Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

[DΣIM]

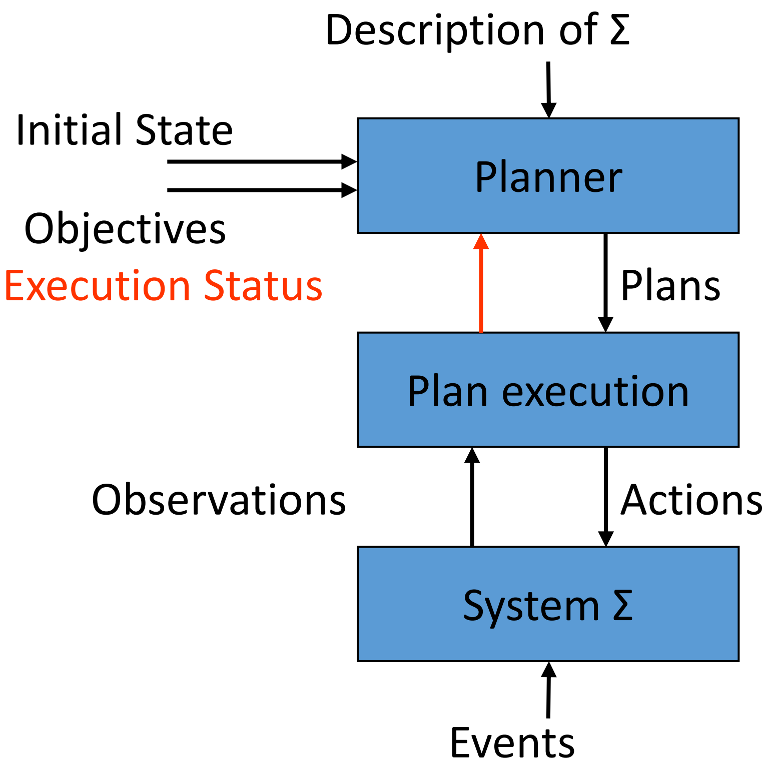
Planning and Approximate Reasoning

Hatem A. Rashwan

Planning under uncertainty: MDPs

MESIIA – MIA

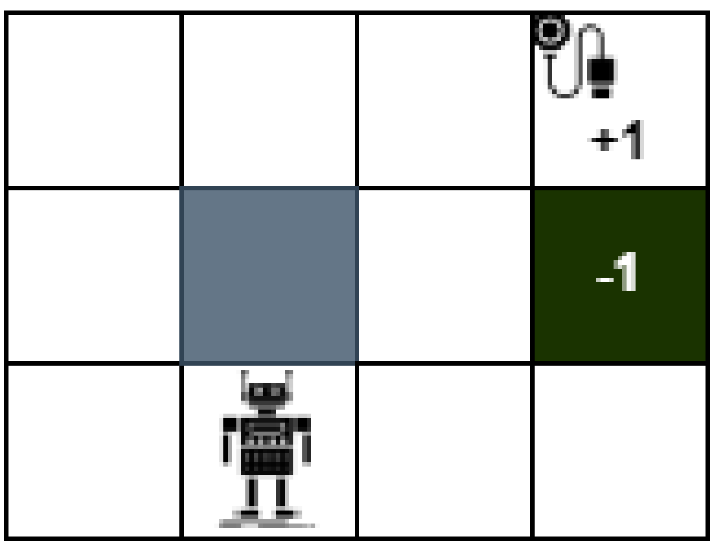
Plan Execution

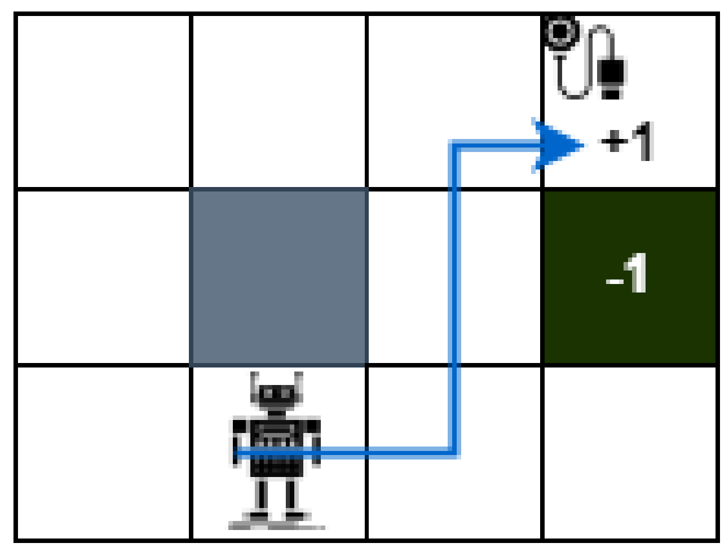


- **problem**: real world differs from model described by Σ
- **more realistic model**: interleaved planning and execution
  - plan supervision
  - plan revision
  - re-planning
- **dynamic planning**: closed loop between planner and controller
  - execution status

Planning Example
Robot navigation

- A robot act in the environment aiming to avoid the obstacles and reach its goal.
- The robot aims to reach the celled labelled with +1 and avoid the cell labelled with -1



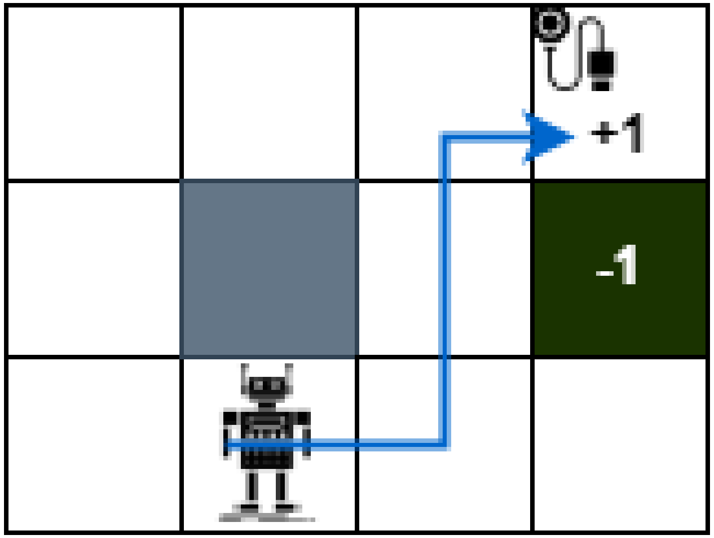


Planning Example
Why the short path is problematic

- Non-perfect execution: actions are performed with a probability < 1
- What are the best actions for a robot (an agent) under this constraint? (move right->up->up->right)
- And what is the maximum performance of the execution? (e.g., 99%)

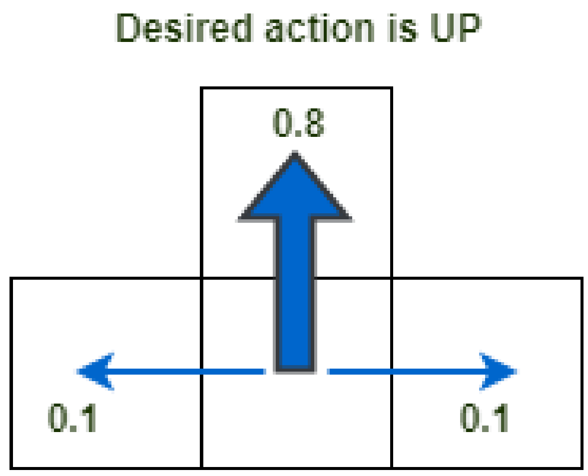
Example

- A robot does not exactly perform the desired motions due to different reasons.
    - Uncertainty about performing actions



Planning Example
Non-Deterministic Transitions

- Consider non-deterministic transition model (UP/R/L)

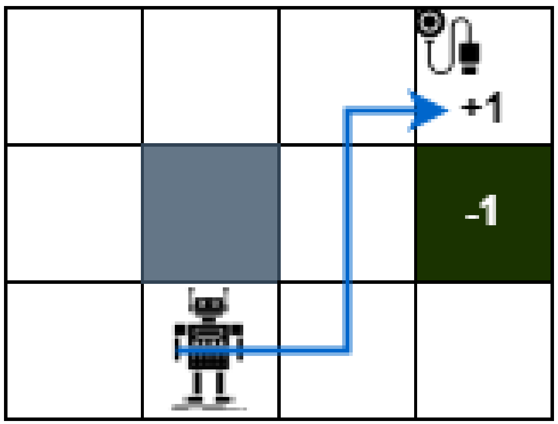


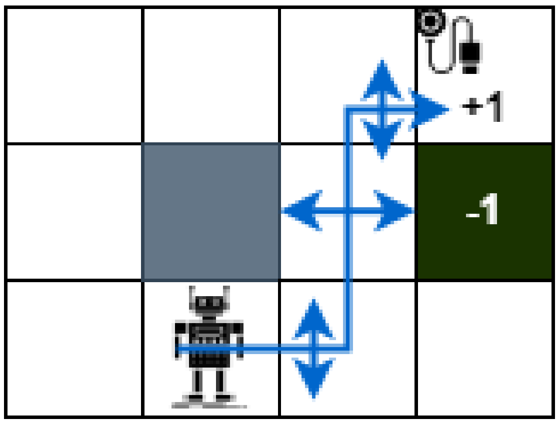
Example

- Intended action is executed with p=0.8
- With p = 0.1, the robot can move right or left.
  - Uncertainty about performing actions

Markov Decision Process (MDP) motivation

- Executing the A* plan



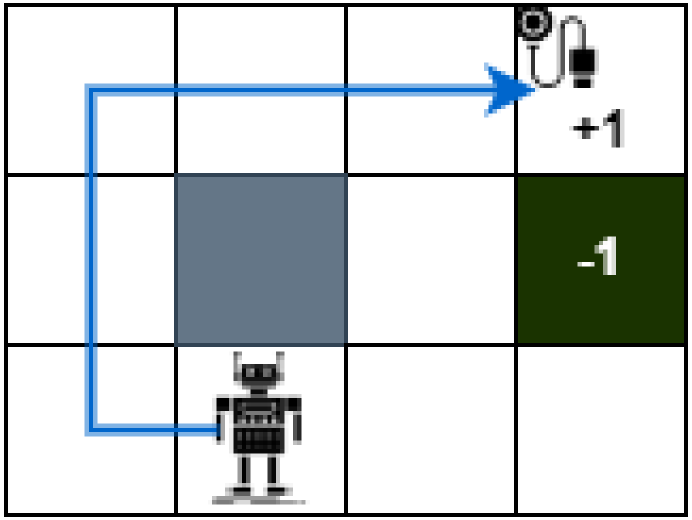


Transitions are non-deterministic

- Uncertainty about performing actions will be occurred

MDP motivation

- Perhaps using longer path with lower probability to not reach the cell labelled -1 is good option.



