Stock Market



Surveying Threatened Species



Health Care



Wireless Sensor Networks



Autonomous Systems



Machine Vision

# Computing Strategies for POMDPs

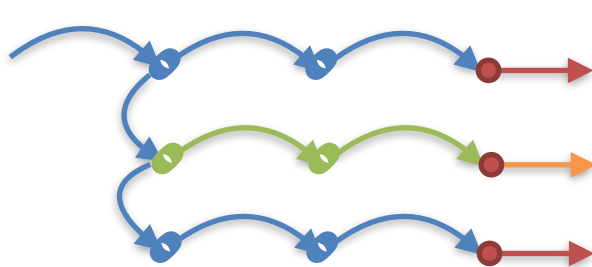
- Randomized with infinite memory: undecidable, optimal results.

# Computing Strategies for POMDPs

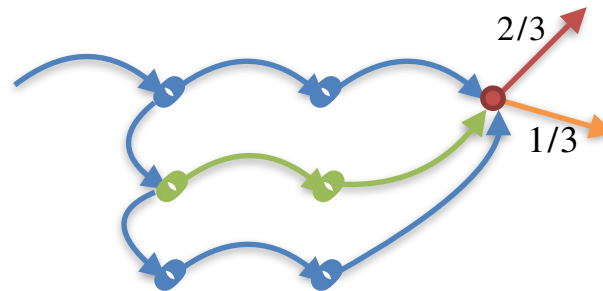
- Randomized with infinite memory: undecidable, optimal results.
- Randomized with finite memory: NP-hard, SQRT-SUM-hard, in PSPACE, not optimal in general, but sufficient for many applications.

# Computing Strategies for POMDPs

- Randomized with **infinite memory**: **undecidable**, optimal results.
- Randomized with **finite memory**: **NP-hard**, **SQRT-SUM-hard**, in **PSPACE**, not optimal in general, but sufficient for many applications.
- Intuitively: **Randomization** can often **trade off memory**.



