Introduction to Decision Making

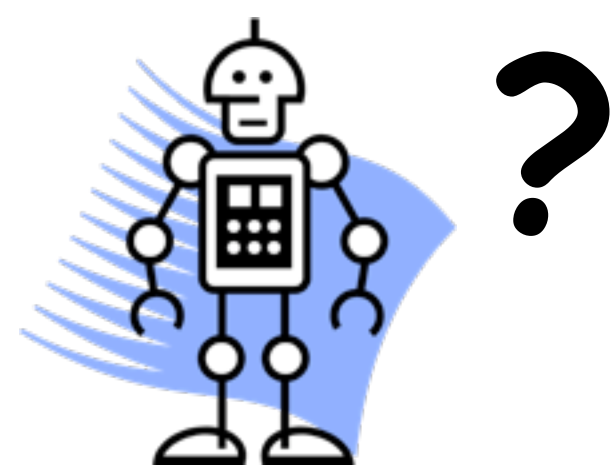
CS 486/686: Introduction to Artificial Intelligence
Fall 2013

Outline

- Utility Theory
- Decision Trees

Decision Making Under Uncertainty

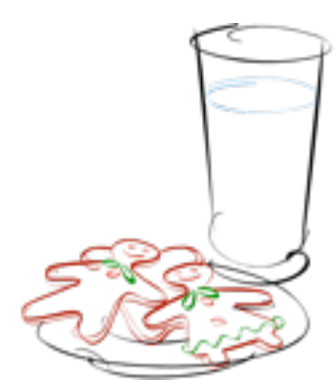
- I give a robot a planning problem: “ I want coffee”
  - But the coffee maker is broken: Robot reports “No plan!”



Decision Making Under Uncertainty

- I want more robust behavior
- I want my robot to know what to do when my primary goal is not satisfied
  - Provide it with some indication of my preferences over alternatives
    - e.g. coffee better than tea, tea better than water, water better than nothing,...





Decision Making Under Uncertainty

- But it is more complicated than that
  - It could wait 45 minutes for the coffee maker to be fixed
- What is better?
  - Tea now?
  - Coffee in 45 minutes?

Preferences

- A preference ordering $\succsim$ is a ranking over all possible states of the world s
- These could be outcomes of actions, truth assignments, states in a search problem, etc
  - s $\succsim$ t: state s is **at least as good as** state t
  - s $\succ$ t: state s is **strictly preferred to** state t
  - s $\sim$ t: agent is **ambivalent between states** s and t

Preferences

- If an agent's actions are deterministic, then we know what states will occur
- If an agent's actions are not deterministic, then we represent this by lotteries
  - Probability distribution over outcomes
  - Lottery $L=[p_1,s_1;p_2,s_2;\ldots;p_n,s_n]$
    - s_1 occurs with probability p_1, s_2 occurs with probability p_2, ...

Axioms

- Orderability: Given 2 states A and B
  - $(A \succsim B) \vee (B \succsim A) \vee (A \sim B)$
- Transitivity: Given 3 states A, B, C
  - $(A \succsim B) \wedge (B \succsim C) \rightarrow (A \succsim C)$
- Continuity:
  - $A \succsim B \succsim C \rightarrow$ Exists p, $[p,A;(1-p),C] \sim B$
- Substitutability
  - $A \sim B \rightarrow [p,A;1-p,C] \sim [p,B,1-p,C]$
- Monotonicity:
  - $(A \succsim B) \rightarrow (p \geq q \leftrightarrow [p,A;1-p,B] \succsim [q,A;1-q,B]$
- Decomposability
  - $[p,A;1-p[q,B;1-q,C]] \sim [p,A;\ (1-p)q,B;(1-p)(1-q),C]$

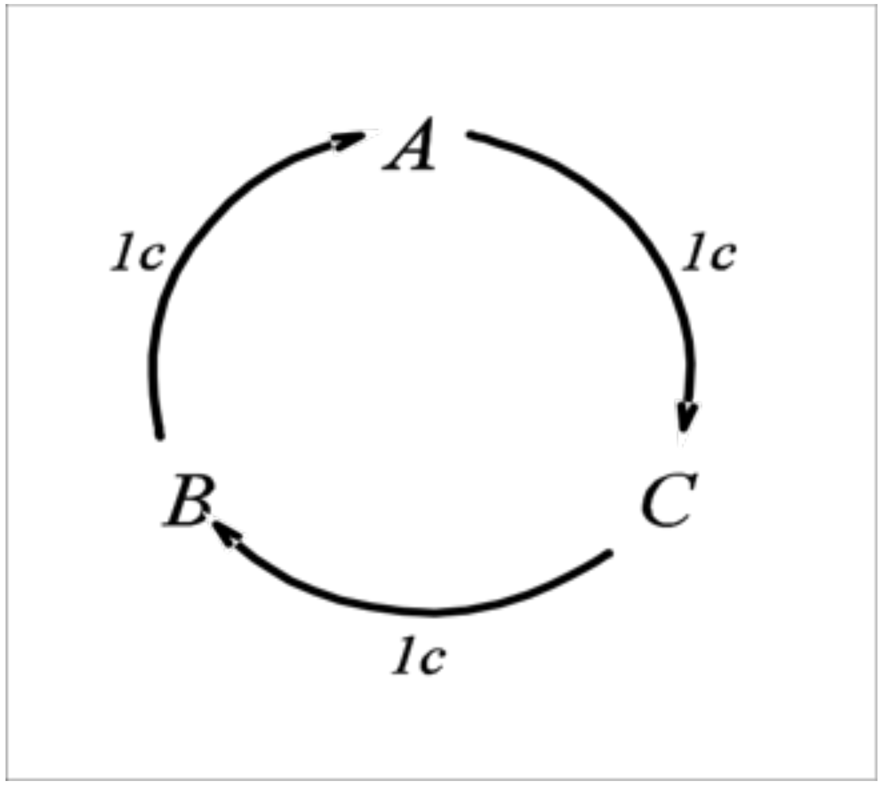
Why Impose These Conditions?

- Structure of preference ordering imposes certain “rationality requirements”
  - It is a weak ordering
- Example: Why transitivity?
  - Without transitivity, I can construct a “Money pump”

Money Pump

A>B>C>A

Assume that agent currently has item A. We offer to sell it item C for some small amount. Since C>A it accepts. Then sell it B. Since B>A it accepts. Sell it A. Since A>B it accepts....

