Widest Paths and Global Propagation in Bounded Value Iteration for Stochastic Games

Kittiphon Phalakarn[1], Toru Takisaka[2],

Thomas Haas[3], Ichiro Hasuo[2,4]

[1] University of Waterloo, Waterloo, Canada
[2] National Institute of Informatics, Tokyo, Japan
[3] Technical University of Braunschweig, Braunschweig, Germany
[4] The Graduate University for Advanced Studies (SOKENDAI), Tokyo, Japan

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST; I.H. is supported by Grant-in-Aid No. 15KT0012, JSPS.

Summary

We introduce a novel algorithm of
Bounded Value Iteration (BVI) for Stochastic Games.

<u>What is BVI?</u>

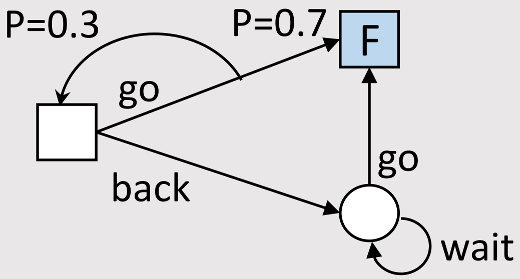
- Approximation technique for reachability
- Approximation with precision guarantee
  - "Compute reachability prob. with 0.01% error range"

<u>Our contribution: faster algorithm</u>

- Existing algorithm [Kelmendi+, CAV'18] requires end component computation
- We omit it by doing global propagation

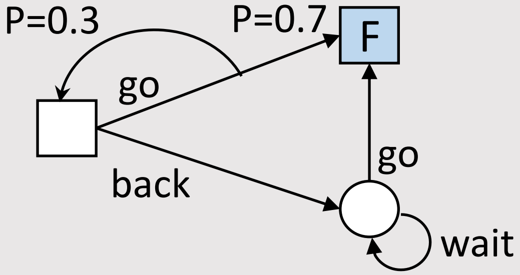
Our model: Stochastic Game (SG)

- A probabilistic system with controller and adversary
- Discrete time, finite states / actions
- Reachability objective



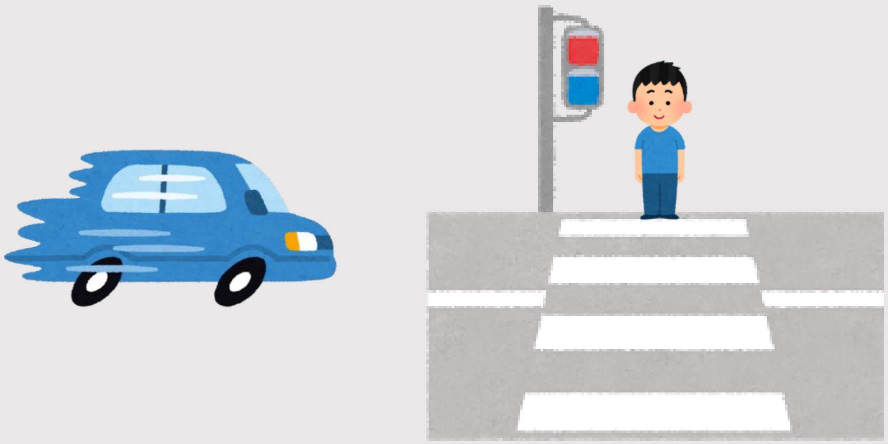
Our model: Stochastic Game (SG)

- A probabilistic system with controller and adversary
- Discrete time, finite states / actions
- Reachability objective

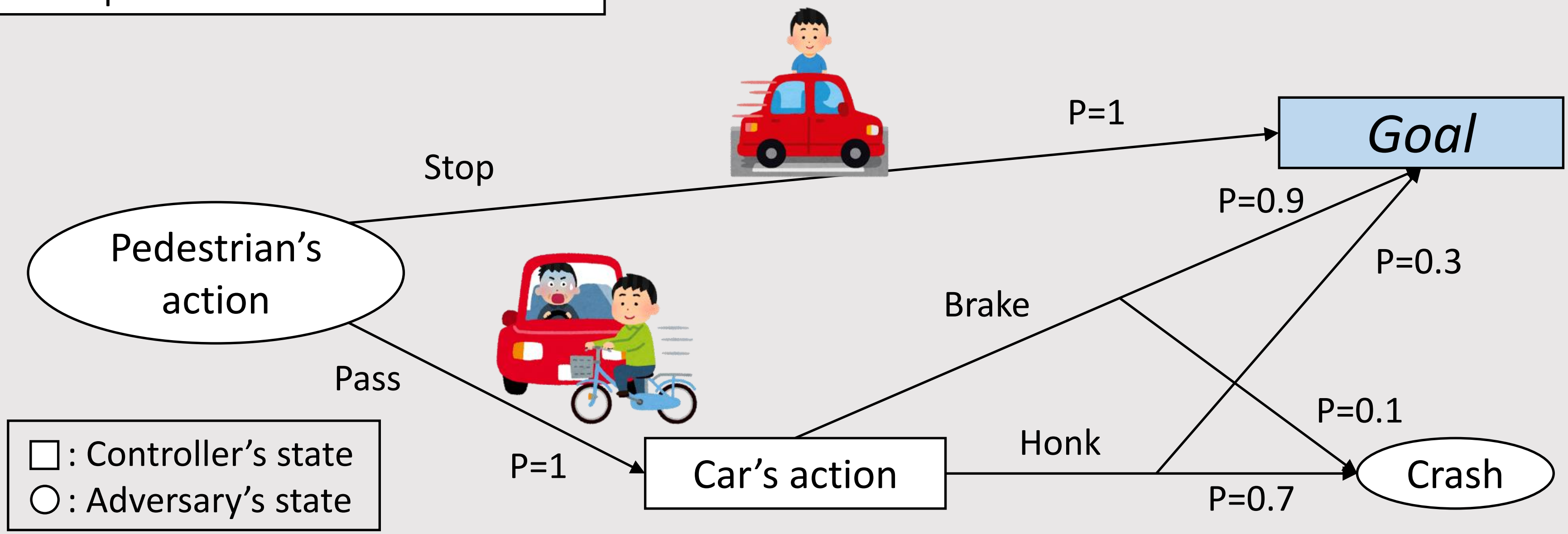


Example: car vs. pedestrian

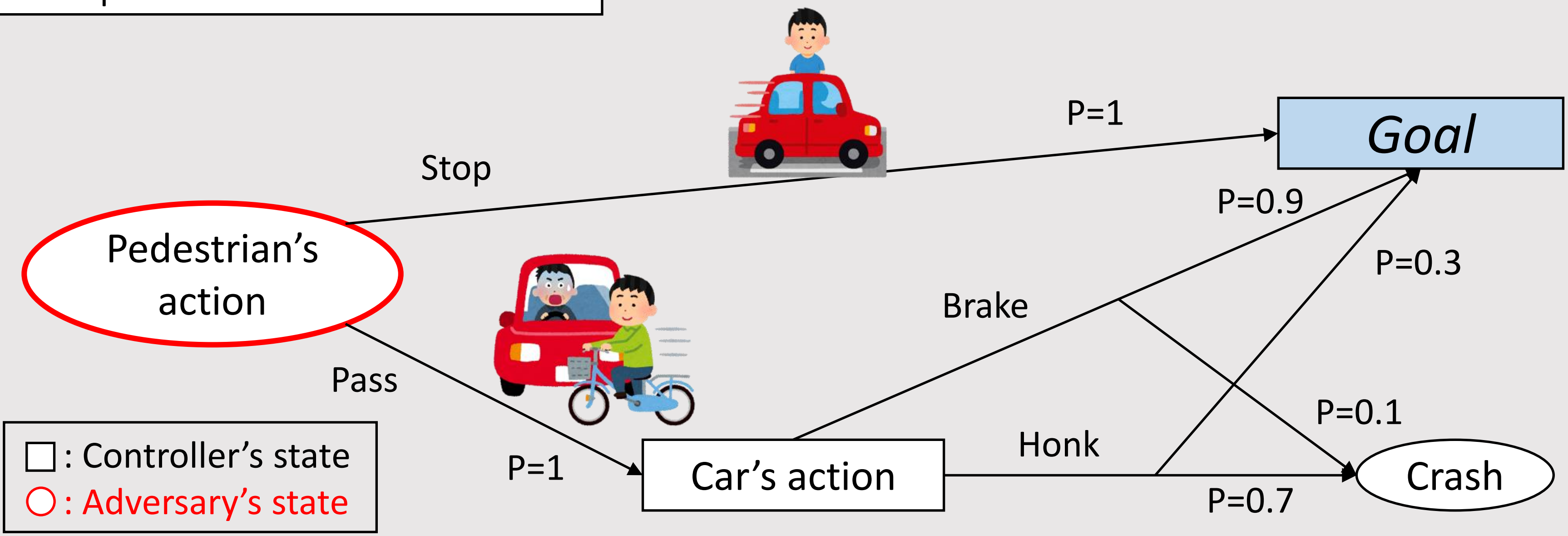
- The car (controller) would like to pass the crossroad without hitting the pedestrian (adversary)



