Symbolic Dynamic Programming

Alan Fern *

* Based in part on slides by Craig Boutilier

Planning in Large State Space MDPs

- You have learned algorithms for computing optimal policies
  - Value Iteration
  - Policy Iteration
- These algorithms explicitly enumerate the state space
  - Often this is impractical
- Simulation-based planning and RL allowed for approximate planning in large MDPs
  - Did not utilize an explicit model of the MDP. Only used a strong or weak simulator.
- How can we get exact solutions to enormous MDPs?

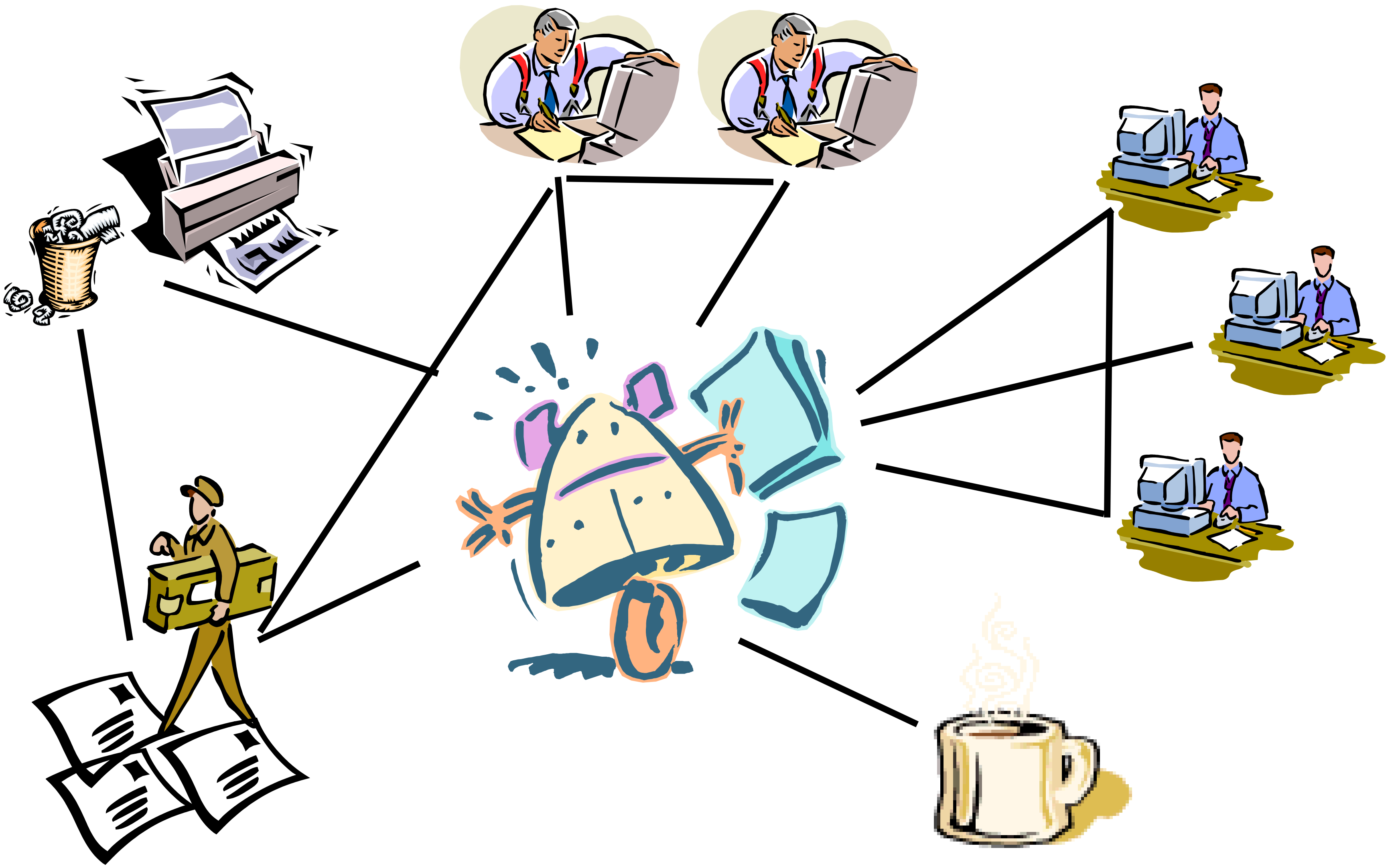
Structured Representations

- Policy iteration and value iteration treat states as atomic entities with no internal structure.

- In most cases, states actually do have internal structure
  - E.g. described by a set of state variables, or objects with properties and relationships
  - Humans exploit this structure to plan effectively

- What if we had a compact, structured representation for a large MDP and could efficiently plan with it?
  - Would allow for exact solutions to very large MDPs

A Planning Problem



Logical or Feature-based Problems

- For most AI problems, states are not viewed as atomic entities.
  - They contain structure. For example, they are described by a set of boolean propositions/variables

$$S = X_1 \times X_2 \times \cdots X_n$$

  - $|S|$ exponential in number of propositions

- Basic policy and value iteration do nothing to exploit the structure of the MDP when it is available

Solution?

- Require structured representations in terms of propositions
  - compactly represent transition function
  - compactly represent reward function
  - compactly represent value functions and policies
- Require structured computation
  - perform steps of PI or VI directly on structured representations
  - can avoid the need to enumerate state space
- We start by representing the transition structure as dynamic Bayesian networks

Propositional Representations

- States decomposable into ***state variables (we will assume boolean variables)***
- $$S = X_1 \times X_2 \times \cdots X_n$$
- *Structured* representations the norm in AI
  - Decision diagrams, Bayesian networks, etc.
  - Describe *how actions affect/depend on features*
  - Natural, concise, can be exploited computationally
- Same ideas can be used for MDPs

Robot Domain as Propositional MDP

- Propositional variables for single user version
  - Loc (robot's locat'n): Office, Entrance
  - T (lab is tidy): boolean
  - CR (coffee request outstanding): boolean
  - RHC (robot holding coffee): boolean
  - RHM (robot holding mail): boolean
  - M (mail waiting for pickup): boolean
- Actions/Events
  - move to an adjacent location, pickup mail, get coffee, deliver mail, deliver coffee, tidy lab
  - mail arrival, coffee request issued, lab gets messy
- Rewards
  - rewarded for tidy lab, satisfying a coffee request, delivering mail
  - (or penalized for their negation)