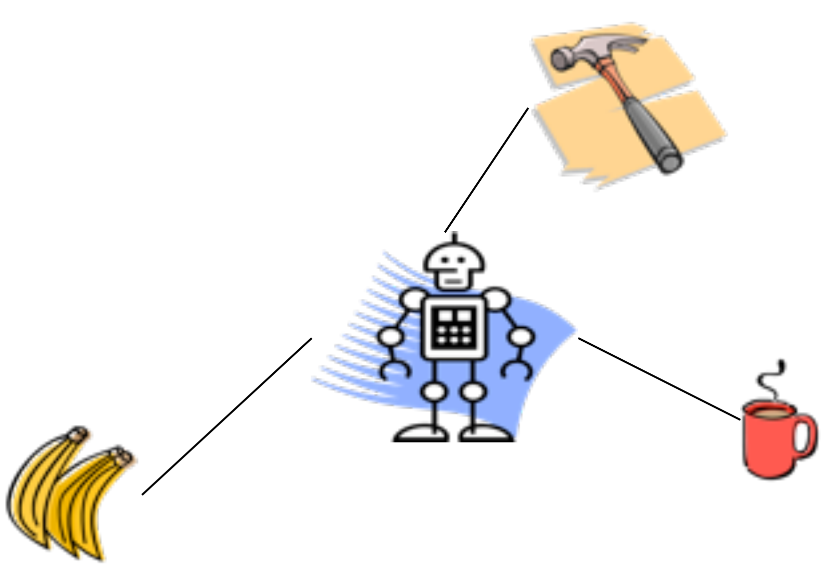


Decision Problem: Certainty

- A **decision problem under certainty** is <D, S, f, ≿> where
  - D is a set of decisions
  - S is a set of outcomes or states
  - f is an outcome function f:D→S
  - ≿ is a preference ordering over S
- A **solution** to a decision problem is any d* in D such that f(d*)≿f(d) for all d in D

Computational Issues

- At some level, a solution to a decision problem is trivial
  - But decisions and outcome functions are rarely specified explicitly
  - For example: In search you construct the set of decisions by exploring search paths
    - Do not know the outcomes in advance



Preferences

c, b, bc

>

c, b, ~bc

>

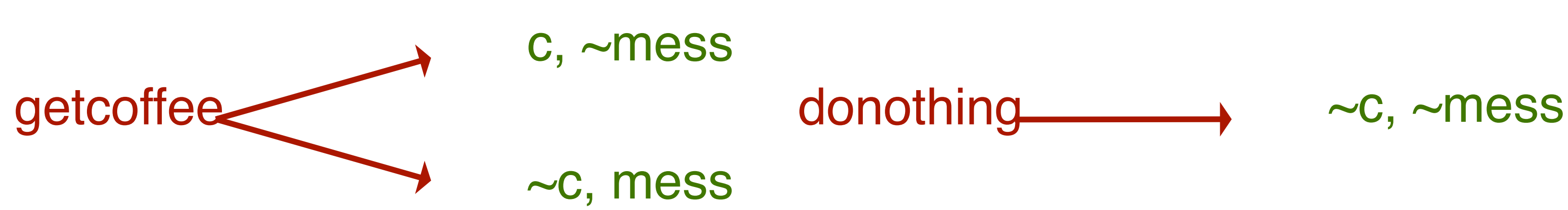
c, ~b, ~bc

>

c, ~b, bc

Decision Making Under Uncertainty

- Suppose actions do not have deterministic outcomes
  - Example: When the robot pours coffee, 20% of the time it spills it, making a mess
  - Preferences: c,~mess>~c,~mess>~c, mess
- What should your robot do?
  - Decision *getcoffee* leads to a good outcome and a bad outcome with some probability
  - Decision *donothing* leads to a medium outcome



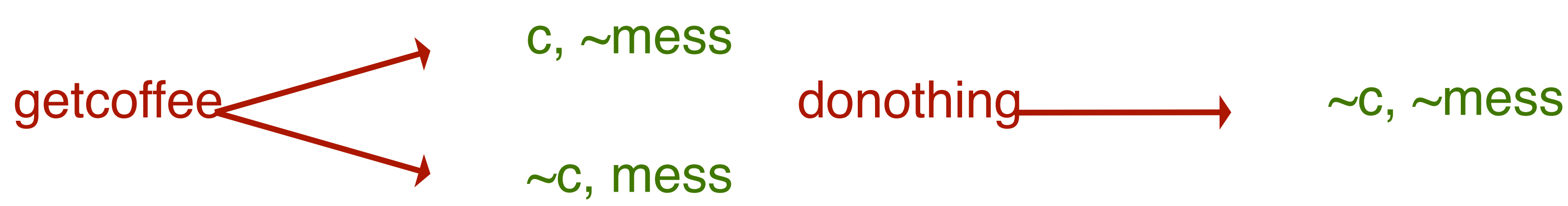
Utilities

- Rather than just ranking outcomes, we need to quantify our degree of preference
  - How much more we prefer one outcome to another (e.g c to $\sim mess$)
- A utility function $U: S \rightarrow \mathbf{R}$ associates a real-valued utility to each outcome
  - Utility measures your degree of preference for s
- U induces a preference ordering $\succsim_U$ over S where $s \succsim_U t$ if and only if $U(s) \geq U(t)$

Expected Utility

- Under conditions of uncertainty, decision d induces a distribution over possible outcomes
  - Pd(s) is the probability of outcome s under decision d
- The **expected utility** of decision d is $EU(d)=\sum_{s \text{ in } S} P_d(s)U(s)$

Example



- When my robot pours coffee, it makes a mess 20% of the time
- If U(c,~ms)=10, U(~c,~ms)=5, U(~c,ms)=0 then
  - EU(*getcoffee*)=(0.8)10+(0.2)0=8
  - EU(*donothing*)=5
- If U(c,~ms)=10, U(~c,~ms)=9, U(~c,ms)=0 then
  - EU(*getcoffee*)=8
  - EU(*donothing*)=9

Maximum Expected Utility Principle

- Principle of Maximum Expected Utility
  - The optimal decision under conditions of uncertainty is that with the greatest expected utility
- Robot example:
  - First case: optimal decision is *getcoffee*
  - Second case: optimal decision is *donothing*