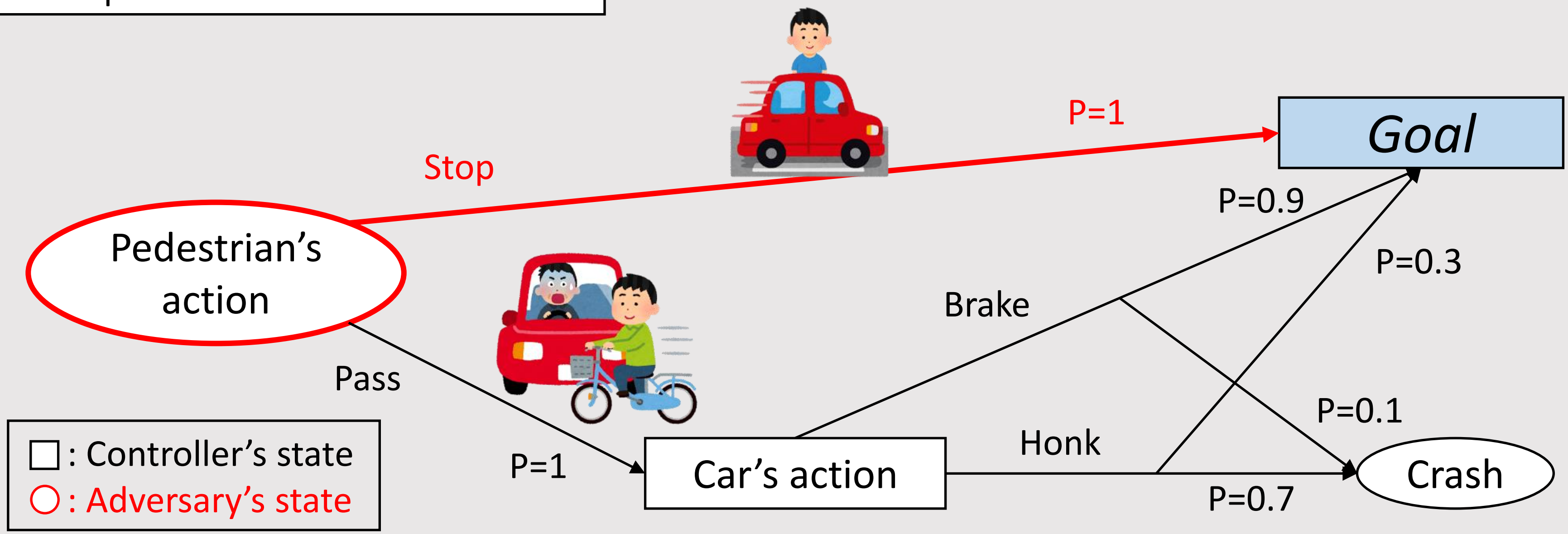
Car vs. pedestrian in Stochastic Game



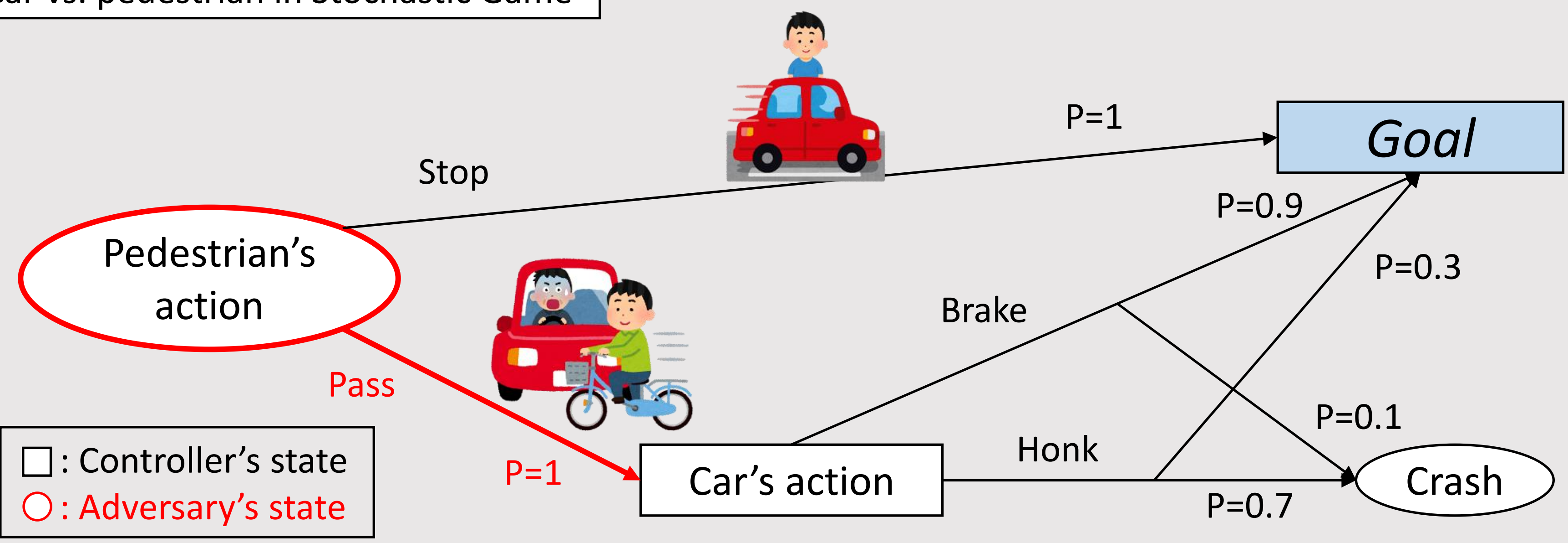
Car vs. pedestrian in Stochastic Game



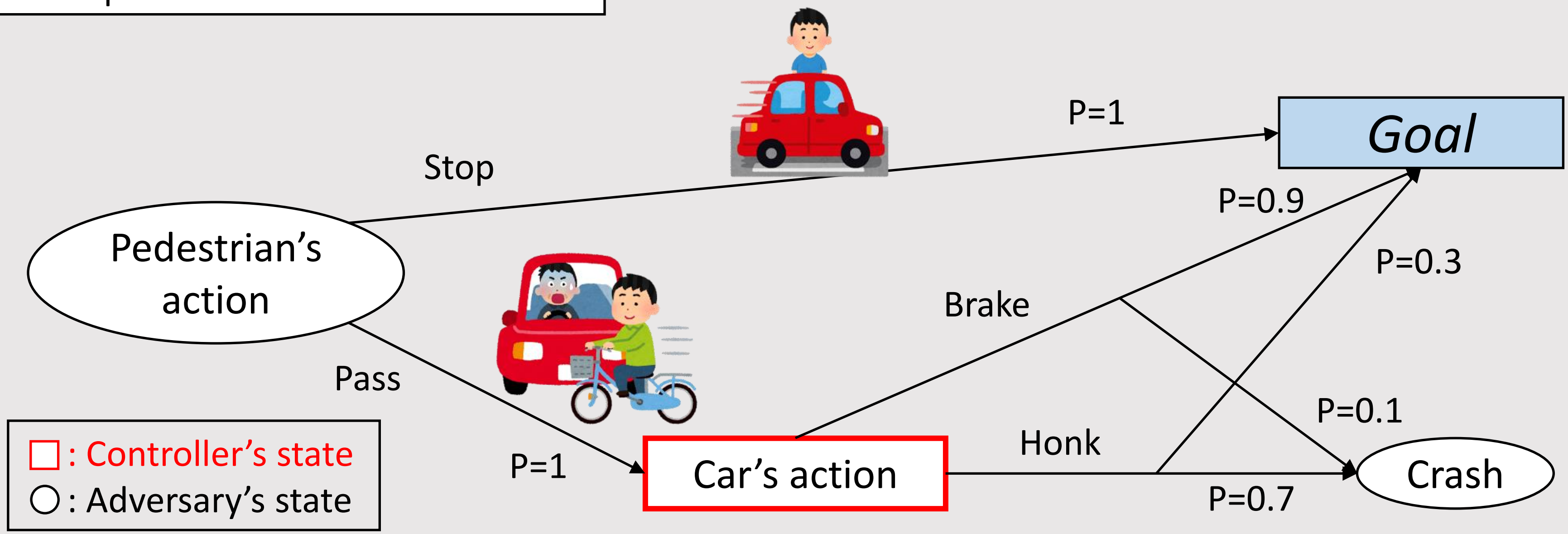
Car vs. pedestrian in Stochastic Game



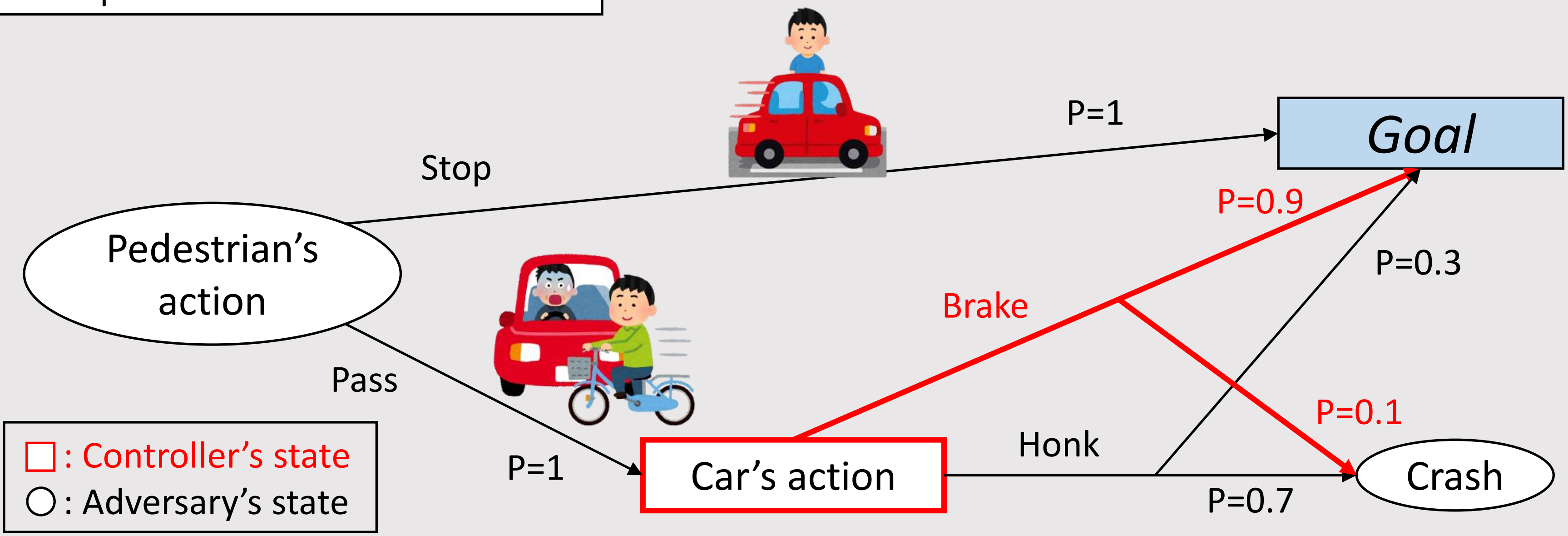
Car vs. pedestrian in Stochastic Game



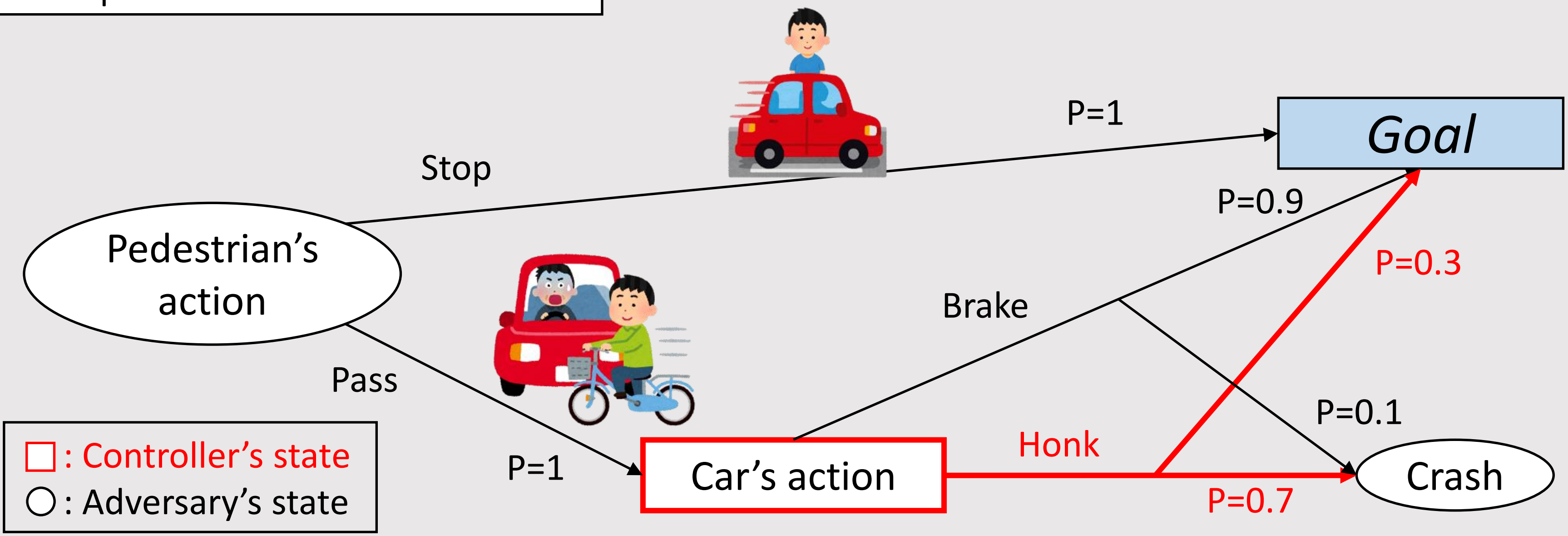
Car vs. pedestrian in Stochastic Game



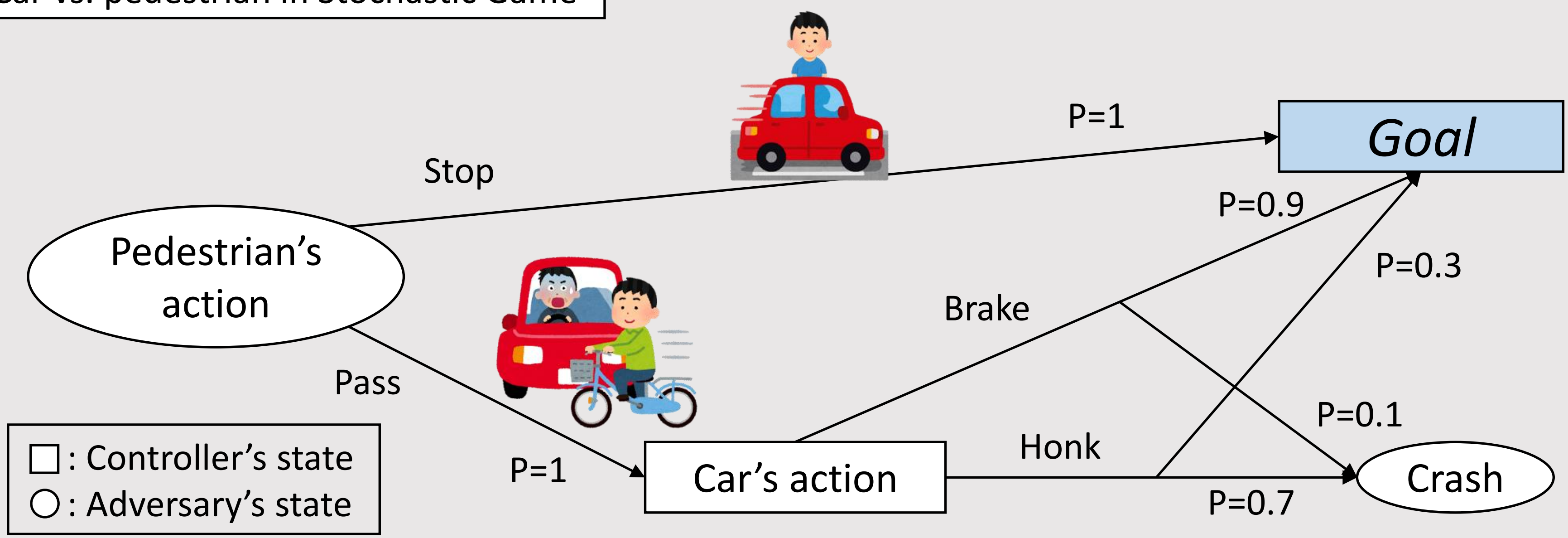
Car vs. pedestrian in Stochastic Game



Car vs. pedestrian in Stochastic Game

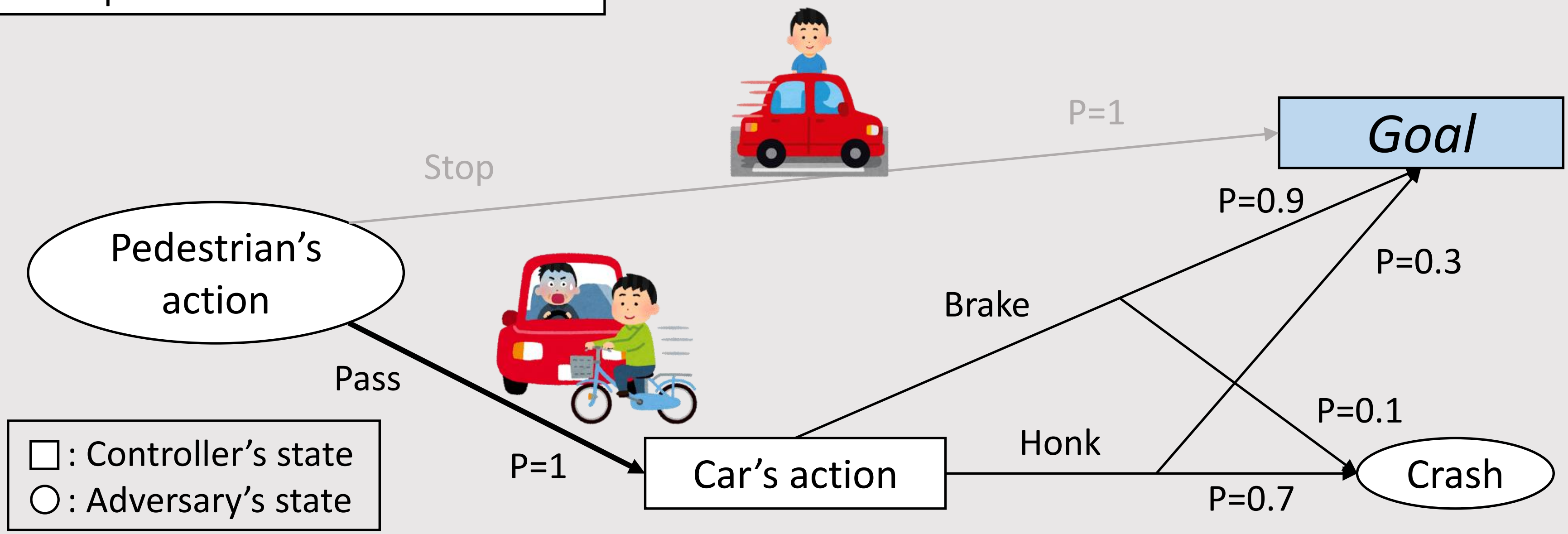


Car vs. pedestrian in Stochastic Game



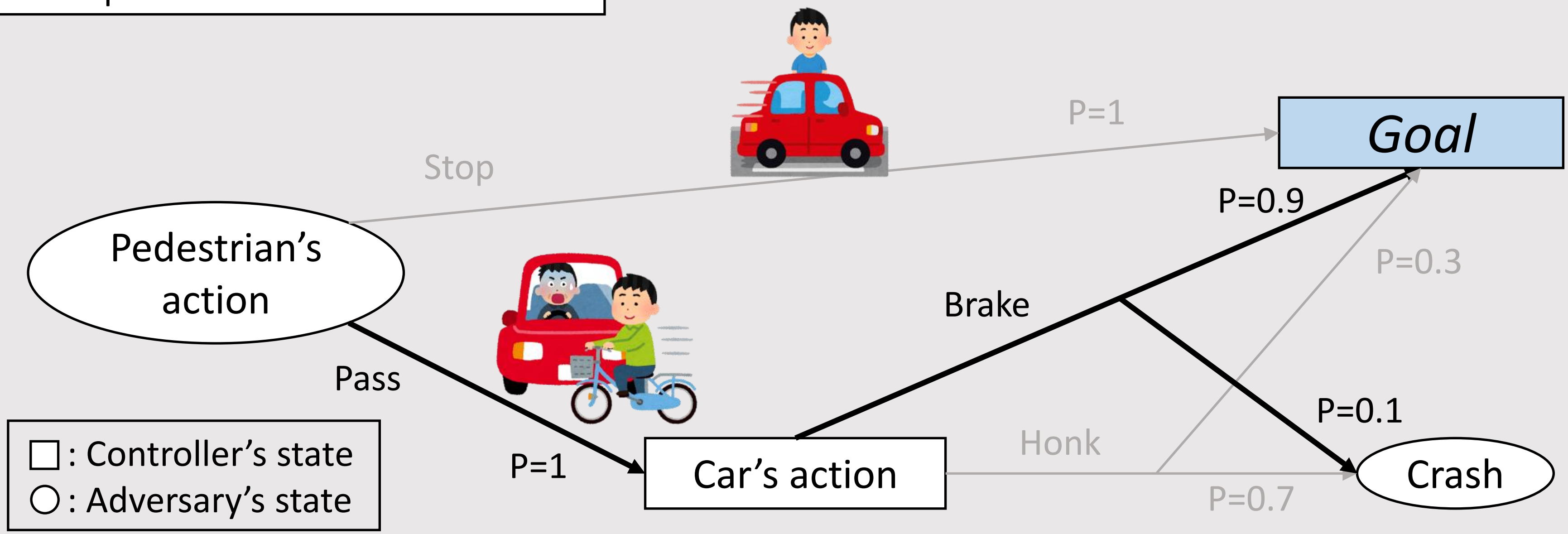
- A (pure positional) strategy of player X … $\sigma: (\mathrm{X}'\mathrm{s\ states}) \to (\mathrm{actions})$

Car vs. pedestrian in Stochastic Game



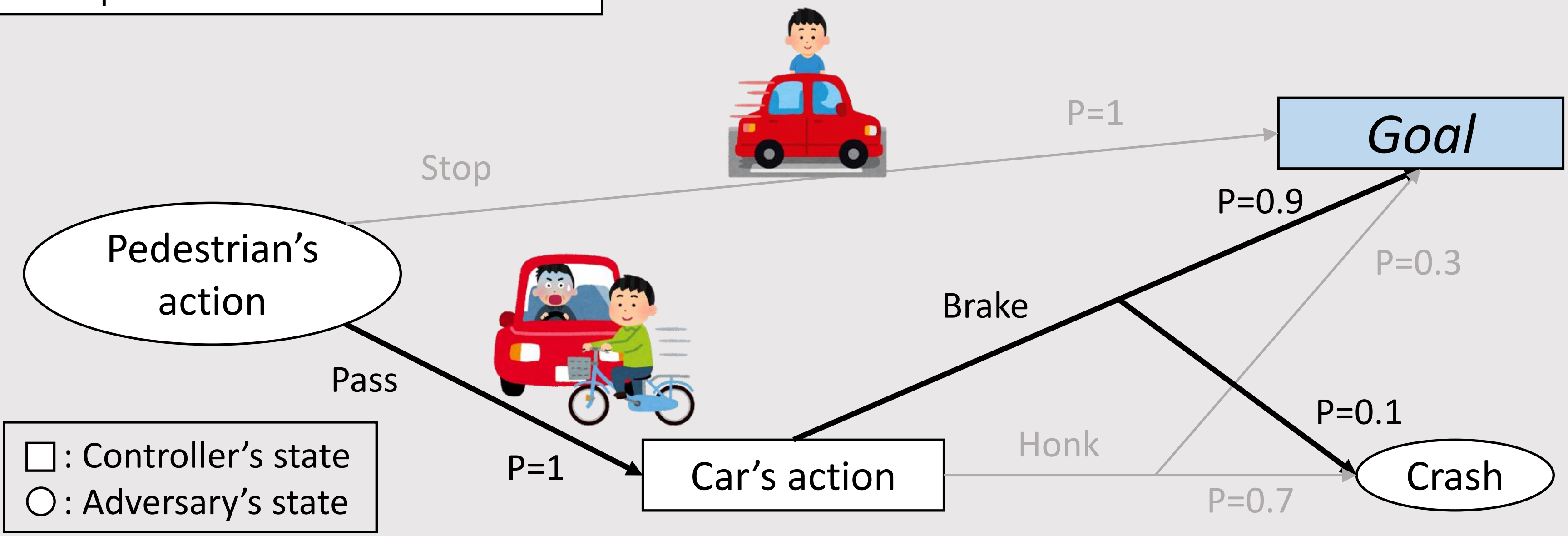
- A (pure positional) strategy of player X … σ: (X′s states) → (actions)

Car vs. pedestrian in Stochastic Game



- A (pure positional) strategy of player X … σ: (X′s states) → (actions)

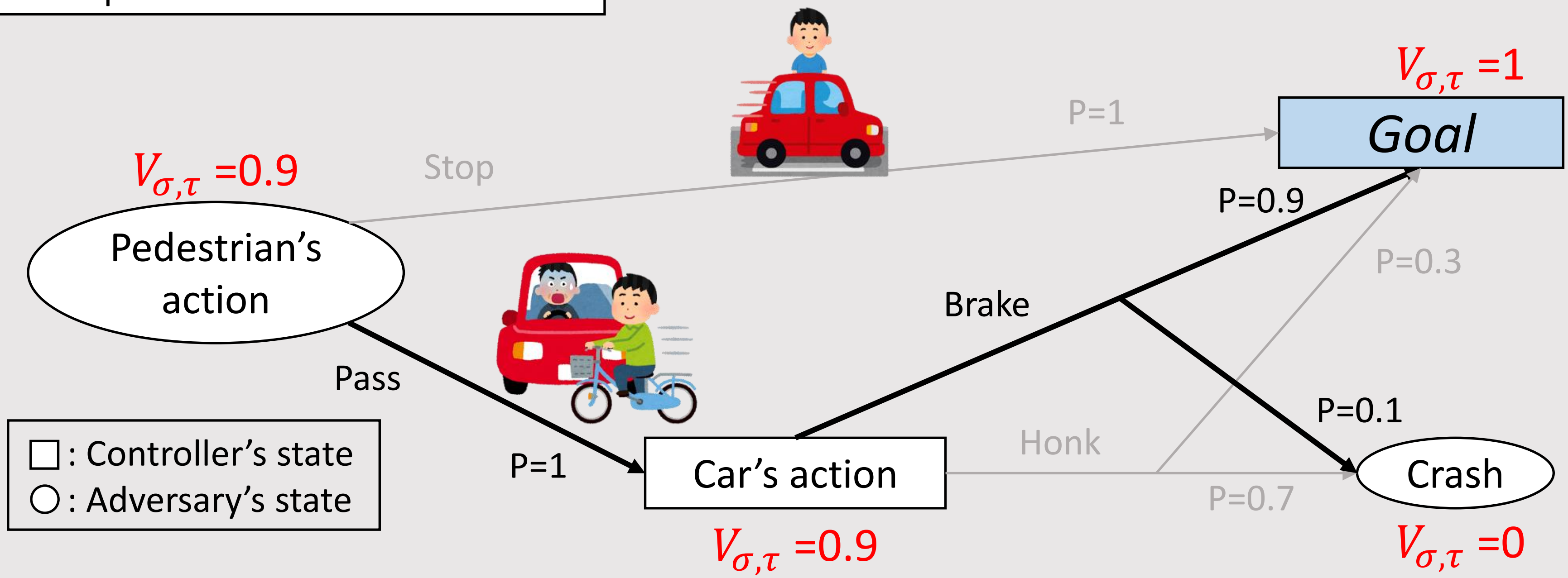
Car vs. pedestrian in Stochastic Game



- A (pure positional) strategy of player X … σ: (X′s states) → (actions)
- Reachability prob. under strategies σ, τ of Controller/Adversary…

$$V_{\sigma,\tau}(s) = \Pr(\textit{Goal} \text{ is visited during the play, starting from } s, \text{under } \sigma \text{ and } \tau)$$