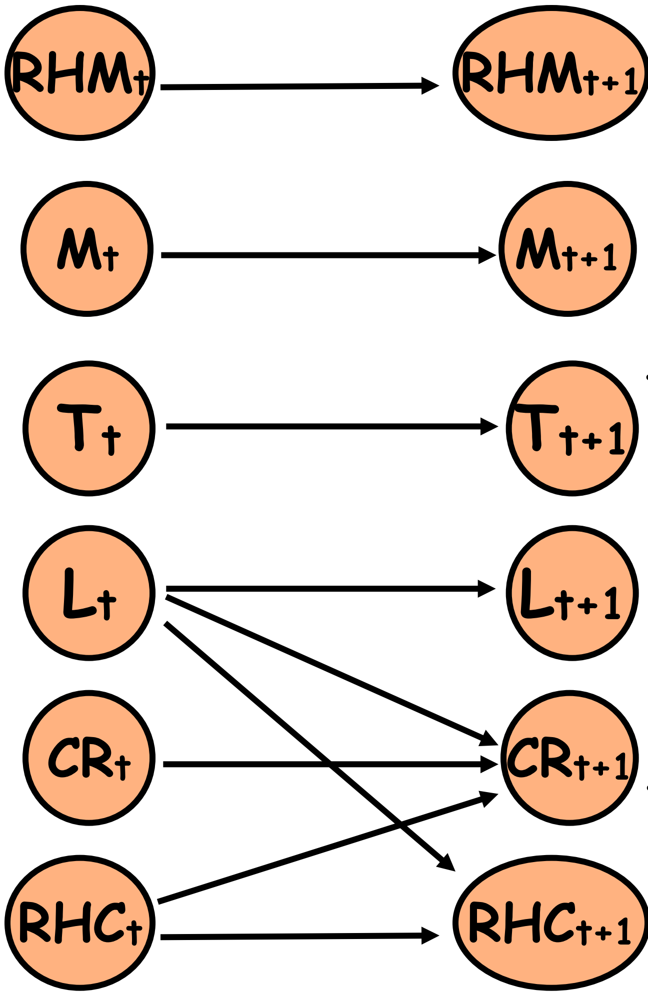
State Space

- State of MDP: assignment to these six variables
  - 64 states
  - grows exponentially with number of variables
- Transition matrices
  - 4032 parameters required per matrix
  - one matrix per action (6 or 7 or more actions)
- Reward function
  - 64 reward values needed
- Factored state and action descriptions will break this exponential dependence (generally)

Dynamic Bayesian Networks (DBNs)

- Bayesian networks (BNs) a common representation for probability distributions
  - A graph (DAG) represents conditional independence
  - Conditional probability tables (CPTs) quantify local probability distributions
- Dynamic Bayes net action representation
  - one Bayes net for each action a, representing the set of conditional distributions $Pr(S^{t+1}|A^t,S^t)$
  - each state variable occurs at time t and t+1
  - dependence of t+1 variables on t variables depicted by directed arcs

DBN Representation: deliver coffee



RHM	R(t+1)	$\overline{R(t+1)}$
T	1.0	0.0
F	0.0	1.0

$Pr(RHM_{t+1} | RHM_t)$

T	T(t+1)	$\overline{T(t+1)}$
T	0.91	0.09
F	0.0	1.0

$Pr(T_{t+1} | T_t)$

L	CR	RHC	CR(t+1)	$\overline{CR(t+1)}$
O	T	T	0.2	0.8
E	T	T	1.0	0.0
O	F	T	0.1	0.9
E	F	T	0.1	0.9
O	T	F	1.0	0.0
E	T	F	1.0	0.0
O	F	F	0.1	0.9
E	F	F	0.1	0.9

$Pr(CR_{t+1} | L_t, CR_t, RHC_t)$

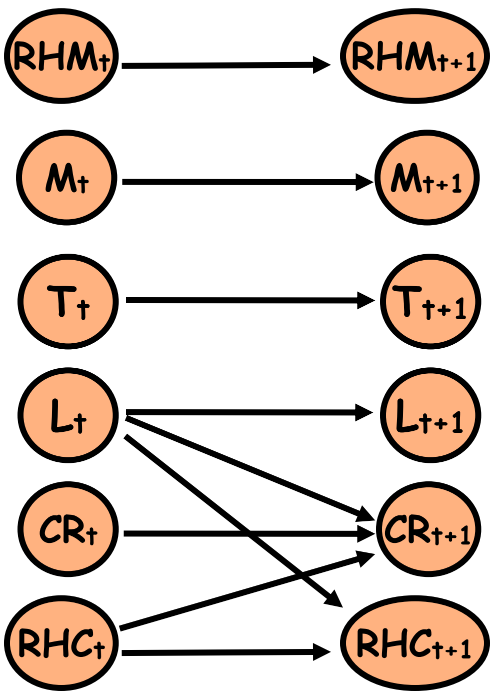
$\Pr(S_{t+1} | S_t, \text{deliver coffee})$ is the product of each of the 6 tables.

Benefits of DBN Representation

$$Pr(S_{t+1} \mid S_t) = Pr(RHM_{t+1}, M_{t+1}, T_{t+1}, L_{t+1}, C_{t+1}, RHC_{t+1} \mid RHM_t, M_t, T_t, L_t, C_t, RHC_t)$$

$$= Pr(RHM_{t+1} \mid RHM_t) * Pr(M_{t+1} \mid M_t) * Pr(T_{t+1} \mid T_t)$$

$$* Pr(L_{t+1} \mid L_t) * Pr(CR_{t+1} \mid CR_t, RHC_t, L_t) * Pr(RHC_{t+1} \mid RHC_t, L_t)$$



- Only 20 parameters vs. 4032 for matrix

Full Matrix

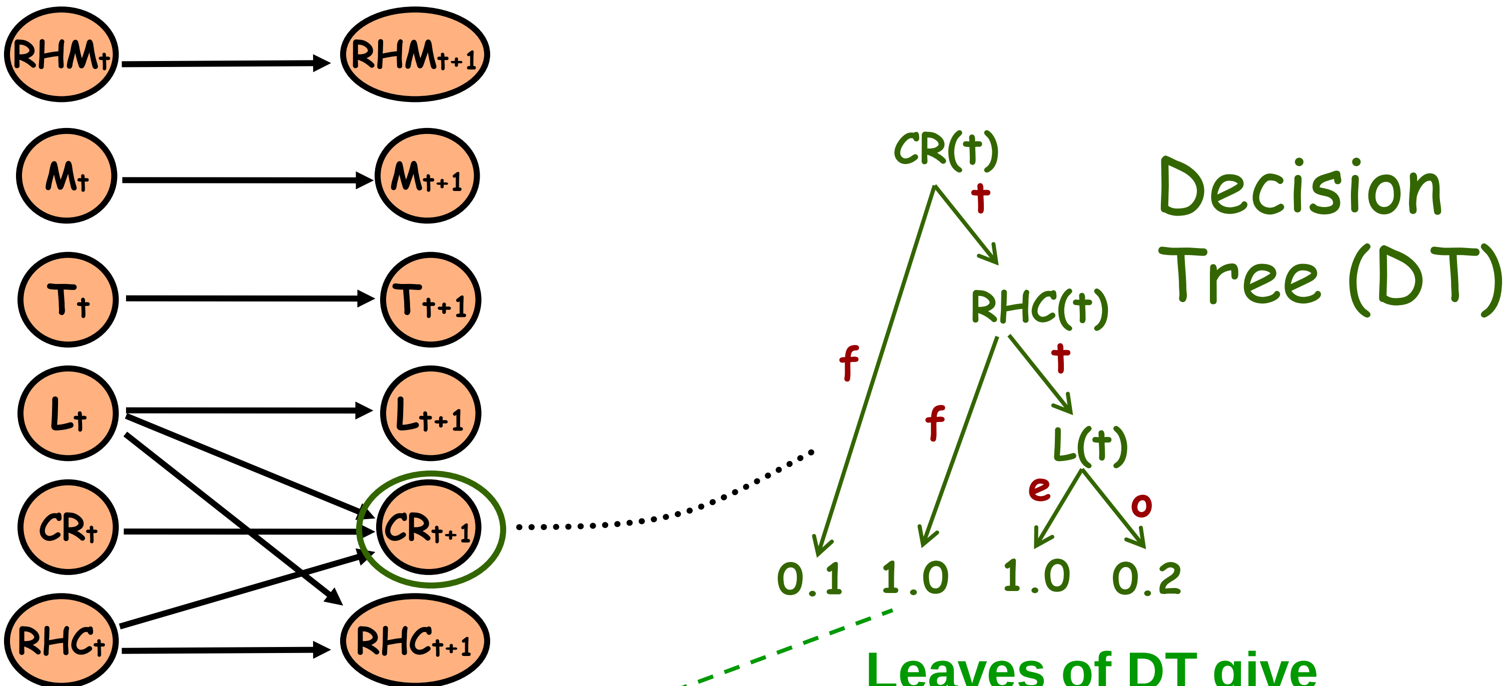
	s_1	s_2	...	s_{64}
s_1	0.9	0.05	...	0.0
s_2	0.0	0.20	...	0.1
⋮				
s_{64}	0.1	0.0	...	0.0