This proposed path can have the highest overall utility.

Axioms of Probability

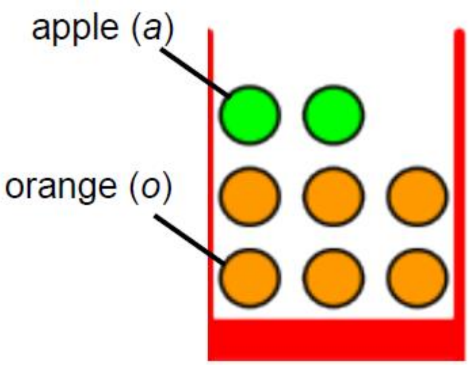
- Let A be a proposition about the world
- P(A) = probability proposition A is true
- 0 <= P(A) <= 1
- P(True) = 1
- P(False) = 0
- P(A or B) = P(A) + P(B) – P(A and B)

Random Variables

- Random Variables: variables in probability to capture phenomena
- A random variable has a **domain of values** it can take on.
- Probability distribution function represents
  “**probability of each value**"

Example – Pick Fruit from Basket

- Random variable: F
- Domain: a, o
- PDF:
  - p(F = a) = ¼
  - p(F = o) = ¾



- The **expected value** of a function of a random variable **is the weighted average of the probability distribution over outcomes.**

- Example: calculate the expected time of waiting for an elevator
- Time: 5ms 2ms 0.5ms
- Probability: 0.2 0.7 0.1

5 x 0.2 + 2 x 0.7+ 0.5 x 0.1 = **2.45ms**

Transition Model

• Given that our agent is in some state s, there is a probability to go to the first state, another probability to go to the second state, and so on for every existing state.

- This is our transition probability.
- Probability to reach the next state *s'* from state *s* by choosing action *a* is $P(s, a, s') \sim P(s|s', a)$ => It is called **Transition model**

Markov Property:

The transition probabilities from *s* to *s'* depend only on the current state *s* and not on the history of earlier states.

Reward:

- In each state s, the robot (agent) receives a reward *R(s)*.
- The reward may be positive or negative but must be bounded
- This can be generalized to be a reward function *R(s, a, s')*

Reward

• In our example, the reward is **-0.04** in all states (e.g**., the cost of the motion**) except the terminal states (**reward is +1/-1**)

A negative reward gives an incentive

- **to reach the goal quickly**
- **Or “living in this environment is not enjoyable”**

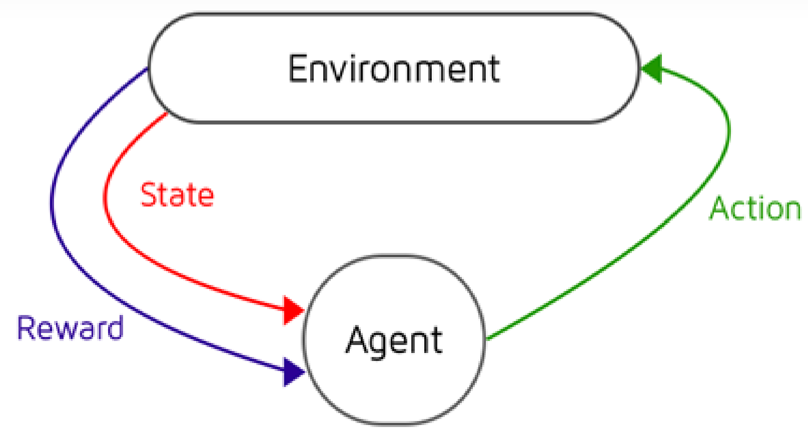
-0.04	-0.04	-0.04	+1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

Markov Decision Process-MDP

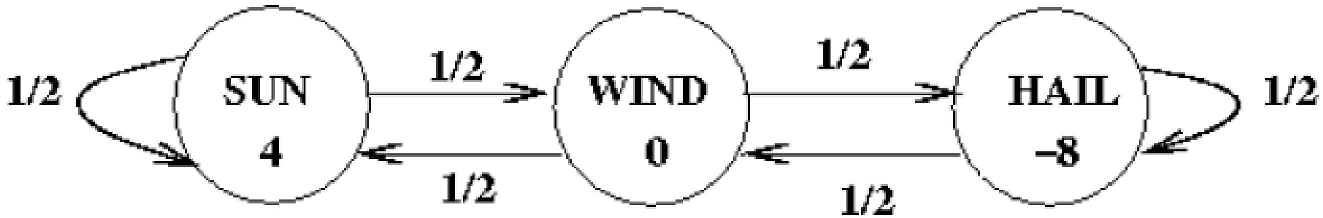
Given a **sequential decision problem** in fully observable, stochastic environment with a known Markovian transition model

Then a Markov Decision Process-MDP ***(S, A, Si, P, R)*** is defined by the components of:

- **Set of states: *S***
- **Set of actions: *A***
- **Initial sates: *Si***
- **Transition model: *P(s,a,s')***
- **Reward function: *R(s)***
    - Here: considering only R(s) (does not change the problem)



Example – Markov System with Reward



- Process/observation:
  - Assume start state s_i
  - Receive immediate reward r_i
  - Move, or observe a move, randomly to a new state according to the probability transition matrix
  - **Future rewards (of next state) are discounted by γ**

probability transition matrix *T* is

sun	wind	hail
1/2	1/2	0
1/2	0	1/2
0	1/2	1/2

Example – Markov System with Reward

Solving a Markov System with Rewards

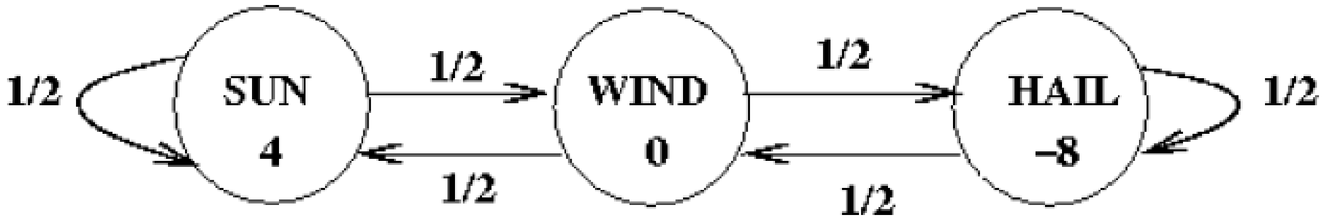
- $V^*(s_i)$ - expected discounted sum of future rewards starting in state s_i
- $V^*(s_i) = r_i + \gamma[p_{i1}V^*(s_1) + p_{i2}V^*(s_2) + \dots\ p_{in}V^*(s_n)]$

- $\boldsymbol{\gamma}$ is a discount factor, where $\boldsymbol{\gamma} \in [0, 1]$.
- It informs the agent of how much it should care about rewards now to rewards in the future.
- If ($\boldsymbol{\gamma}$ = 0), that means the agent is short-sighted, in other words, it only cares about the first reward.
- If ($\boldsymbol{\gamma}$ = 1), that means the agent is far-sighted, i.e. it cares about all future rewards.
- What we care about is the total rewards that we're going to get.

Value Iteration to Solve a Markov System with Rewards

- $V_1(s_i)$ - expected discounted sum of future rewards starting in state s_i *for one step*.
- $V_2(s_i)$ - expected discounted sum of future rewards starting in state s_i *for two steps*.
- ...
- $V_\mathrm{k}(s_i)$ - expected discounted sum of future rewards starting in state *si for k steps*.
- As $k \rightarrow \infty V_\mathrm{k}(s_i) \rightarrow V^*(s_i)$
- Stop when difference of $k + 1$ and k values is smaller than some $\in$.