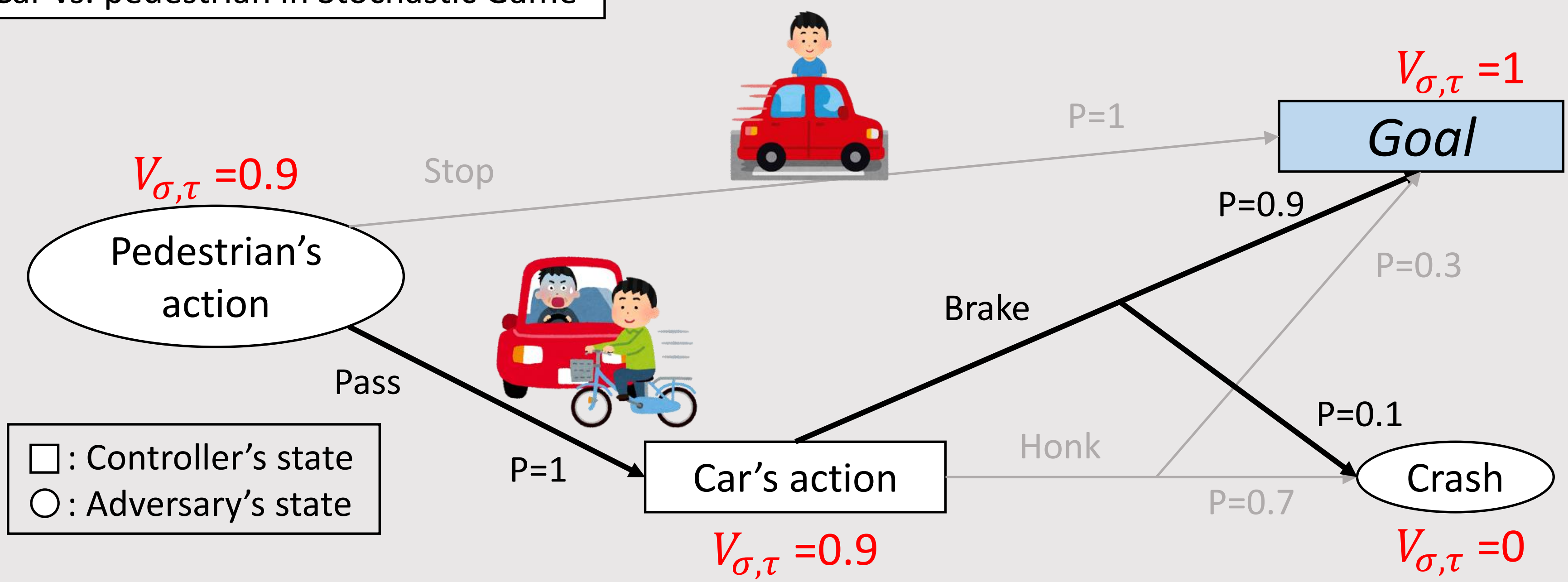
Car vs. pedestrian in Stochastic Game



- A (pure positional) strategy of player X … σ: (X′s states) → (actions)
- Reachability prob. under strategies σ, τ of Controller/Adversary…

$$V_{\sigma,\tau}(s) = \Pr(\mathit{Goal} \text{ is visited during the play, starting from } s, \text{under } \sigma \text{ and } \tau)$$

Car vs. pedestrian in Stochastic Game



- A (pure positional) strategy of player X … $\sigma\colon (\text{X's states}) \to (\text{actions})$
- Reachability prob. under strategies σ, τ of Controller/Adversary…

$$V_{\sigma,\tau}(s) = \Pr(\textit{Goal} \text{ is visited during the play, starting from } s, \text{under } \sigma \text{ and } \tau)$$

Problem

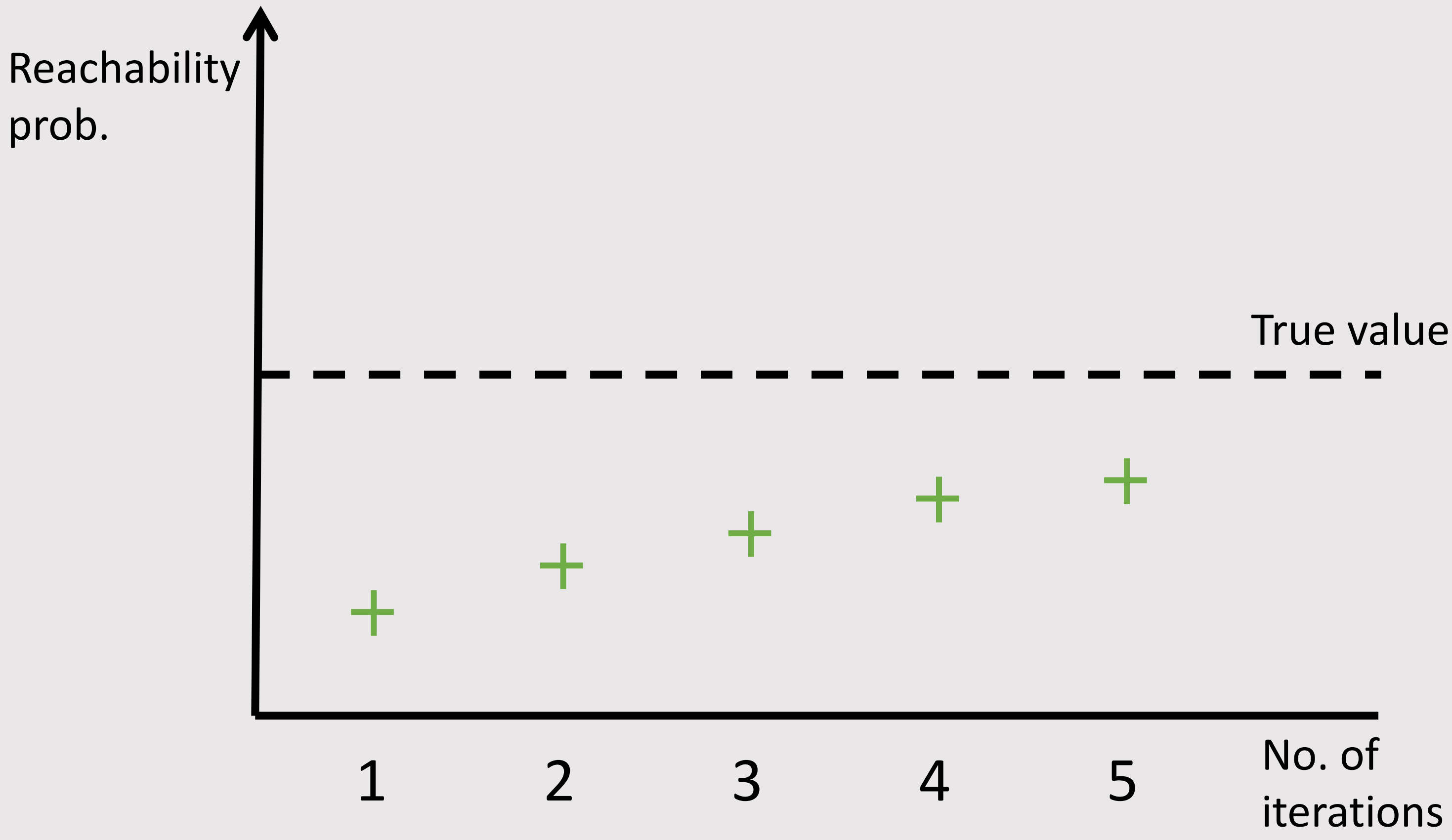
- Approximate the following $V\colon (\text{states}) \to [0,1]$

Controller/Adversary tries to maximize/minimize $V_{\sigma,\tau}$

$$V(s) = \max_{\sigma} \min_{\tau} V_{\sigma,\tau}(s)$$

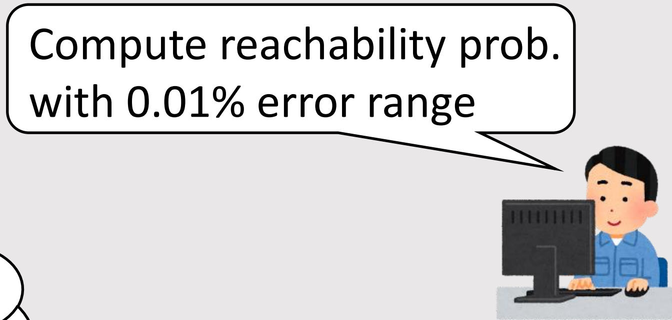
Existing technique 1 : **Value Iteration (VI)**

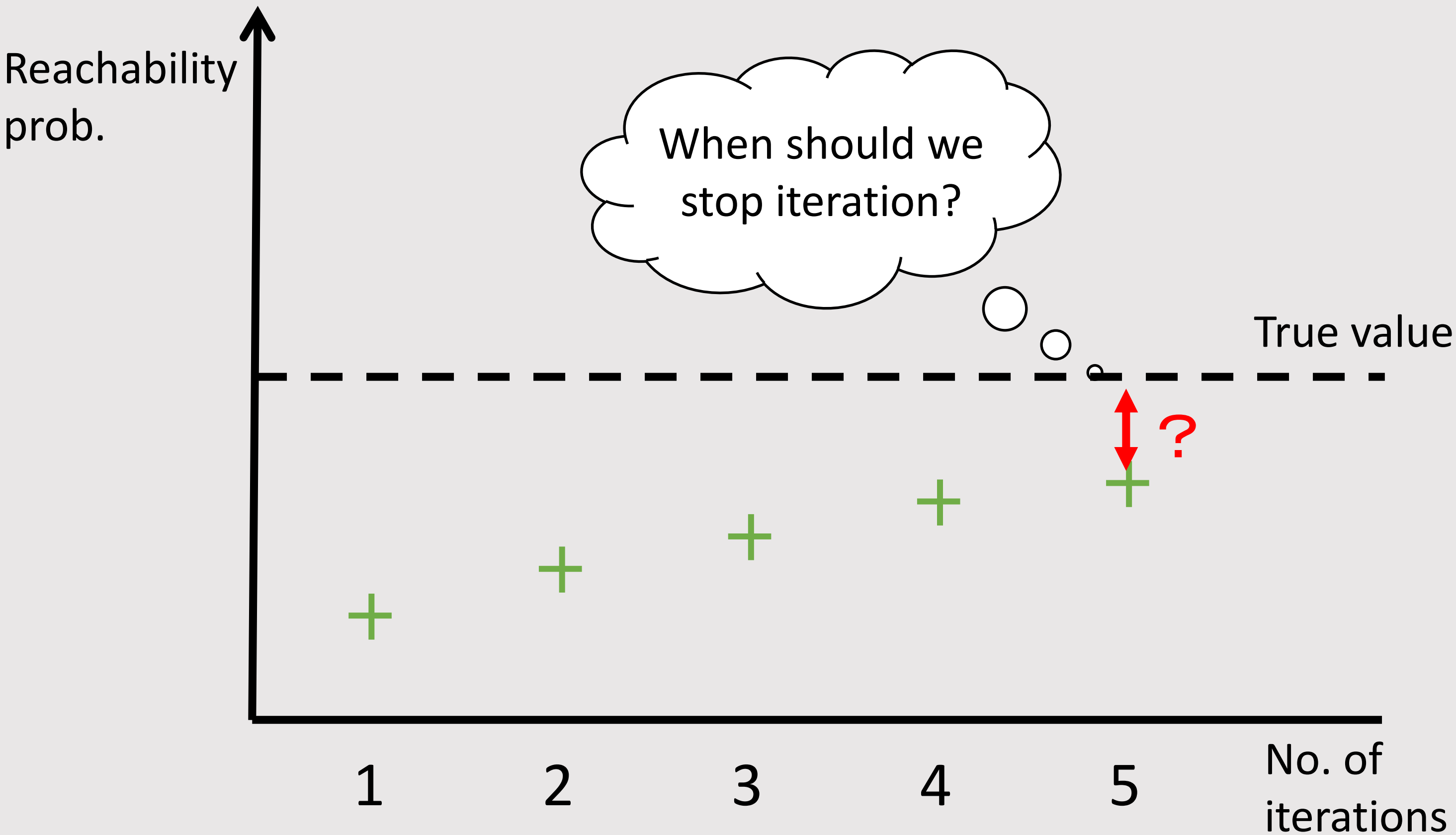
- Generates an increasing sequence of lower bounds
- Converges to the true value
- No precision guarantee



Existing technique 1 : **Value Iteration (VI)**

- Generates an increasing sequence of lower bounds
- Converges to the true value
- No precision guarantee

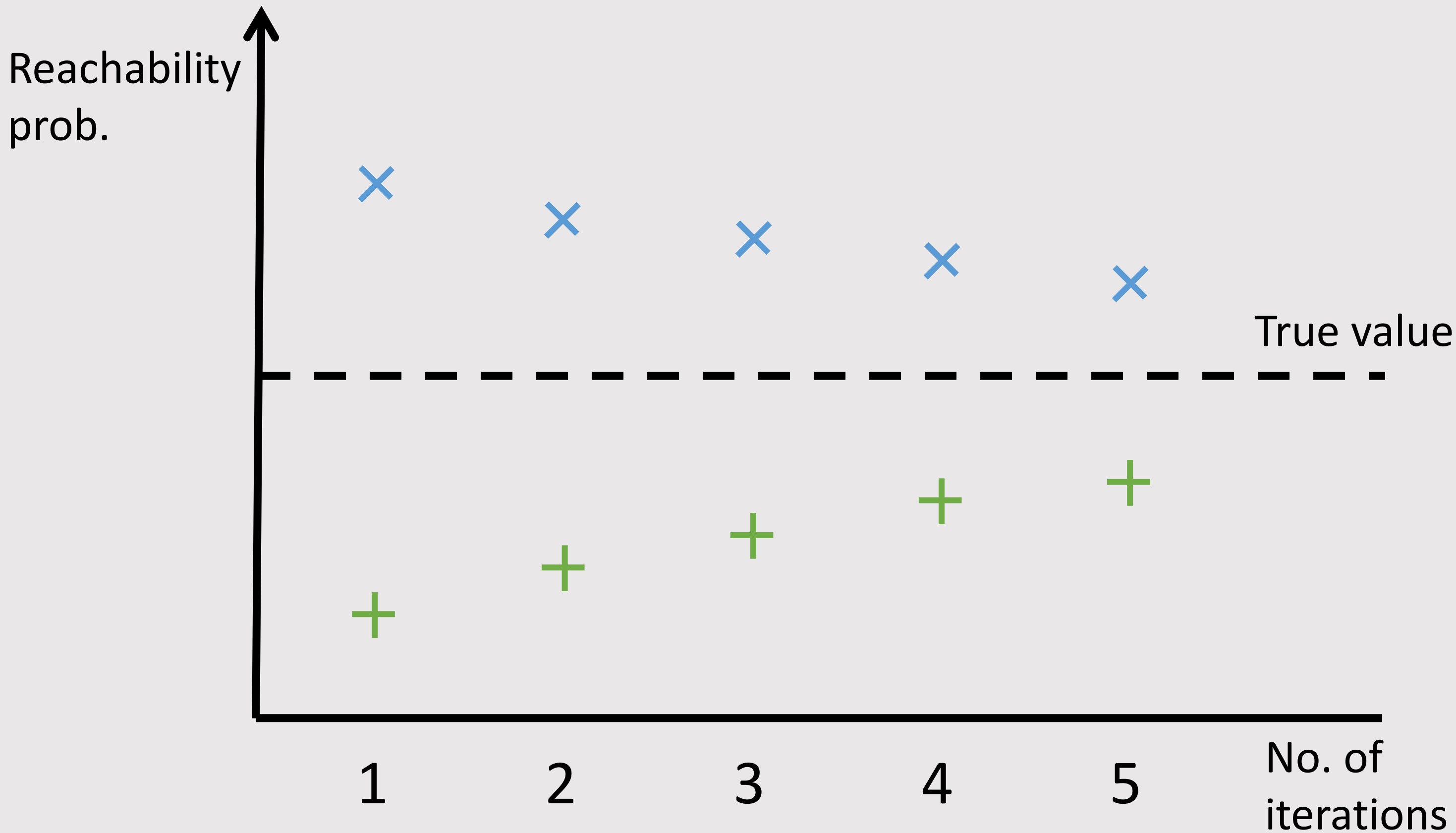




Existing technique 2 : **Bounded Value Iteration (BVI)**

[McMahan+,'05][Brazdil+,'14][Ujma, '15][Haddad+,'18][Kelmendi+,'18]

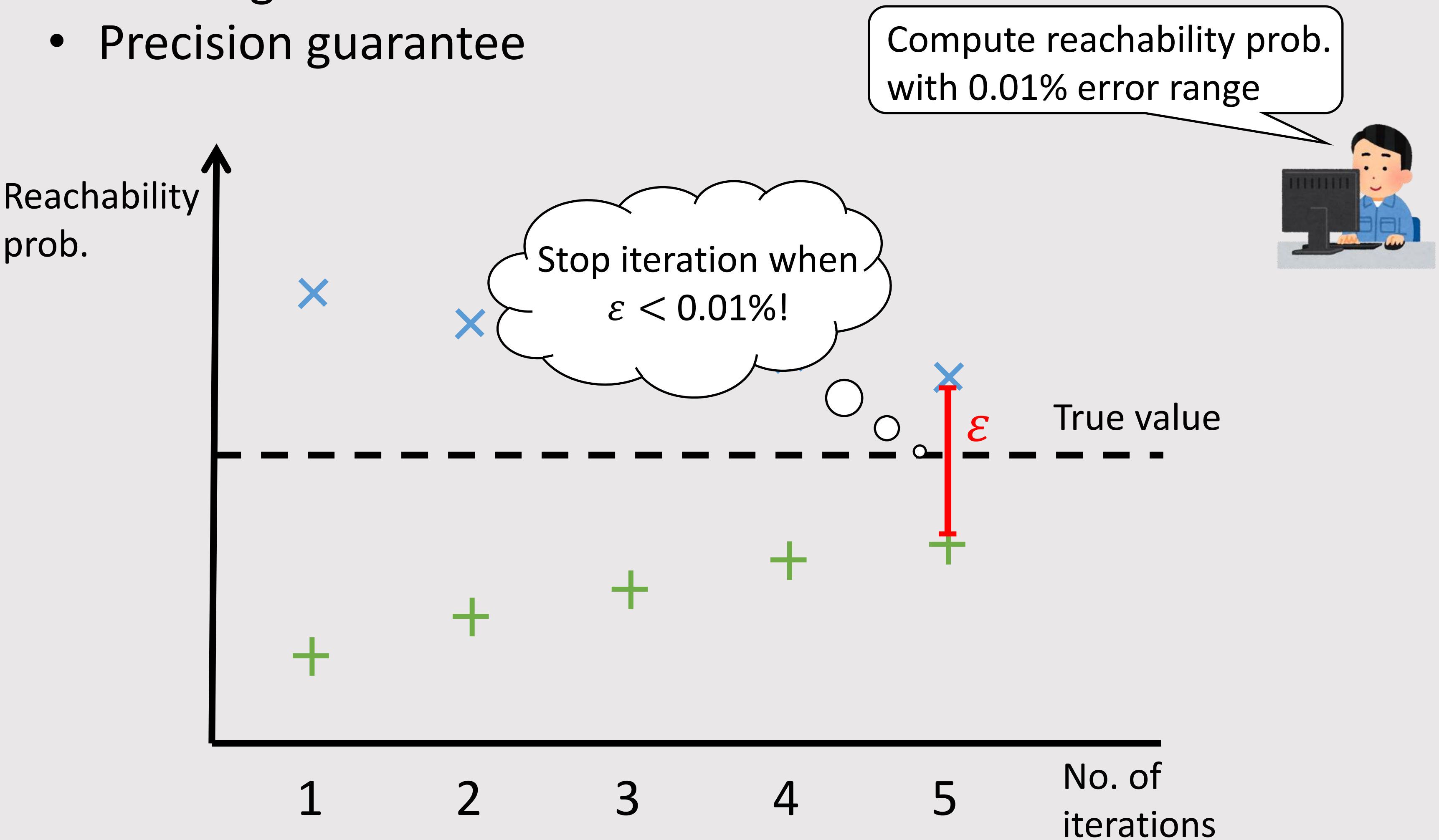
- Generates a decreasing sequence of upper bounds, too
- Converges to the true value
- Precision guarantee



Existing technique 2 : **Bounded Value Iteration (BVI)**

[McMahan+,'05][Brazdil+,'14][Ujma, '15][Haddad+,'18][Kelmendi+,'18]

- Generates a decreasing sequence of upper bounds, too
- Converges to the true value
- Precision guarantee



Technical challenge

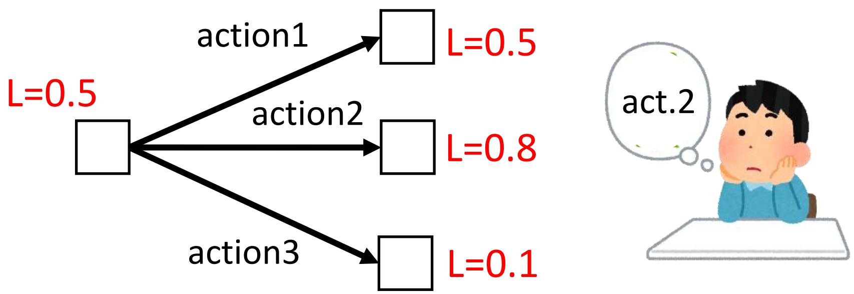
Q: Is BVI a technique that merely performs VI twice in parallel,
starting from some lower and upper bound?