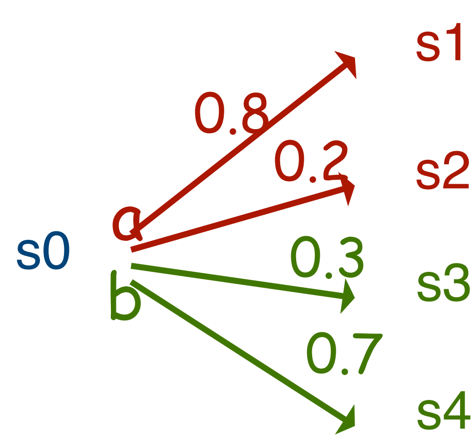
Decision Problem: Uncertainty

- A decision problem under uncertainty is
  - Set of decisions D
  - Set of outcomes S
  - Outcome function $P: D \rightarrow \Delta(S)$
    - $\Delta(S)$ is the set of distributions over S
  - Utility function U over S

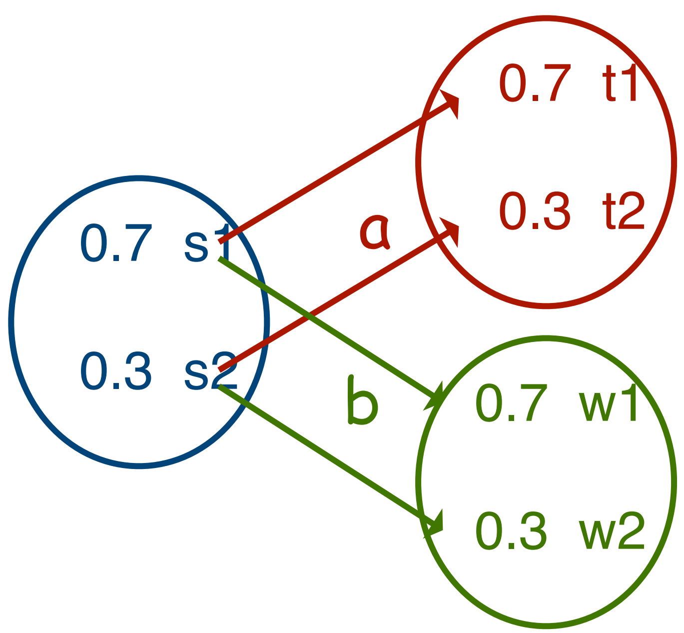
- A **solution** is any d* in D such that $EU(d^*) \geq EU(d)$ for all d in D

Notes: Expected Utility

- This viewpoint accounts for
  - Uncertainty in action outcomes
  - Uncertainty in state of knowledge
  - Any combination of the two



Stochastic actions



Uncertain knowledge

Notes: Expected Utility

- Why Maximum Expected Utility?

- Where do these utilities come from?
  - Preference elicitation

Notes: Expected Utility

- Utility functions need not be unique
  - If you multiply U by a positive constant, all decisions have the same relative utility
  - If you add a constant to U, then the same thing is true
- U is unique up to a positive affine transformation

If $d^{*} = \text{argmax} \sum_d \Pr(d)U(d)$

then

$d^{*} = \text{argmax} \sum_d \Pr(d)[aU(d)+b]$

$a>0$

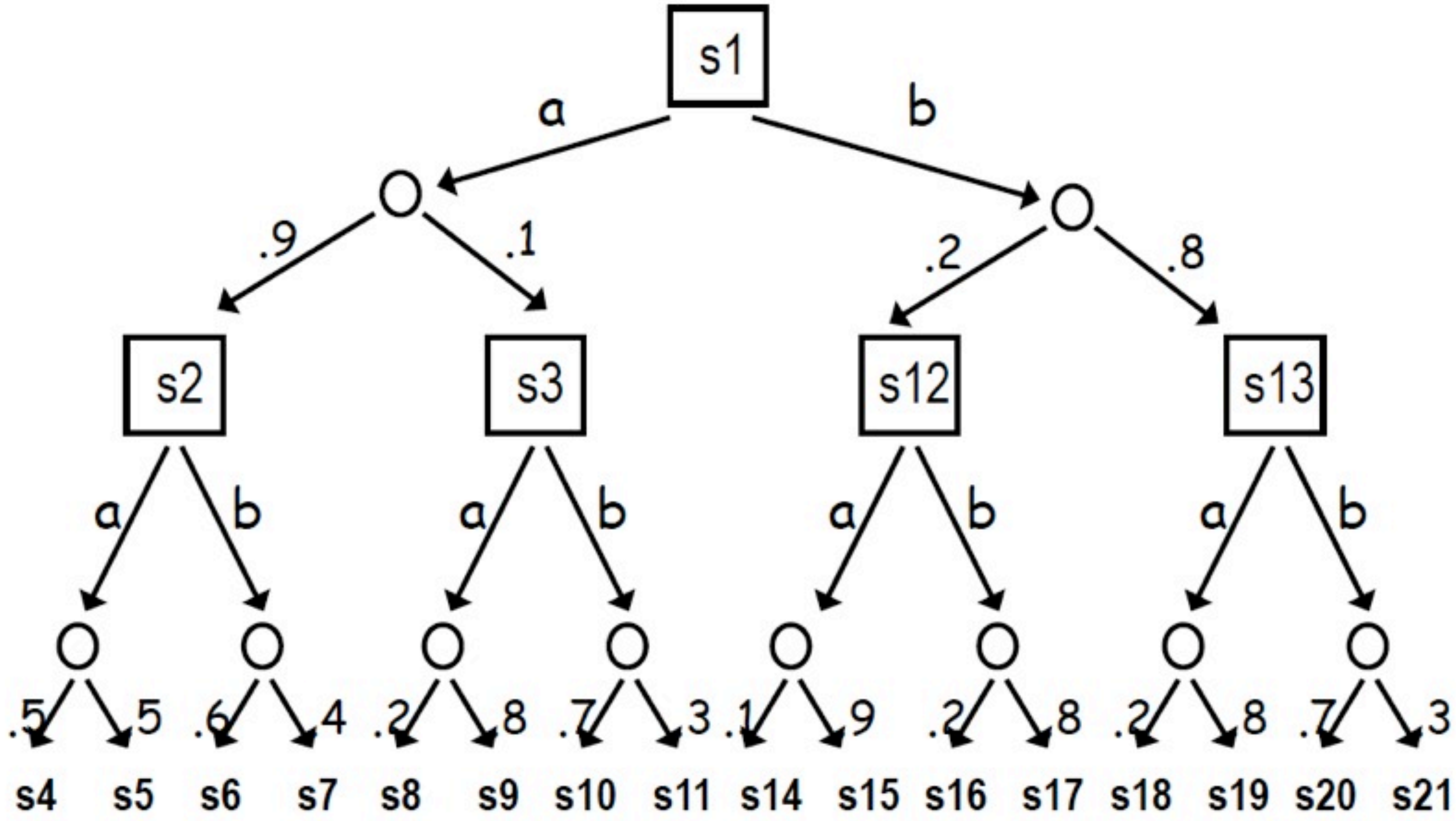
What are the Complications?