$$\sigma: \text{ObsSeq}_{fin} \rightarrow \text{Distr}(\text{Act})$$



$$\sigma: \text{Obs} \rightarrow \text{Distr}(\text{Act})$$

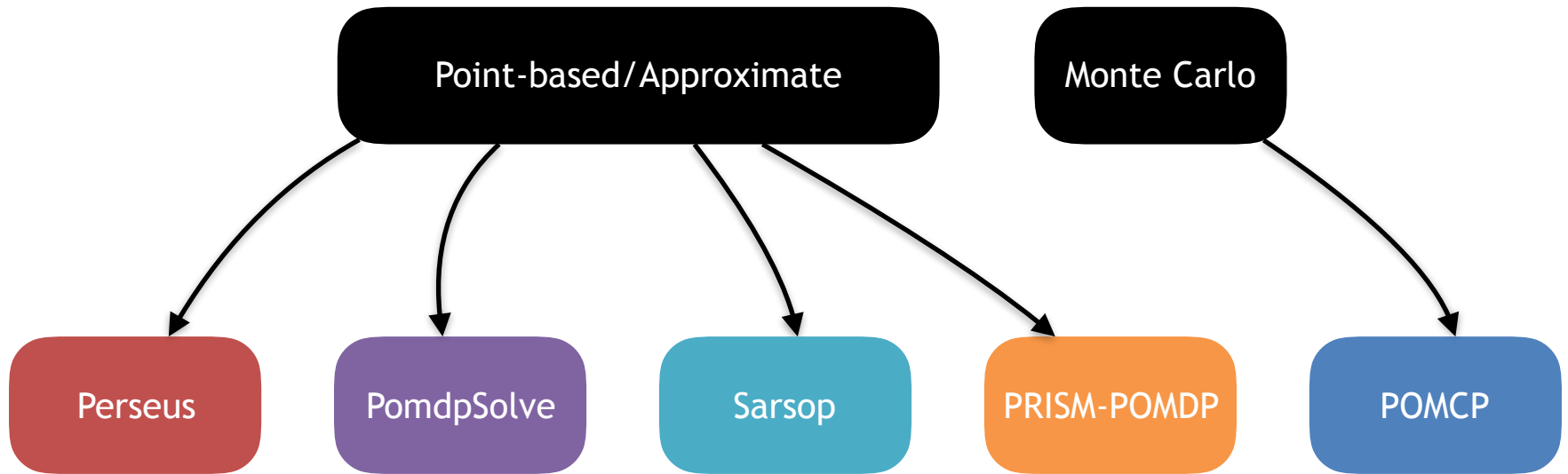


# POMDP Solving - State of The Art

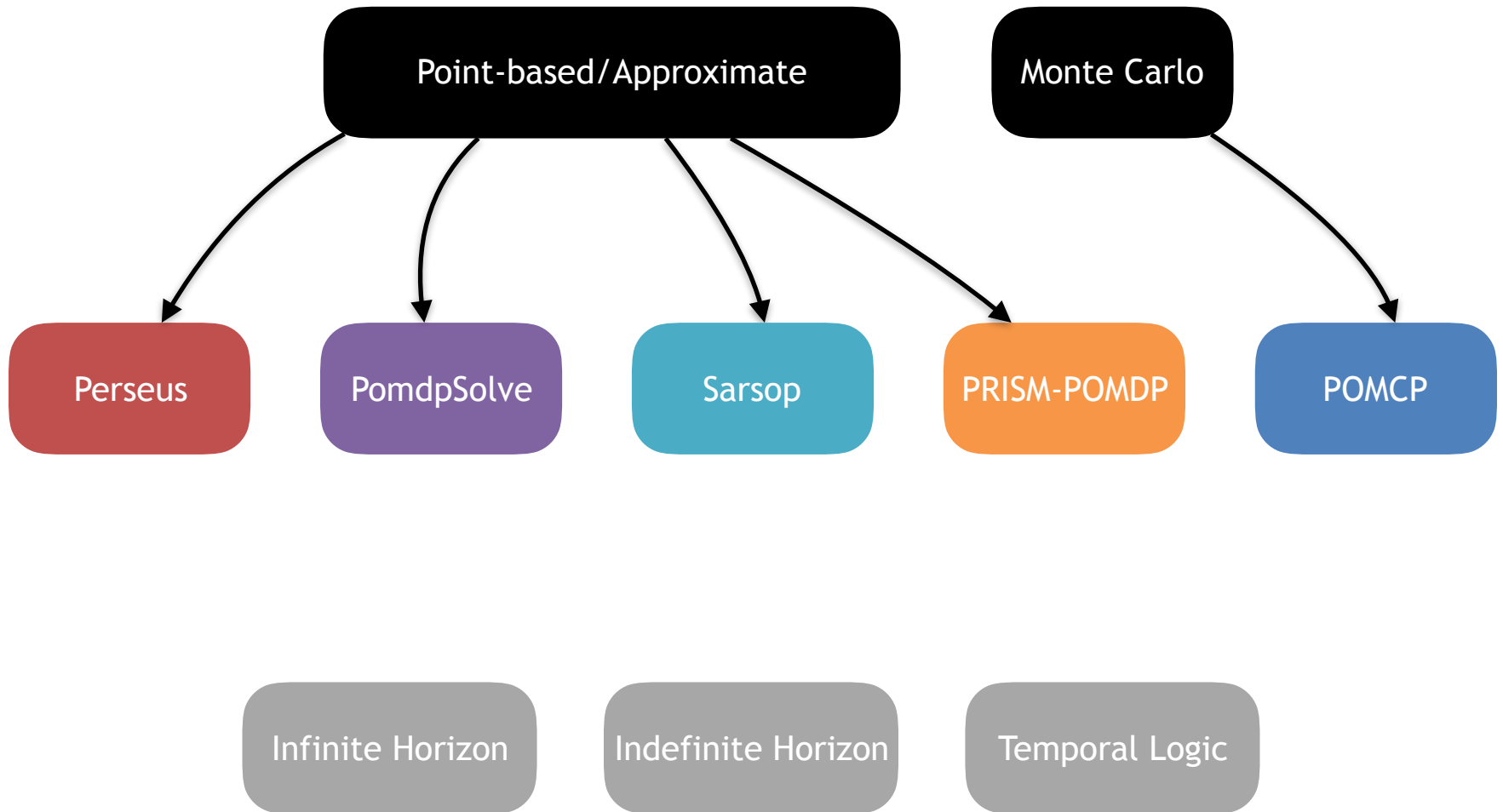
Point-based / Approximate

Monte Carlo

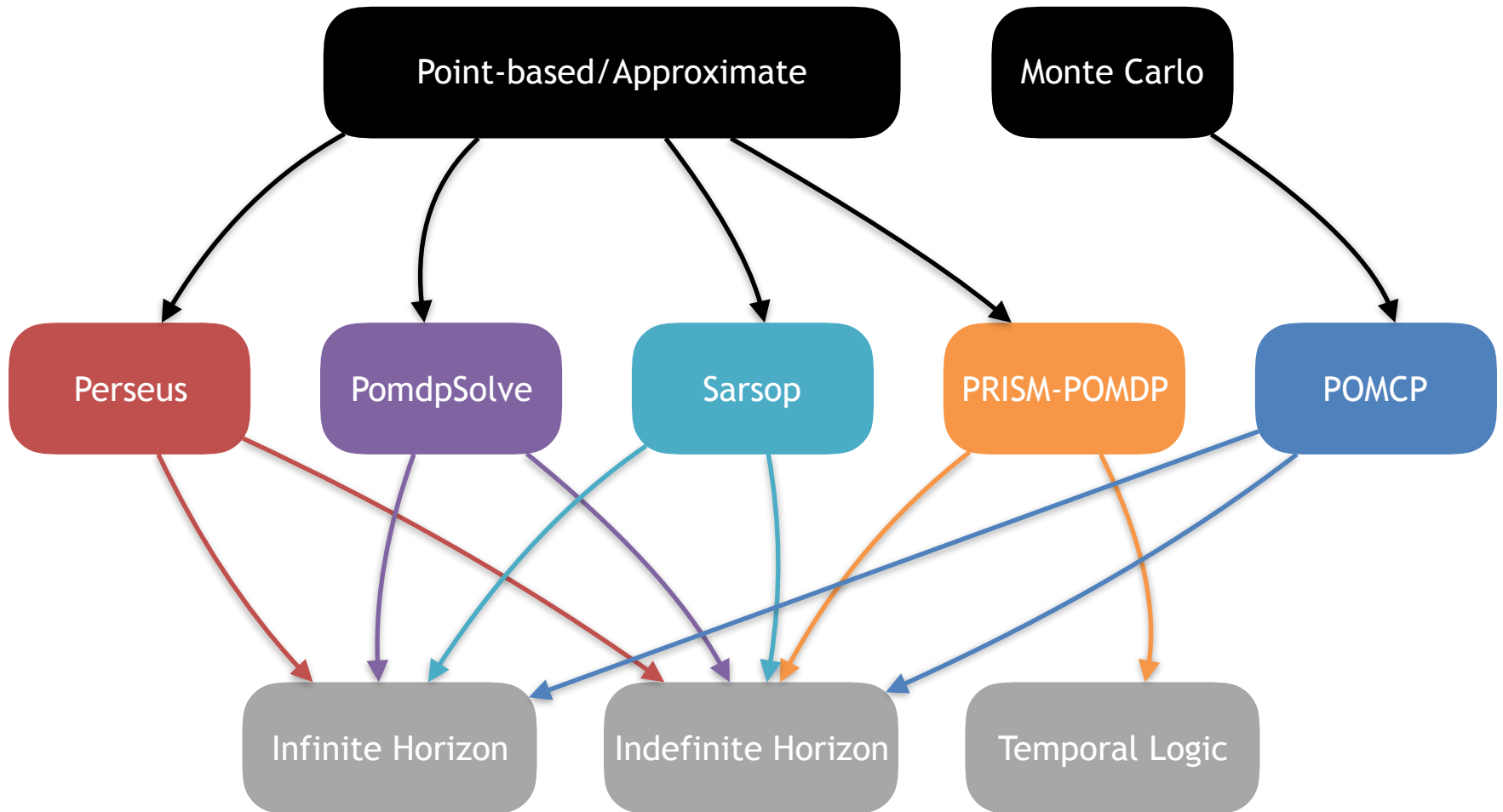
# POMDP Solving - State of The Art



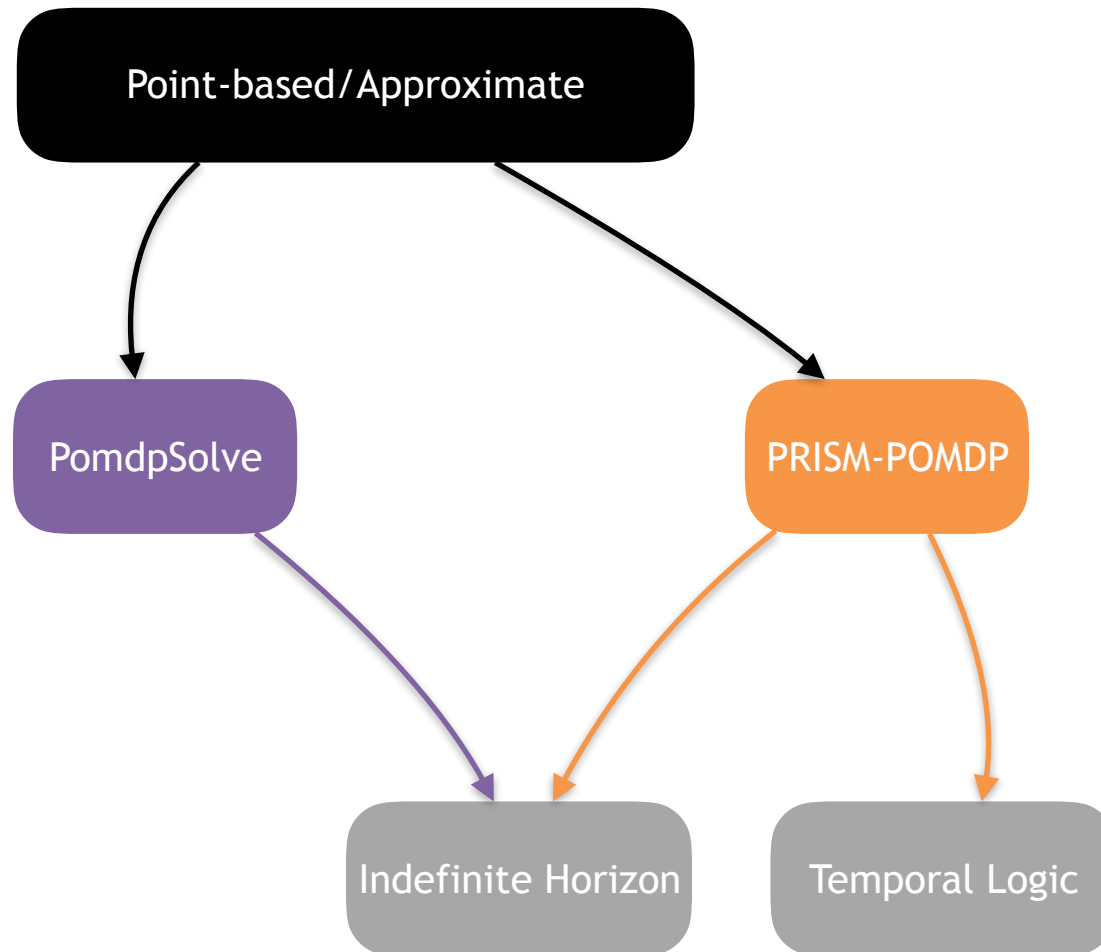
# POMDP Solving - State of The Art



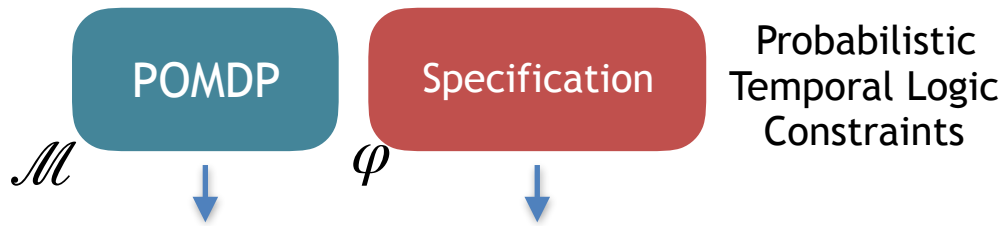
# POMDP Solving - State of The Art



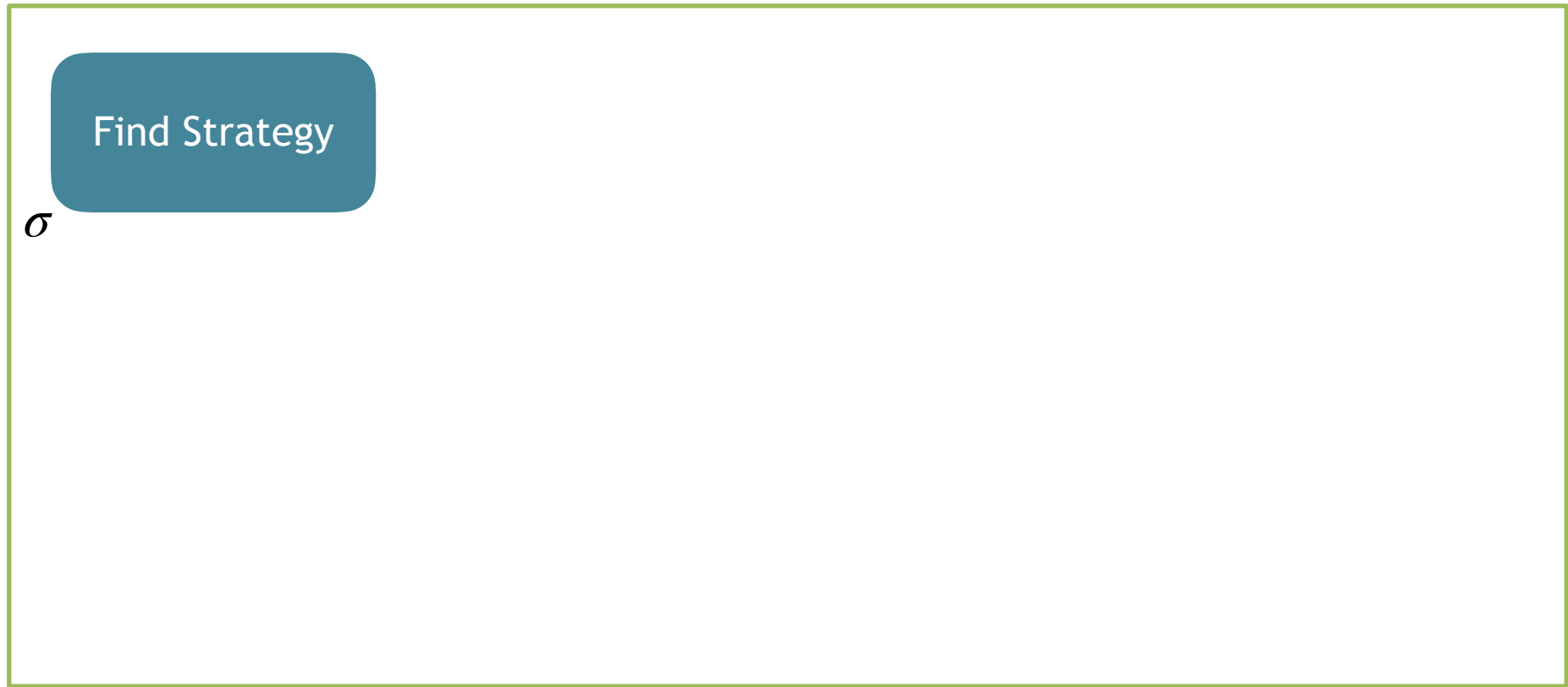
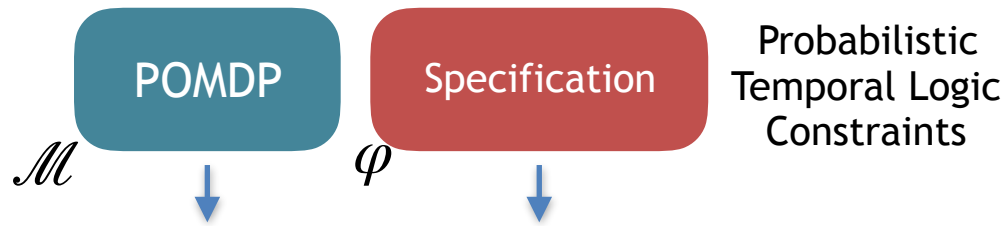
# POMDP Solving - State of The Art