A: No, it's more than that.
To assure convergence of upper bound, we need some trick.

How a lower bound L: (states) $\to [0,1]$ is updated via VI

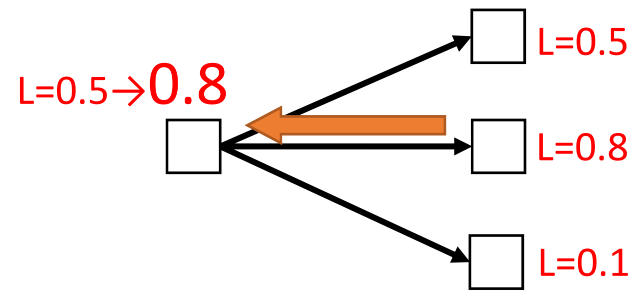
(at Controller's states)

Strategy estimation via local optimality



Choose an action that maximizes L after a transition

Bound update via estimated strategy

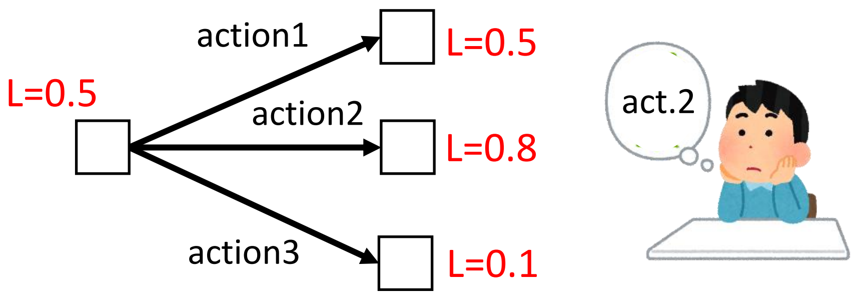


Back-propagate the value of L along the transition arrow

How a lower bound L: (states) $\to [0,1]$ is updated via VI

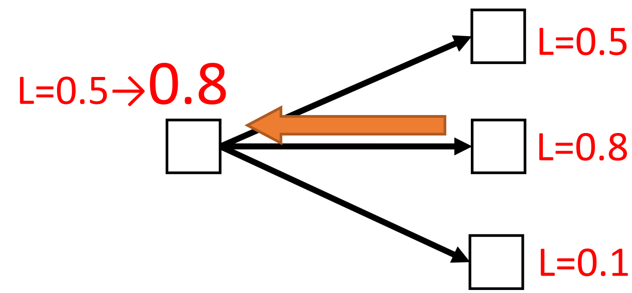
(at Controller's states)

Strategy estimation via local optimality



Choose an action that maximizes L after a transition

Bound update via estimated strategy



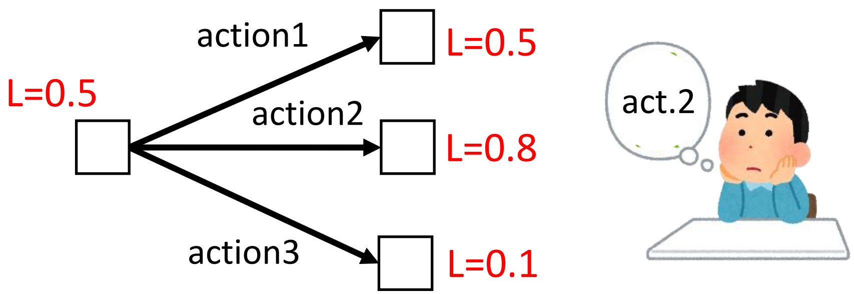
Back-propagate the value of L along the transition arrow

Bellman operator $\mathbb{X}: L \mapsto \mathbb{X}L$

How a lower bound $L: (\mathrm{states}) \to [0,1]$ is updated via VI

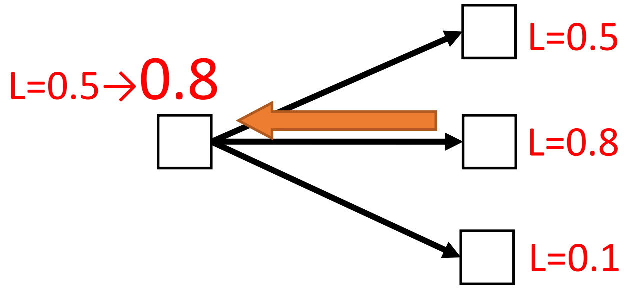
(at Controller's states)

Strategy estimation via local optimality



Choose an action that maximizes
L after a transition

Bound update via estimated strategy



Back-propagate the value of L
along the transition arrow

Bellman operator $\mathbb{X}: L \mapsto \mathbb{X}L$

VI performs $L_0 \mapsto \mathbb{X}L_0 \mapsto \mathbb{X}(\mathbb{X}L_0) \mapsto \mathbb{X}\big(\mathbb{X}(\mathbb{X}L_0)\big) \to \cdots$

How the non-convergence issue of an upper bound occurs

- Bellman operator is monotone over the set

$$\{f : (states) \to [0,1] \mid f(\mathrm{final}) = 1, f(\mathrm{sink}) = 0\}$$

How the non-convergence issue of an upper bound occurs

$f \le g \Leftrightarrow \forall s. f(s) \le g(s)$

- Bellman operator is monotone over the set
  $$\{f: (states) \to [0,1] \mid f(\text{final}) = 1, f(\text{sink}) = 0\}$$

How the non-convergence issue of an upper bound occurs

$f \leq g \Leftrightarrow \forall s. f(s) \leq g(s)$

- Bellman operator is monotone over the set

$$\{f : (states) \to [0,1] \mid f(\text{final}) = 1, f(\text{sink}) = 0\}$$

- Optimal reachability probability is the least fixed point of Bellman operator:

$$V = \mu\mathbb{X}$$

How the non-convergence issue of an upper bound occurs

$f \leq g \Leftrightarrow \forall s. f(s) \leq g(s)$

- Bellman operator is monotone over the set

$$\{f : (states) \to [0,1] \mid f(\text{final}) = 1, f(\text{sink}) = 0\}$$

- Optimal reachability probability is the least fixed point of Bellman operator:

$$V = \mu\mathbb{X}$$

Convergence of lower bound

- Starting from $L_0 = \bot$, VI generates a sequence

$$L_0 \leq \mathbb{X}L_0 \leq \mathbb{X}(\mathbb{X}L_0) \leq \cdots \to \mu\mathbb{X} = V$$

How the non-convergence issue of an upper bound occurs

$f \le g \Leftrightarrow \forall s. f(s) \le g(s)$

- Bellman operator is monotone over the set

$$\{f : (states) \to [0,1] \mid f(\text{final}) = 1, f(\text{sink}) = 0\}$$

- Optimal reachability probability is the least fixed point of Bellman operator:

$$V = \mu \mathbb{X}$$

Convergence of lower bound

- Starting from $L_0 = \bot$, VI generates a sequence

$$L_0 \le \mathbb{X}L_0 \le \mathbb{X}(\mathbb{X}L_0) \le \cdots \to \mu\mathbb{X} = V$$

Non-convergence of upper bound

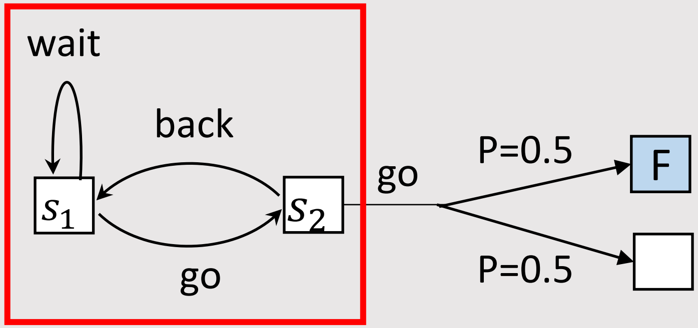
- Starting from $U_0 = \top$, VI generates a sequence

$$U_0 \ge \mathbb{X}U_0 \ge \mathbb{X}(\mathbb{X}U_0) \ge \cdots \to \nu\mathbb{X} \ge V$$

Existing technique to address the problem

- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

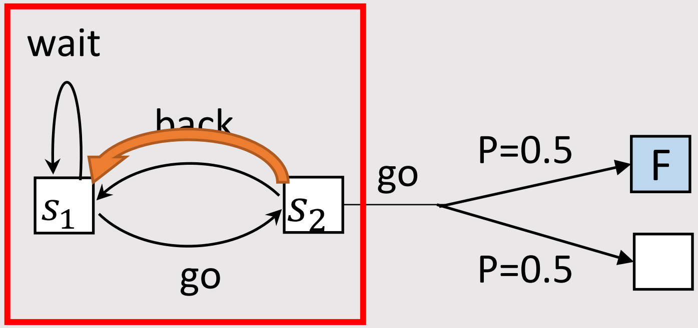
Sub-MDP that constitutes a loop



Existing technique to address the problem

- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

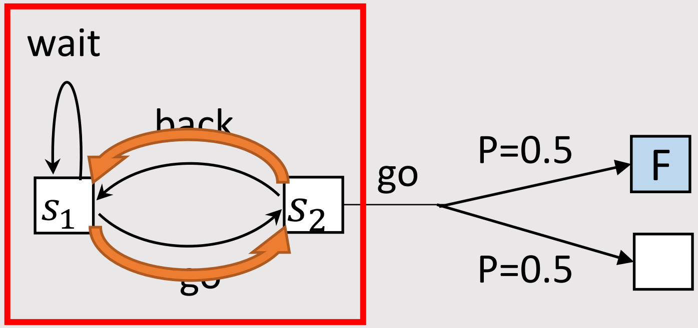
Sub-MDP that constitutes a loop



Existing technique to address the problem

- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

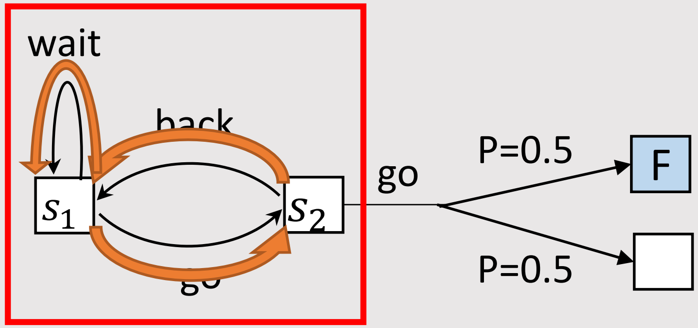
Sub-MDP that constitutes a loop



Existing technique to address the problem

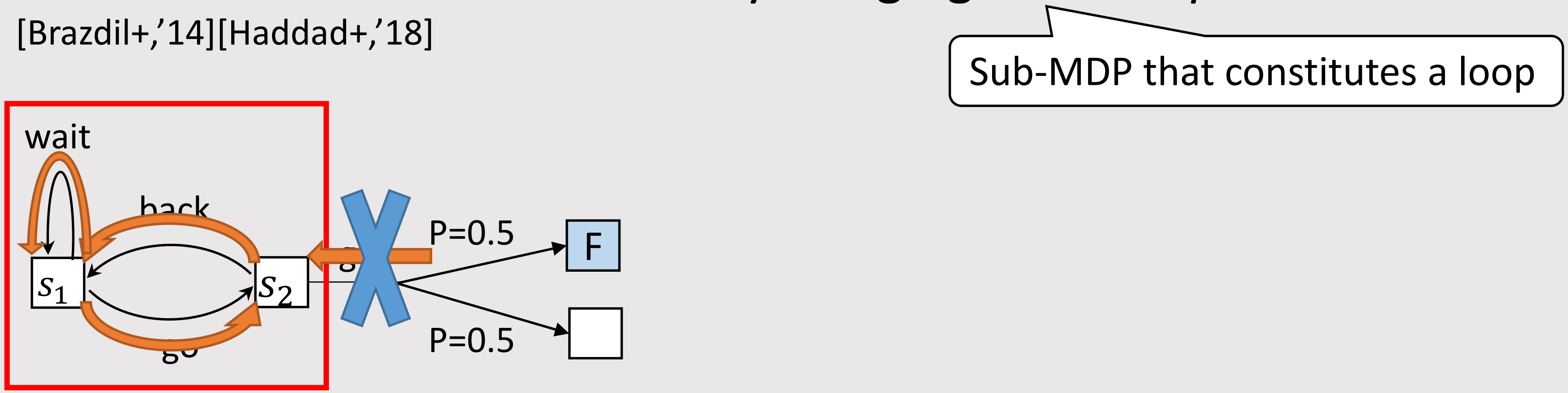
- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

Sub-MDP that constitutes a loop



Existing technique to address the problem

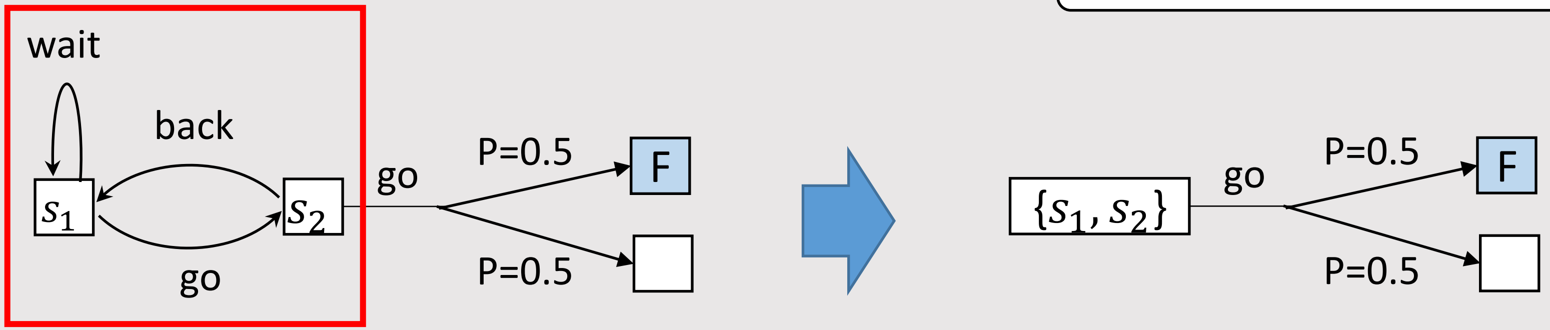
- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]



Existing technique to address the problem

- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

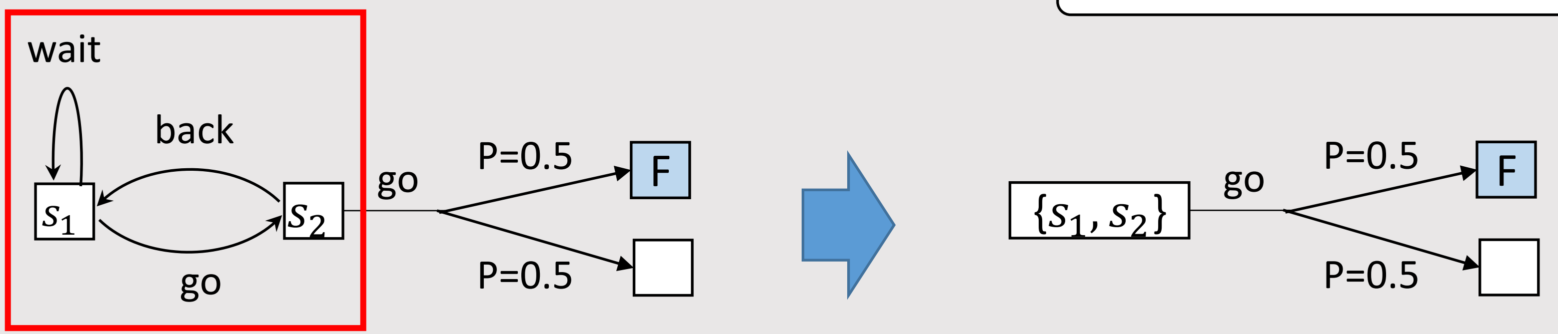
Sub-MDP that constitutes a loop



Existing technique to address the problem

- If the system is an MDP (i.e. there is no Adversary's state), GFP can be matched with LFP by merging *End Components* [Brazdil+,'14][Haddad+,'18]

Sub-MDP that constitutes a loop



- For an arbitrary SG, we periodically *deflate* an upper bound while running the standard VI [Kelmendi+,'18]

Singular update of bound over (a sound approx. of) specific ECs

Overview of our algorithm

- Every existing technique involves EC computation
  (or restrict the model so that non-convergence problem does not occur)
- EC computation can be a bottleneck of execution time of BVI
  - Especially for SGs… EC computation is invoked many times

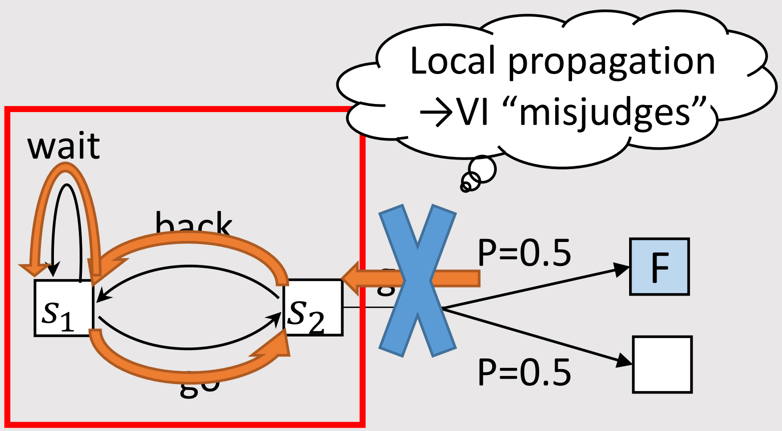
Overview of our algorithm

- Every existing technique involves EC computation
  (or restrict the model so that non-convergence problem does not occur)

- EC computation can be a bottleneck of execution time of BVI
  - Especially for SGs… EC computation is invoked many times