- Outcome space can be large
  - State space can be huge
  - Do not want to spell out distributions explicitly
  - **Solution**: Use Bayes Nets (or related Influence diagrams)
- Decision space is large
  - Usually decisions are not one-shot
    - Sequential choice
    - If we treat each plan as a distinct decision, then the space is too large to handle directly
  - **Solution**: Use dynamic programming to construct optimal plans

Simple Example

- Two actions: a,b
  - That is, either [a,a], [a,b], [b,a], [b,b]
- We can execute two actions in sequence
- Actions are stochastic: action a induces distribution $P_a(s_i|s_j)$ over states
  - $P_a(s_2|s_1)=0.9$ means that the prob. of moving to state s2 when taking action a in state s1 is 0.9
  - Similar distribution for action b
- How good is a particular plan?

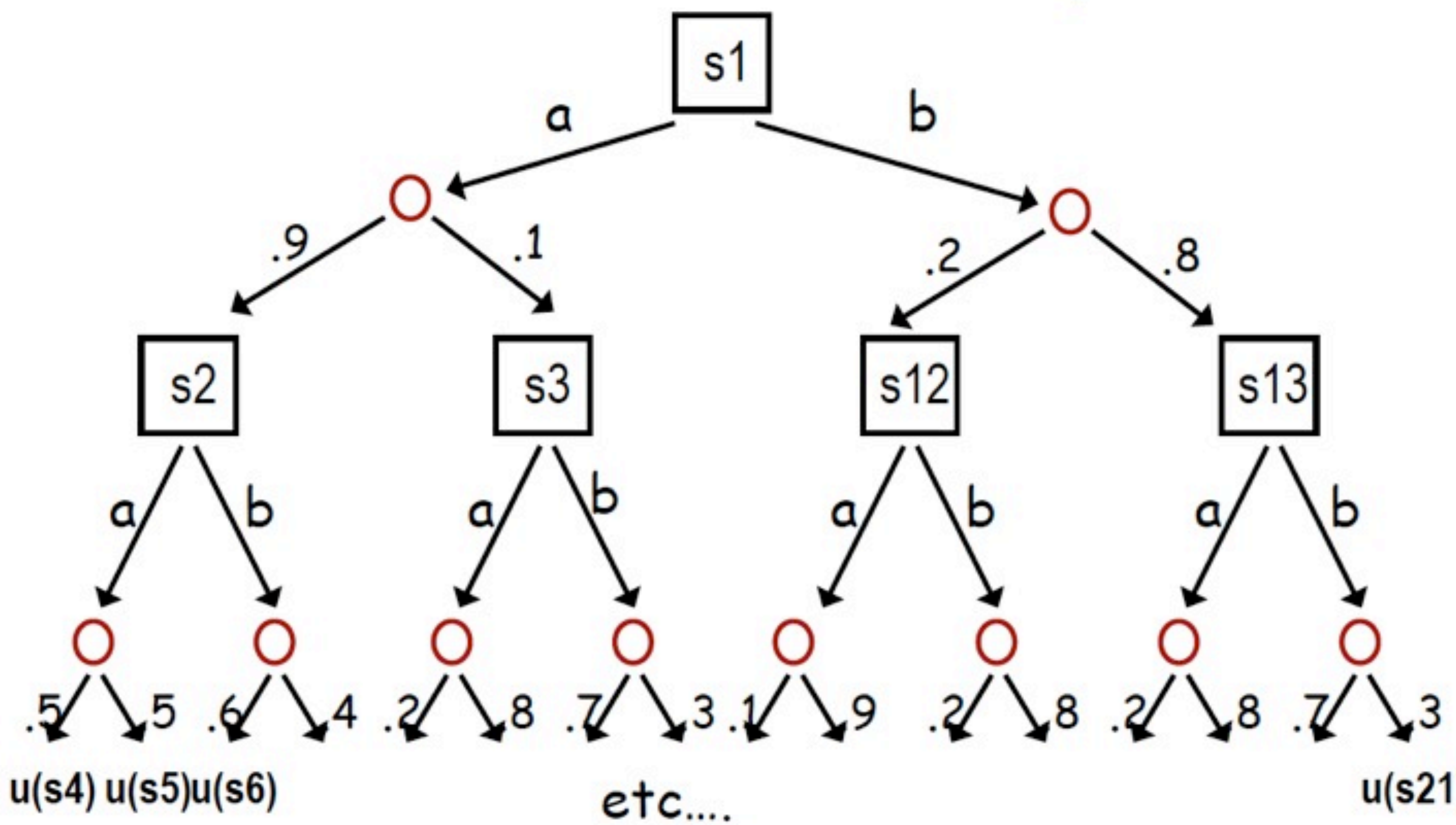
Distributions for Action Sequences



How Good is a Sequence?