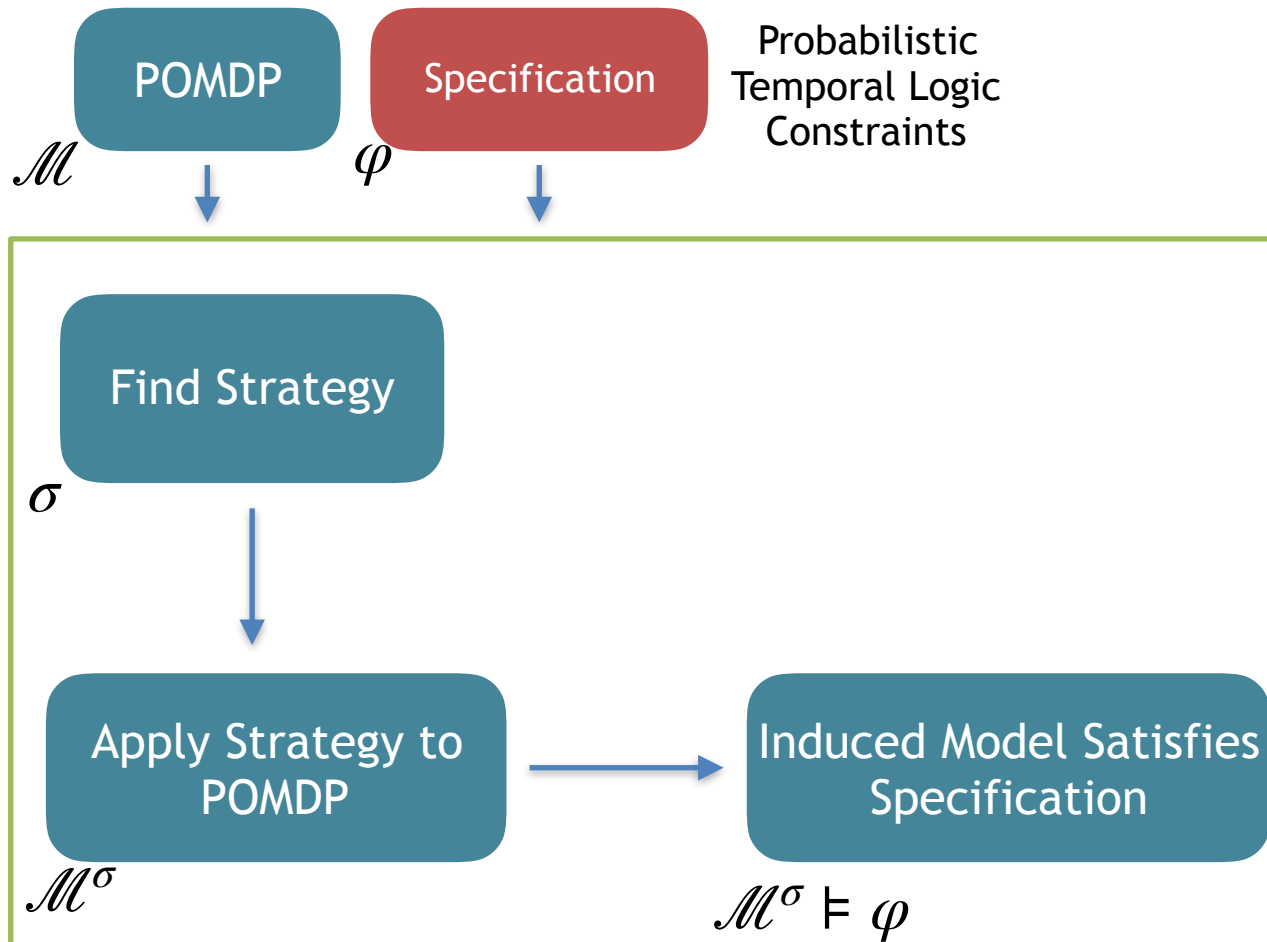
# The Problem: Strategy Synthesis



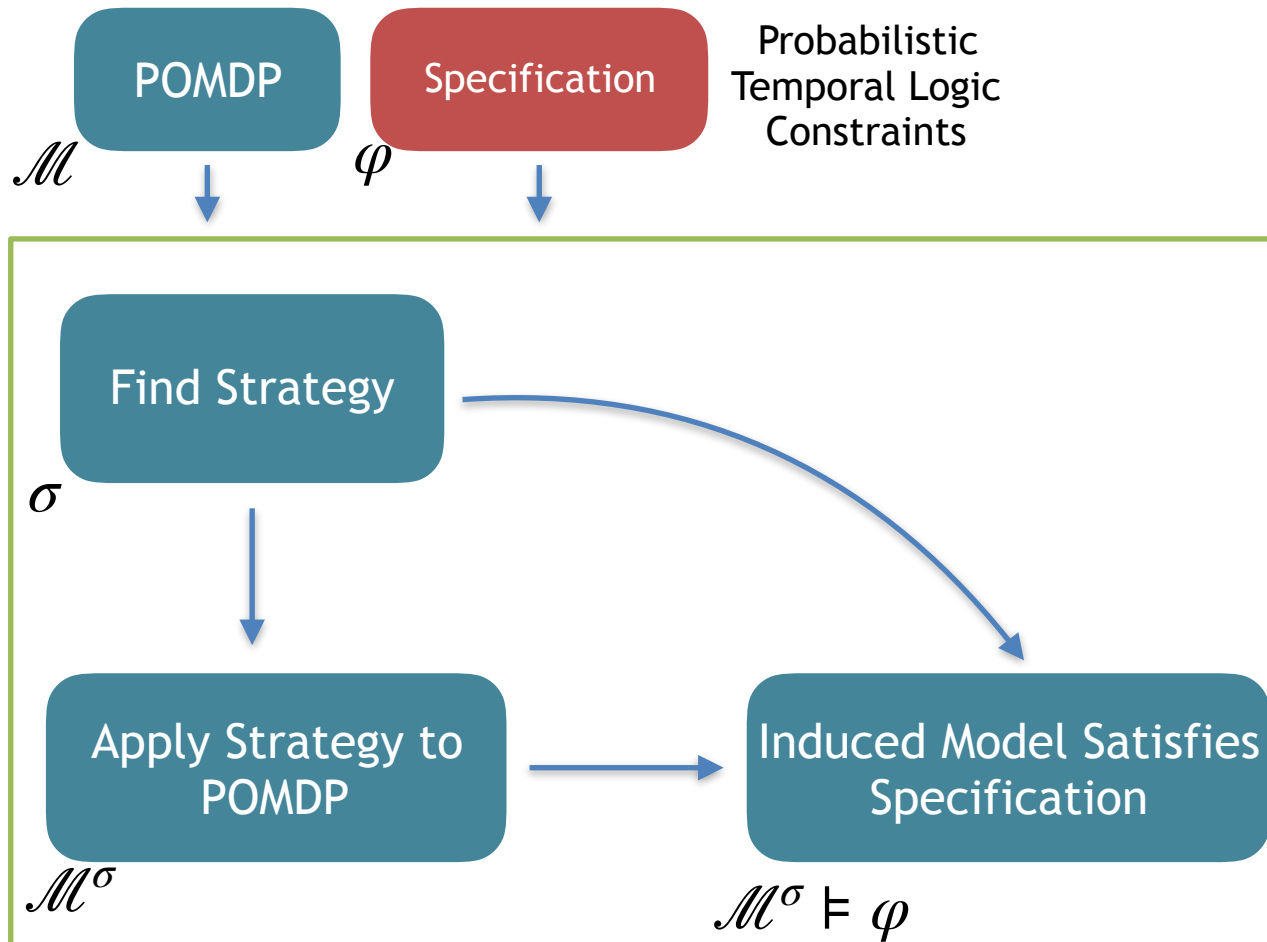
# The Problem: Strategy Synthesis



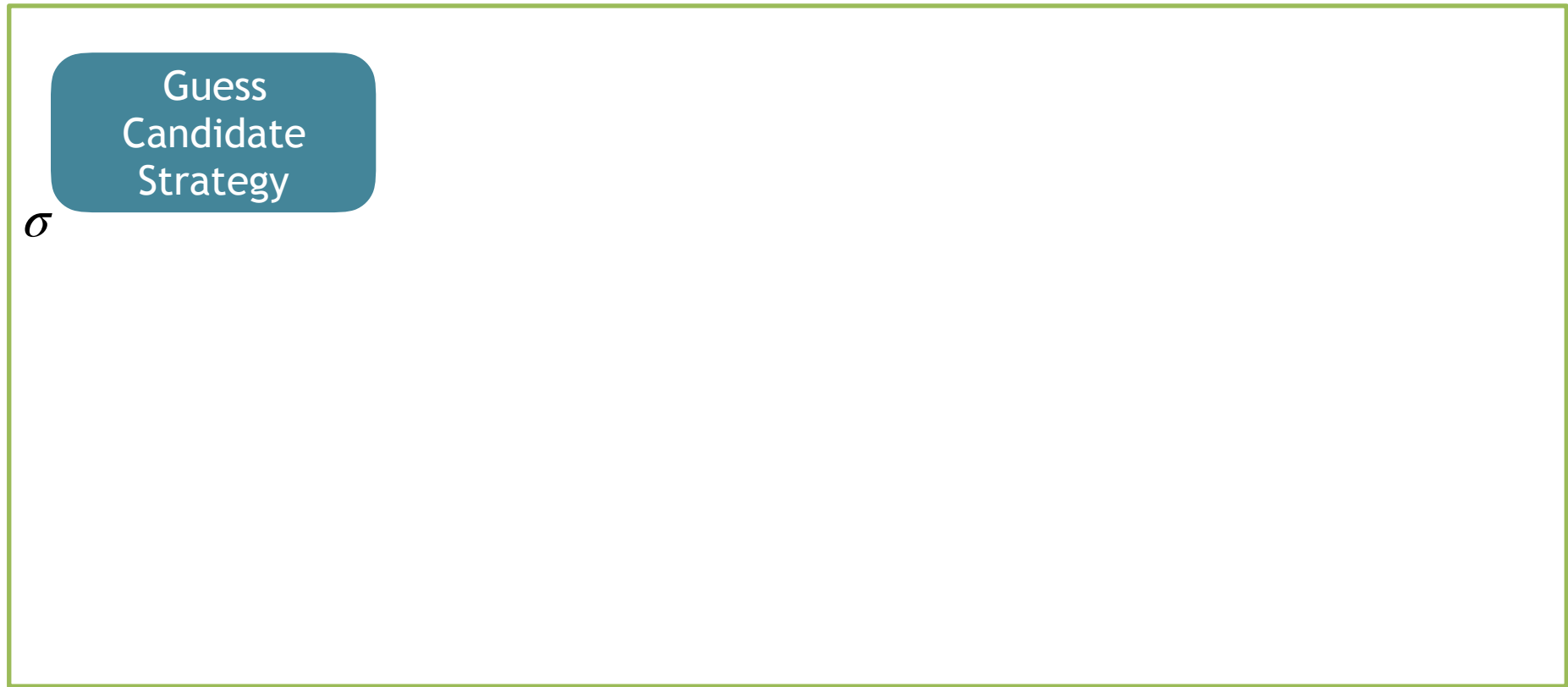
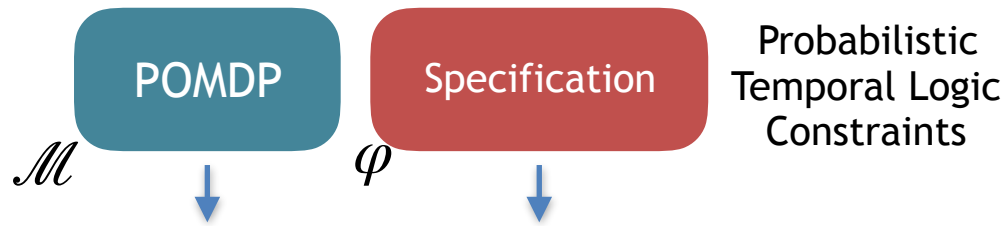
# The Problem: Strategy Synthesis