Our idea: ignore ECs, rather than compute

- Global propagation along the path to the final state



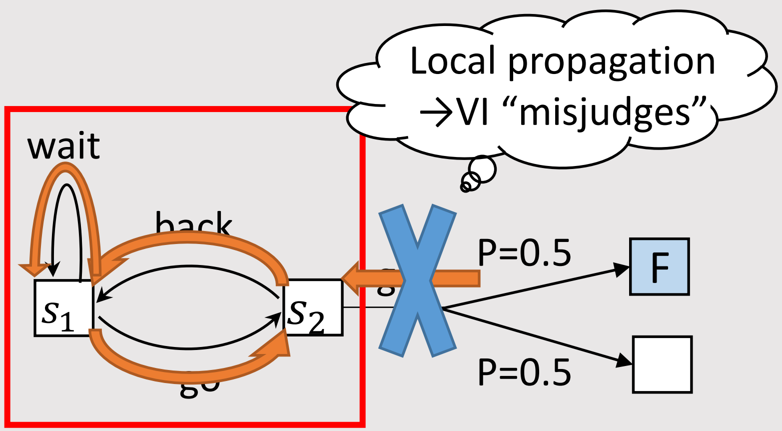
Overview of our algorithm

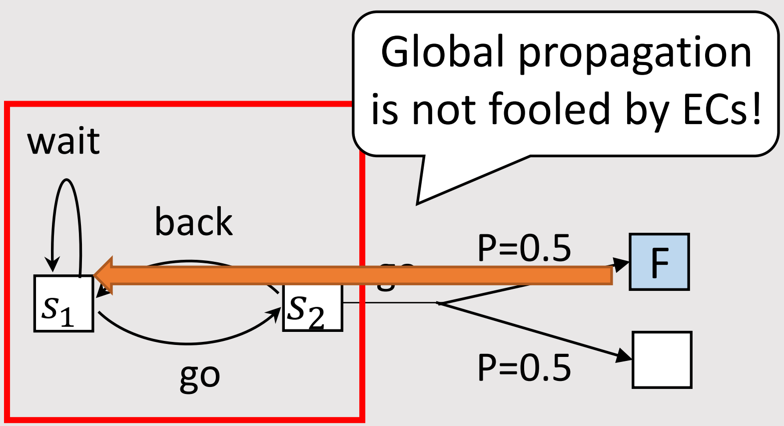
- Every existing technique involves EC computation
  (or restrict the model so that non-convergence problem does not occur)

- EC computation can be a bottleneck of execution time of BVI
  - Especially for SGs… EC computation is invoked many times

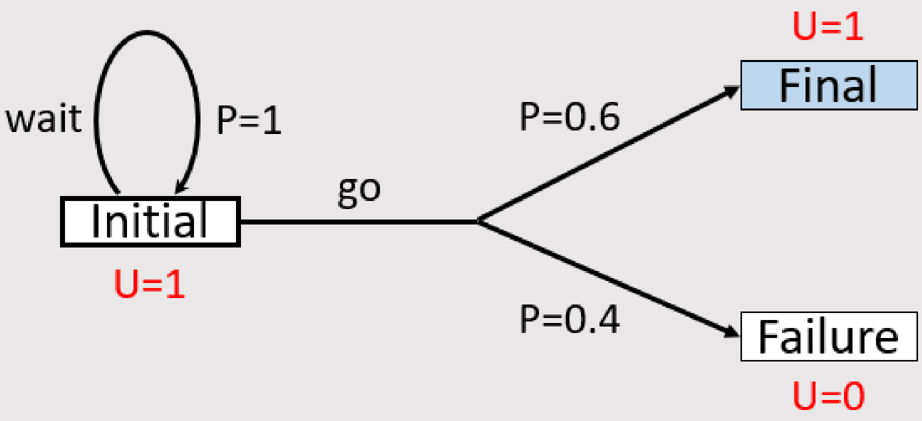
Our idea: ignore ECs, rather than compute

- Global propagation along the path to the final state

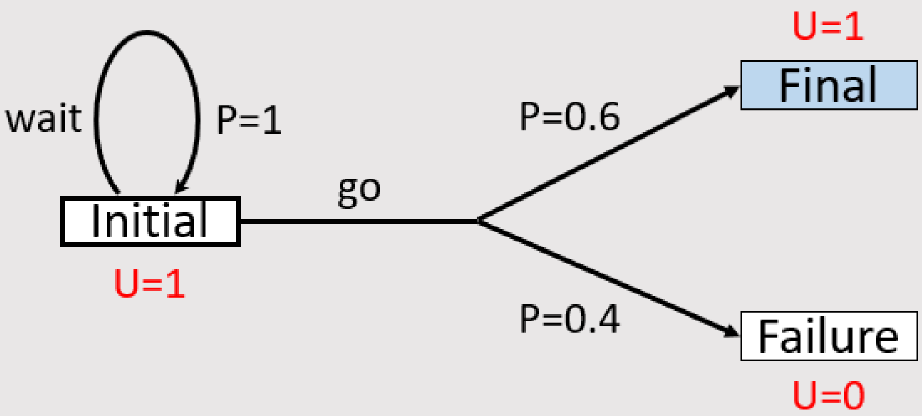




Step 1: Construct a weighted graph from MDP and upper bound U

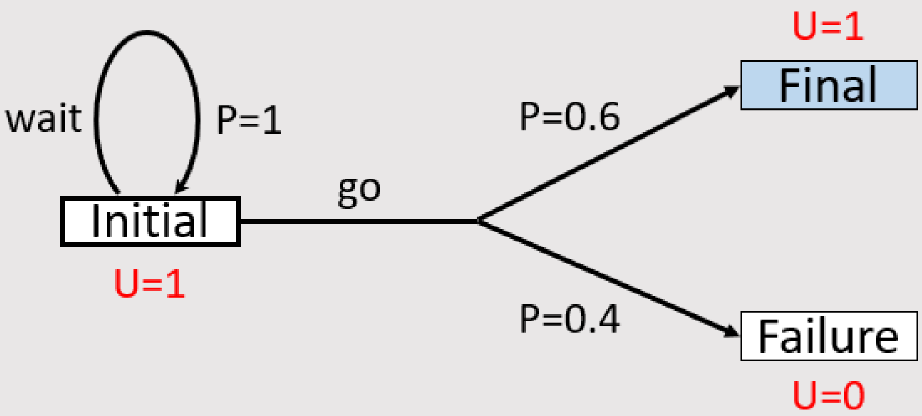


Step 1: Construct a weighted graph from MDP and upper bound U



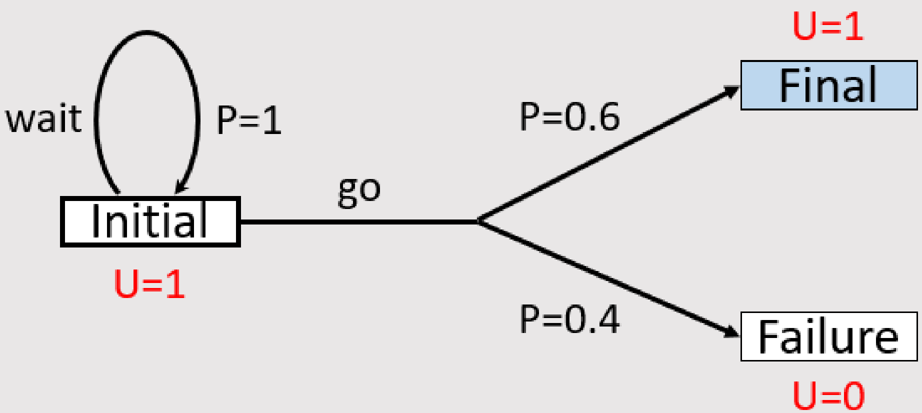
State	Action	Val. of U after a transition	Possible next state
Initial	go	0.6	Final, Failure
Initial	wait	1	Initial

Step 1: Construct a weighted graph from MDP and upper bound U

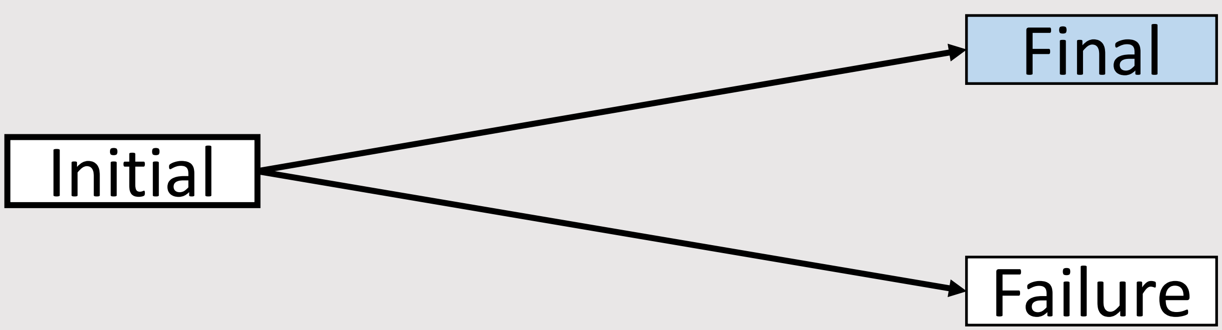


State	Action	Val. of U after a transition	Possible next state
Initial	go	0.6	Final, Failure
Initial	wait	1	Initial

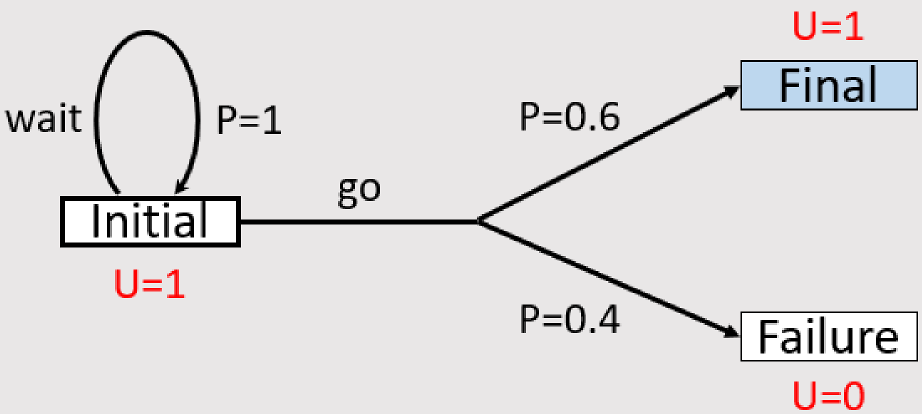
Step 1: Construct a weighted graph from MDP and upper bound U



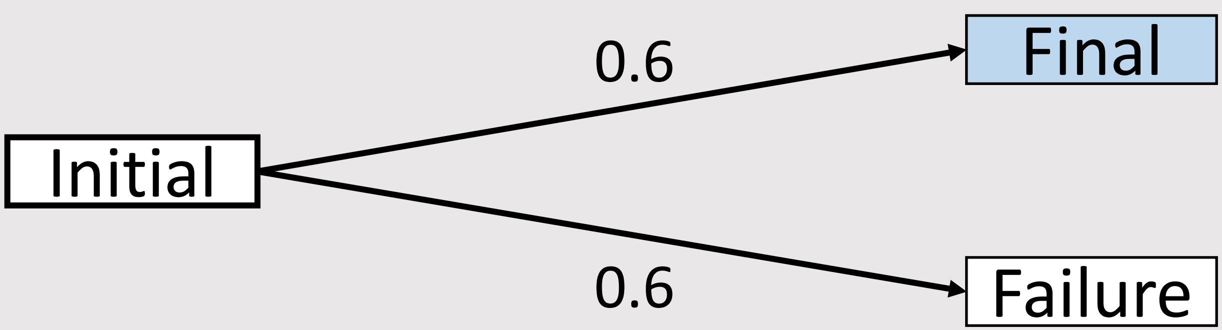
State	Action	Val. of U after a transition	Possible next state
Initial	go	0.6	Final, Failure
Initial	wait	1	Initial



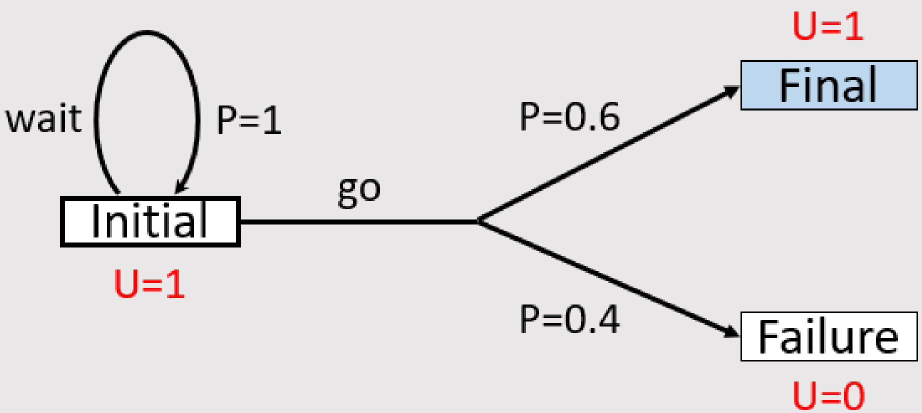
Step 1: Construct a weighted graph from MDP and upper bound U



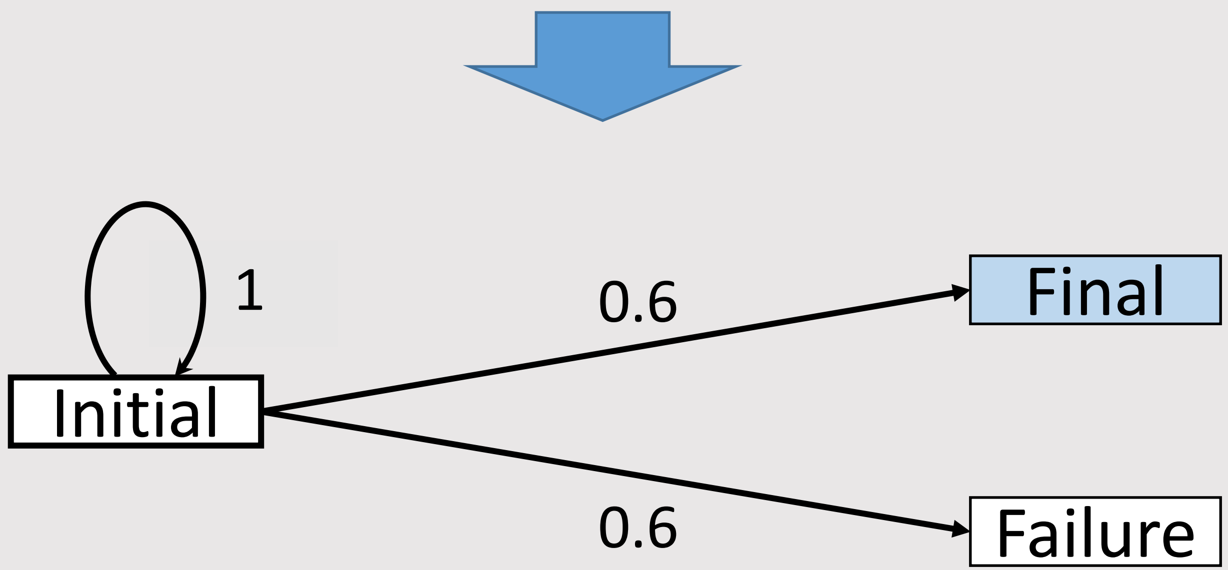
State	Action	Val. of U after a transition	Possible next state
Initial	go	0.6	Final, Failure
Initial	wait	1	Initial



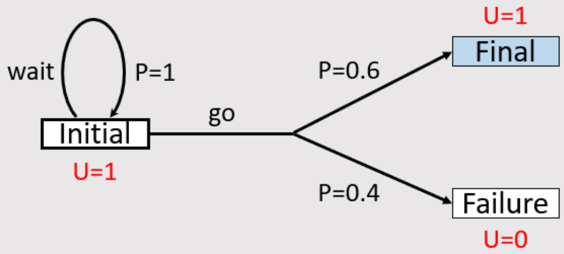
Step 1: Construct a weighted graph from MDP and upper bound U

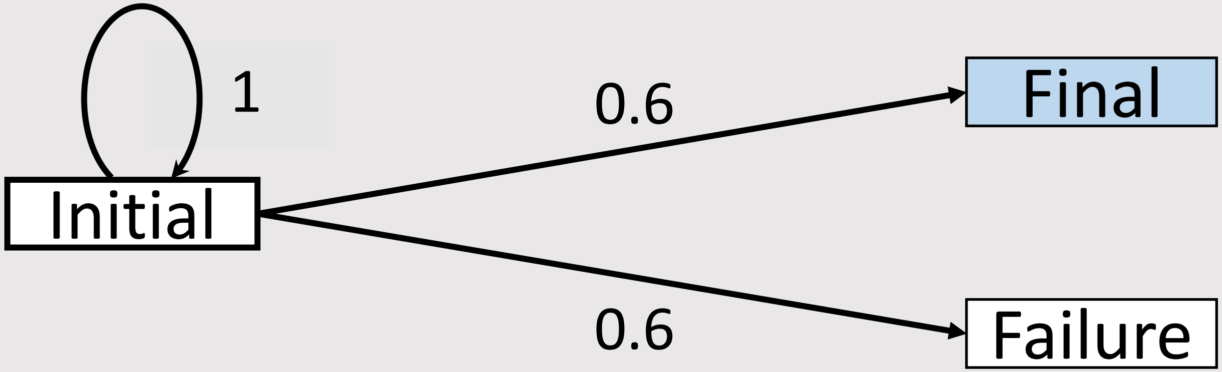


State	Action	Val. of U after a transition	Possible next state
Initial	go	0.6	Final, Failure
Initial	wait	1	Initial

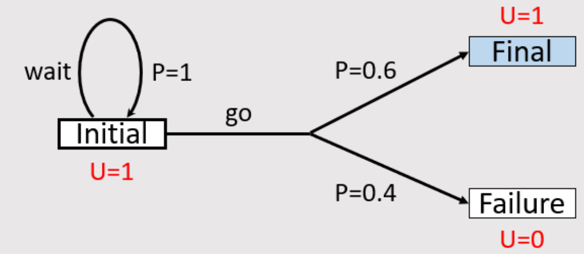


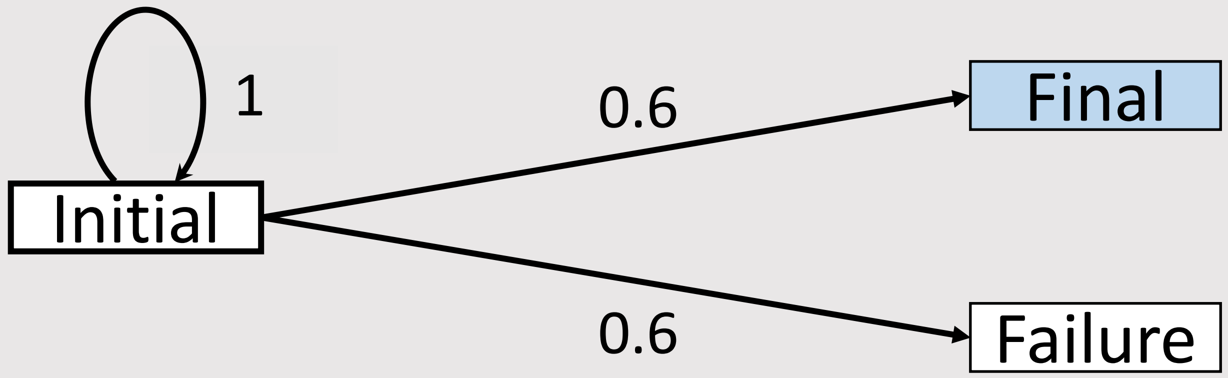
Step 2: update an upper bound U





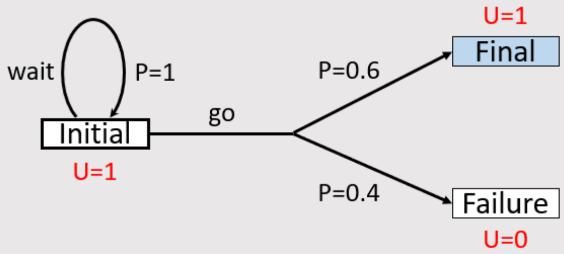
Step 2: update an upper bound U

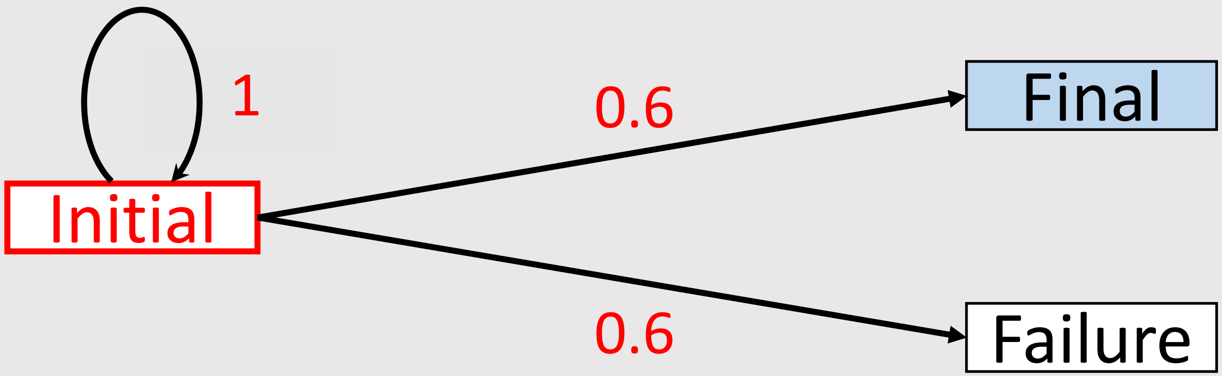




VI: “Compare outgoing edges, and propagate the largest weight”