-Removes global exponential dependence

Structure in CPTs

- So far we have represented each CPT as a table of size exponential in the number of parents
- Notice that there's regularity in CPTs
  - e.g., $\mathbf{Pr(CR_{t+1} \mid L_t, CR_t, RHC_t)}$ has many similar entries
- Compact function representations for CPTs can be used to great effect
  - decision trees
  - algebraic decision diagrams (ADDs/BDDs)
- Here we show examples of decision trees (DTs)

Action Representation – DBN/DT



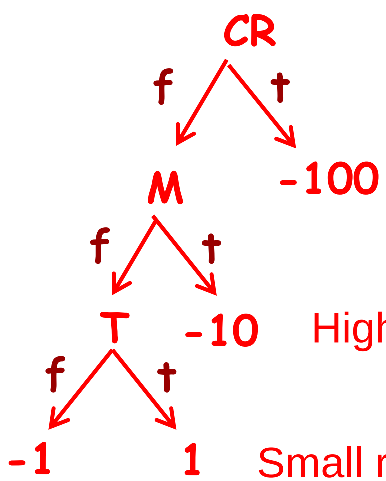
Leaves of DT give
$\Pr(CR_{t+1}=\text{true} \mid L_t, CR_t, RHC_t)$

e.g. If CR(t) = true & RHC(t) = false then
CR(t+1)=TRUE with prob. 1

DTs can often represent conditional probabilities much more compactly than a full conditional probability table

Reward Representation

- Rewards represented with DTs in a similar fashion
  - Would require vector of size 2^n for explicit representation



High cost for unsatisfied coffee request

High, but lower, cost for undelivered mail

Cost for lab being untidy

Small reward for satisfying all of these conditions

Structured Computation

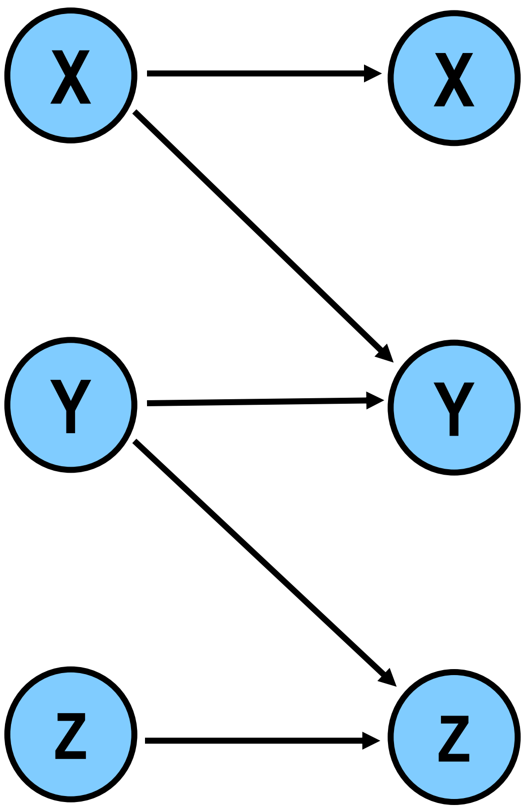
- Given our compact decision tree (DBN) representation, can we solve MDP without explicit state space enumeration?
- Can we avoid $O(|S|)$-computations by exploiting regularities made explicit by representation?
- We will study a general approach for doing this called structured dynamic programming

Structured Dynamic Programming

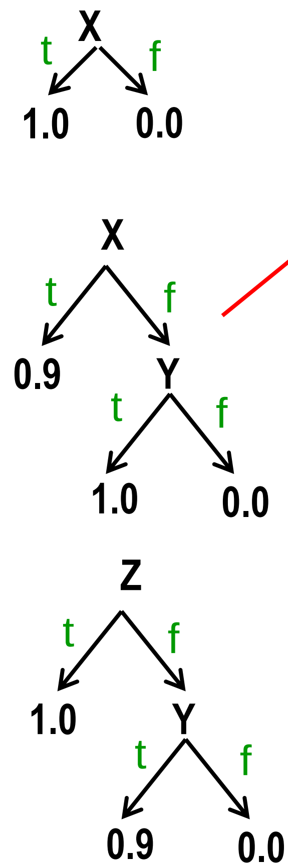
- We now consider how to perform dynamic programming techniques such as VI and PI using the problem structure
- VI and PI are based on a few basic operations.
  - Here we will show how to perform these operations directly on tree representations of value functions, policies, and transitions functions
- The approach is very general and can be applied to other representations (e.g. algebraic decision diagrams, situation calculus) and other problems after the main idea is understood
- We will focus on VI here, but the paper also describes a version of modified policy iteration

Recall Tree-Based Representations

Note: we are leaving off time subscripts for readability and using X(t), Y(t), …, instead.

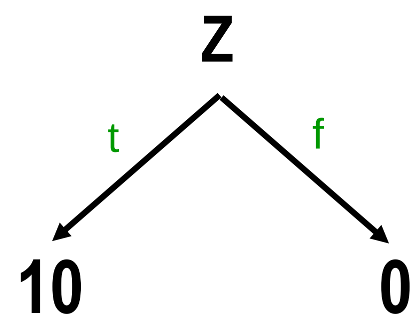


DBN for Action A



e.g. If X(t)=false & Y(t) = true then
Y(t+1)=true w/ prob 1

e.g. If X(t)=true THEN
Y(t+1)=true w/ prob 0.9



Reward Function R

Recall that each action of the MDP has its own DBN.

Structured Dynamic Programming

- Value functions and policies can also have tree representations
  - Often much more compact representations than tables

- **Our Goal:** compute the tree representations of policy and value function given the tree representations of the transitions and rewards

Recall Value Iteration

Value Iteration:

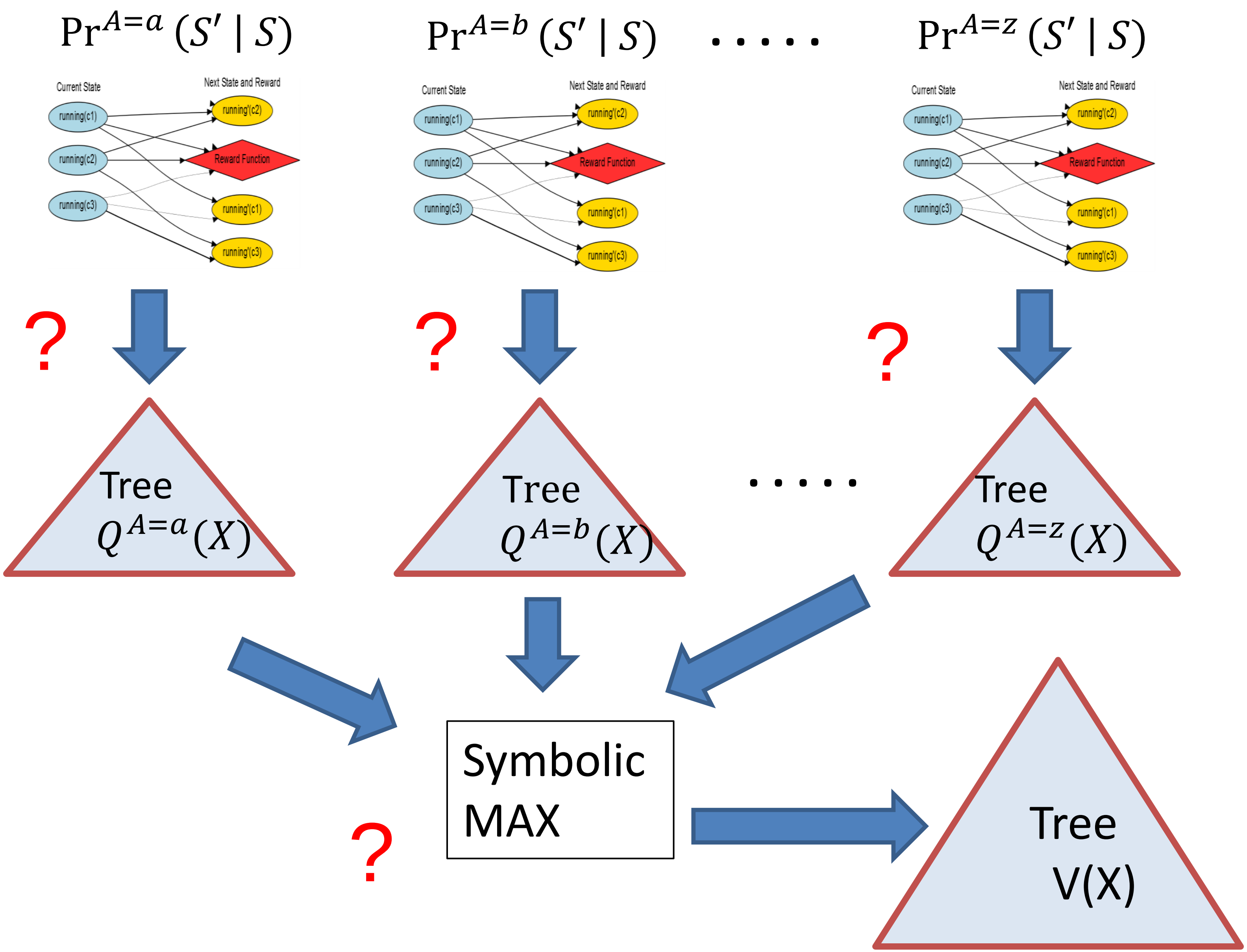
$$V_0(s) = R(s) \quad \text{;; could initialize to 0}$$

$$\left.\begin{aligned} Q_{k+1}^{a}(s) &= R(s) + \gamma \sum_{s'} \Pr(s'|s,a) V_k(s') \\ V_{k+1}(s) &= \max_a Q_{k+1}^{a}(s) \end{aligned}\right\} \text{Bellman Backup}$$

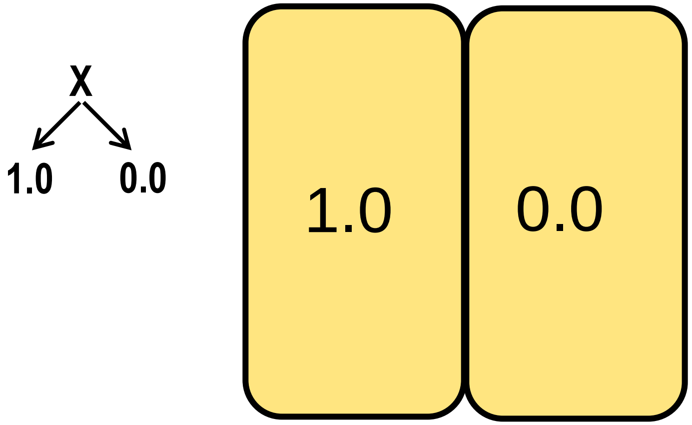
Suppose that initial V_k is compactly represented as a tree.

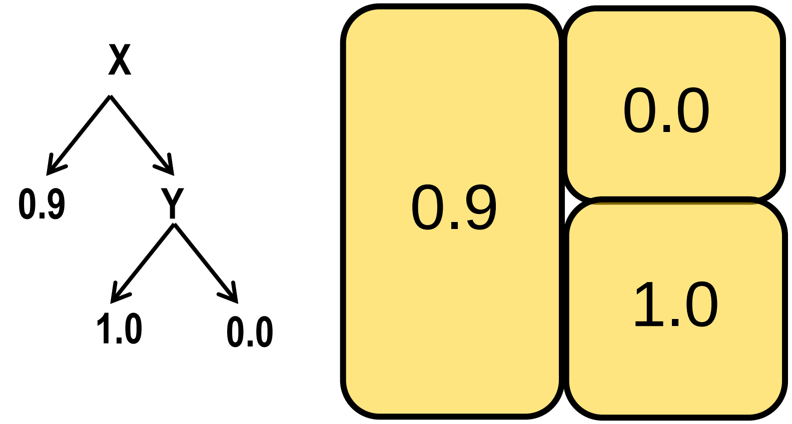
1. Show how to compute compact trees for $Q_{k+1}^{a_1}, \ldots, Q_{k+1}^{a_n}$
2. Use a max operation on the Q-trees (returns a single tree)

Symbolic Value Iteration



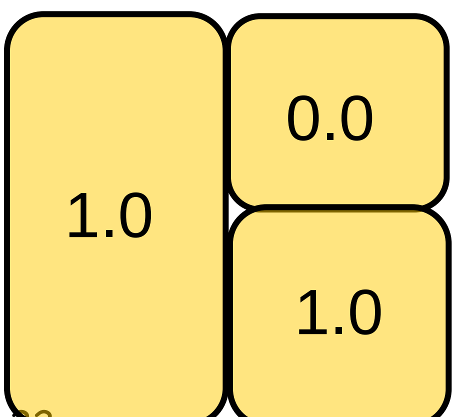
The MAX Trees Operation





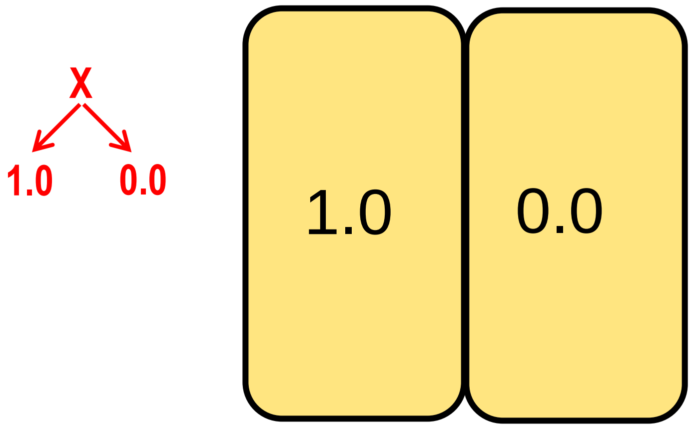
Tree partitions the state space, assigning values to each region

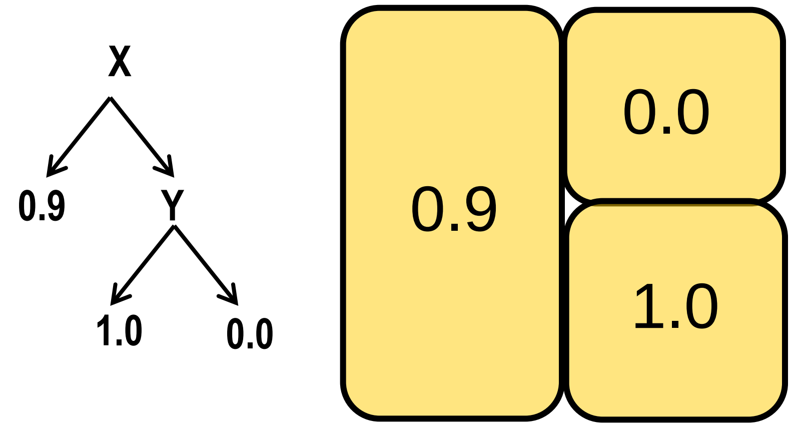
The state space max for the above trees is:



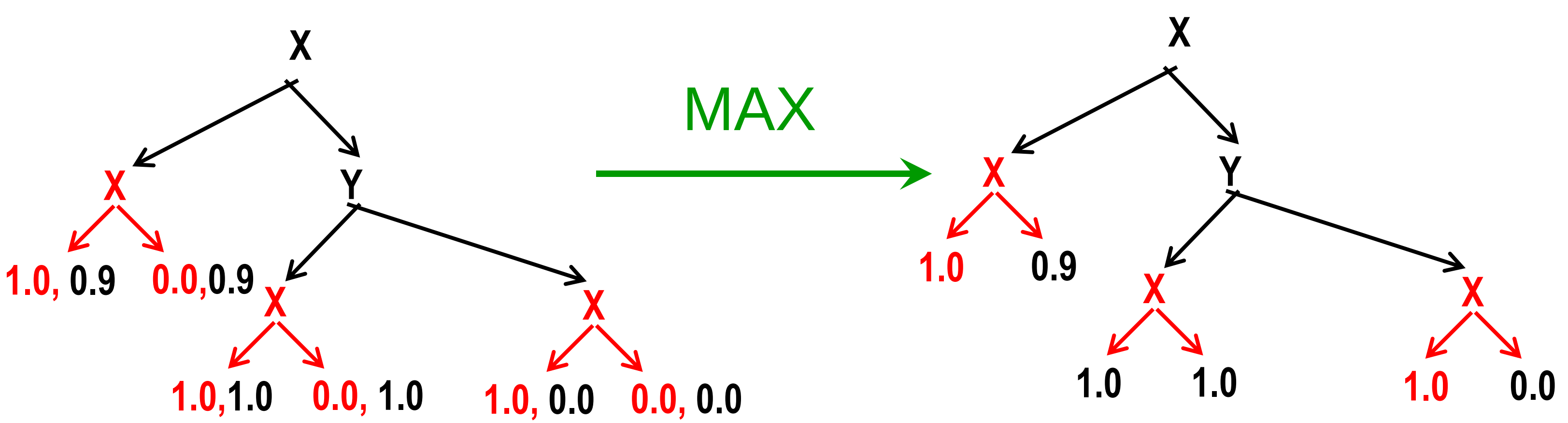
In general, how can we compute the tree representing the max?

The MAX Tree Operation

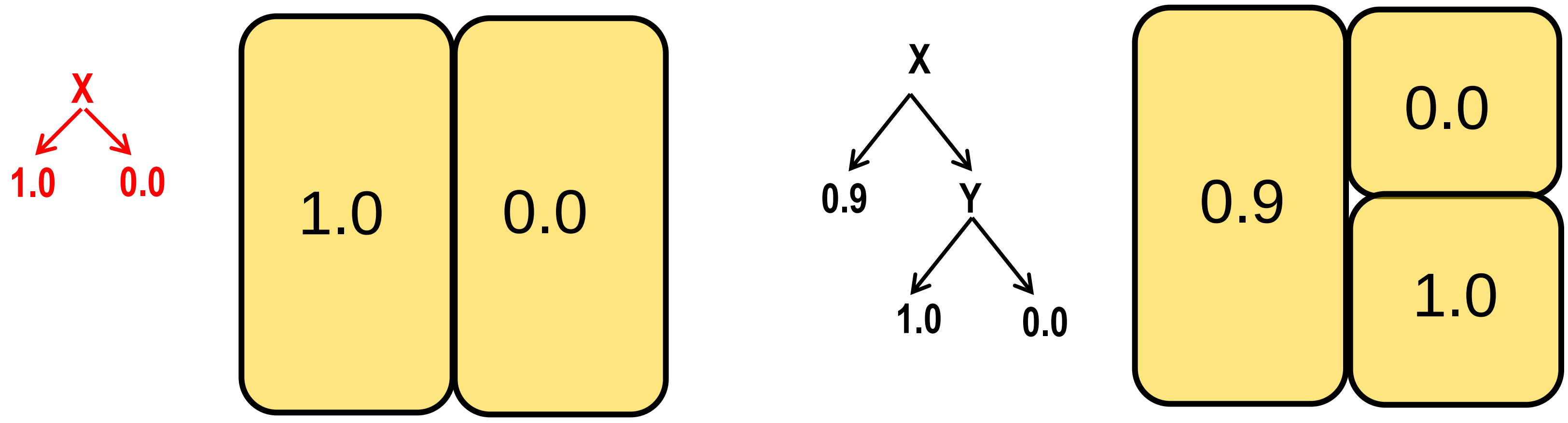




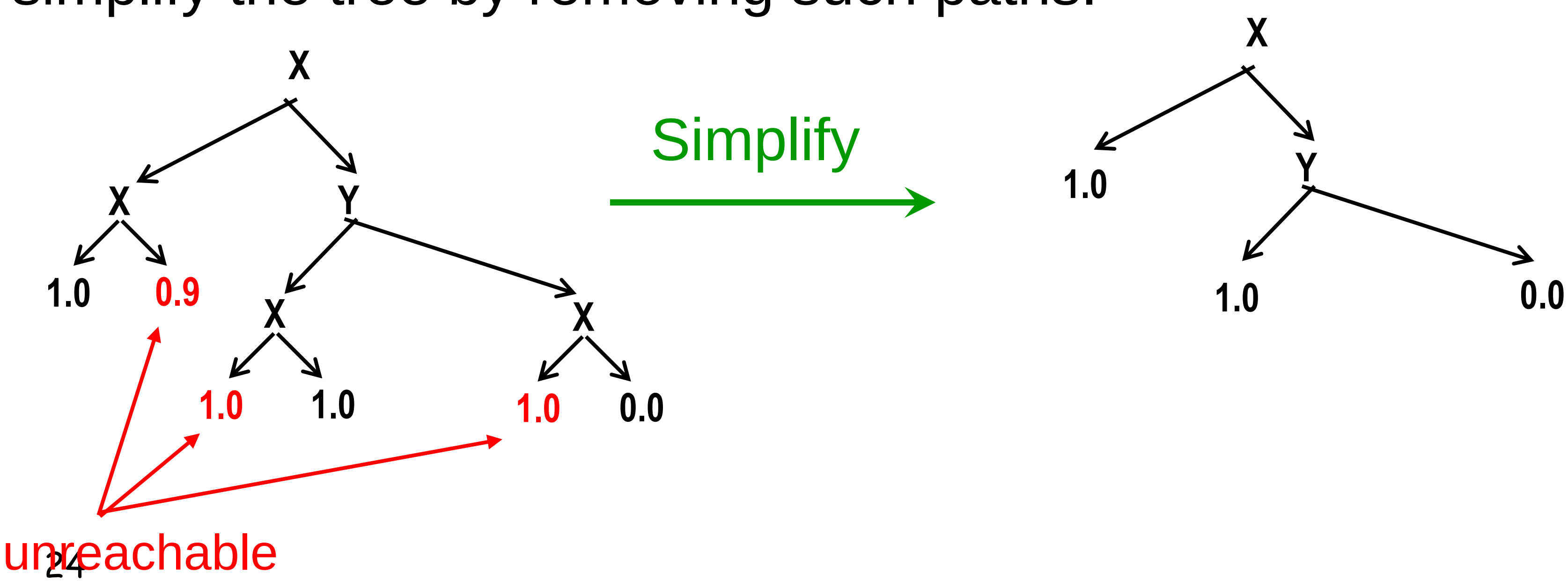
Can simply append one tree to leaves of other. Makes all the distinctions that either tree makes. Max operation is taken at leaves of result.