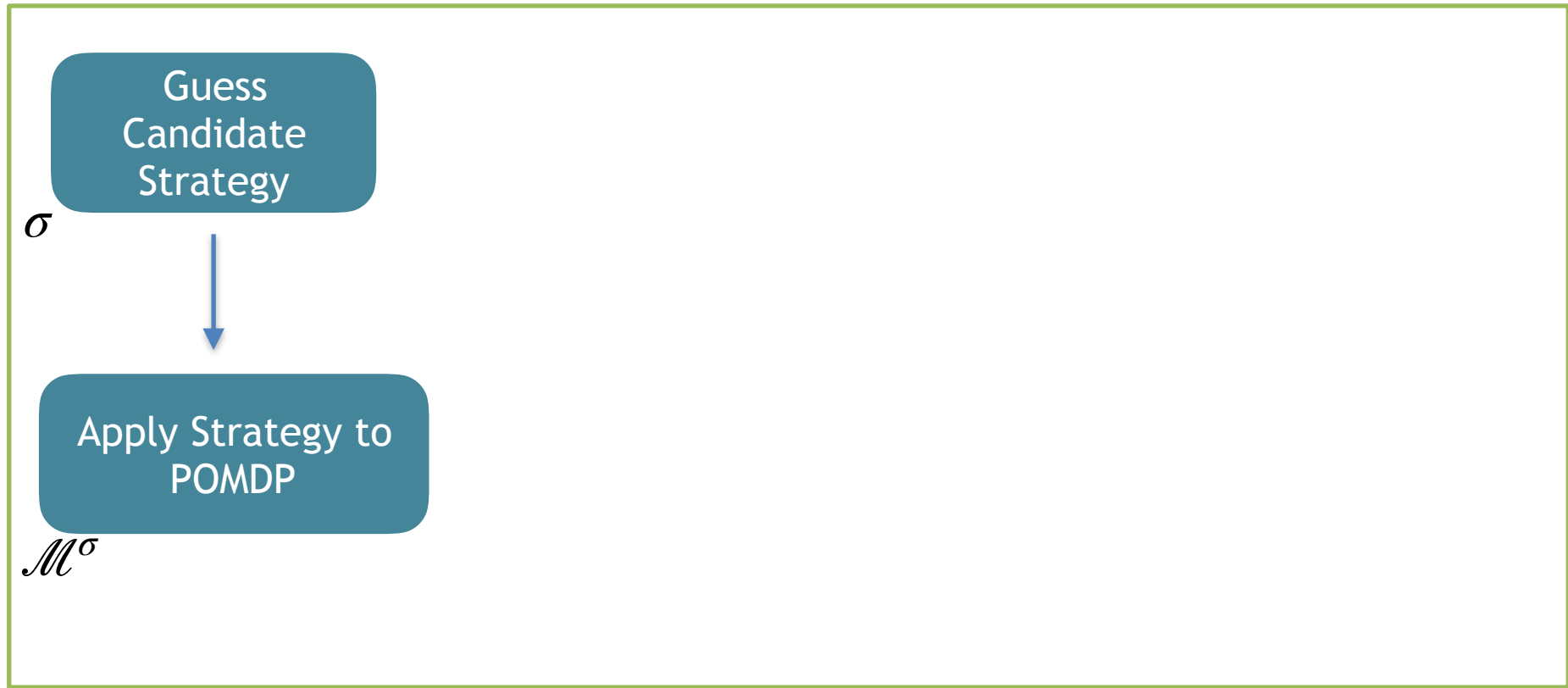
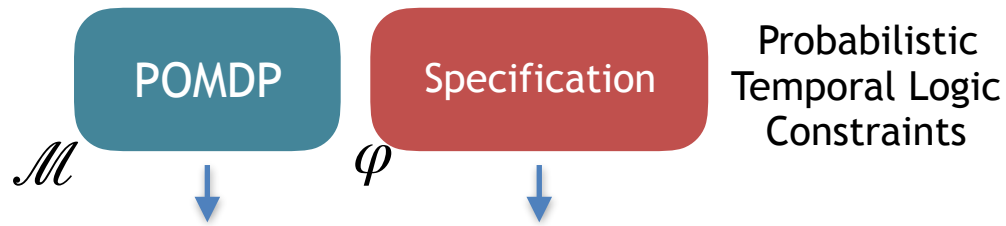
# The Problem: Strategy Synthesis



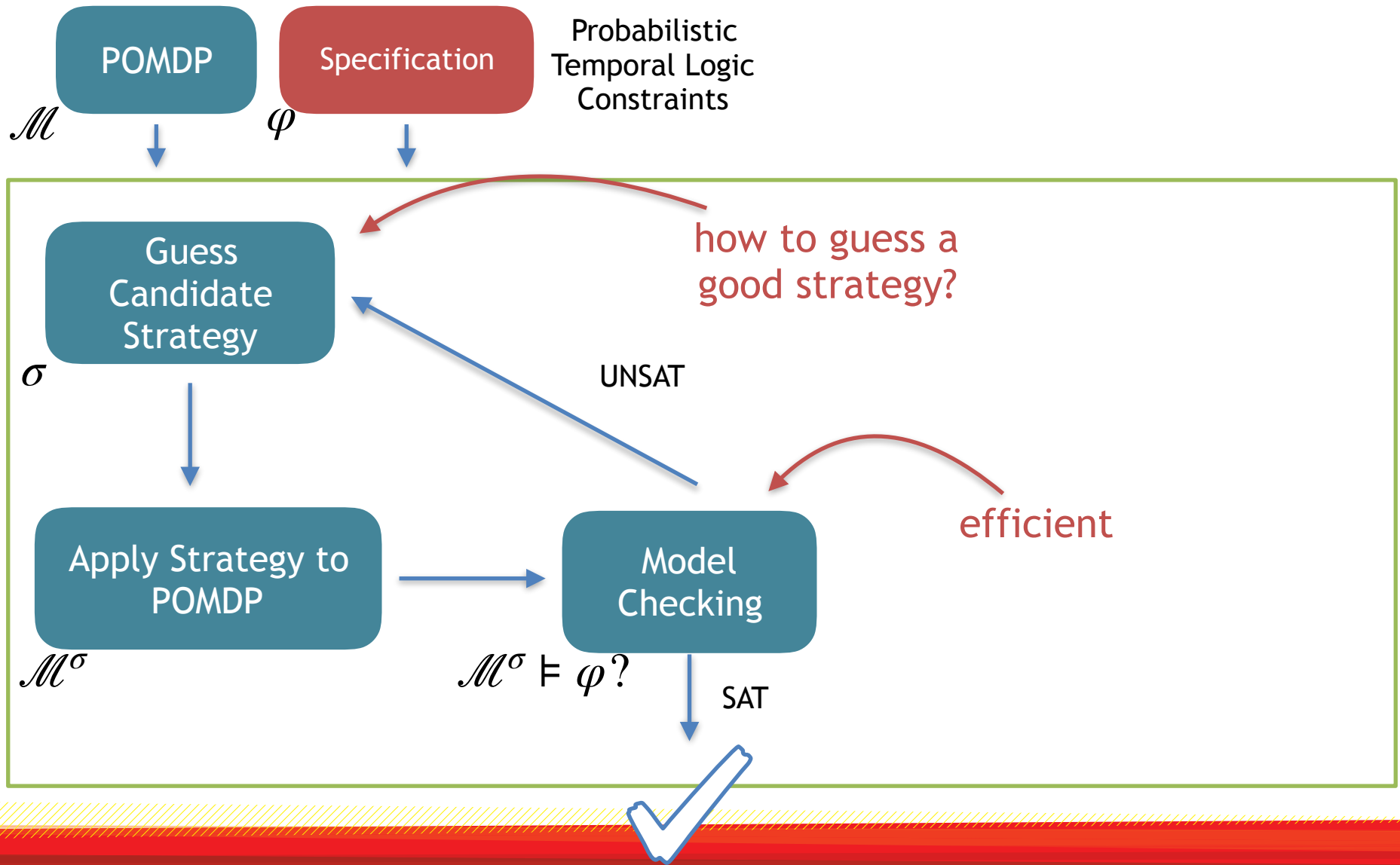
# Be Lazy: Guess a Strategy and Verify!



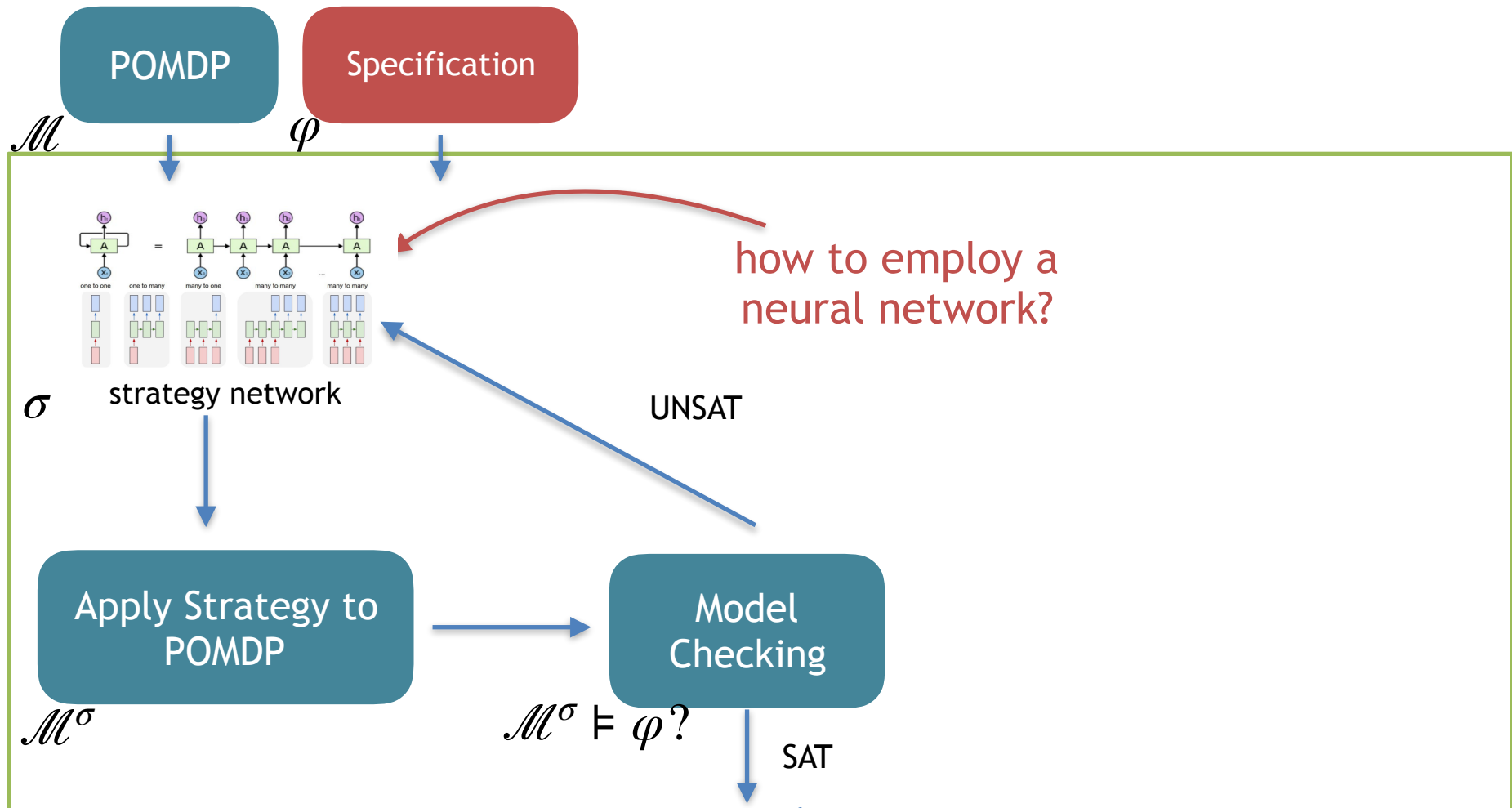
# Be Lazy: Guess a Strategy and Verify!