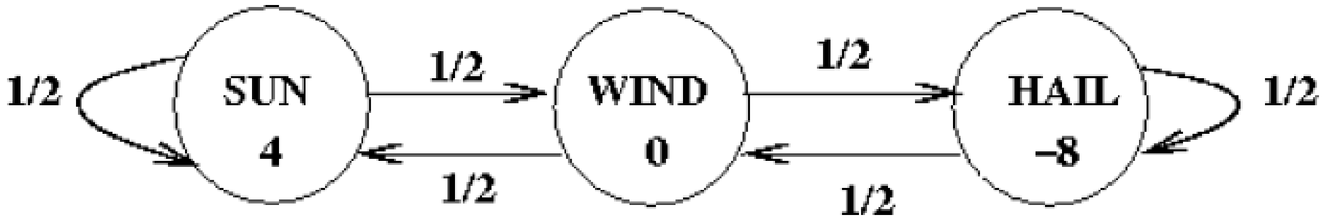
Example – Markov System with Reward



3-State Example: Values γ = 0.5

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	5.0	-1.0	-10.0
3	5.0	-1.25	-10.75
4	4.9375	-1.4375	-11.0
5	4.875	-1.515625	-11.109375
6	4.8398437	-1.5585937	-11.15625
7	4.8203125	-1.5791016	-11.178711
8	4.8103027	-1.5895996	-11.189453
9	4.805176	-1.5947876	-11.194763
10	4.802597	-1.5973969	-11.197388
11	4.8013	-1.5986977	-11.198696
12	4.8006506	-1.599349	-11.199348
13	4.8003254	-1.5996745	-11.199675
14	4.800163	-1.5998373	-11.199837
15	4.8000813	-1.5999185	-11.199919

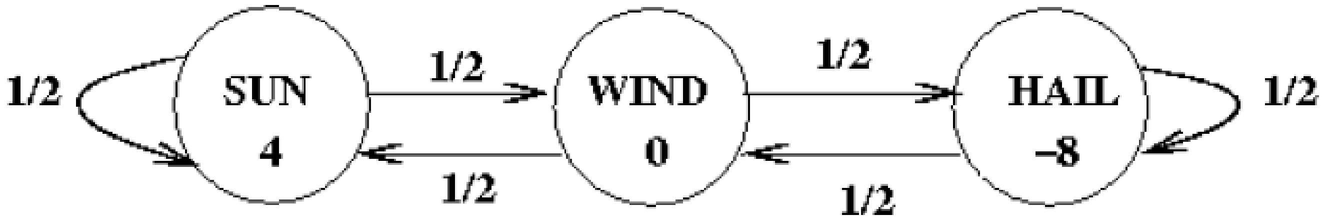
Example – Markov System with Reward



3-State Example: Values $\gamma = 0.9$

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	5.8	-1.8	-11.6
3	5.8	-2.6100001	-14.030001
4	5.4355	-3.7035	-15.488001
5	4.7794	-4.5236254	-16.636175
6	4.1150985	-5.335549	-17.521912
7	3.4507973	-6.0330653	-18.285858
8	2.8379793	-6.6757774	-18.943516
9	2.272991	-7.247492	-19.528683
...	...	...	...
50	-2.8152928	-12.345073	-24.633476
51	-2.8221645	-12.351946	-24.640347
52	-2.8283496	-12.3581295	-24.646532
...	...	...	...
86	-2.882461	-12.412242	-24.700644
87	-2.882616	-12.412397	-24.700798
88	-2.8827558	-12.412536	-24.70094

Example – Markov System with Reward

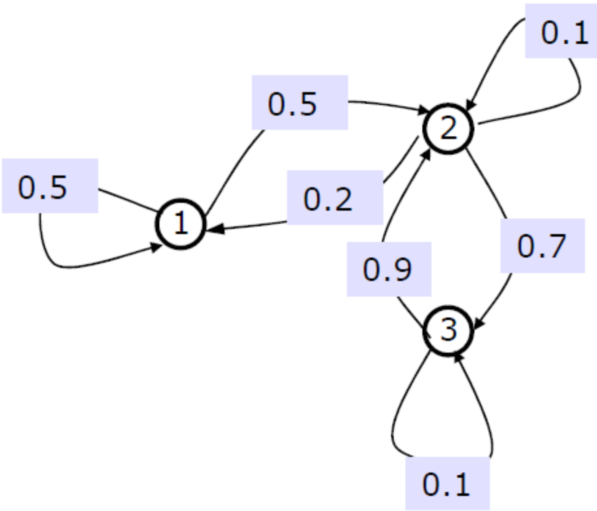


3-State Example: Values $\gamma = 0.2$

Iteration	SUN	WIND	HAIL
0	0	0	0
1	4	0	-8
2	4.4	-0.4	-8.8
3	4.4	-0.44000003	-8.92
4	4.396	-0.452	-8.936
5	4.3944	-0.454	-8.9388
6	4.39404	-0.45443997	-8.93928
7	4.39396	-0.45452395	-8.939372
8	4.393944	-0.4545412	-8.939389
9	4.3939404	-0.45454454	-8.939393
10	4.3939395	-0.45454526	-8.939394
11	4.3939395	-0.45454547	-8.939394
12	4.3939395	-0.45454547	-8.939394

Markov Chain - Example

- Markov Chain
  - states
  - transitions
  - rewards
  - no actions



- Value of a state, using infinite discounted horizon

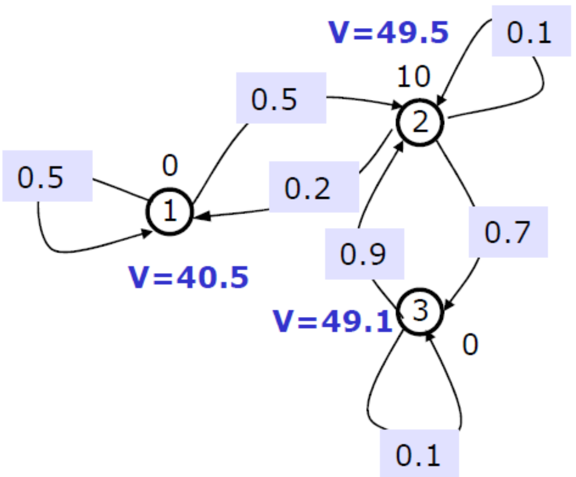
$V^*(s_i) = r_i + \gamma[p_{i1}V^*(s_1) + p_{i2}V^*(s_2) + \dots p_{in}V^*(s_n)]$

- Assume γ=0.9

V(1)= 0+ .9(.5 V(1)+.5 V(2))

V(2)=10+ .9(.2 V(1)+.1 V(2)+.7 V(3))

V(3)= 0+ .9(.9 V(2)+.1 V(3))



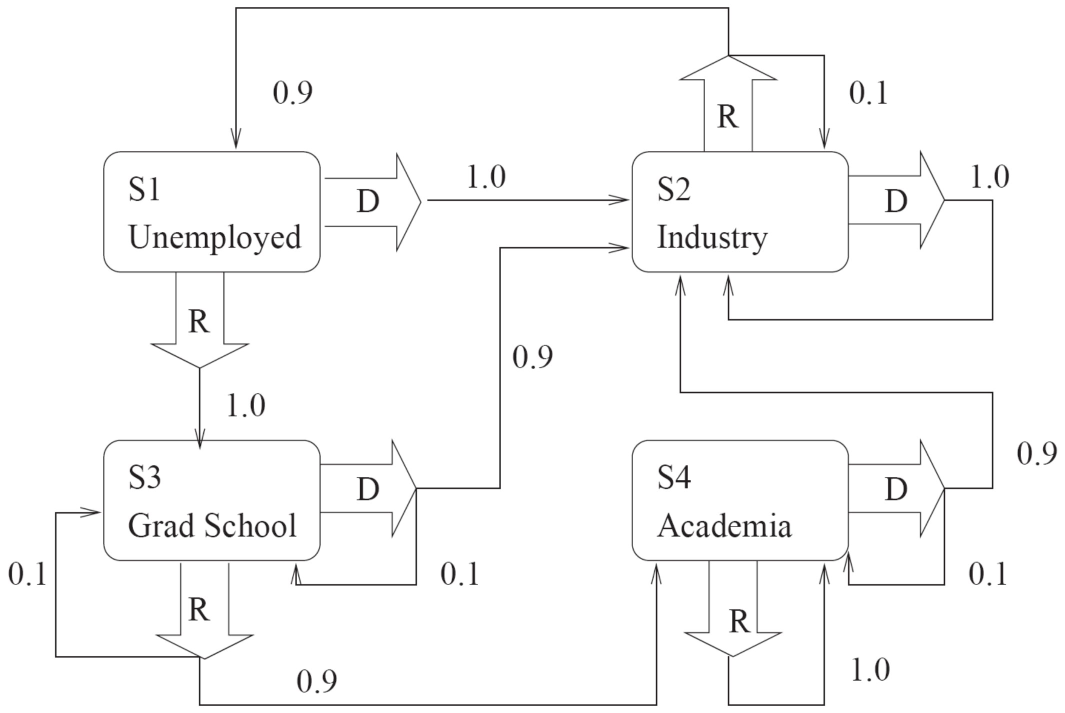
Markov Decision Processes

- Finite set of states, $s_1,\ldots, s_n$
- **Finite set of actions, $a_1,\ldots, a_m$**
- Probabilistic state, action transitions:
- $P_{ij}^{K} = prob\ (next = s_j \mid current = s_i \mid take\ action = a_k)$
- ***Markov property***: State transition function only dependent on current state, not on the "history" of how the state was reached.
- Reward for each state, $r1,\ldots, r_n$
- Process:
  - Start in state s_i
  - Receive immediate reward r_i
  - Choose action $a_k \in A$
  - Change to state s_j with probability P_{ij}^{k}
  - Discount future rewards

Solving an MDP

- Find an action to apply to each state.
- A policy is a mapping from states to actions.
- Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.
- For every MDP there exists an optimal policy.
- Solving an MDP is finding an optimal policy.
- A specific policy converts an MDP into a plain Markov system with rewards.

Solving an MDP - Example



- Note the need to have a finite set of states and actions (R or D).
  R=> Research, and D=> Development
- Note the need to have all transition probabilities.