The MAX Tree Operation

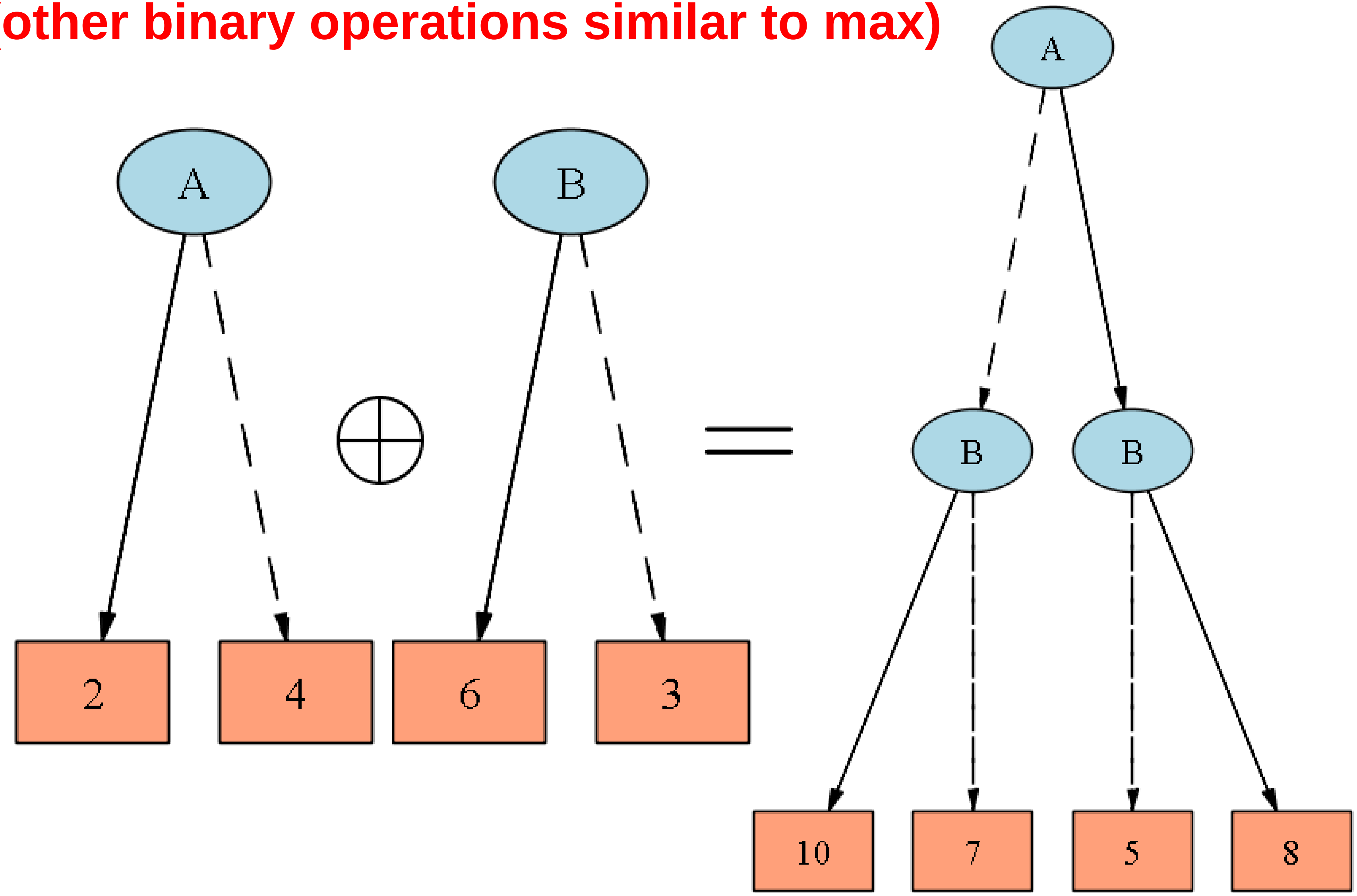


The resulting tree may have unreachable leaves. We can simplify the tree by removing such paths.



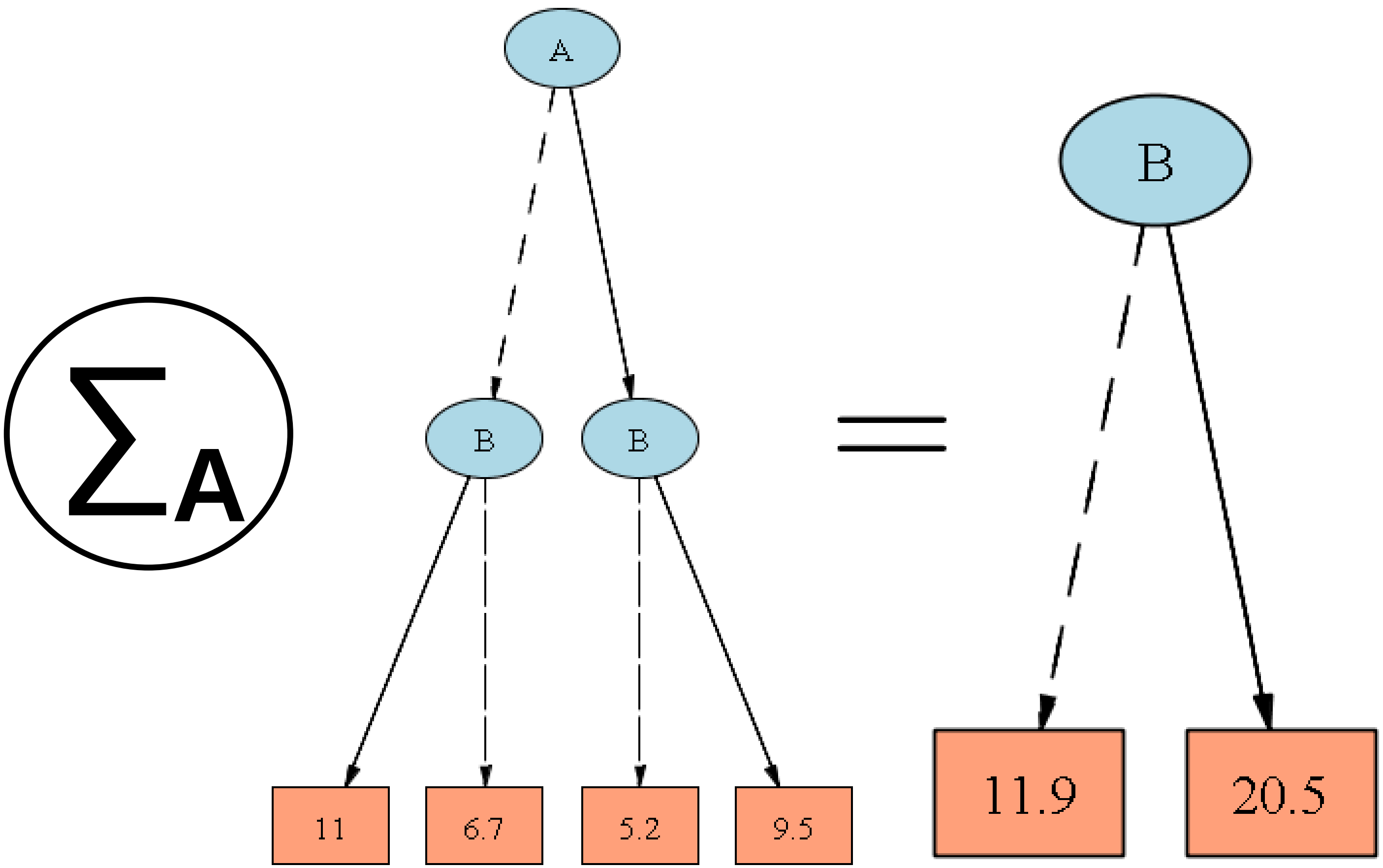
BINARY OPERATIONS

(other binary operations similar to max)



MARGINALIZATION

Compute diagram representing $G(B) = \sum_A F(A, B)$



There are libraries for doing this.