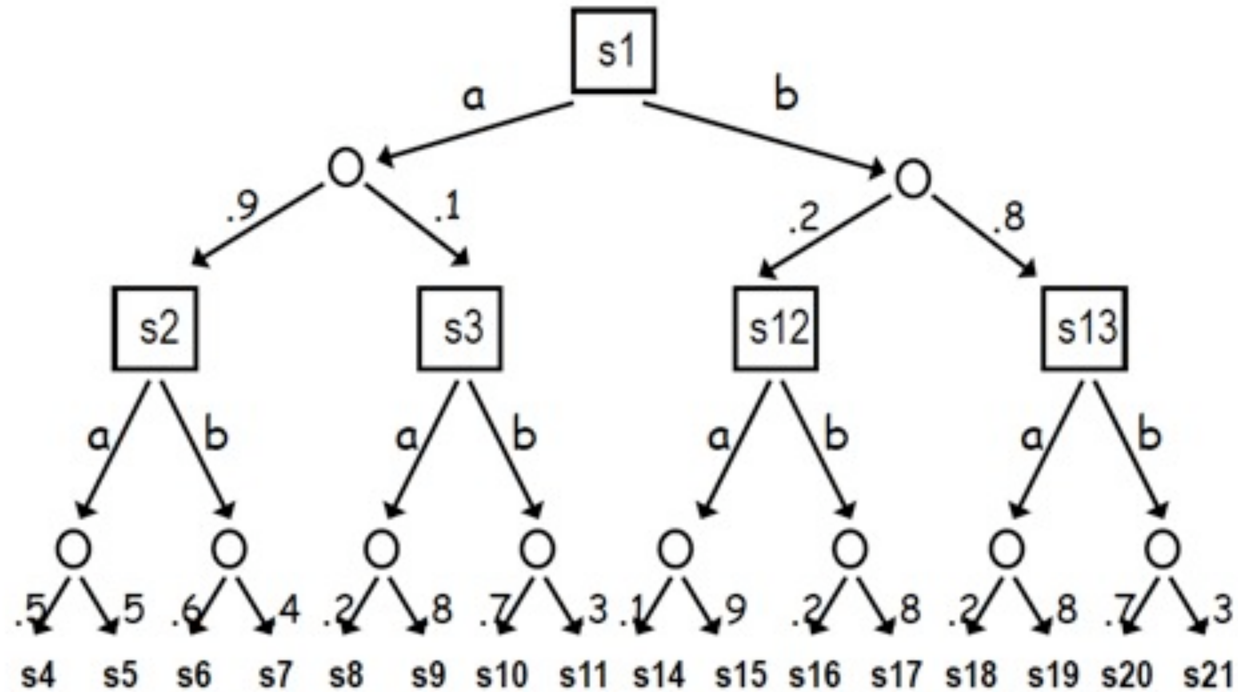
- We associate utilities with the **final outcome**
  - How good is it to end up at s_4, s_5, s_6, ...
- Now we have:
  - $EU(aa)=.45U(s_4)+.45U(s_5)+.02U(s_8)+.08(s_9)$
  - $EU(ab)=.54U(s_6)+.36U(s_7)+.07U(s_{10})+.03U(s_{11})$
  - etc

Utilities for Action Sequences



Looks a lot like a game tree, but with chance nodes instead of min nodes. (We average instead of minimizing)

Why Sequences Might Be Bad



- Suppose we do *a* first; we could reach s_2 or s_3
  - At s2, assume: EU(a)=.5U(s4)+.5U(s 5)>EU(b)=.6U(s6)+.4U(s7)
  - At s3 assume: EU(a)=.2U(s8)+.8U(s9)<EU(b)=.7U(s10)+.3U(s11)
- After doing *a* first, we want to do *a* next if we reach s_2, but we want to be *b* second if we reach s_3

Policies

- We want to consider **policies**, not sequences of actions (plans)
- We have 8 policies for the decision tree:

[a; if s2 a, if s3 a]	[b; if s12 a, if s13 a]
[a; if s2 a, if s3 b]	[b; if s12 a, if s13 b]
[a; if s2 b, is s3 a]	[b; if s12 b, if s13 a]
[a; if s2 b, if s3 b]	[b; if s12 b. if s13 b]

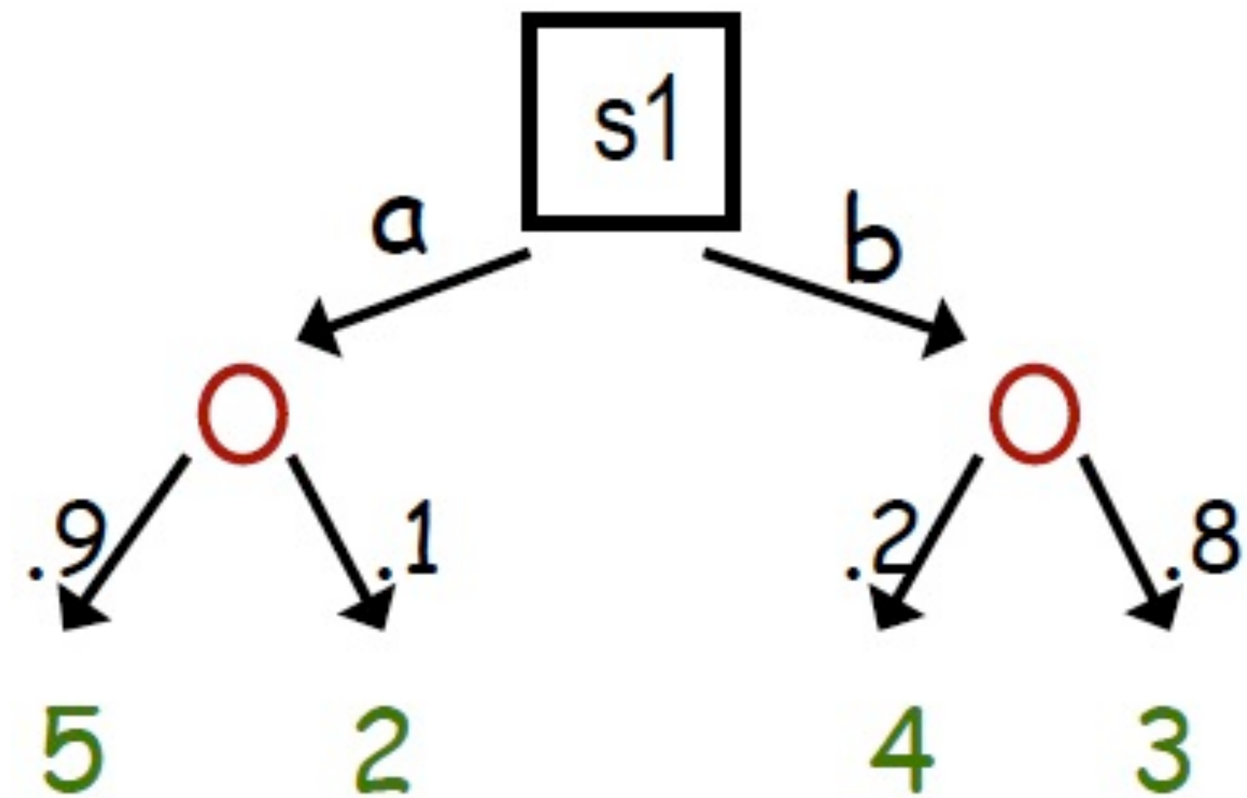
- We have 4 plans
  - [a;a], [a;b], [b;a], [b;b]
  - **Note**: each plans corresponds to a policy so we can only **gain** by allowing the decision maker to use policies