Value Iteration

- $V^*(s_i)$ - expected discounted future rewards, if we start from state s_i, and we follow the optimal policy.
- Compute V^* with value iteration:

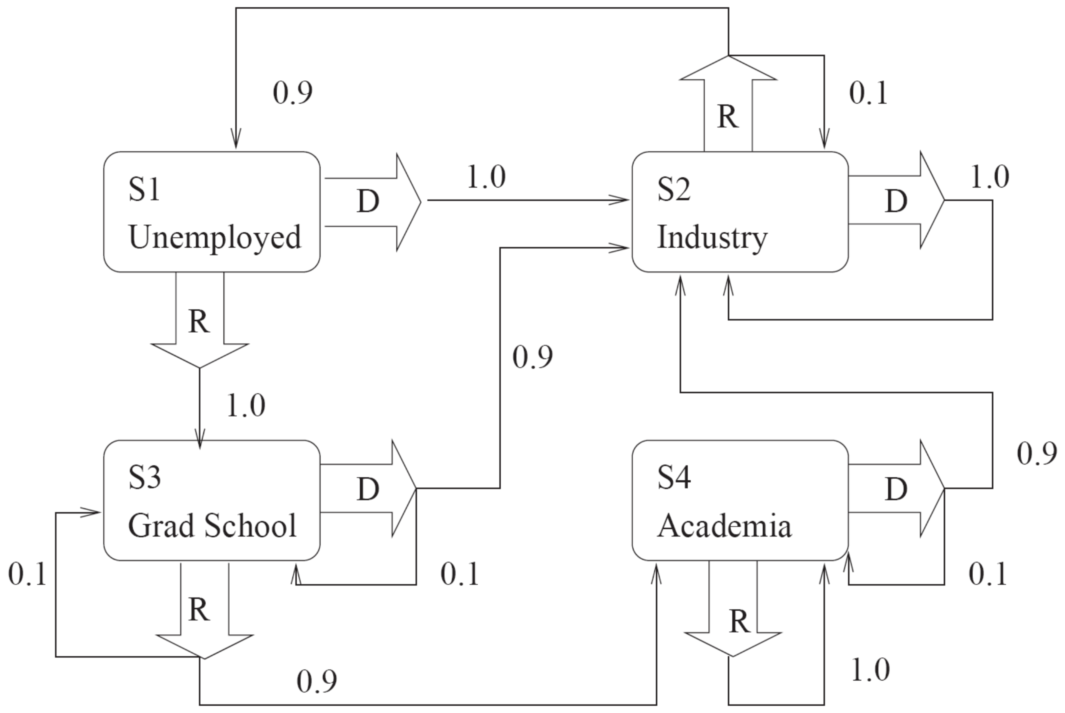
– $V_k(s_i)$ = maximum possible future sum of rewards starting from state s_i for k steps.

- Bellman's Equation:

$$V^{n+1}(s_i) = max_k\{r_i + \gamma \sum_{j=1}^{N} P_{ij}^{k} V^n(s_j)\}$$

- Dynamic programming

Nondeterministic Example



- Reward and discount factor to be decided.
- Note the need to have a finite set of states and actions.
- Note the need to have all transition probabilities.

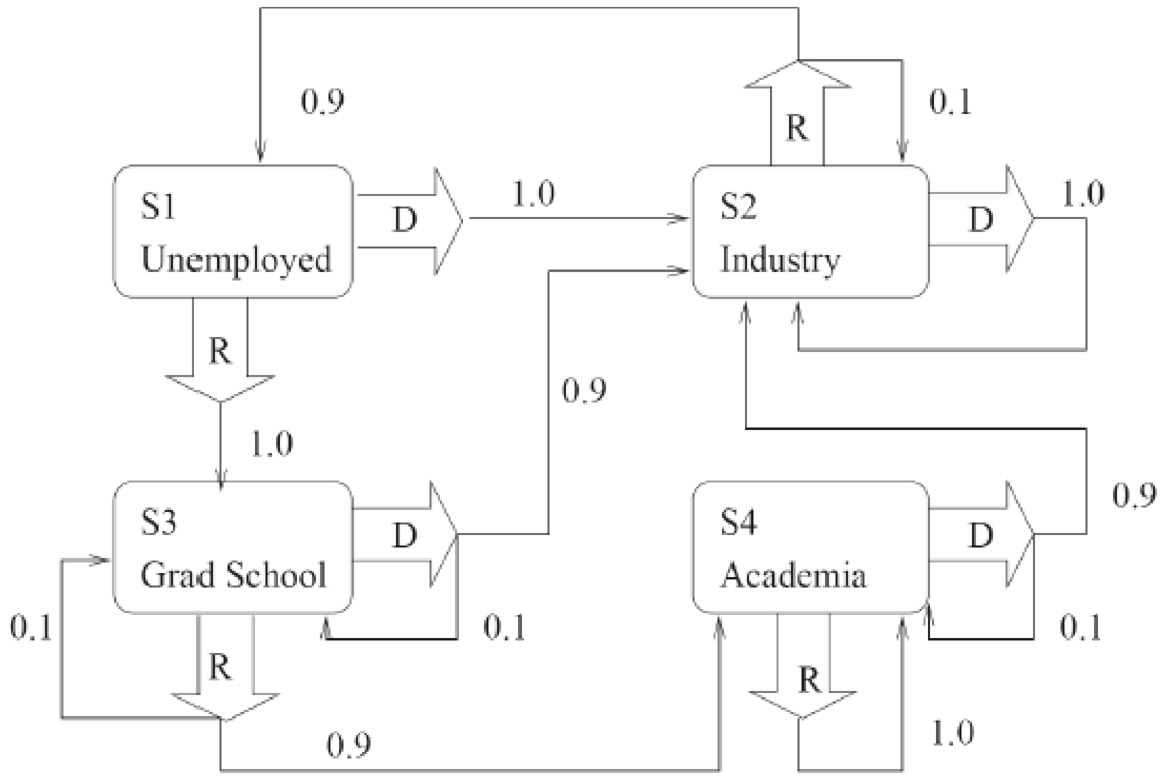
Value Iteration-Bellman equation

- Start with some policy $\pi_0(s_i)$.
- Such policy transforms the MDP into a plain Markov system with rewards.
- Compute the values of the states according to the current policy.
- Update policy:

$$\pi_{k+1}(s_i) = r_i + \arg max_a\{\gamma \sum_{j=1}^{N} P_{ij}^{a} V^{\pi_n}(s_j)\}$$

- Keep computing
- Stop when $\pi_{k+1} = \pi_k$.

Nondeterministic Example



REWARD

S1	S2	S3	S4
0	100	0	10

Nondeterministic Example

$\pi^*(s) = D$, for any s= S1, S2, S3, and S4, $\gamma = 0.9$.

```
V*(S2) = r(S2,D) + 0.9 (1.0 V*(S2))
V*(S2) = 100 + 0.9 V*(S2)
V*(S2) = 1000.

V*(S1) = r(S1,D) + 0.9 (1.0 V*(S2))
V*(S1) = 0 + 0.9 x 1000
V*(S1) = 900.

V*(S3) = r(S3,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S3))
V*(S3) = 0 + 0.9 (0.9 x 1000 + 0.1 V*(S3))
V*(S3) = 81000/91.

V*(S4) = r(S4,D) + 0.9 (0.9 V*(S2) + 0.1 V*(S4))
V*(S4) = 40 + 0.9 (0.9 x 1000 + 0.1 V*(S4))
V*(S4) = 85000/91.
```

Solve the MDP

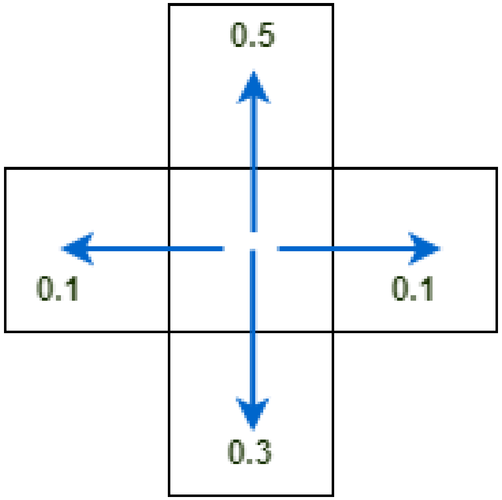
Markov Decision Processes satisfy both mentioned properties.

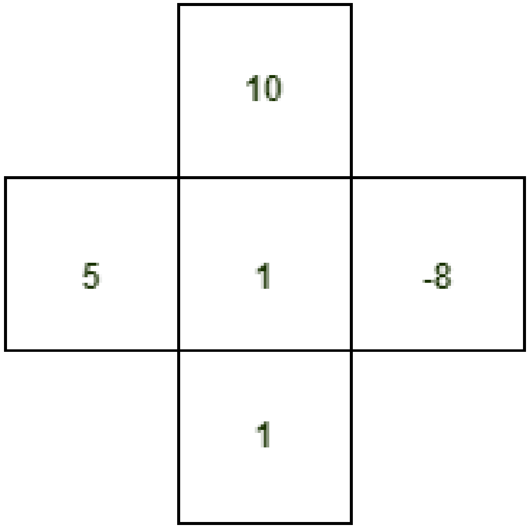
- Bellman equation gives us recursive decomposition (the first property).
- Bellman equation tells us how to break down the optimal value function into two pieces,
  - the optimal behaviour for one step followed by
  - the optimal behaviour after that step.
- We can do prediction, i.e., evaluate the given policy to get the value function on that policy (Dynamic Programming).
- ***Evaluating a random policy***
- ***Policy Update***

MDP solving - Value Iteration Example

- In this Grid World, we get a reward of as shown in the figure for each transition we make (actions we take). And the actions that we can take are **north, south, east and west** .
- Our agent is following some random policy π **with a weight of 0.5, 0.3, 0.1 and 0.1 for moving to north, south, east and west directions, respectively**.