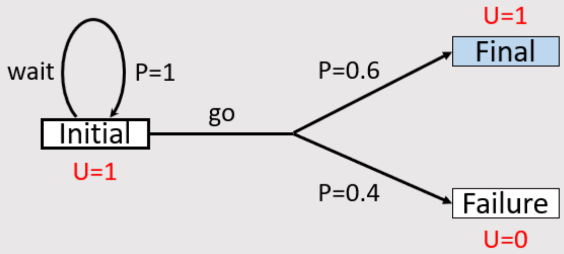
Step 2: update an upper bound U

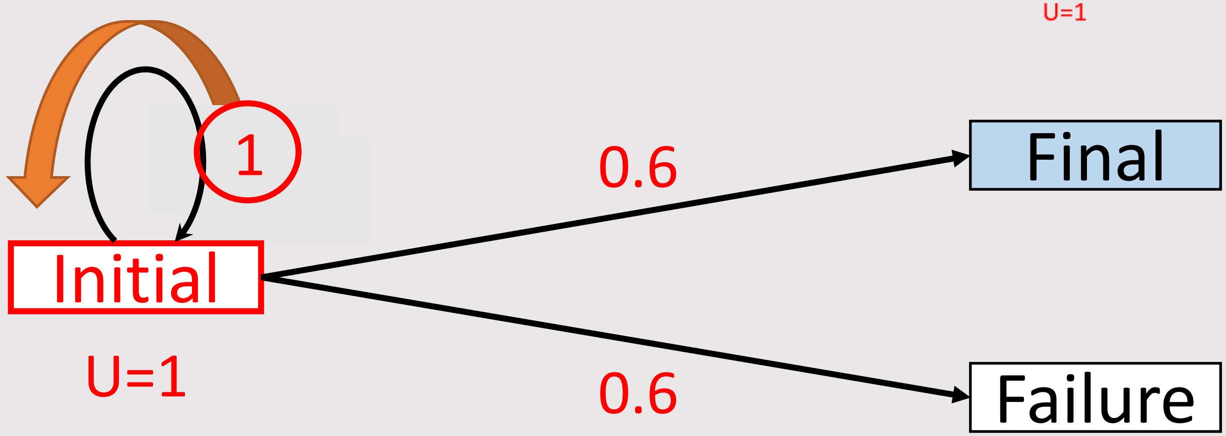




VI: “Compare outgoing edges, and propagate the largest weight”

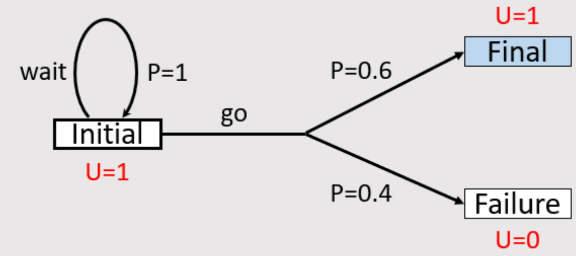
Step 2: update an upper bound U

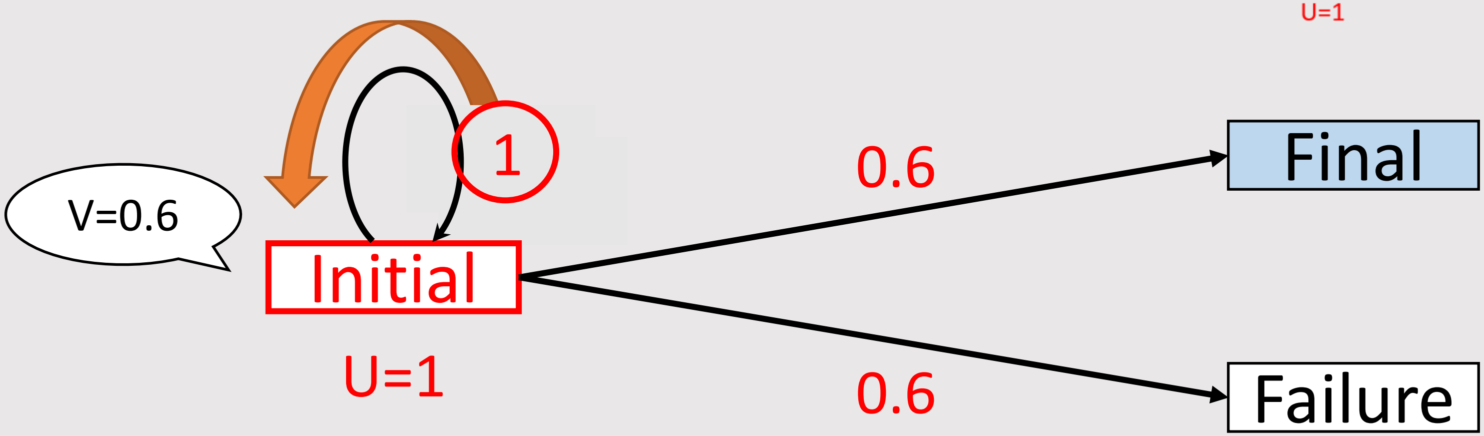




VI: “Compare outgoing edges, and propagate the largest weight”

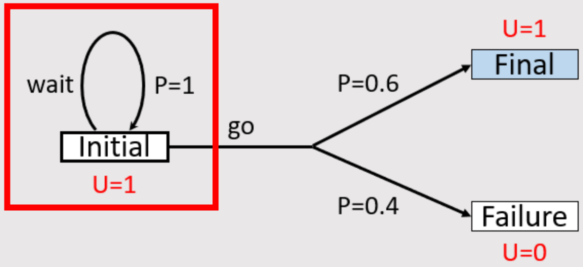
Step 2: update an upper bound U

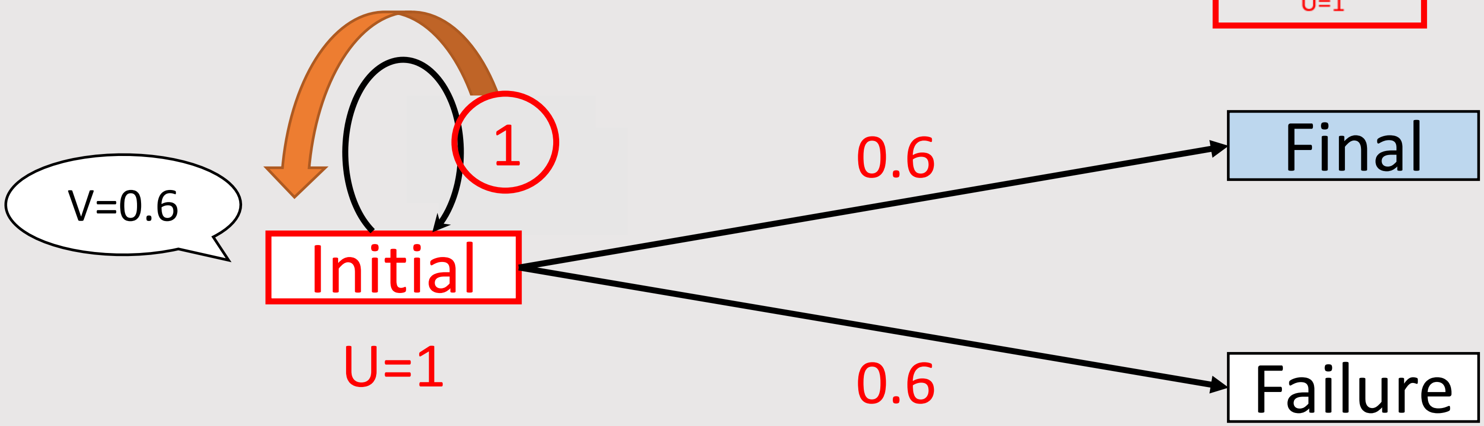




VI: “Compare outgoing edges, and propagate the largest weight”

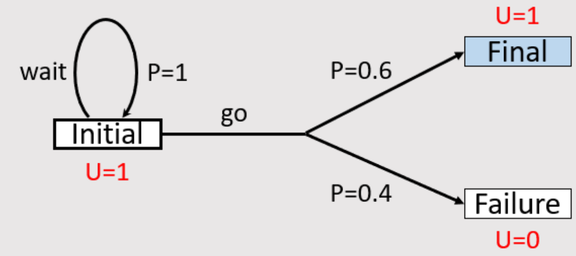
Step 2: update an upper bound U

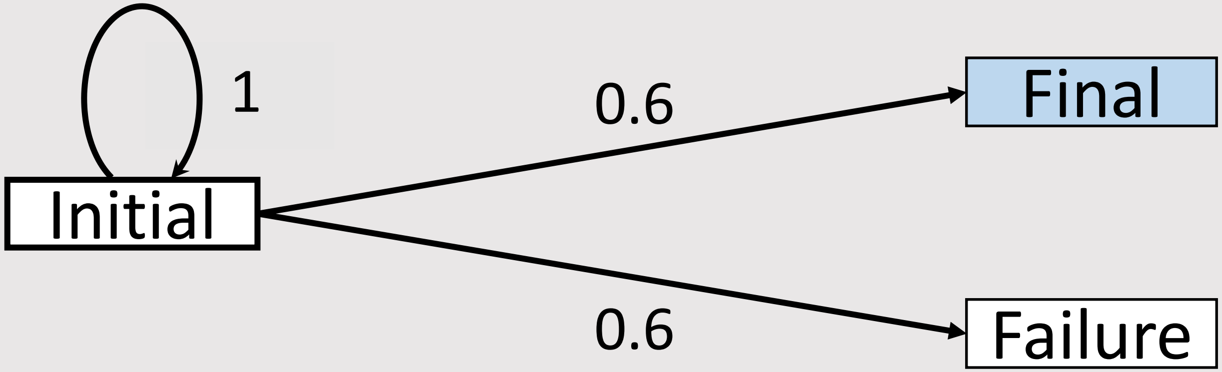




VI: “Compare outgoing edges, and propagate the largest weight”

Step 2: update an upper bound U

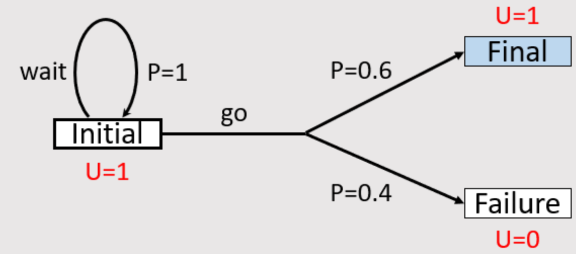


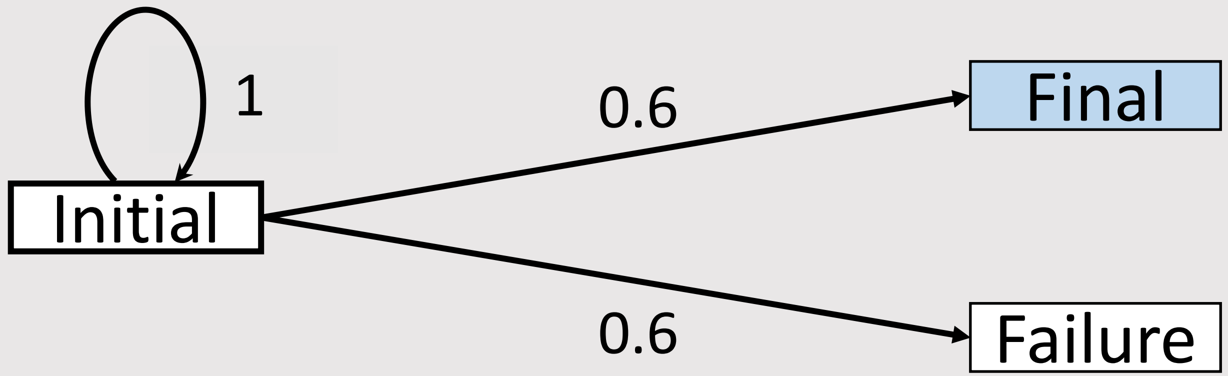


Our alg.: “Compare paths to Final, and propagate the largest width”

the minimum weight of constituting edges

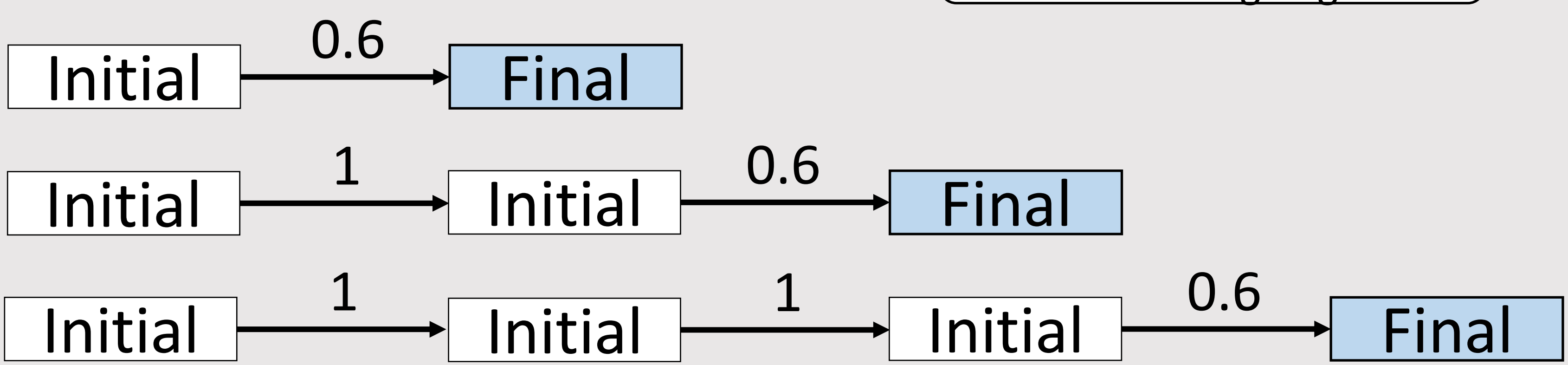
Step 2: update an upper bound U



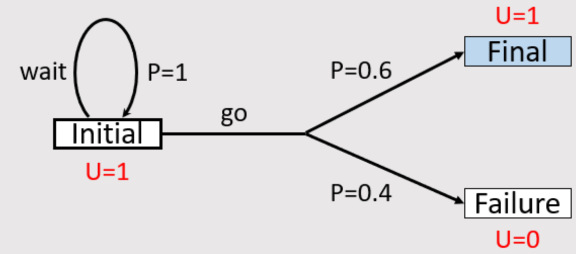


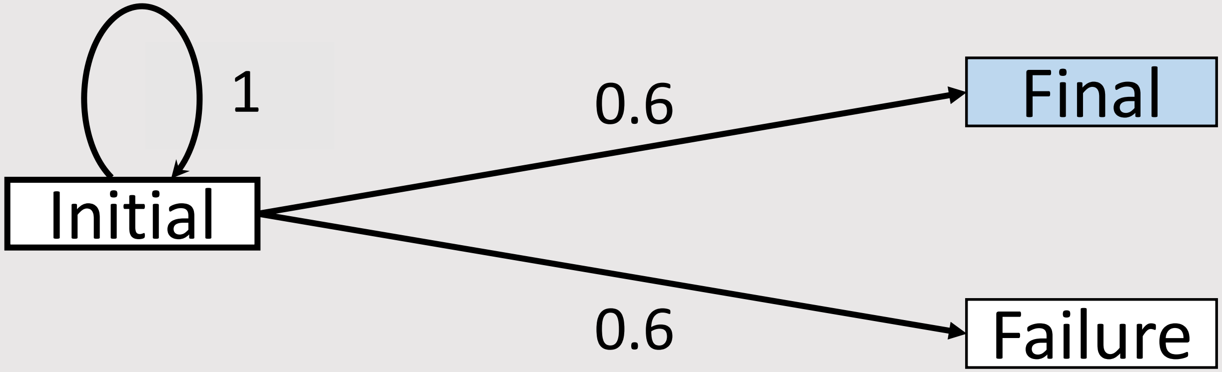
Our alg.: “Compare paths to Final, and propagate the largest width”

the minimum weight of constituting edges



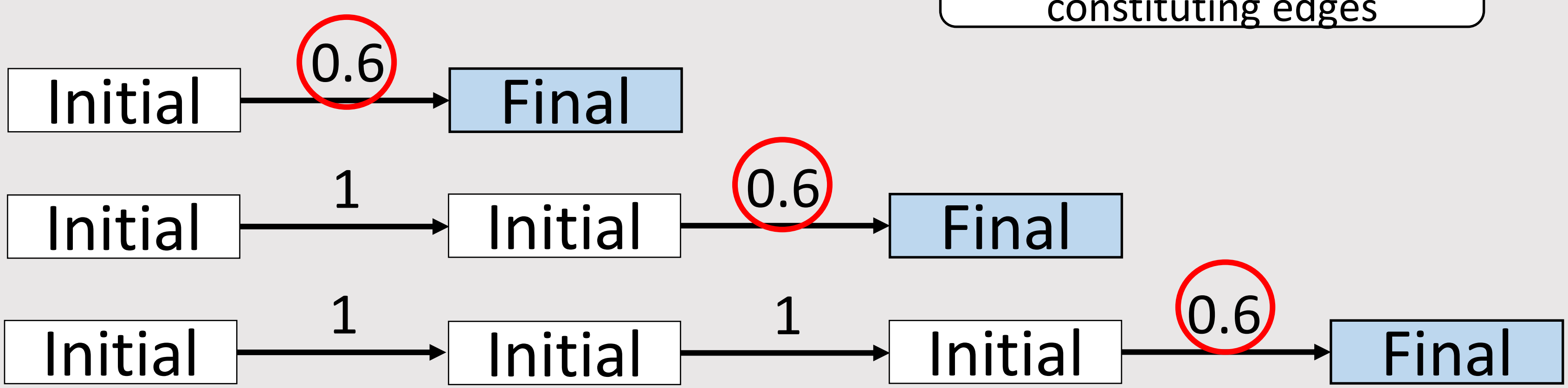
Step 2: update an upper bound U



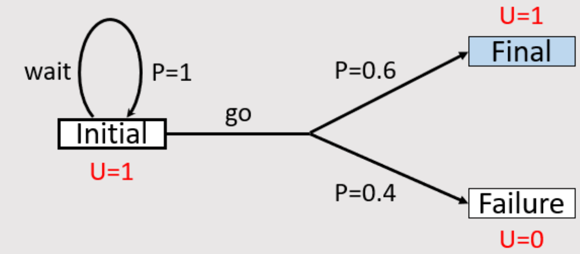


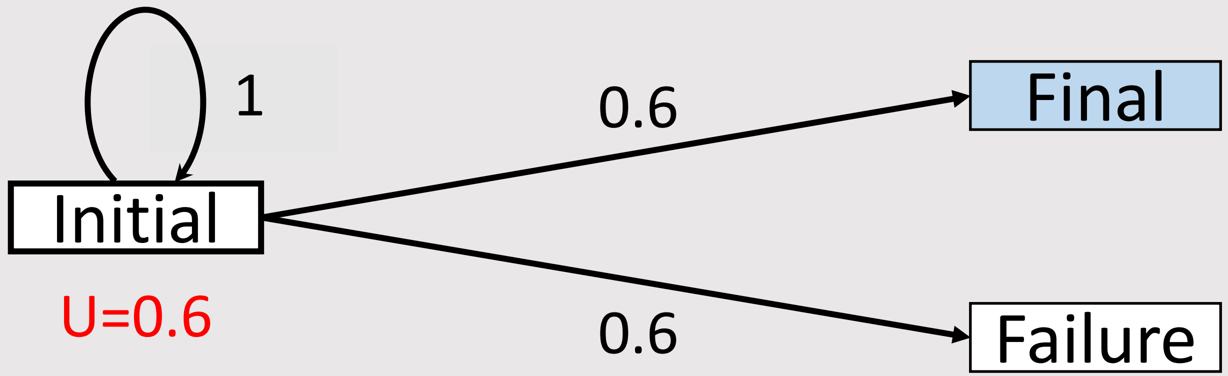
Our alg.: “Compare paths to Final, and propagate the largest width”