Evaluating Policies

- Number of plans (sequences) of length k
  - Exponential in k: $|A|^k$ if A is the action set
- Number of policies is much larger
  - If A is the action set and O is the outcome set, then we have $(|A||O|)^k$ policies
- Fortunately, dynamic programming can be used
  - Suppose EU(a)>EU(b) at s2
  - Never consider a policy that does anything else at s2
- How to do this?
  - Back values up the tree much like minimax search

Decision Trees

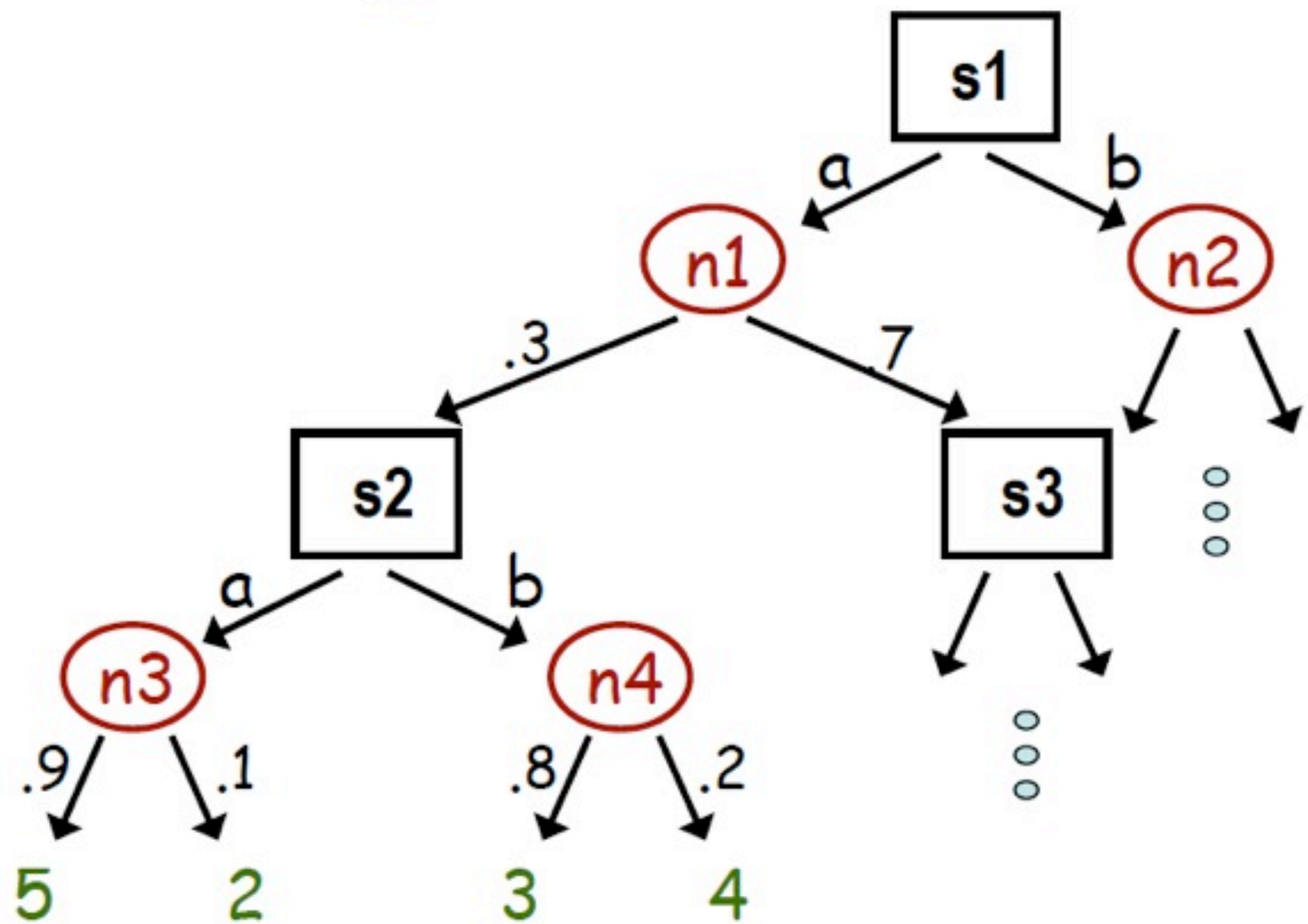
- Squares denote **choice** nodes (**decision** nodes)
- Circles denote **chance** nodes
    - Uncertainty regarding action effects
- Terminal nodes labelled with **utilities**



Evaluating Decision Trees

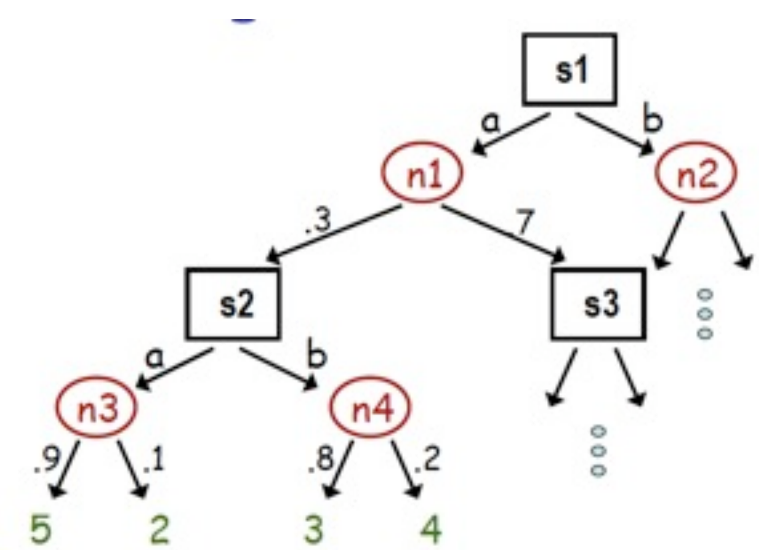
- Procedure is exactly like game trees except
  - “MIN” is “nature” who chooses outcomes at chance nodes with specified probability
    - Average instead of minimize
- Back values up the tree
  - U(t) defined for terminal nodes
  - U(n)=avg {U(c):c a child of n} if n is chance node
  - U(n)=max{U(c:c is child of n} if n is a choice node