We need to calculate the policy for four actions, **up, left, right and down with the center node with a reward of 1**

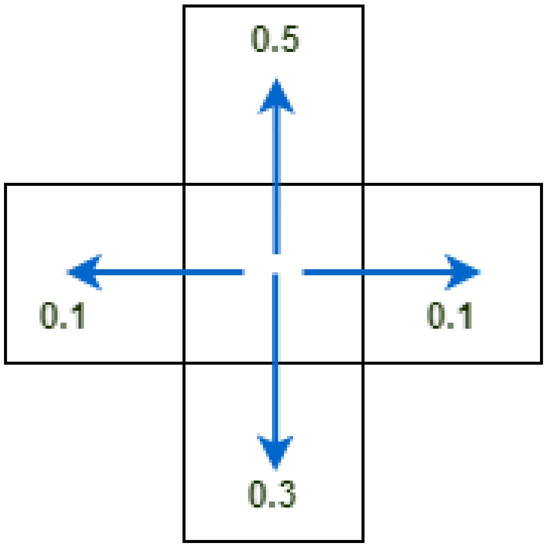


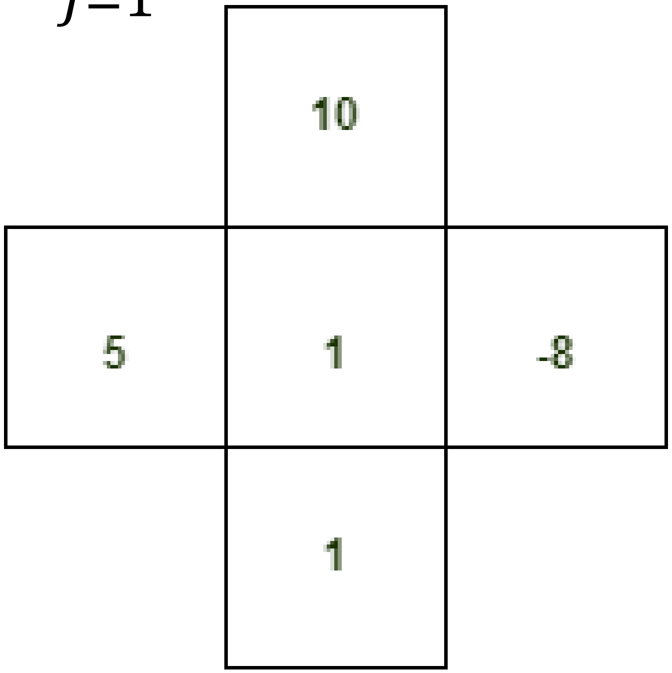


state space and reword

MDP solving – Value Iteration Example

$$\pi_{k+1}(s_i) = r_i + \arg max_a\{\gamma \sum_{j=1}^{N} P_{ij}^{a} V^{\pi_n}(s_j)\}$$

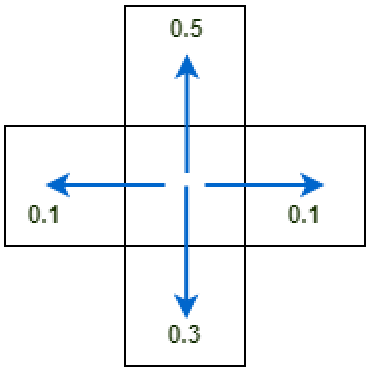




state space and reword

MDP solving – Value Iteration Example

$$\pi_{k+1}(s_i) = r_i + \arg max_a\{\gamma \sum_{j=1}^{N} P_{ij}^{a} V^{\pi_n}(s_j)\}$$