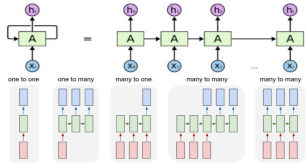
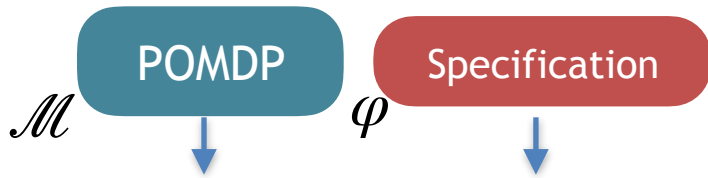
# Be Lazy: Guess a Strategy and Verify!



# Let Machine Learning do the Guessing?

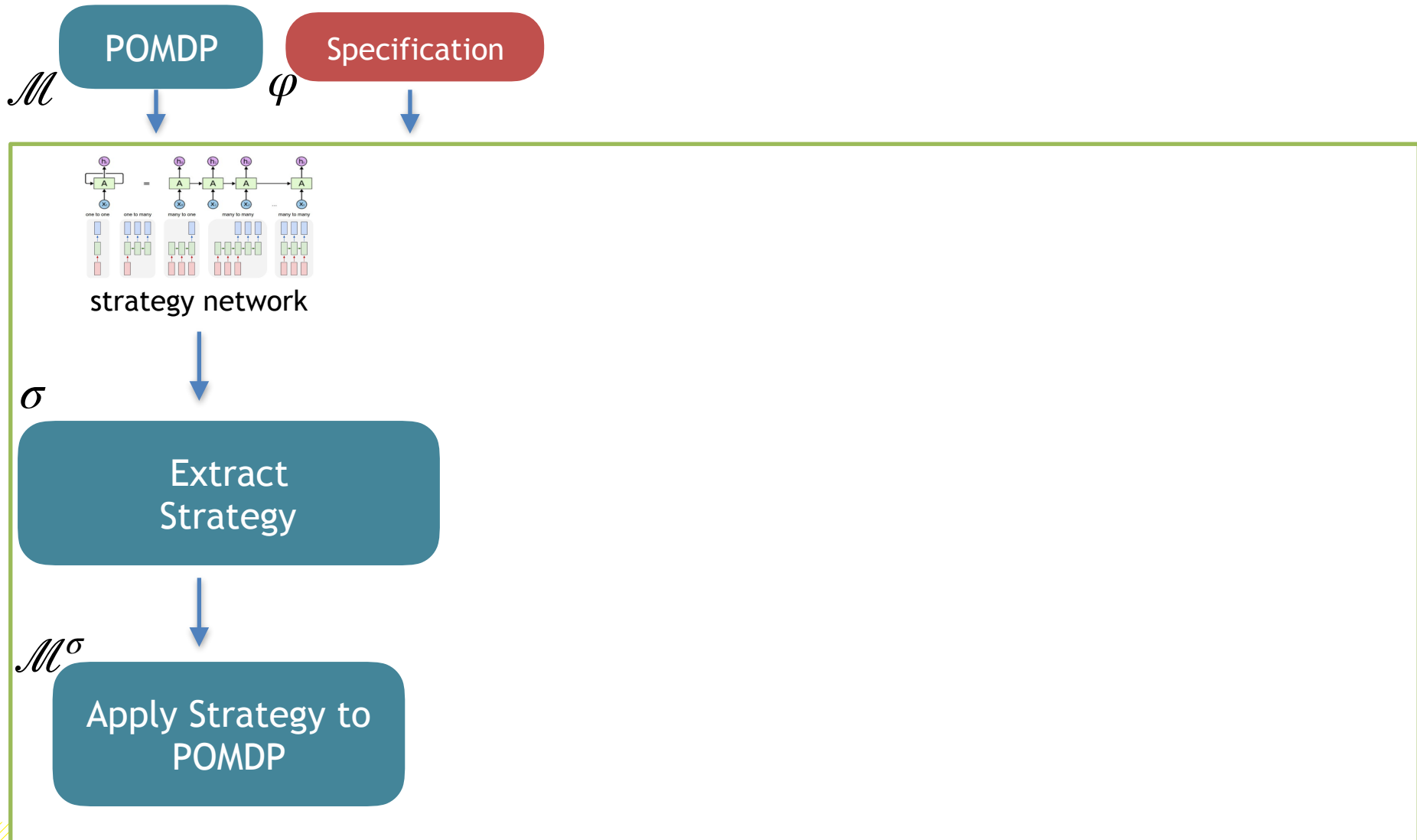


# RNN Strategy Improvement

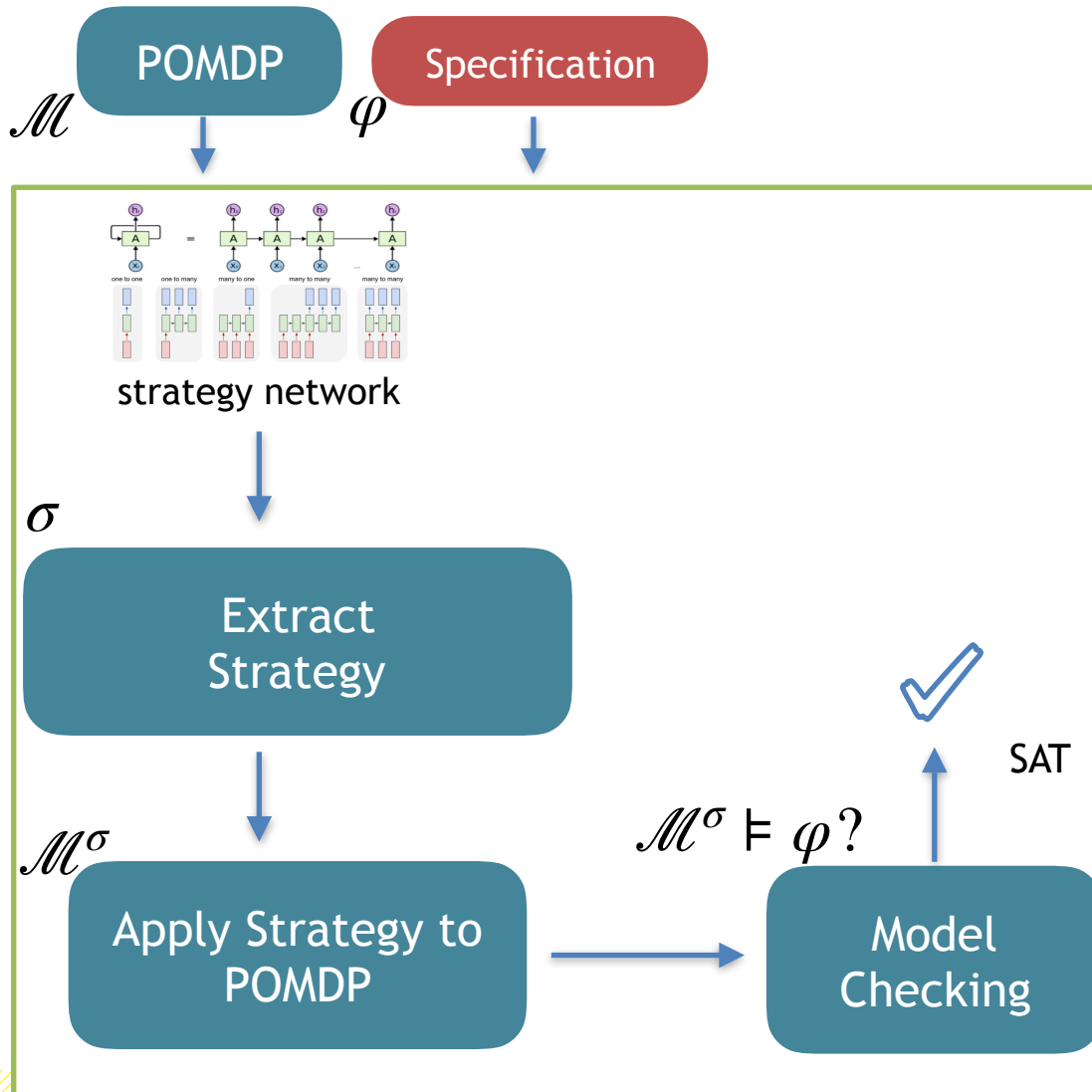


strategy network

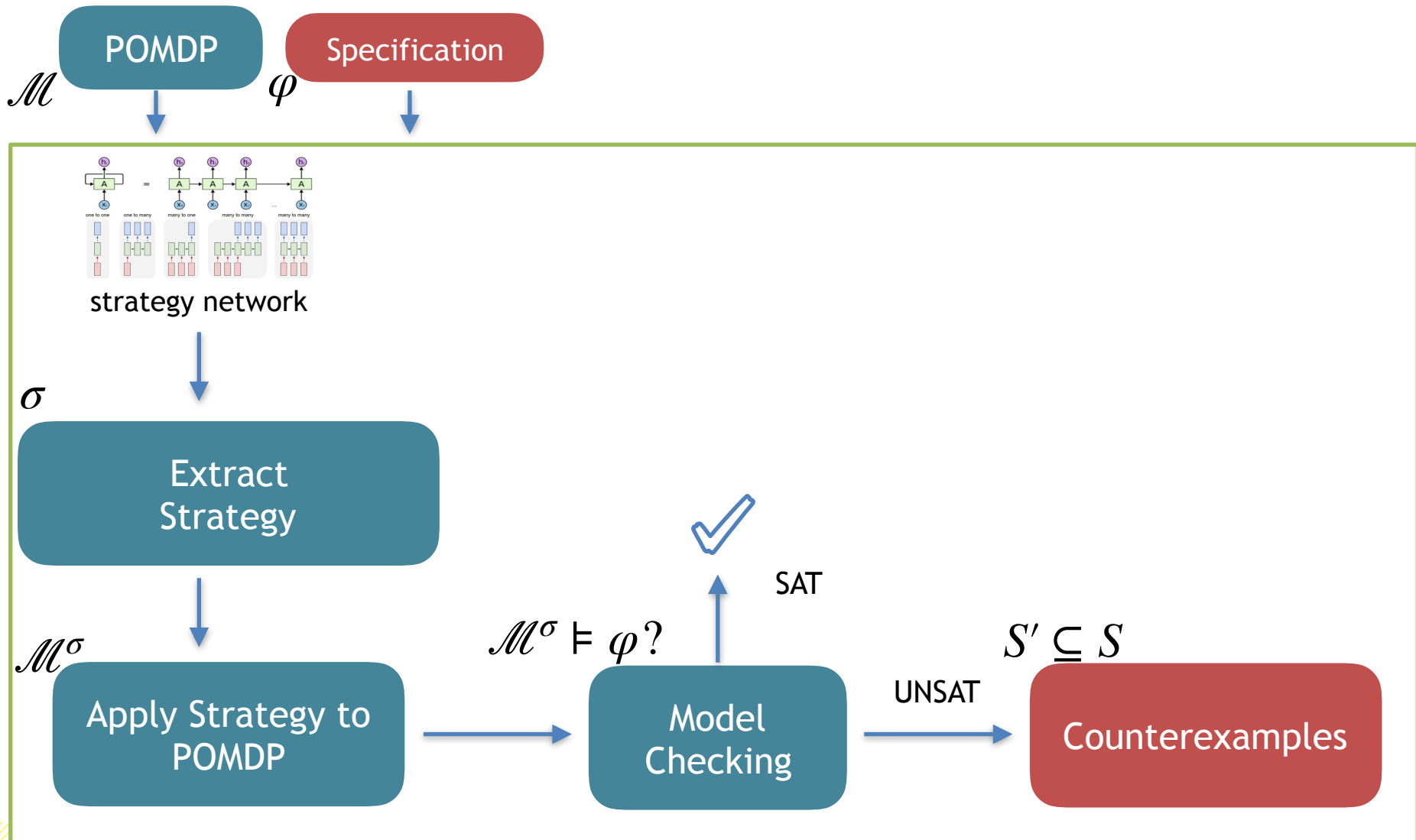
# RNN Strategy Improvement



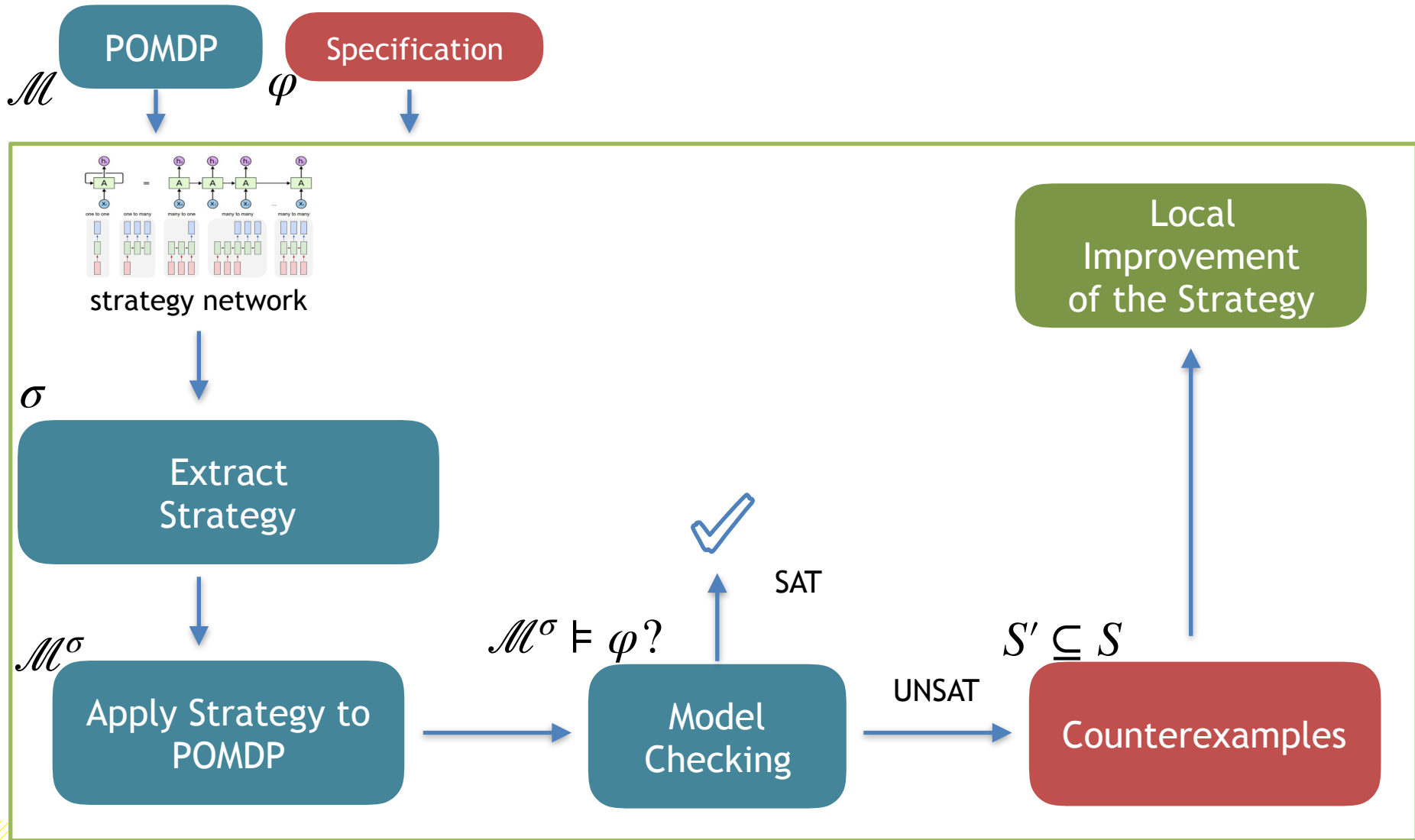
# RNN Strategy Improvement



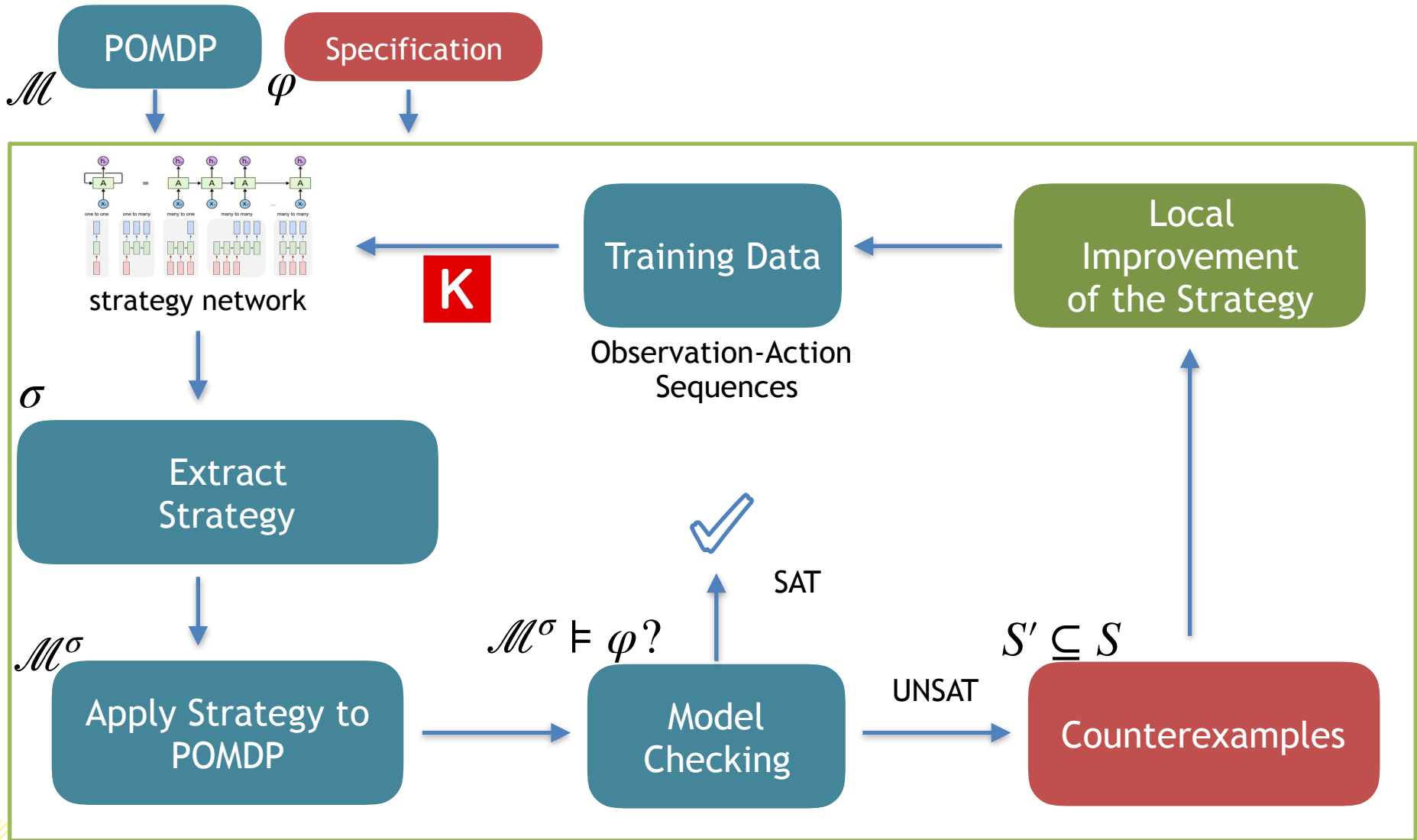
# RNN Strategy Improvement



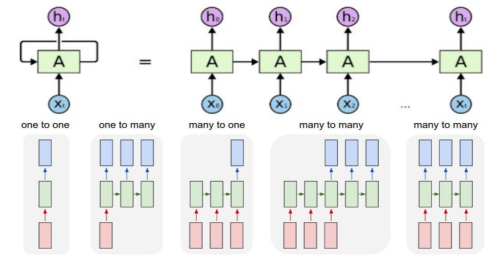
# RNN Strategy Improvement



# RNN Strategy Improvement



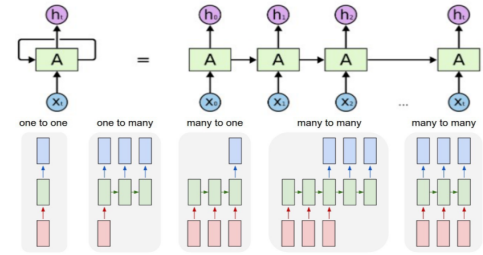
# Learning Strategies with RNNs



# Learning Strategies with RNNs

## Recurrent Neural Network

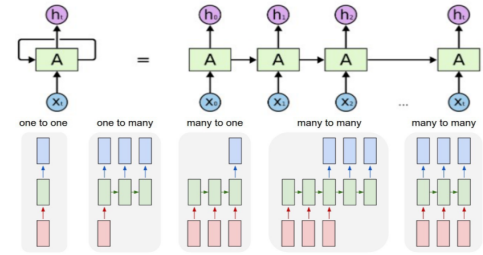
- long short-term memory (LSTM) architecture to learn dependencies in sequential data
- trained with observation-action sequences  $ObsSeq_{fin}$
- strategy network  $\sigma: ObsSeq_{fin} \rightarrow Distr(Act)$
- observations are input labels, actions are output labels



# Learning Strategies with RNNs

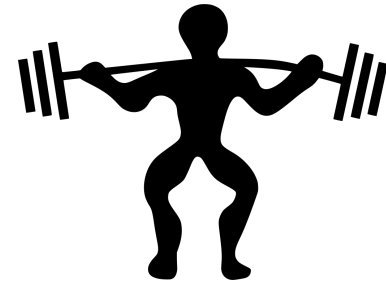
## Recurrent Neural Network

- long short-term memory (LSTM) architecture to learn dependencies in sequential data
- trained with observation-action sequences  $ObsSeq_{fin}$
- strategy network  $\sigma: ObsSeq_{fin} \rightarrow Distr(Act)$
- observations are input labels, actions are output labels



## Initial Training

- compute optimal MDP strategy
- generate (possible) observation-action sequences



# Improving the Strategy

# Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.

# Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.
- For each observation with critical decision, **minimize** the number of different critical actions.

Local linear program

$$\begin{aligned} & \max_{\gamma(z)(a), a \in Act} \min_{s \in S} p_s & (1) \\ \text{subject to} & \\ \forall s \in O^{-1}(z). & \quad p_s = \sum_{a \in Act} \gamma(z)(a) \cdot \sum_{s' \in S} \mathcal{P}(s, a, s') \cdot p^*(s') \end{aligned}$$

# Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.
- For each observation with critical decision, **minimize** the number of different critical actions.
- **Retrain** with the new (locally improved) strategy.

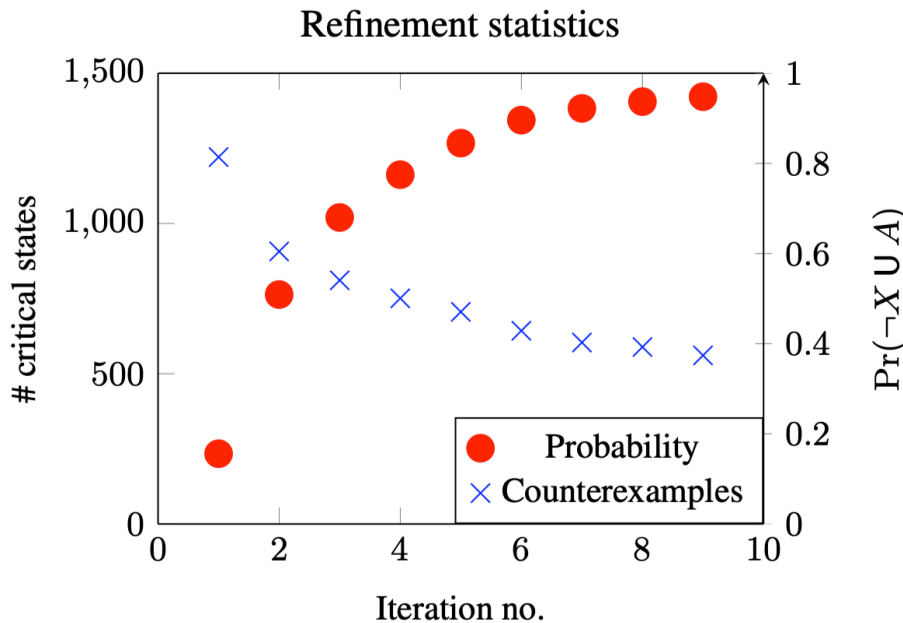
Local linear program

$$\begin{aligned} & \max_{\gamma(z)(a), a \in Act} \min_{s \in S} p_s & (1) \\ \text{subject to} & \\ \forall s \in O^{-1}(z). & \quad p_s = \sum_{a \in Act} \gamma(z)(a) \cdot \sum_{s' \in S} \mathcal{P}(s, a, s') \cdot p^*(s') \end{aligned}$$

# Improving the Strategy

- Identify **critical decisions** that lead to states with high probability of **violating** the specification.
- For each observation with critical decision, **minimize** the number of different critical actions.
- **Retrain** with the new (locally improved) strategy.

Local linear program

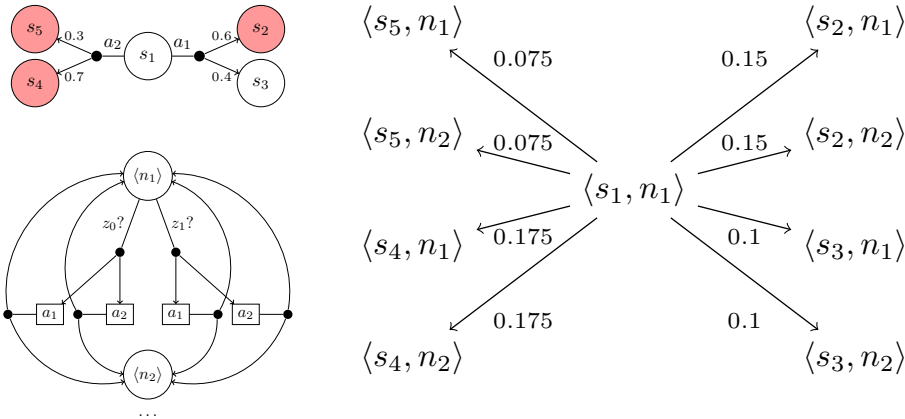


$$\begin{aligned} & \max_{\gamma(z)(a), a \in Act} \min_{s \in S} p_s \\ & \text{subject to} \\ & \forall s \in O^{-1}(z). \quad p_s = \sum_{a \in Act} \gamma(z)(a) \cdot \sum_{s' \in S} \mathcal{P}(s, a, s') \cdot p^*(s') \end{aligned} \quad (1)$$

Even if specification is satisfied, there may be critical states and decisions!

# Finite-memory Strategies (FSC)

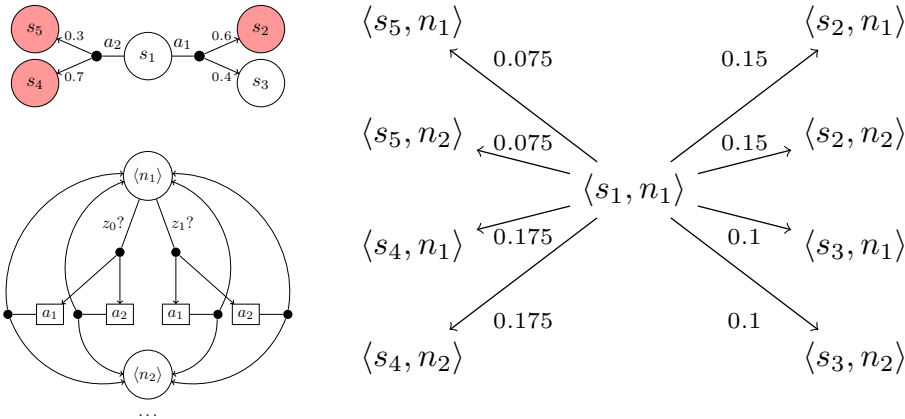
- Encode **finite memory** directly into the state space:



- Strategy network is of the form  $\sigma: ObsSeq_{fin} \rightarrow Distr(Act)$
- But: How to infer a **memory-update function** to construct an FSC?

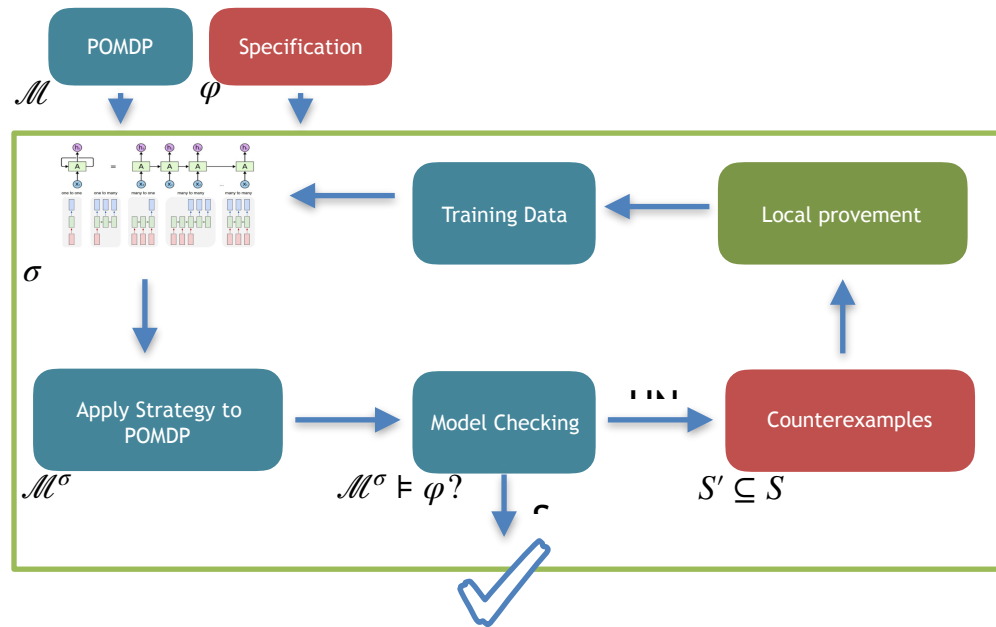
# Finite-memory Strategies (FSC)

- Encode **finite memory** directly into the state space:

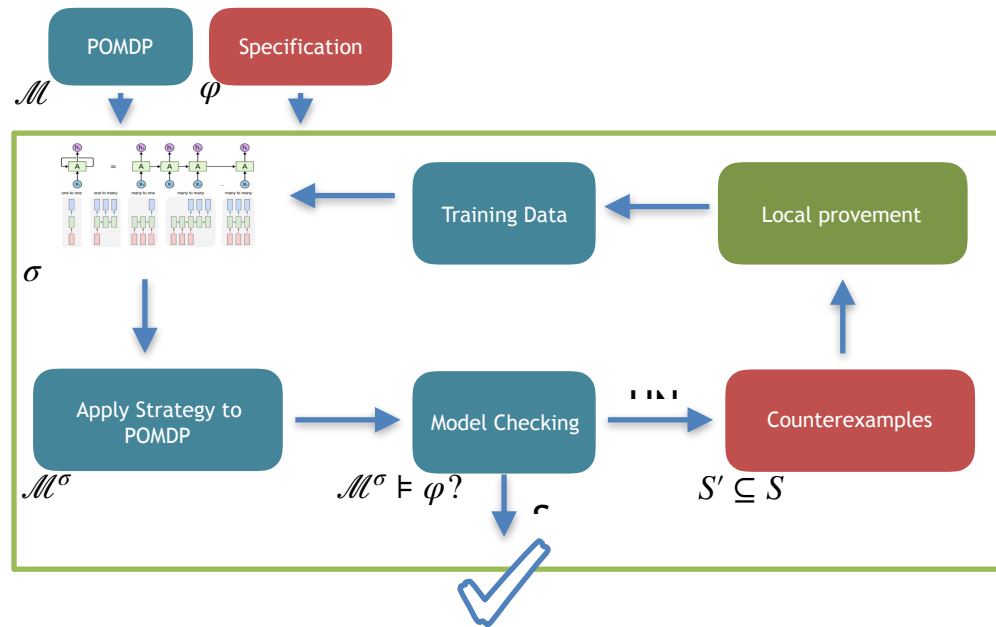


- Strategy network is of the form  $\sigma: ObsSeq_{fin} \rightarrow Distr(Act)$
- But: How to infer a **memory-update function** to construct an FSC?
- **Predefine memory update**, for instance (deterministic) transition upon repetition of an observation.
- Compute product of FSC and POMDP and compute memoryless strategy.