the minimum weight of constituting edges



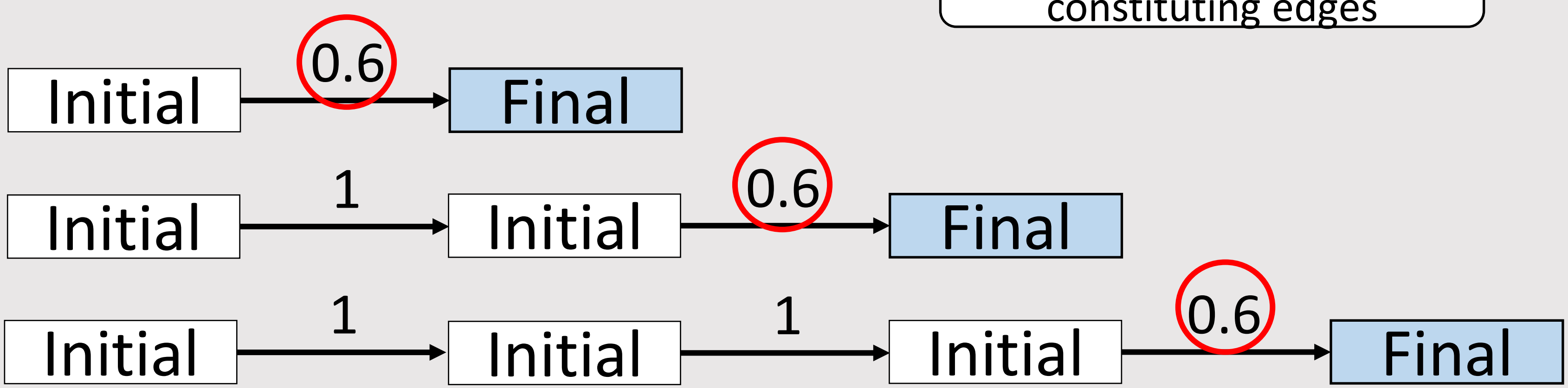
Step 2: update an upper bound U



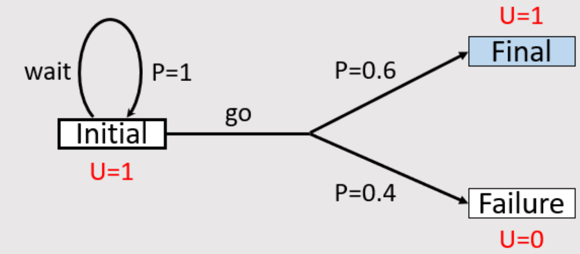


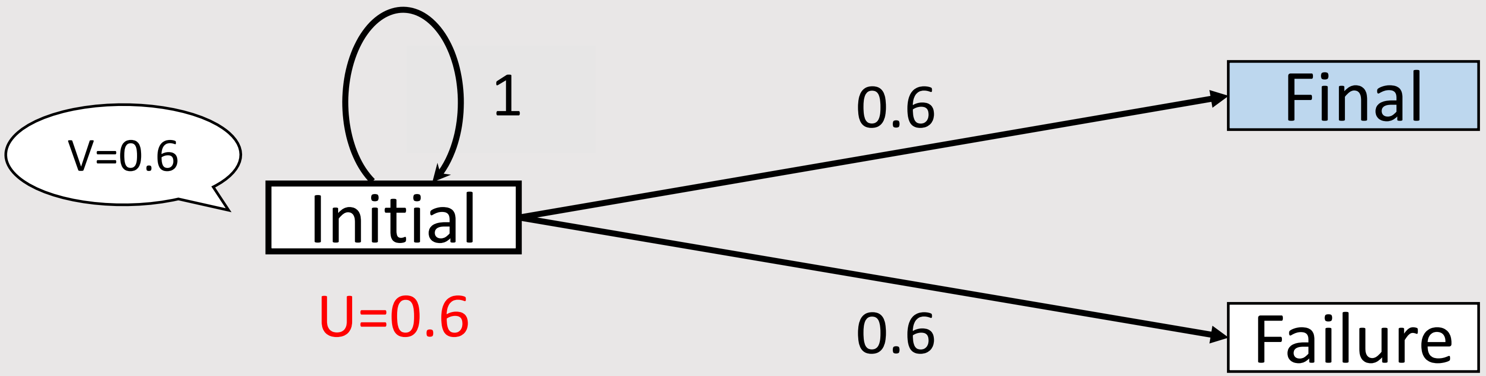
Our alg.: “Compare paths to Final, and propagate the largest width”

the minimum weight of constituting edges



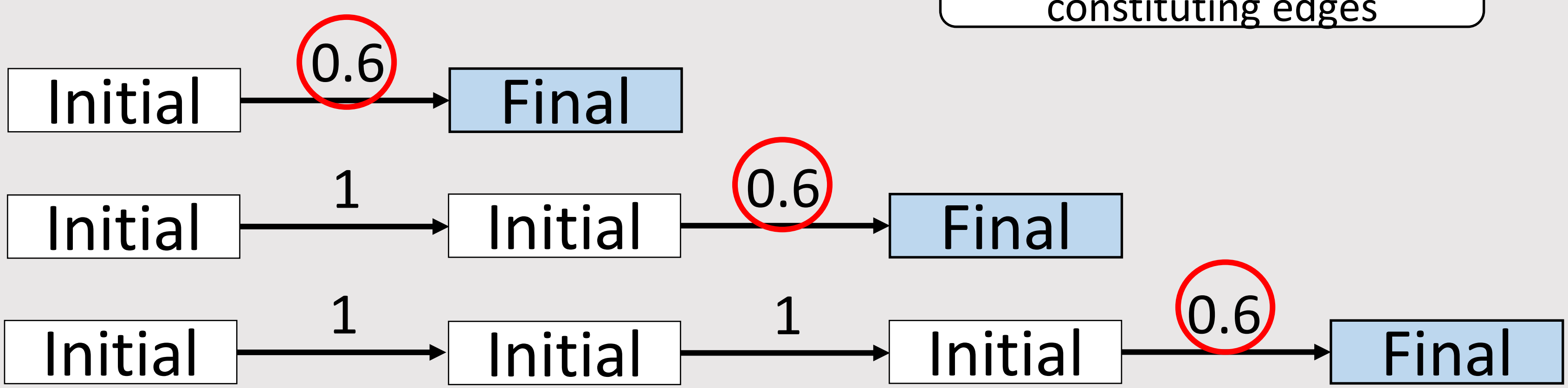
Step 2: update an upper bound U



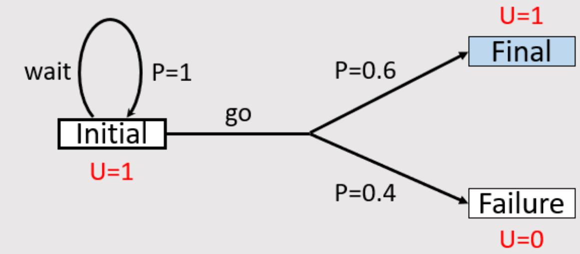


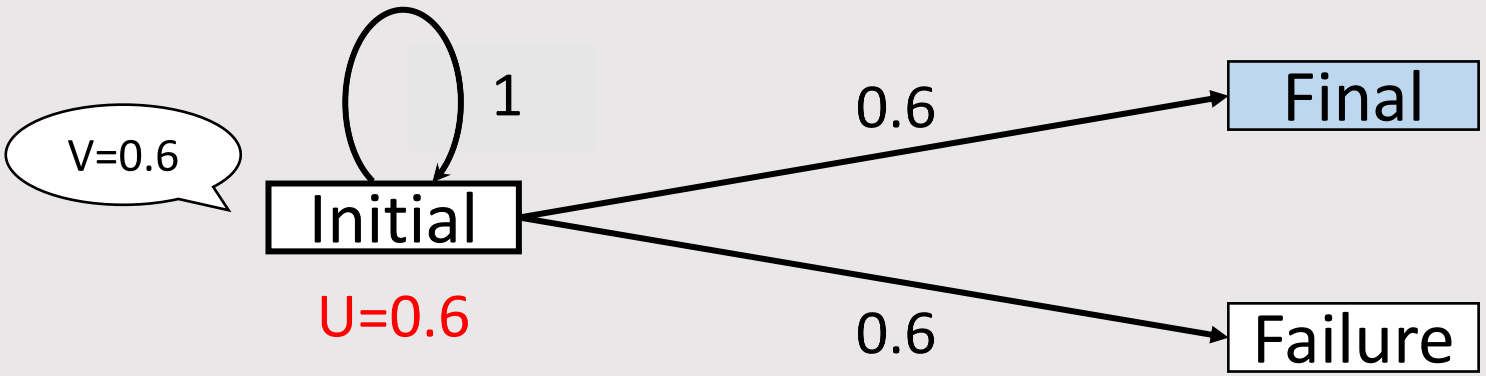
Our alg.: “Compare paths to Final, and propagate the largest width”

the minimum weight of constituting edges

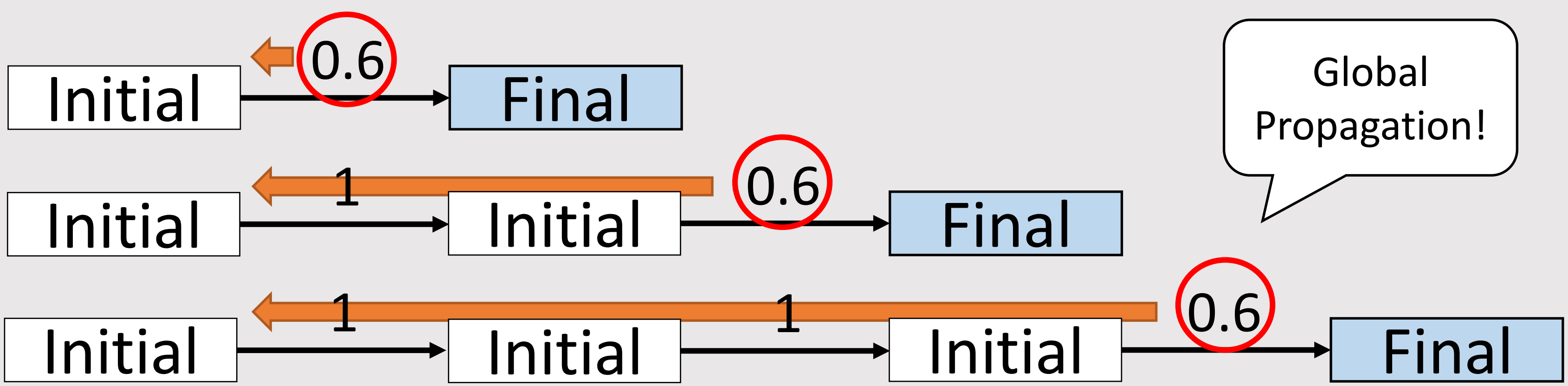


Step 2: update an upper bound U

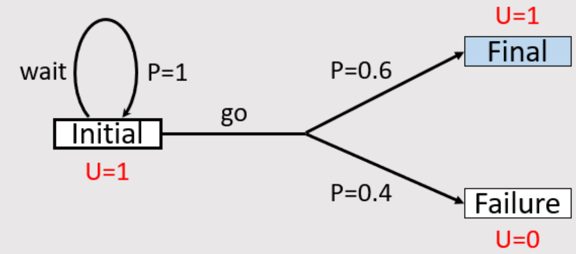


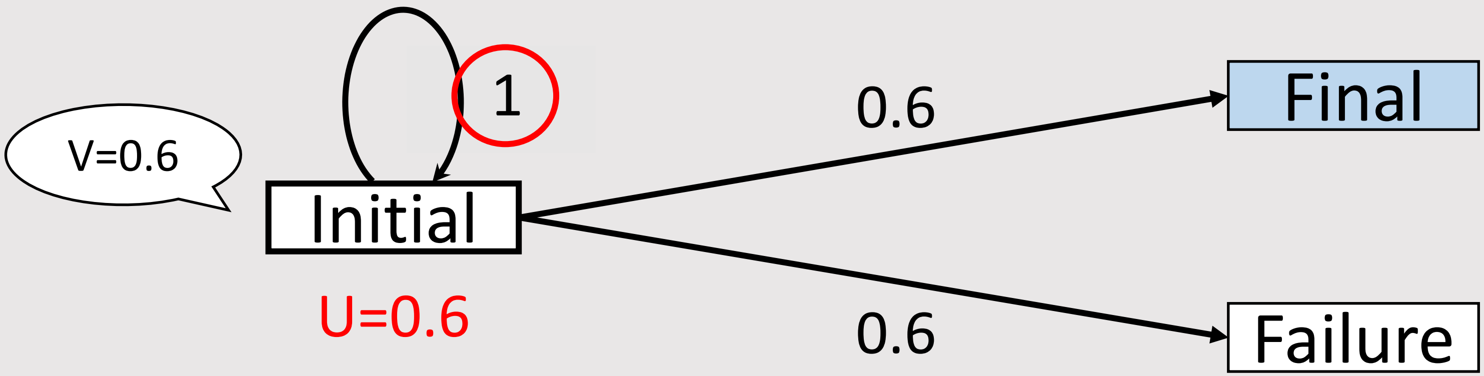


Our alg.: “Compare paths to Final, and propagate the largest width”

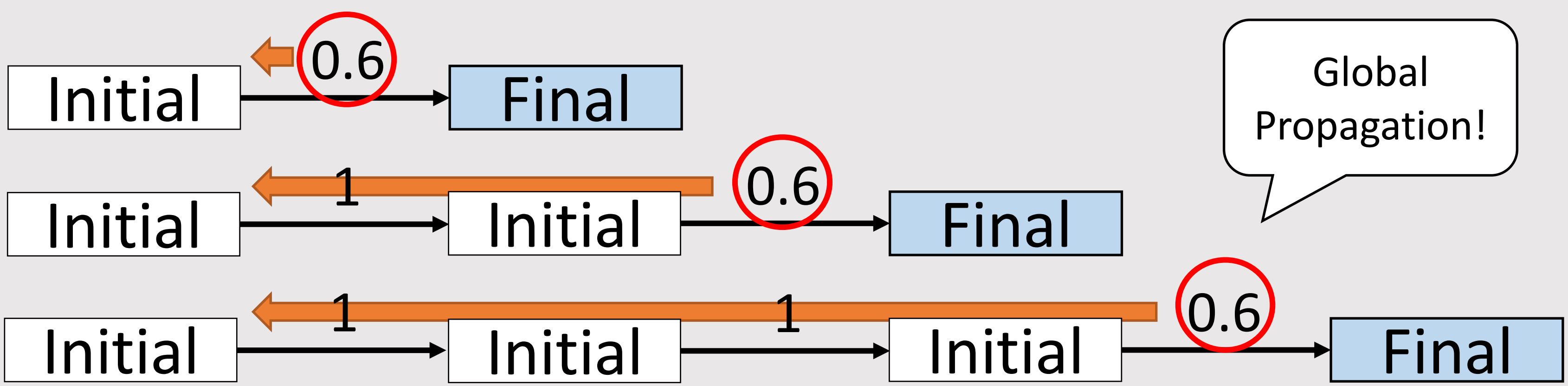


Step 2: update an upper bound U

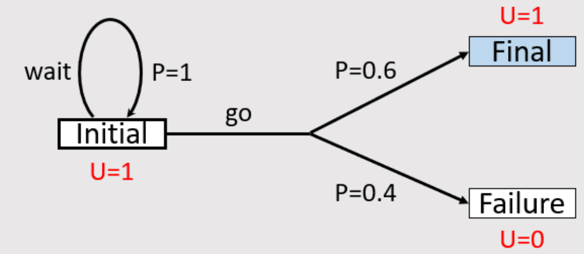


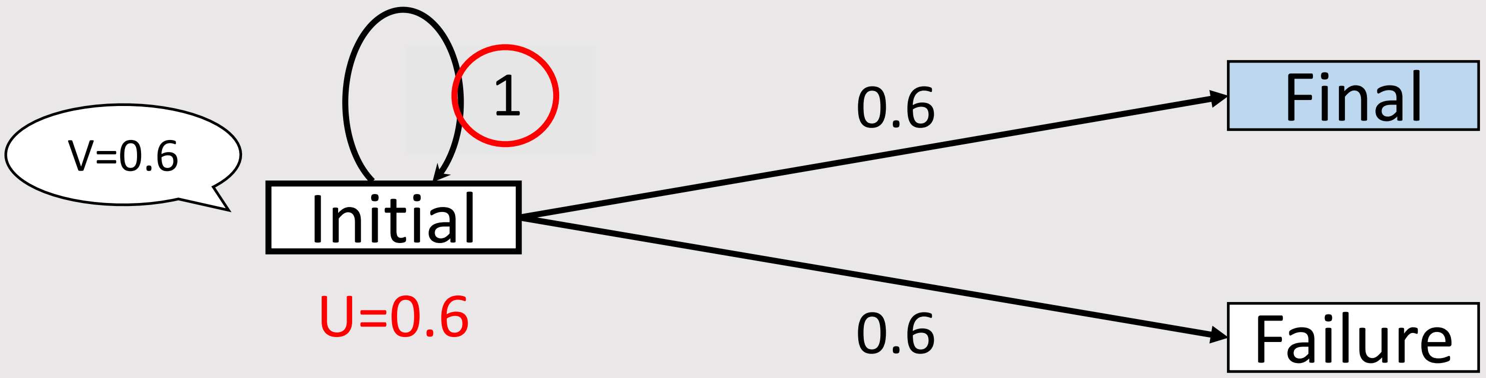


Our alg.: “Compare paths to Final, and propagate the largest width”