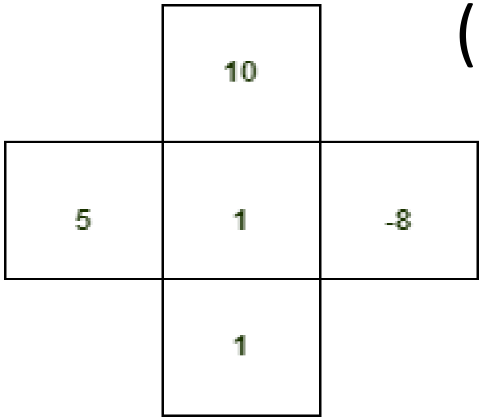


(left←) 0.5*5+0.1*10+0.1*1+0.3*-8 = 1.2

(up ↑) 0.5*10+0.1*5+0.1*-8+0.3*1 = 5.0

(right⟶) 0.5*-8+0.1*10+0.1*1+0.3*5 = -1.4

(Down ↓) 0.5*1+0.1*5+0.1*-8+0.3*10 = 3.2



state space and reword

$$\pi_{1=1+\max\{1.2,5.0,-1.4,3.2\}=6.0} \quad \uparrow$$

Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

Planning and Approximate Reasoning

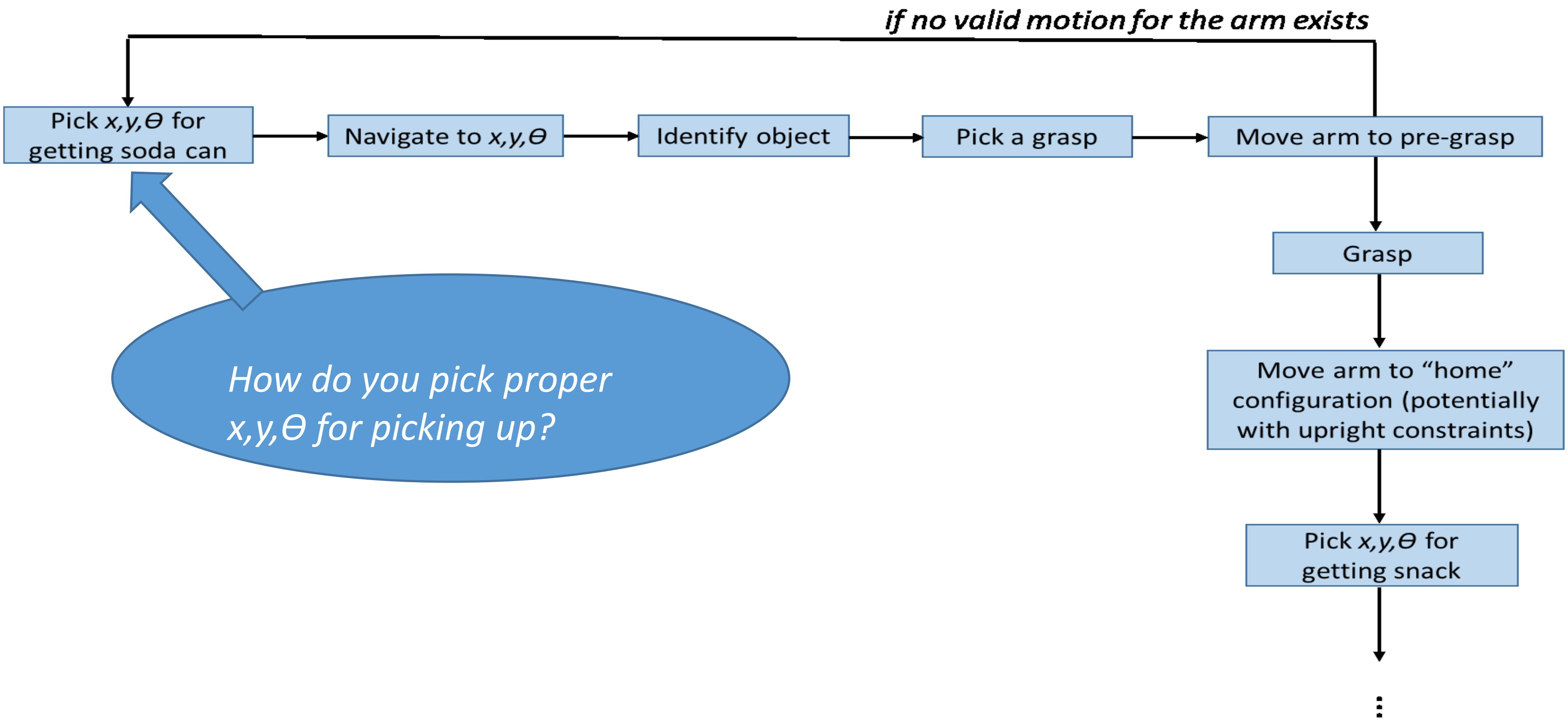
Hatem A. Rashwan

Application: Planning for Mobile Robot Manipulation

MESIIA – MIA

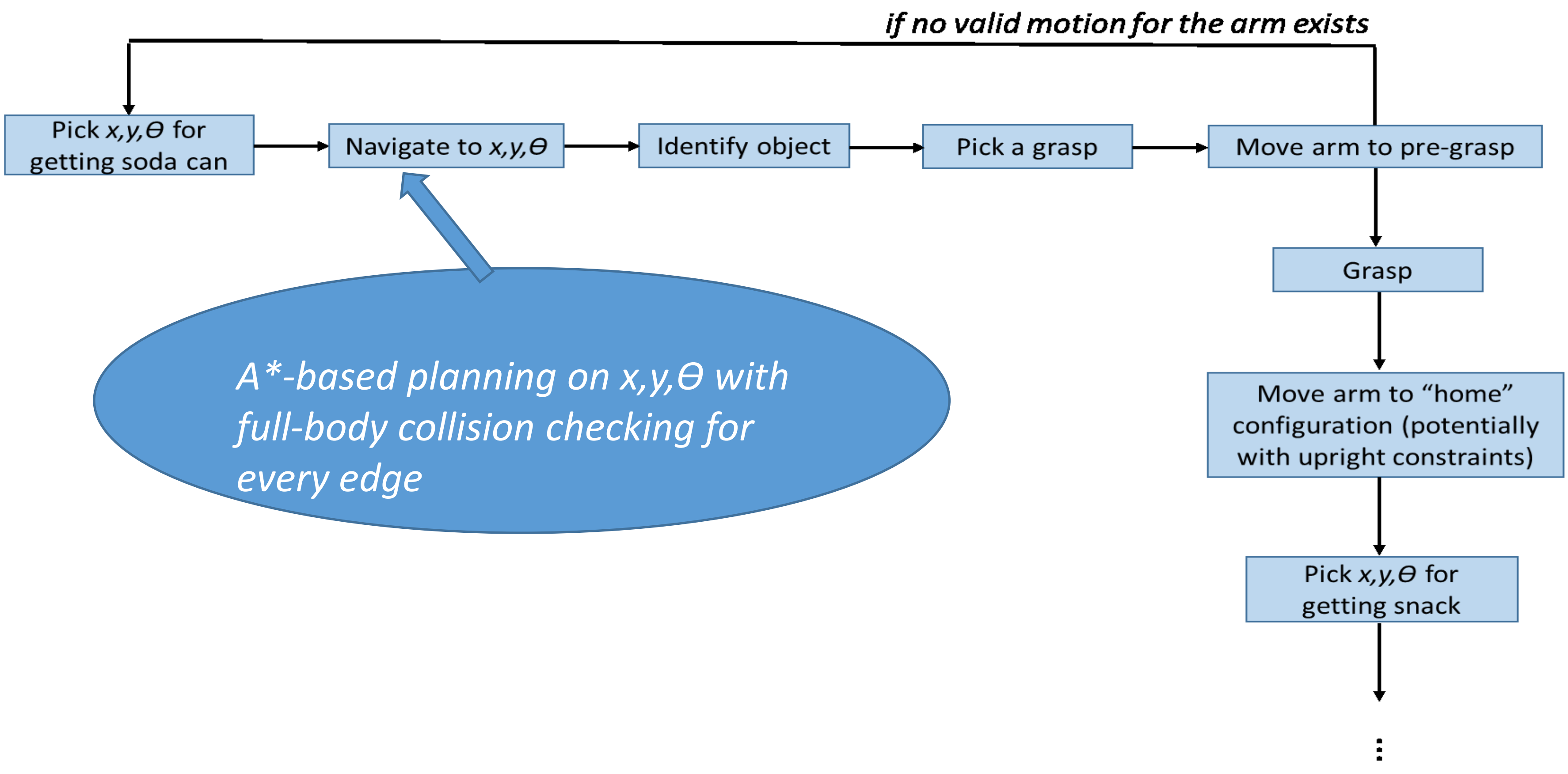
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



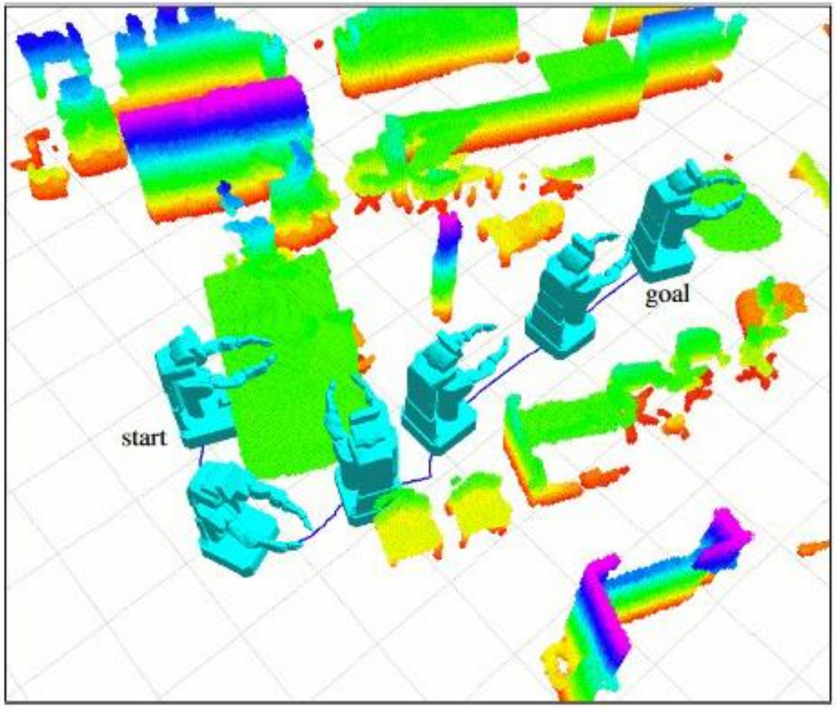
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



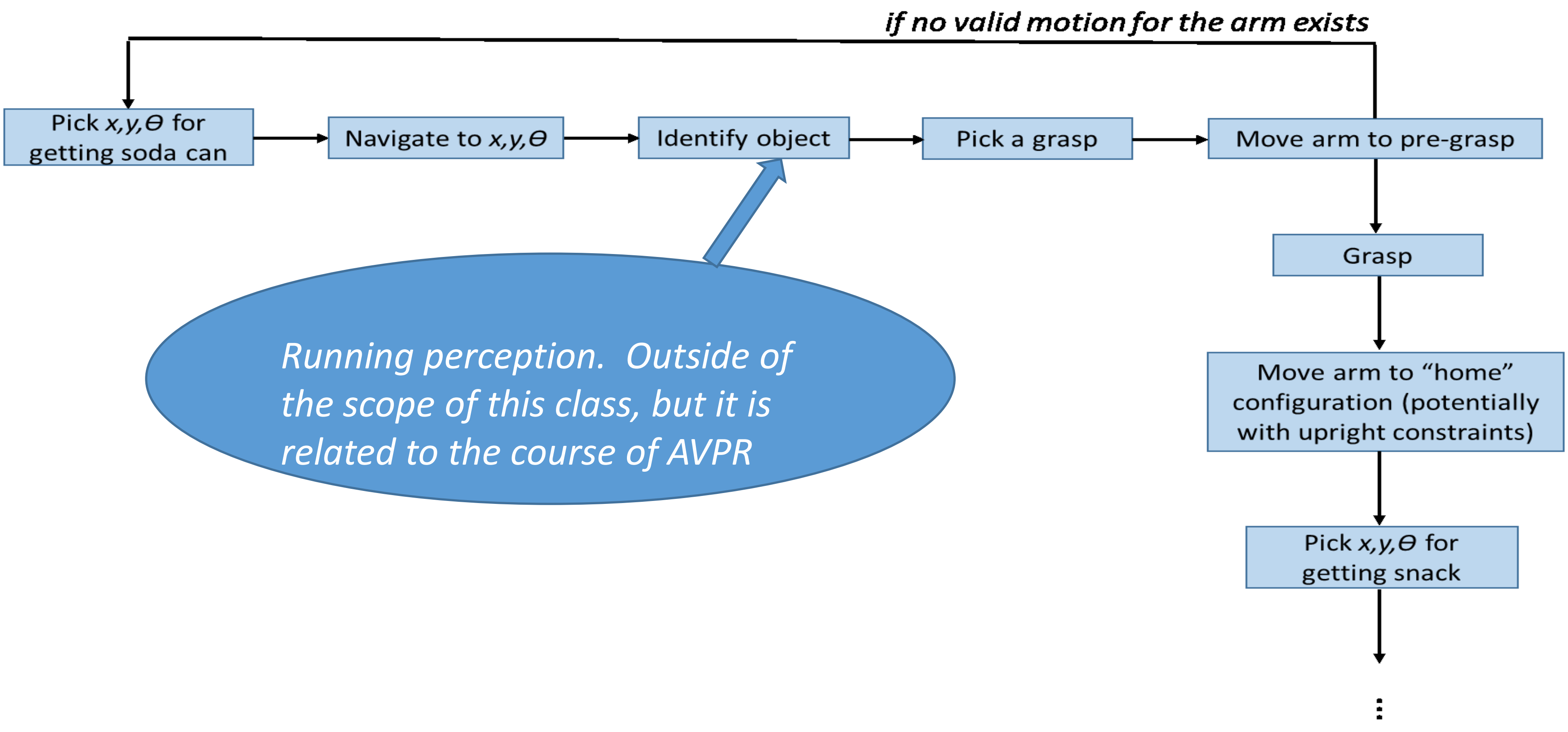
Graph for Navigation with Complex 3D Body [Hornung et al., '12]

- 3D (x,y,θ) lattice-based graph representation for full-body collision checking
  –takes set of motion primitives as input
  –takes *N*footprints of the robot defined as polygons as input
  –each footprint corresponds to the projection of a part of the body onto x,y plane
  –collision checking/cost computation is done for each footprint at the corresponding projection of the 3D map