Symbolic Bellman Backup

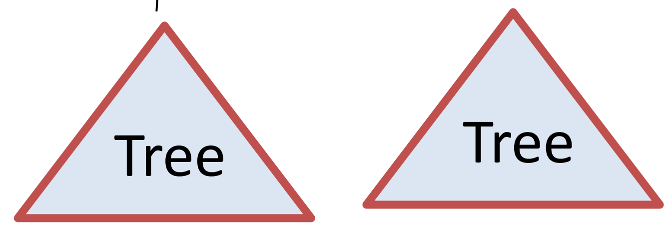
for each action a

$$Q^a_{n+1} = R^a \oplus \gamma \sum_{X_1'} Pr^a(X_1' \mid X) \cdots \sum_{X_l'} Pr^a(X_l' \mid X) \otimes (V_n')$$

$$V_{n+1} = \max\{V_{n+1}, Q^a_{n+1}\}.$$

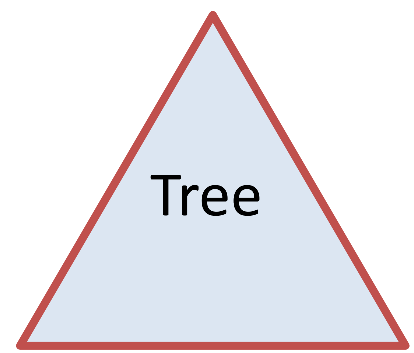
Symbol Q^a_{n+1}

$$Q^a_{n+1} = R^a \oplus \gamma \sum_{X_1'} Pr^a(X_1' \mid X) \cdots \sum_{X_l'} Pr^a(X_l' \mid X) \otimes (V_n')$$



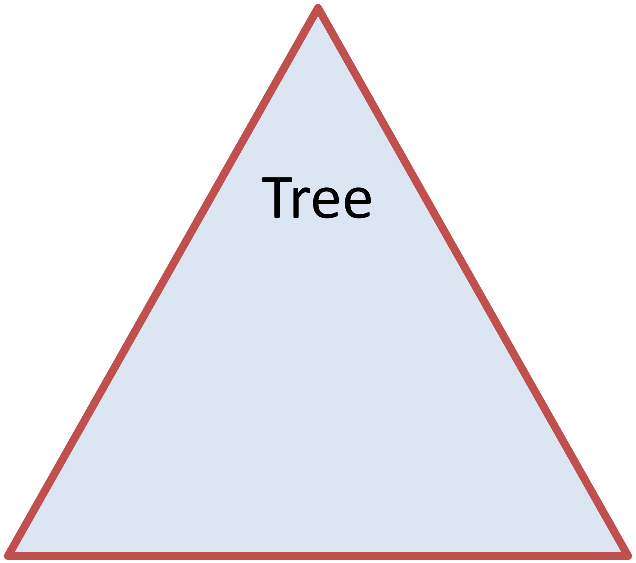
Symbol Q_{n+1}^{a}

$$Q_{n+1}^{a} = R^{a} \oplus \gamma \sum_{X_1'} Pr^{a}(X_1' \mid X) \cdots \sum_{X_l'} Pr^{a}(X_l' \mid X) \otimes (V_n')$$



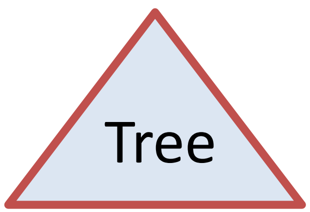
Symbol Q_{n+1}^{a}

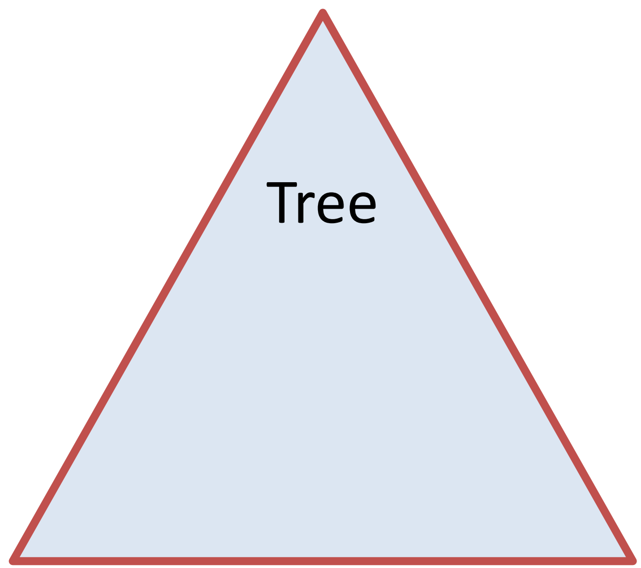
$$Q_{n+1}^{a} = R^{a} \oplus \gamma \sum_{X_1'} Pr^{a}(X_1' \mid X) \cdots \sum_{X_l'} Pr^{a}(X_l' \mid X) \otimes (V_n')$$



Symbol Q^a_{n+1}

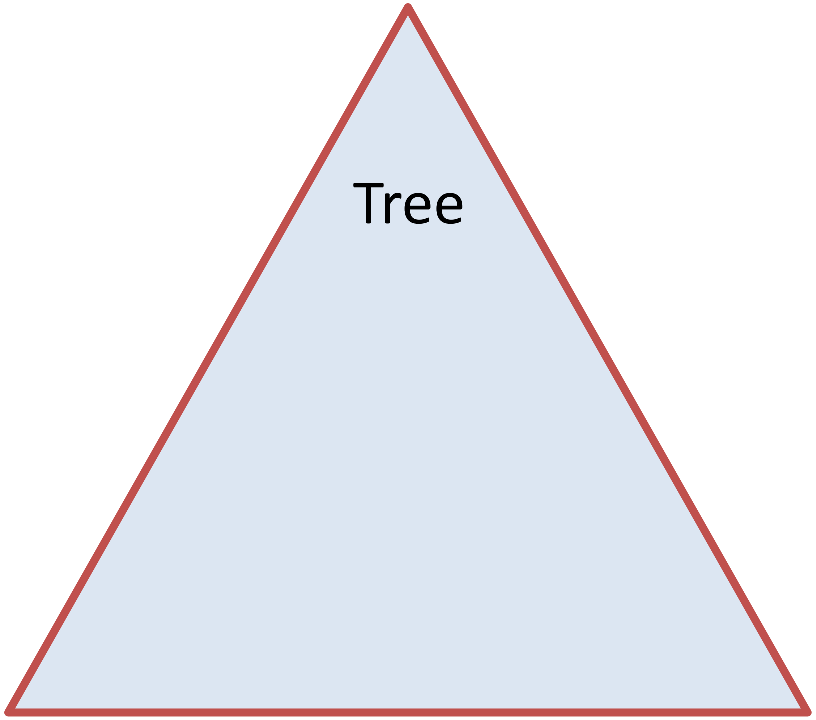
$$Q^a_{n+1} = R^a \oplus \gamma \sum_{X_1'} Pr^a(X_1' \mid X) \cdots \sum_{X_l'} Pr^a(X_l' \mid X) \otimes (V_n')$$