Evaluating a Decision Tree



Decision Tree Policies

- Note that we don't just compute values, but policies for the tree
- A **policy** assigns a decision to each choice node in the tree
- Some policies can't be distinguished in terms of their expected values
  - Example: If a policy chooses *a* at s1, the choice at s4 does not matter because it won't be reached
  - Two policies are **implementationally indistinguishable** if they disagree only on unreachable nodes



Computational Issues

- Savings compared to explicit policy evaluation is substantial
- Let n=|A| and m=|O|
  - Evaluate only $O((nm)^d)$ nodes in tree of depth d
    - Total computational cost is thus $O((nm)^d)$
  - Note that there are also $(nm)^d$ policies
    - Evaluating a single policy requires $O(m^d)$
    - Total computation for explicitly evaluating each policy would be $O(n^d m^{2d})$

Computational Issues

- **Tree size**: Grows exponentially with depth
  - Possible solutions: Bounded lookahead, heuristic search procedures
- **Full Observability**: We must know the initial state and outcome of each action
  - Possible solutions: Handcrafted decision trees, more general policies based on observations

Other Issues

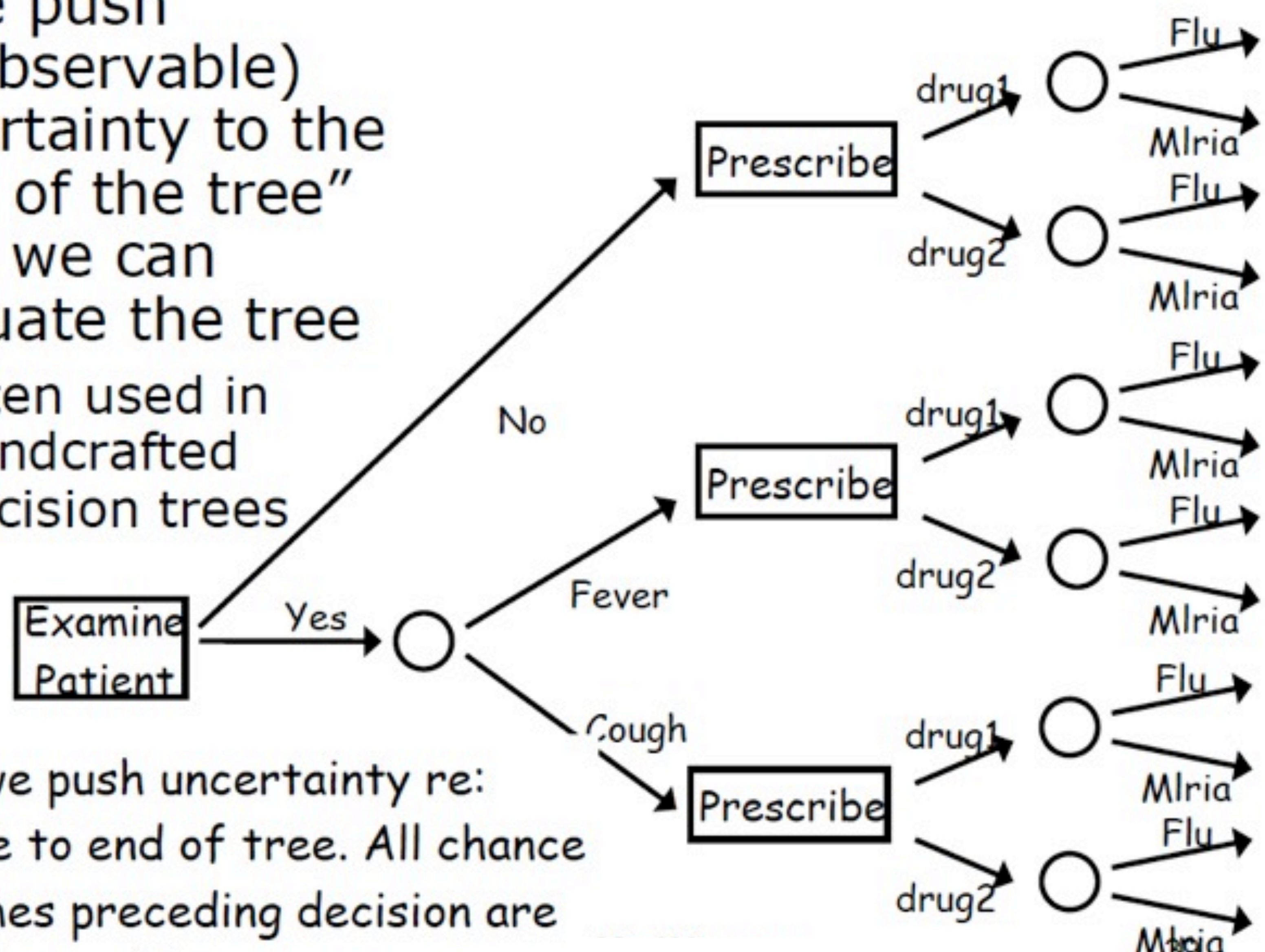
- **Specification:** Suppose each state is an assignment of values to variables
  - Representing action probability distributions is complex
    - Large branching factor
- Possible solutions:
  - Bayes Net representations
  - Solve problems using decision networks

Key Assumption: Observability

- **Full observability**: We must know the initial state and outcome of each action
  - To implement a policy we must be able to resolve the uncertainty of any chance node that is followed by a decision node
    - e.g. After doing a at s1, we must know which of the outcomes (s2 or s3) was realized so that we know what action to take next
  - Note: We don't need to resolve the uncertainty at a chance node if no decision follows it

Partial Observability

- If we push (unobservable) uncertainty to the "end of the tree" then we can evaluate the tree
  - often used in handcrafted decision trees



Here we push uncertainty re: disease to end of tree. All chance outcomes preceding decision are fully observable.