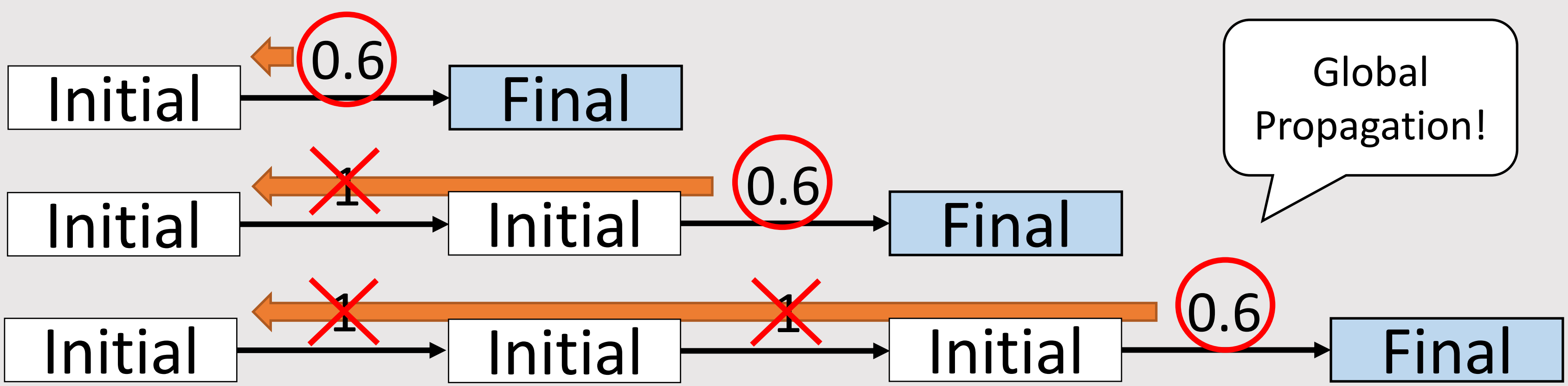


Step 2: update an upper bound U

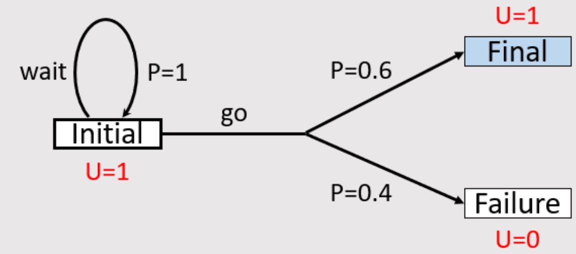


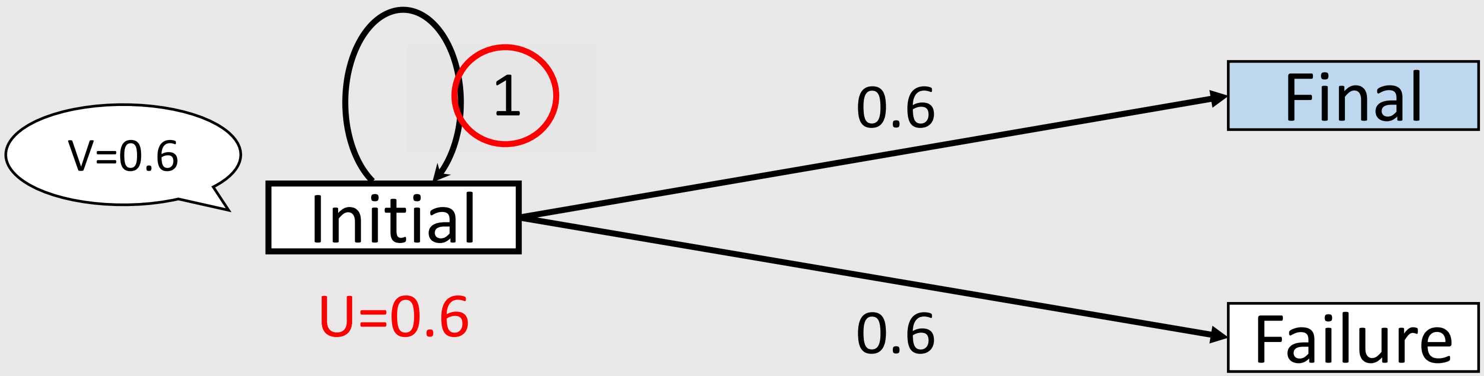


Our alg.: "Compare paths to Final, and propagate the largest width"

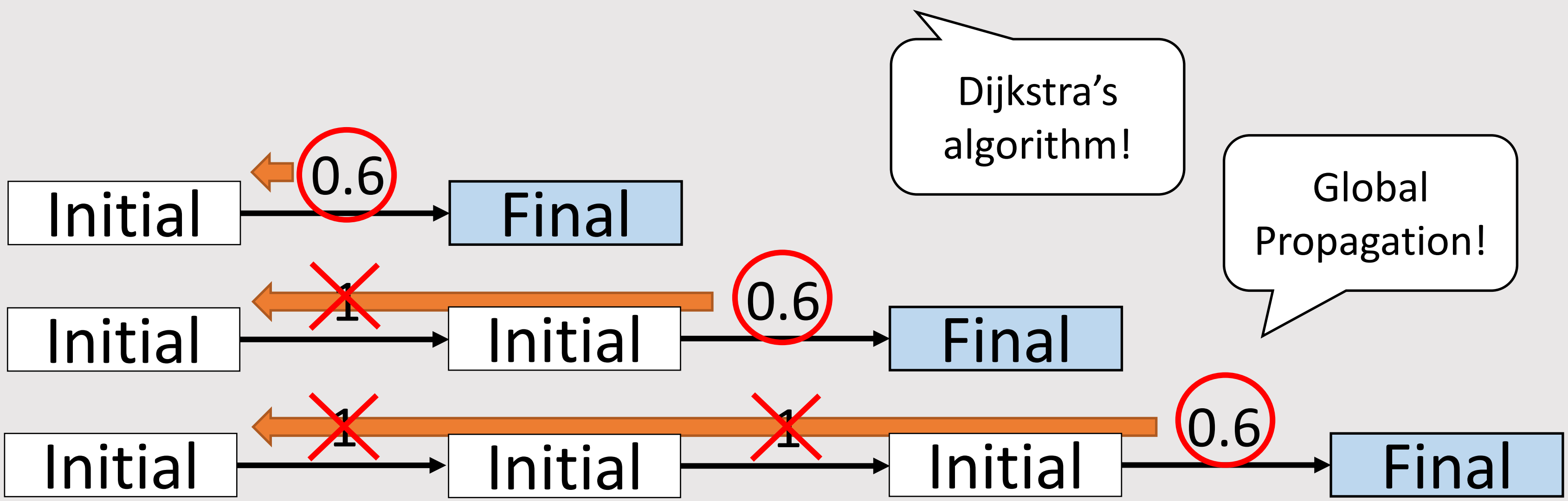


Step 2: update an upper bound U





Our alg.: "Compare paths to Final, and propagate the largest width"



Our algorithm for SGs

procedure $\mathrm{BVI\_WP}(\mathcal{G}, \varepsilon)$

$L_0 \leftarrow \bot,\ U_0 \leftarrow \top,\ i \leftarrow 0$

while $U_i(s_I) - L_i(s_I) > \varepsilon$ **do**

- $i{+}{+}$
- $L_i \leftarrow \mathbb{X}L_{i-1}$ `// value iteration for lower bounds`
- $\mathcal{M}_i \leftarrow \mathcal{M}_{\mathrm{PlRd}}(\mathcal{G}, L_i)$ `// player reduction`
- $\mathcal{W}_i \leftarrow \mathcal{W}_{\mathrm{LcPg}}(\mathcal{M}_i, U_{i-1})$ `// local propagation`
- $U_i \leftarrow \min\{U_{i-1}, \mathrm{WPW}(\mathcal{W}_i)\}$ `// widest path computation`

return $U_i(s_I)$

Our algorithm for SGs

procedure $\underline{\mathrm{BVI\_WP}(\mathcal{G}, \varepsilon)}$ $\mathcal{G}$: SG, ε: precision requirement

$L_0 \leftarrow \bot,\ U_0 \leftarrow \top,\ i \leftarrow 0$

while $U_i(s_I) - L_i(s_I) > \varepsilon$ **do**

$i{+}{+}$

$L_i \leftarrow \mathbb{X} L_{i-1}$ // value iteration for lower bounds

$\mathcal{M}_i \leftarrow \mathcal{M}_{\mathrm{PlRd}}(\mathcal{G}, L_i)$ // player reduction

$\mathcal{W}_i \leftarrow \mathcal{W}_{\mathrm{LcPg}}(\mathcal{M}_i, U_{i-1})$ // local propagation

$U_i \leftarrow \min\{U_{i-1}, \mathrm{WPW}(\mathcal{W}_i)\}$ // widest path computation

return $U_i(s_I)$

Our algorithm for SGs

```
procedure BVI_WP(𝒢, ε)
    L_0 ← ⊥, U_0 ← ⊤, i ← 0
    while U_i(s_I) − L_i(s_I) > ε do
        i++
        L_i ← 𝕏L_{i−1}                          // value iteration for lower bounds
        ℳ_i ← ℳ_PlRd(𝒢, L_i)                    // player reduction
        𝒲_i ← 𝒲_LcPg(ℳ_i, U_{i−1})              // local propagation
        U_i ← min{U_{i−1}, WPW(𝒲_i)}            // widest path computation
    return U_i(s_I)
```

$\mathcal{G}$: SG, ε: precision requirement

Our algorithm for SGs

procedure $\mathrm{BVI\_WP}(\mathcal{G}, \varepsilon)$ — $\mathcal{G}$: SG, ε: precision requirement

$L_0 \leftarrow \bot,\ U_0 \leftarrow \top,\ i \leftarrow 0$

while $U_i(s_I) - L_i(s_I) > \varepsilon$ **do**

$i{+}{+}$

$L_i \leftarrow \mathbb{X} L_{i-1}$ // value iteration for lower bounds

$\mathcal{M}_i \leftarrow \mathcal{M}_{\mathrm{PlRd}}(\mathcal{G}, L_i)$ // player reduction

$\mathcal{W}_i \leftarrow \mathcal{W}_{\mathrm{LcPg}}(\mathcal{M}_i, U_{i-1})$ // local propagation

$U_i \leftarrow \min\{U_{i-1}, \mathrm{WPW}(\mathcal{W}_i)\}$ // widest path computation

return $U_i(s_I)$

Our algorithm for SGs

procedure BVI_WP$(\mathcal{G}, \varepsilon)$ — $\mathcal{G}$: SG, ε: precision requirement

  $L_0 \leftarrow \bot,\ U_0 \leftarrow \top,\ i \leftarrow 0$

  while $U_i(s_I) - L_i(s_I) > \varepsilon$ **do**

    $i{+}{+}$

    $L_i \leftarrow \mathbb{X} L_{i-1}$ `// value iteration for lower bounds`

    MDP $\mathcal{M}_i \leftarrow \mathcal{M}_{\mathrm{PlRd}}(\mathcal{G}, L_i)$ `// player reduction`

    $\mathcal{W}_i \leftarrow \mathcal{W}_{\mathrm{LcPg}}(\mathcal{M}_i, U_{i-1})$ `// local propagation`

    $U_i \leftarrow \min\{U_{i-1}, \mathrm{WPW}(\mathcal{W}_i)\}$ `// widest path computation`

  return $U_i(s_I)$

Our algorithm for SGs

procedure BVI_WP($\mathcal{G}, \varepsilon$)　　$\mathcal{G}$: SG, ε: precision requirement

　$L_0 \leftarrow \bot,\ U_0 \leftarrow \top,\ i \leftarrow 0$

　while $U_i(s_I) - L_i(s_I) > \varepsilon$ **do**

　　i++

　　$L_i \leftarrow \mathbb{X} L_{i-1}$　　`// value iteration for lower bounds`

　　$\mathcal{M}_i \leftarrow \mathcal{M}_{\mathrm{PlRd}}(\mathcal{G}, L_i)$　　`// player reduction`

　　$\mathcal{W}_i \leftarrow \mathcal{W}_{\mathrm{LcPg}}(\mathcal{M}_i, U_{i-1})$　　`// local propagation`

　　$U_i \leftarrow \min\{U_{i-1}, \mathrm{WPW}(\mathcal{W}_i)\}$　　`// widest path computation`

　return $U_i(s_I)$

MDP

For every sufficiently large i,
reachability prob. of $\mathcal{M}_i$ and $\mathcal{G}$ are the same

Our algorithm for SGs

```
procedure BVI_WP(𝒢, ε)
    L_0 ← ⊥, U_0 ← ⊤, i ← 0
    while U_i(s_I) − L_i(s_I) > ε do
        i++
        L_i ← 𝕏L_{i−1}                          // value iteration for lower bounds
        ℳ_i ← ℳ_PlRd(𝒢, L_i)                    // player reduction
        𝒲_i ← 𝒲_LcPg(ℳ_i, U_{i−1})              // local propagation
        U_i ← min{U_{i−1}, WPW(𝒲_i)}            // widest path computation
    return U_i(s_I)
```

$\mathcal{G}$: SG, ε: precision requirement

Our algorithm for SGs

```
procedure BVI_WP(𝒢, ε)        𝒢: SG, ε: precision requirement
    L_0 ← ⊥, U_0 ← ⊤, i ← 0
    while U_i(s_I) − L_i(s_I) > ε do
        i++
        L_i ← 𝕏L_{i−1}                          // value iteration for lower bounds
        ℳ_i ← ℳ_PlRd(𝒢, L_i)                    // player reduction
        𝒲_i ← 𝒲_LcPg(ℳ_i, U_{i−1})              // local propagation
        U_i ← min{U_{i−1}, WPW(𝒲_i)}            // widest path computation
    return U_i(s_I)
```

<u>Theorem(P,T,H,H,2020).</u> Let the while loop iterate forever in the above algorithm. Then it generates a decreasing sequence of functions that converges to optimal reachability probability:

$$U_0 \geq U_1 \geq \cdots \geq U_i \xrightarrow{i \to \infty} V$$

Experimental result

model	Param.	#states	#trans	#EC	[K+, Ver.1] itr	[K+, Ver.1] time(s)	[K+, Ver.2] itr	[K+, Ver.2] time(s)	[K+, learning] itr	[K+, learning] visit%	[K+, learning] time(s)	**Our alg.** itr	**Our alg.** time(s)
mdsm	3	62245	151143	1	121	3	121	4	17339	49.3	15	120	5
	4	335211	882765	1	125	15	125	47	91301	42.1	86	124	38
cloud	5	8842	60437	4421	7	7	7	1	167	6.9	14	7	<1
	6	34954	274965	17477	11	177	11	5	41	0.6	3	11	1
	7	139402	1237525	69701	11	19721	11	62	41	0.2	4	11	5
teamform	3	12475	15228	2754	2	<1	2	<1	972	49.0	137	2	<1
	4	96665	116464	19800	2	<1	2	<1	4154	34.6	9603	2	<1
	5	907993	1084752	176760	2	<1	2	<1			TO	2	<1
investor	50	211321	673810	29690	441	184	441	249			TO	364	48
	100	807521	2587510	114390	801	3318		OOM			TO	688	736
manyECs	500	1004	3007	502	6	7	6	7			TO	5	<1
	1000	2004	6007	1002	6	51	6	51			TO	5	<1
	5000	10004	30007	5002		SO		SO			TO	5	<1

[K+] Kelmendi, E., Kramer, J., Kretnsky, J., Weininger, M.: *Value iteration for simple stochastic games: stopping criterion and learning algorithm*. Proc. CAV 2018

- Precision constant = 10^{-6}
  (i.e. an approx. with 10^{-6} error range is returned for each successful runs)
- Green shaded = fastest
- Gray shaded = computation failure
  (TO=timeout (6hours), OOM= out of memory, SO=stack overflow)

Experimental result

model	Param.	#states	#trans	#EC	[K+, Ver.1] itr	[K+, Ver.1] time(s)	[K+, Ver.2] itr	[K+, Ver.2] time(s)	[K+, learning] itr	[K+, learning] visit%	[K+, learning] time(s)	**Our alg.** itr	**Our alg.** time(s)
mdsm	3	62245	151143	1	121	3	121	4	17339	49.3	15	120	5
	4	335211	882765	1	125	15	125	47	91301	42.1	86	124	38
cloud	5	8842	60437	4421	7	7	7	1	167	6.9	14	7	<1
	6	34954	274965	17477	11	177	11	5	41	0.6	3	11	1
	7	139402	1237525	69701	11	19721	11	62	41	0.2	4	11	5
teamform	3	12475	15228	2754	2	<1	2	<1	972	49.0	137	2	<1
	4	96665	116464	19800	2	<1	2	<1	4154	34.6	9603	2	<1
	5	907993	1084752	176760	2	<1	2	<1			TO	2	<1
investor	50	211321	673810	29690	441	184	441	249			TO	364	48
	100	807521	2587510	114390	801	3318		OOM			TO	688	736
manyECs	500	1004	3007	502	6	7	6	7			TO	5	<1
	1000	2004	6007	1002	6	51	6	51			TO	5	<1
	5000	10004	30007	5002		SO		SO			TO	5	<1

[K+] Kelmendi, E., Kramer, J., Kretnsky, J., Weininger, M.: *Value iteration for simple stochastic games: stopping criterion and learning algorithm*. Proc. CAV 2018

Fastest in 7/13 instances

- Precision constant = 10^{-6}
  (i.e. an approx. with 10^{-6} error range is returned for each successful runs)
- Green shaded = fastest
- Gray shaded = computation failure
  (TO=timeout (6hours), OOM= out of memory, SO=stack overflow)

Experimental result

model	Param.	#states	#trans	#EC	[K+, Ver.1] itr	[K+, Ver.1] time(s)	[K+, Ver.2] itr	[K+, Ver.2] time(s)	[K+, learning] itr	[K+, learning] visit%	[K+, learning] time(s)	**Our alg.** itr	**Our alg.** time(s)
mdsm	3	62245	151143	1	121	3	121	4	17339	49.3	15	120	5
	4	335211	882765	1	125	15	125	47	91301	42.1	86	124	38
cloud	5	8842	60437	4421	7	7	7	1	167	6.9	14	7	<1
	6	34954	274965	17477	11	177	11	5	41	0.6	3	11	1
	7	139402	1237525	69701	11	19721	11	62	41	0.2	4	11	5
teamform	3	12475	15228	2754	2	<1	2	<1	972	49.0	137	2	<1
	4	96665	116464	19800	2	<1	2	<1	4154	34.6	9603	2	<1
	5	907993	1084752	176760	2	<1	2	<1			TO	2	<1
investor	50	211321	673810	29690	441	184	441	249			TO	364	48
	100	807521	2587510	114390	801	3318		OOM			TO	688	736
manyECs	500	1004	3007	502	6	7	6	7			TO	5	<1
	1000	2004	6007	1002	6	51	6	51			TO	5	<1
	5000	10004	30007	5002		SO		SO			TO	5	<1

[K+] Kelmendi, E., Kramer, J., Kretnsky, J., Weininger, M.: *Value iteration for simple stochastic games: stopping criterion and learning algorithm*. Proc. CAV 2018

Slow/failure sometimes

Stably fast

- Precision constant = 10^{-6}
  (i.e. an approx. with 10^{-6} error range is returned for each successful runs)
- Green shaded = fastest
- Gray shaded = computation failure
  (TO=timeout (6hours), OOM= out of memory, SO=stack overflow)

Summary

We introduced a novel algorithm of Bounded Value Iteration (BVI) which is faster than the existing one.

Future works

- Adapt the technique to a general reward setting (currently reachability only)
- Extend applicability of the technique to more complicated systems (e.g. black box ones)