Symbol Q^a_{n+1}

$$Q^a_{n+1} = R^a \oplus \gamma \sum_{X_1'} Pr^a(X_1' \mid X) \cdots \sum_{X_l'} Pr^a(X_l' \mid X) \otimes (V_n')$$

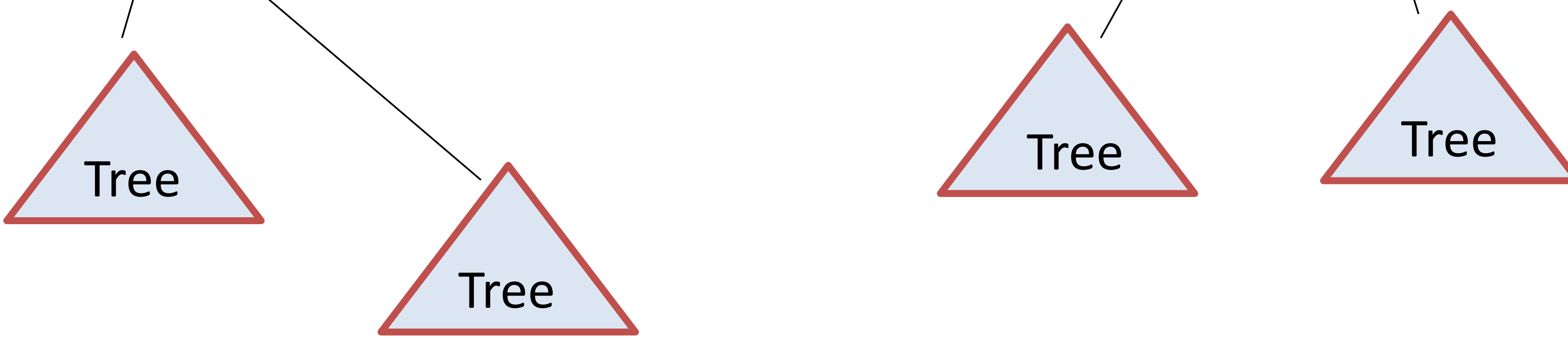


Symbolic Bellman Backup

for each action a

$$Q^a_{n+1} = R^a \oplus \gamma \sum_{X_1'} Pr^a(X_1' \mid X) \cdots \sum_{X_l'} Pr^a(X_l' \mid X) \otimes (V_n')$$

$$V_{n+1} = \max\{V_{n+1}, Q^a_{n+1}\}.$$



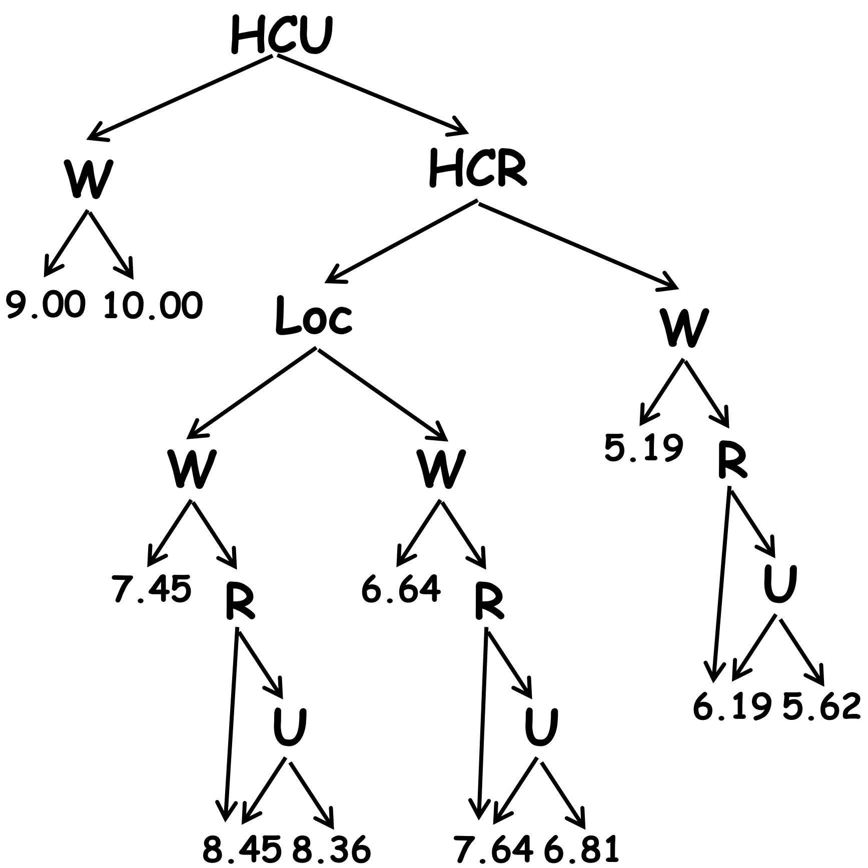
SDP: Relative Merits

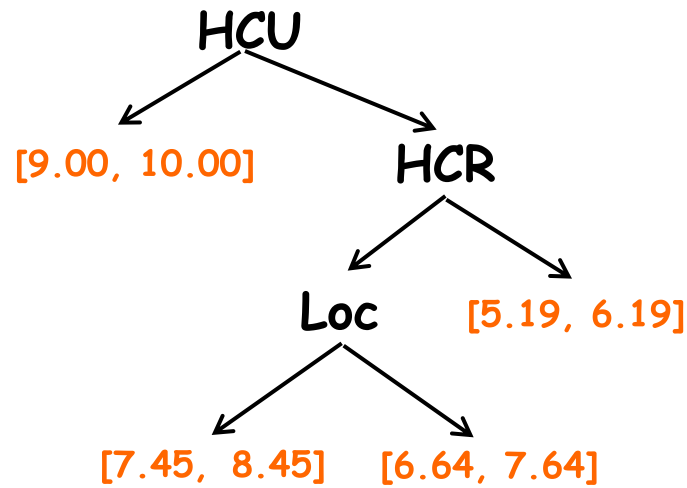
- Adaptive, nonuniform, exact abstraction method
  - provides exact solution to MDP
  - much more efficient on certain problems (time/space)
  - 400 million state problems in a couple hrs
- Can formulate a similar procedure for modified policy iteration
- Some drawbacks
  - produces piecewise constant VF
  - some problems admit no compact solution representation
    - so the sizes of trees blows up with enough iterations
  - approximation may be desirable or necessary

Approximate SDP

- Easy to approximate solution using SDP
- Simple ***pruning*** of value function
  - Simply “merge” leaves that have similar values
  - Can prune trees **[BouDearden96]** or ADDs **[StaubinHoeyBou00]**
- Gives regions of ***approximately same value***

A Pruned Value ADD





Approximate SDP: Relative Merits

- Relative merits of ASDP fewer regions implies faster computation
  - 30-40 billion state problems in a couple hours
  - allows fine-grained control of time vs. solution quality with dynamic error bounds
  - technical challenges: variable ordering, convergence, fixed vs. adaptive tolerance, etc.
- Some drawbacks
  - (still) produces piecewise constant VF
  - doesn’t exploit additive structure of VF at all
- **Bottom-line**: When a problem matches the structural assumptions of SDP then we can gain much. But many problems do not match assumptions.

Ongoing Work

- Factored action spaces
  - Sometimes the action space is large, but has structure.
  - For example, cooperative multi-agent systems
- Recent work (at OSU) has studied SDP for factored action spaces
  - Include action variables in the DBNs