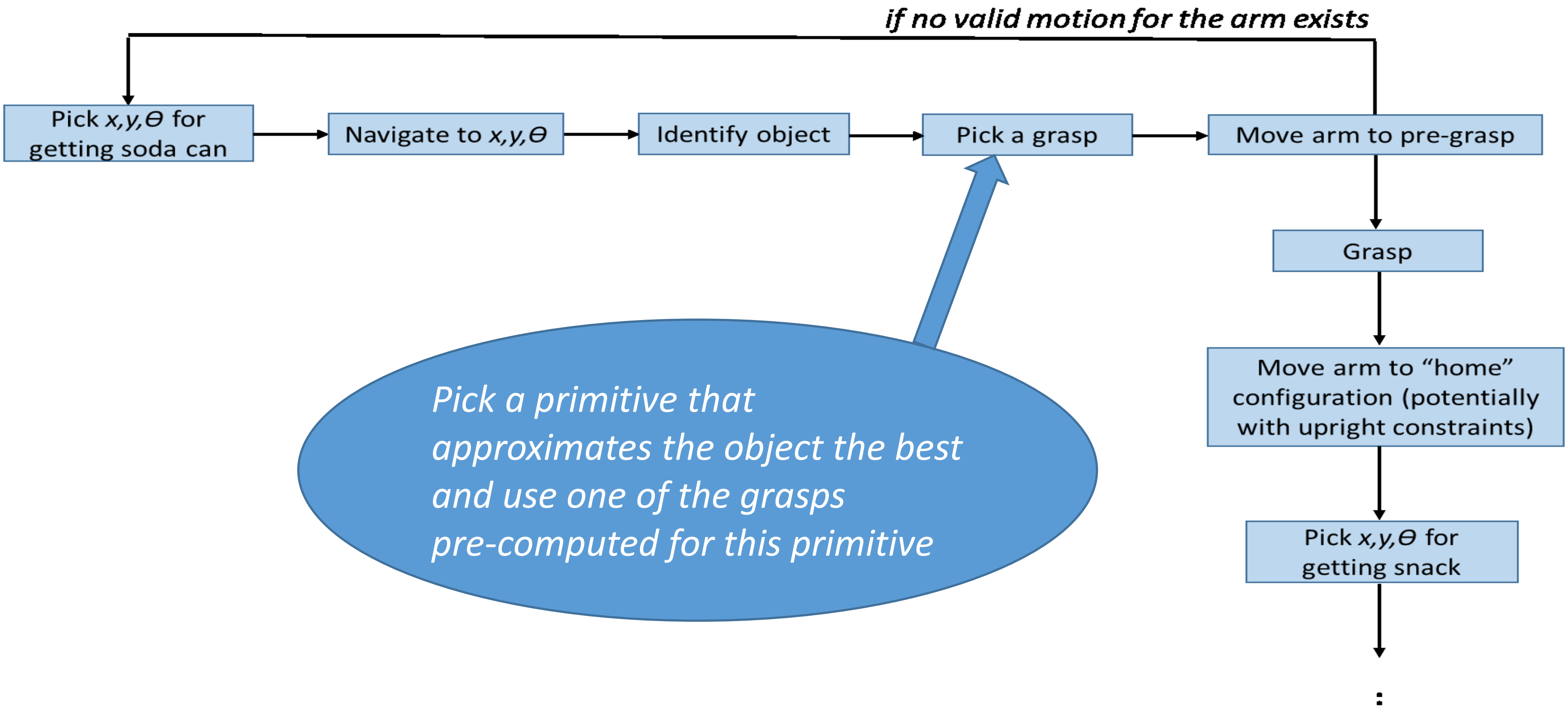
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



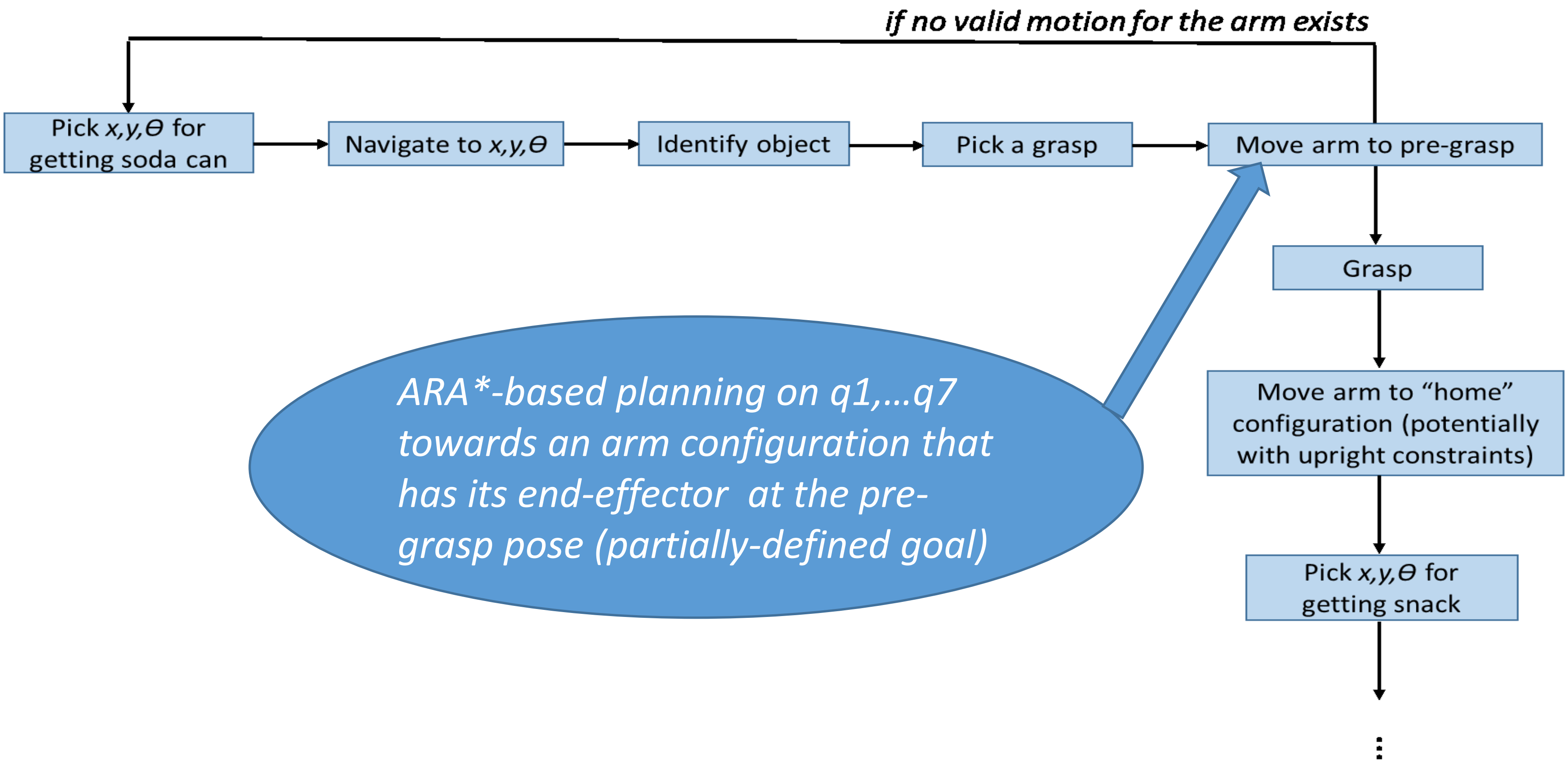
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



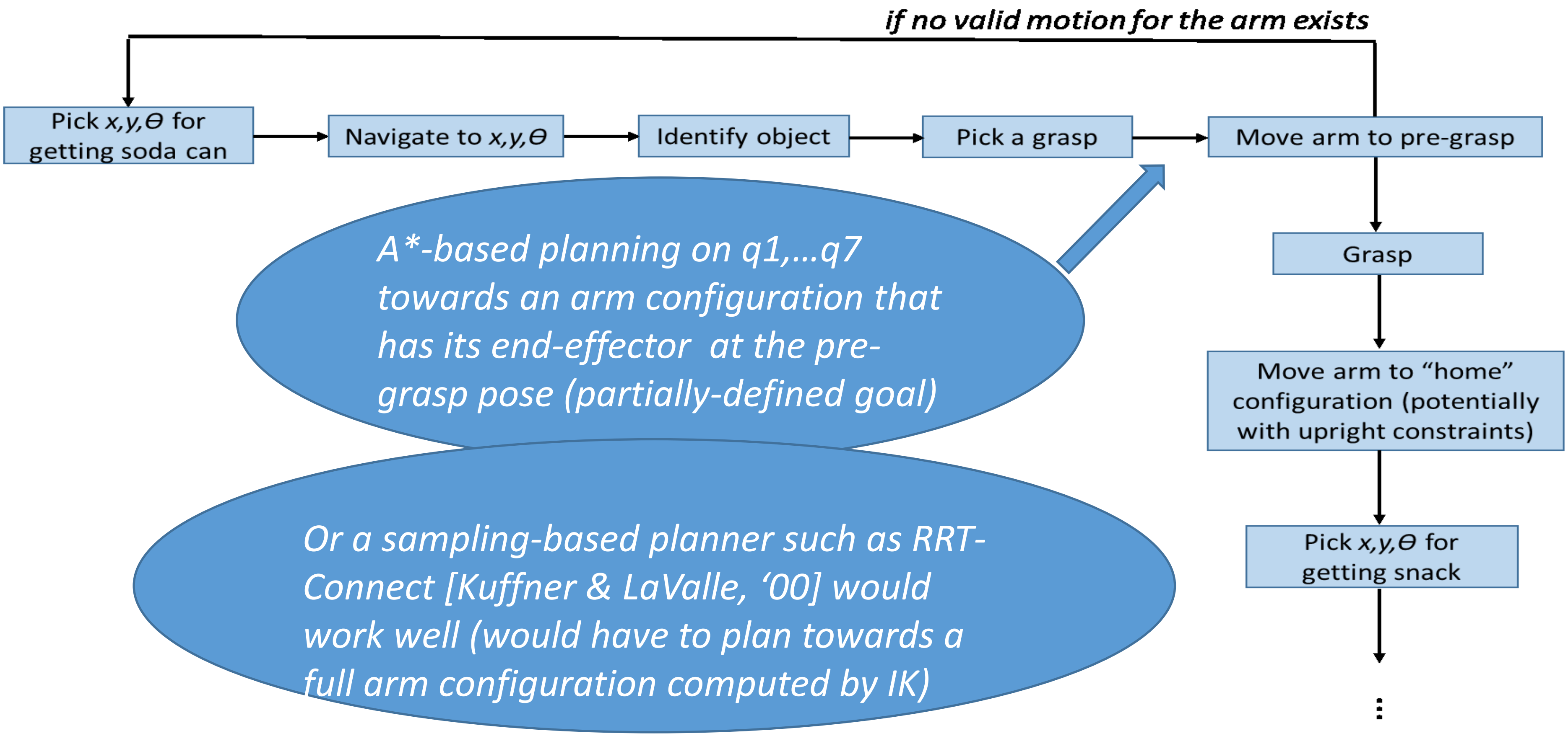
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