# Correctness and Completeness?

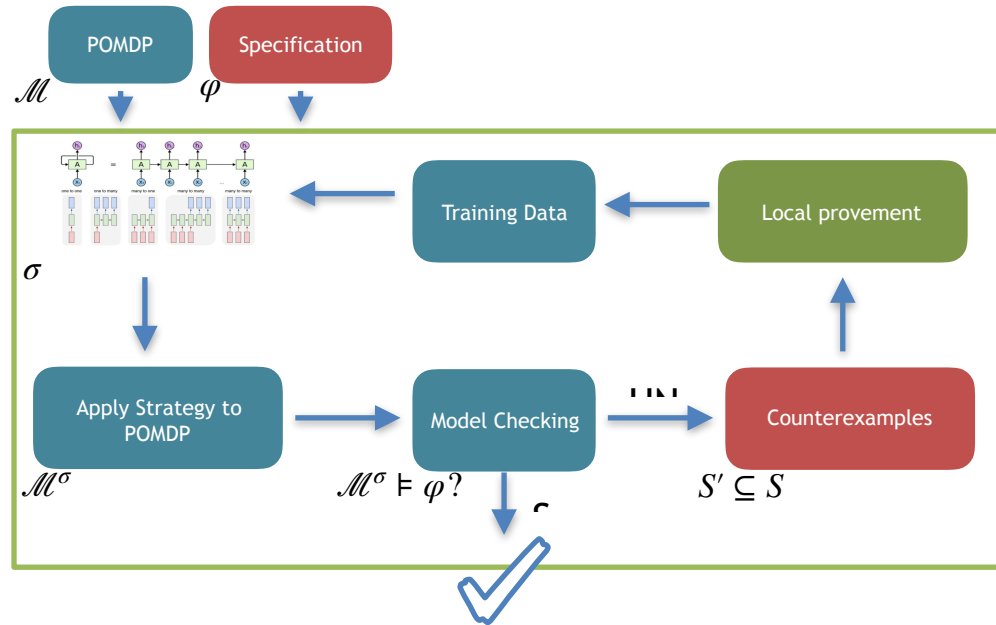


# Correctness and Completeness?



**Correct**, as each strategy prediction is evaluated using model checking.

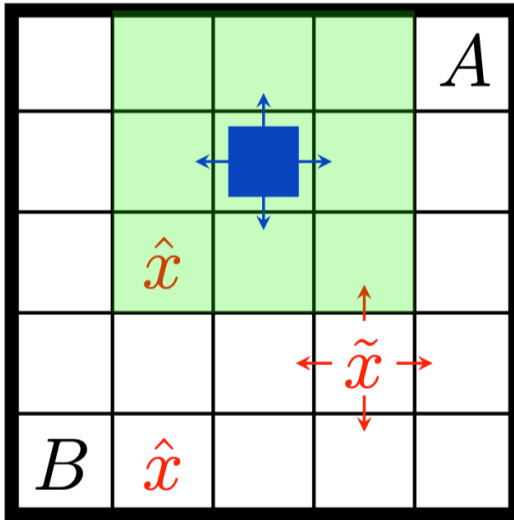
# Correctness and Completeness?



**Correct**, as each strategy prediction is evaluated using model checking.

**Not complete**, as we may never find a feasible strategy. Also, problem is undecidable (or hard) anyways :).

# Experiments - LTL



Problem	$ S $	$ Act $	$ Z $
Navigation ( $c$ )	$c^4$	4	256
Delivery ( $c$ )	$c^2$	4	256
Slippery ( $c$ )	$c^2$	4	256
Maze( $c$ )	$3c + 8$	4	7
Grid( $c$ )	$c^2$	4	2
RockSample[4, 4]	257	9	2
RockSample[5, 5]	801	10	2
RockSample[7, 8]	12545	13	2

Problem	States	Type, $\varphi$	RNN-based Synthesis		PRISM-POMDP	
			Res.	Time (s)	Res.	Time (s)
Navigation (3)	333	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.74	<b>14.16</b>	<b>0.84</b>	73.88
Navigation (4)	1088	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.82	<b>22.67</b>	<b>0.93</b>	1034.64
Navigation (4) [2-FSC]	13373	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.91	47.26	–	–
Navigation (4) [4-FSC]	26741	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.92	59.42	–	–
Navigation (4) [8-FSC]	53477	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.92</b>	85.26	–	–
Navigation (5)	2725	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.91	<b>34.34</b>	MO	MO
Navigation (5) [2-FSC]	33357	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.92	115.16	–	–
Navigation (5) [4-FSC]	66709	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.92	159.61	–	–
Navigation (5) [8-FSC]	133413	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.92</b>	250.91	–	–
Navigation (10)	49060	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.79	<b>822.87</b>	MO	MO
Navigation (10) [2-FSC]	475053	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.83	1185.41	–	–
Navigation (10) [4-FSC]	950101	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.85</b>	1488.77	–	–
Navigation (10) [8-FSC]	1900197	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	0.81	1805.22	–	–
Navigation (15)	251965	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.91</b>	<b>1271.80*</b>	MO	MO
Navigation (20)	798040	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.96</b>	<b>4712.25*</b>	MO	MO
Navigation (30)	4045840	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	<b>0.95</b>	<b>25191.05*</b>	MO	MO
Navigation (40)	–	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_1$	TO	TO	MO	MO
Delivery (4) [2-FSC]	80	$\mathbb{E}_{\min}^{\mathcal{M}}, \varphi_2$	6.02	35.35	<b>6.0</b>	<b>28.53</b>
Delivery (5) [2-FSC]	125	$\mathbb{E}_{\min}^{\mathcal{M}}, \varphi_2$	8.11	<b>78.32</b>	<b>8.0</b>	102.41
Delivery (10) [2-FSC]	500	$\mathbb{E}_{\min}^{\mathcal{M}}, \varphi_2$	<b>18.13</b>	<b>120.34</b>	MO	MO
Slippery (4) [2-FSC]	460	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_3$	0.78	67.51	<b>0.90</b>	<b>5.10</b>
Slippery (5) [2-FSC]	730	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_3$	0.89	84.32	<b>0.93</b>	<b>83.24</b>
Slippery (10) [2-FSC]	2980	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_3$	<b>0.98</b>	<b>119.14</b>	MO	MO
Slippery (20) [2-FSC]	11980	$\mathbb{P}_{\max}^{\mathcal{M}}, \varphi_3$	<b>0.99</b>	<b>1580.42</b>	MO	MO

# Experiments - Standard POMDPs

Problem	Type	RNN-based Synthesis			PRISM-POMDP		pomdpSolve	
		States	Res	Time (s)	Res	Time (s)	Res	Time (s)
Maze (1)	$E_{\min}^{\mathcal{M}}$	68	4.31	31.70	<b>4.30</b>	<b>0.09</b>	4.30	0.30
Maze (2)	$E_{\min}^{\mathcal{M}}$	83	5.31	46.65	5.23	2.176	<b>5.23</b>	<b>0.67</b>
Maze (3)	$E_{\min}^{\mathcal{M}}$	98	8.10	58.75	7.13	38.82	<b>7.13</b>	<b>2.39</b>
Maze (4)	$E_{\min}^{\mathcal{M}}$	113	11.53	58.09	8.58	543.06	<b>8.58</b>	<b>7.15</b>
Maze (5)	$E_{\min}^{\mathcal{M}}$	128	14.40	<b>68.09</b>	13.00	4110.50	<b>12.04</b>	132.12
Maze (6)	$E_{\min}^{\mathcal{M}}$	143	22.34	<b>71.89</b>	MO	MO	<b>18.52</b>	1546.02
Maze (10)	$E_{\min}^{\mathcal{M}}$	203	100.21	<b>158.33</b>	MO	MO	MO	MO
Grid (3)	$E_{\min}^{\mathcal{M}}$	165	2.90	38.94	2.88	2.332	<b>2.88</b>	<b>0.07</b>
Grid (4)	$E_{\min}^{\mathcal{M}}$	381	4.32	79.99	4.13	1032.53	<b>4.13</b>	<b>0.77</b>
Grid (5)	$E_{\min}^{\mathcal{M}}$	727	6.623	91.42	MO	MO	<b>5.42</b>	<b>1.94</b>
Grid (10)	$E_{\min}^{\mathcal{M}}$	5457	<b>13.630</b>	<b>268.40</b>	MO	MO	MO	MO
RockSample[4, 4]	$E_{\max}^{\mathcal{M}}$	2432	17.71	35.35	N/A	N/A	<b>18.04</b>	<b>0.43</b>
RockSample[5, 5]	$E_{\max}^{\mathcal{M}}$	8320	18.40	<b>43.74</b>	N/A	N/A	<b>19.23</b>	621.28
RockSample[7, 8]	$E_{\max}^{\mathcal{M}}$	166656	20.32	<b>860.53</b>	N/A	N/A	<b>21.64</b>	20458.41

# Conclusion

- Novel way to generate **provably correct** POMDP strategies
- **Good** scalability, **not optimal**
- Results **transferrable**
- Future work: **More principled** approach to finite-memory strategies → Extract FSC directly from RNN

**Counterexample-Guided Strategy Improvement for POMDPs Using RNNs**  
Steven Carr, Nils Jansen, Ralf Wimmer, Alexandru Serban, Bernd Becker, and Ufuk Topcu

**MOTIVATION**

- Synthesizing POMDP strategies is very complex, especially under constraints → verifying a given strategy is less so
- **Machine learning** generates a candidate strategy
- **Formal verification** certifies its quality against temporal logic constraints

**PROCESS FLOW**

**EXPERIMENTAL EXAMPLES**

- **Gridworld** with moving obstacles
- **Temporal logic** constraints

$\phi_1 = R_{max}(\neg X \cup A)$   
 $\phi_2 = E_{max}(A \wedge B)$

**NEURAL NETWORK DESIGN**

- LSTM architecture
- **Input:** Finite observation sequences
- **Output:** Probability distribution over action-space
- **Trained** from initial strategy guess

**Problem Statement:** For a POMDP  $\mathcal{M}$  and specification  $\phi$ , determine finite-memory strategy  $\gamma$  such that  $\mathcal{M}^\gamma \models \phi$

**REFINEMENT METRICS**

- With data from **counterexamples**, strategies iteratively improve

**EMPIRICAL EVALUATION**

- Outperforms existing POMDP solvers (PRESM-POMDP and pomdpSolve)

Spec/Problem	RNN-based Synthesis		PRESM-POMDP/pomdpSolve	
	States	Relevant States (%)	Mem.	Time (s)
$\phi_1$	256	0.92	81.29	0.93 1034.64
$\phi_2$	$10^6$	0.85	1488.37	AMD-AMD
$\phi_3$	$81.1 \times 10^6$	0.85	$2.51 \times 10^6$	AMD-AMD
$\phi_4$	125	8.11	78.12	8.0 102.41
$\phi_5$	500	18.10	120.34	AMD-AMD
Mem(5)	120	14.40	60.00	11.08 131.12
Mem(10)	203	100.21	138.33	AMD-AMD
grid(10)	5407	13.63	158.48	AMD-AMD

**REFERENCES**

[1] Geffrin Norman et al. Verification and control of partially observable probabilistic systems. Real-Time Systems, 53(3):354–402, 2017.  
[2] Matthew Hauswirth and Peter Stone. Deep recurrent reinforcement learning for partially observable MDPs. CoRR, abs/1507.06527, 2015.  
[3] Erwin Wolpert and Matthew Spaniol. Accelerated vector pruning for optimal POMDP solvers. In AAAI, pages 3672–3678. AAAI Press, 2017.  
[4] Dean Woelcke et al. Solving deep-memory POMDPs with recurrent policy gradients. In ICANN, pages 657–706. Springer, 2007.