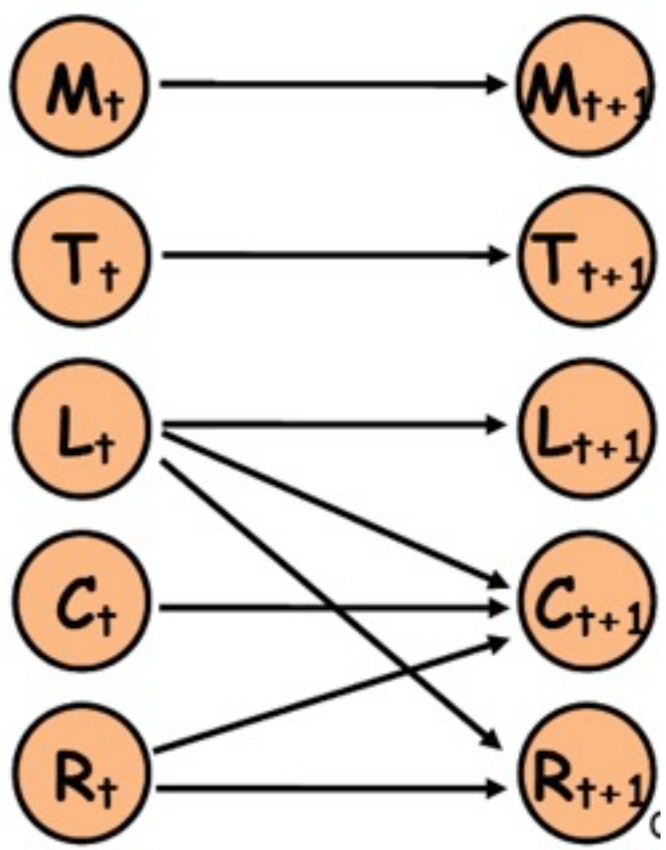
Large State Spaces (Variables)

- To represent outcomes of actions or decisions, we need to specify distributions
  - P(s|d): probability of outcome s given decision d
  - P(s|a,s'): probability of state s given action a was taken in state s'
- Note that the state space is exponential in the number of variables
  - Spelling out distributions explicitly is intractable
- Bayes Nets can be used to represent actions
  - Joint distribution over variables, conditioned on action/decision and previous state

Example Action Using a Dynamic Bayes Net

Deliver Coffee action

M – mail waiting C – Kate has coffee
T – lab tidy R – robot has coffee
L – robot located in Kate's office



T	T(t+1)	T(t+1)
T	1.0	0.0
F	0.0	1.0

$f_J(T_t, T_{t+1})$

L	R	C	C(t+1)	C(t+1)
T	T	T	1.0	0.0
F	T	T	1.0	0.0
T	F	T	1.0	0.0
F	F	T	1.0	0.0
T	T	F	0.8	0.2

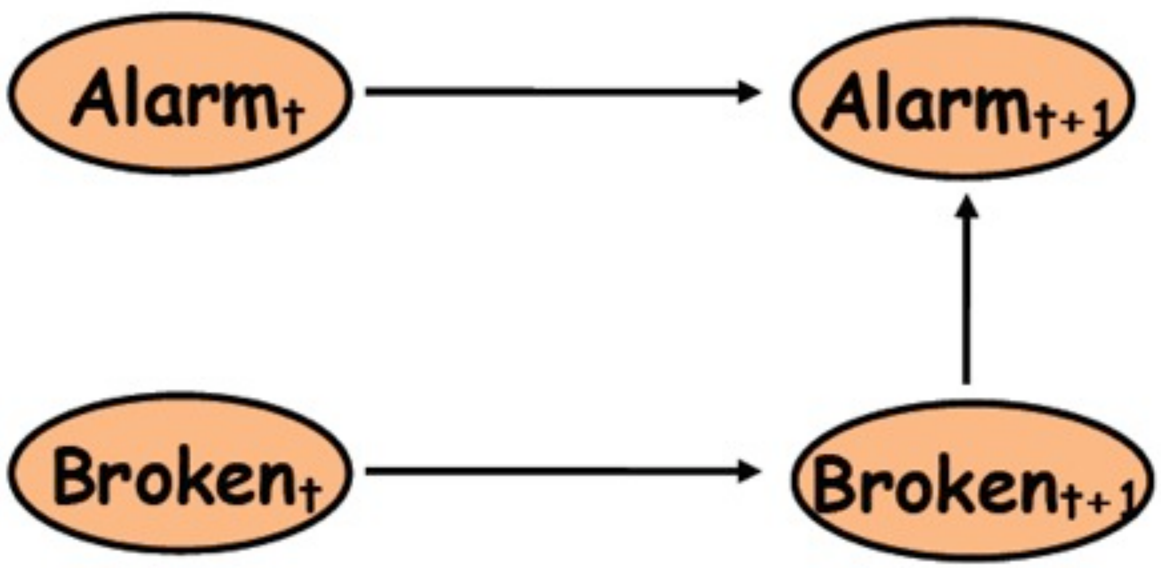
$f_R(L_t, R_t, C_t, C_{t+1})$

Dynamic BN Action Representation

- Dynamic Bayes Nets (DBN)
  - List all state variables for time t (pre-action)
  - List all state variables for time t+1 (post-action)
  - Indicate parents of all t+1 variables
    - Can include time t and t+1 variables, but network must be acyclic
  - Specify CPT for each time t+1 variable
- Note: Generally **no prior given** for time t variables
  - We are generally interested in **conditional distributions** over post-action states given pre-action states
  - Time t variables are instantiated as “evidence” when using a DBN (generally)

Example

Throw rock at window action



$P(al_{t+1} \mid al_t, Br_{t+1}) = 1$
$P(al_{t+1} \mid \sim al_t, \sim br_{t+1}) = 0$
$P(al_{t+1} \mid \sim al_t, br_{t+1}) = .95$

$P(broken_{t+1} \mid broken_t) = 1$
$P(broken_{t+1} \mid \sim broken_t) = .6$

Throwing rock has certain probability of breaking window and setting off alarm; but whether alarm is triggered depends on whether rock actually broke the window.

Use of BN Action Representation

- DBNs: Actions concisely, naturally specified
- Can be used in two ways
  - To generate “expectimax” search tree to solve decision problems
  - Used directly in stochastic decision making algorithms
- First use does not buy us that much computationally when solving decision problems
- Second use allows us to compute expected utilities without enumerating the outcome space (tree)
  - Decision networks (next week)

Summary

- Basic properties of preferences
- Relationship between preferences and utilities
- Principle of Maximum Expected Utility
- Decision Trees