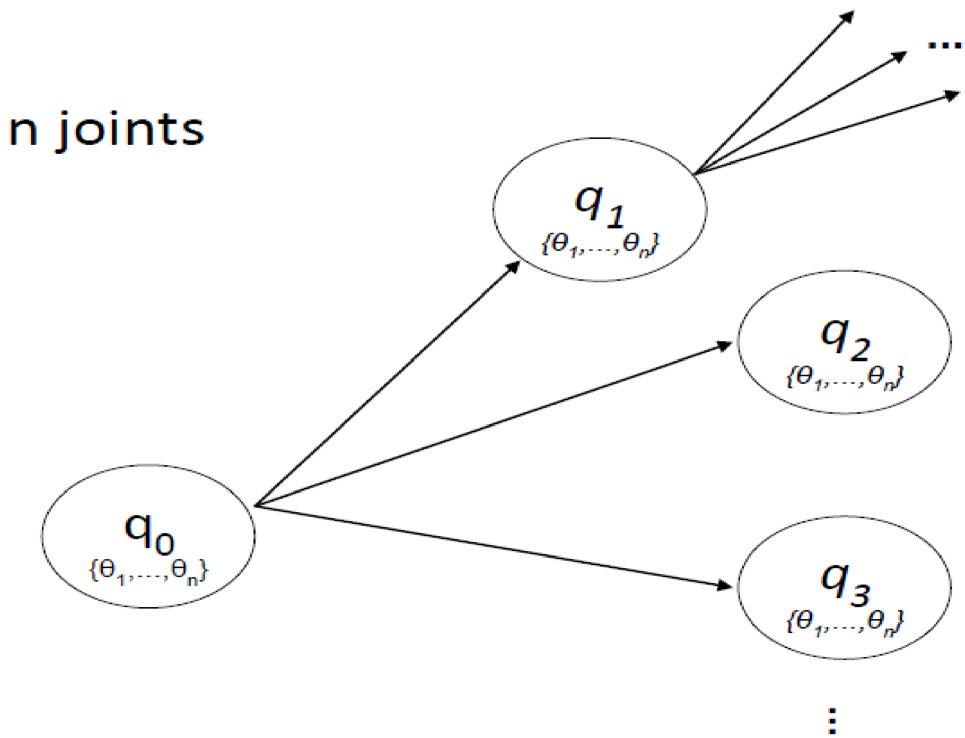
Robotic Bartender Demo ([Phillips et al.])

- Robot takes in a command from User Interface as to what soda can and snack to deliver



Manipulation Lattice for Arm Planning [Cohen et al., '13]

- Representation
  - ex. Single arm with n joints {θ1,...,θn}

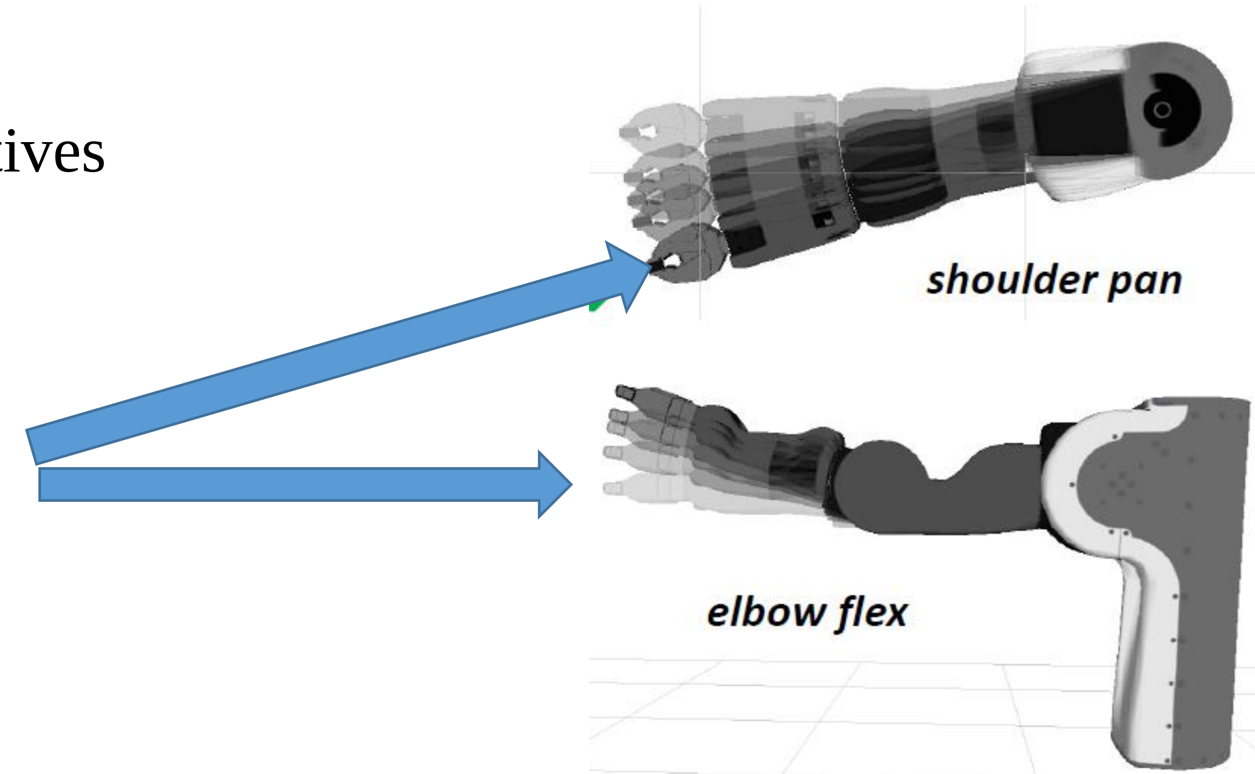


Manipulation Lattice for Arm Planning [Cohen et al., '13]

- Representation
  - –ex. Single arm with n joints {θ1,...,θn}

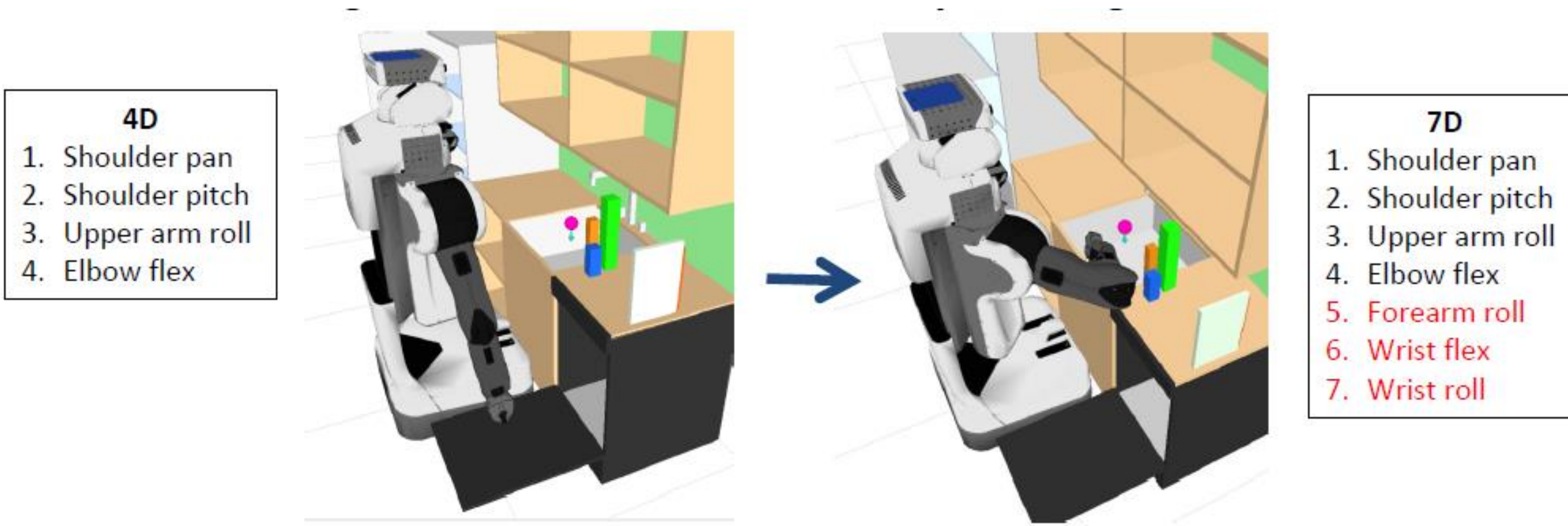
- Motion Primitives

Static



Manipulation Lattice for Arm Planning [Cohen et al., ‘13]

- Non-uniform Dimensionality
  - –far from goal: only 4 DoFs active
  - –around goal: all 7 DoFs are active
- Non-uniform Resolution
  - –far from goal: larger discretization of joint angles
  - –around goal: finer discretization of joint angles



Summary

- Multiple planners used for both domains
- Start and goal configurations are often most constrained
  –can be exploited by the planners
- Planning is higher-dimensional but can take longer than on ground and aerial vehicles
- Design of proper heuristics is a key to efficiency

End