CSC384: Intro to Artificial Intelligence

Decision Making Under Uncertainty

- Based on the slides by Maryam-Fazel Zarandi from Fall 2010.

- This material is covered in chapter 16.

Decision Making

- We want agents that can make rational decisions based on what they believe and what they want.
- Example: I give robot a planning problem: I want coffee
  - But coffee maker is broken: robot reports "No plan!"
- We really want more robust behavior.
  - Robot to know what to do if my primary goal can't be satisfied
  - Need to provide it with some indication of my *preferences over alternatives*
    - e.g., coffee better than tea, tea better than water, water better than nothing, etc.

Decision Making

- Decision making: Choosing among alternatives

- It's difficult due to:
  - complexity of decisions,
  - presence of uncertainty,
  - existence of multiple and sometimes opposing objectives in these environments

- Preferences guide decision making and are essential for making intelligent choices in complex situations.

Preference Orderings

- A *preference ordering* $\succcurlyeq$ is a ranking of all possible states of affairs (worlds) S
  - These could be outcomes of actions, truth assts, states in a search problem, etc.
  - $s \succcurlyeq t$: means that state s is *at least as good as* t
  - $s \succ t$: means that state s is *strictly preferred to* t
- We insist that $\succcurlyeq$ is
  - Reflexive: i.e., $s \succcurlyeq s$ for all states s
  - Transitive: i.e., if $s \succcurlyeq t$ and $t \succcurlyeq w$, then $s \succcurlyeq w$
  - Connected: for all states s,t, either $s \succcurlyeq t$ or $t \succcurlyeq s$

Decision Problems

- Two categories:
  - Decision making under certainty
    - Decision maker has complete and accurate knowledge of the consequences that will follow on each alternative
  - Decision making under uncertainty
    - Decision maker has accurate knowledge of a probability distribution of the outcomes of each alternative
    - Our focus here!

Decision Making under Uncertainty

- Suppose actions don't have deterministic outcomes
  - e.g., when robot pours coffee, it spills 20% of time, making a mess
  - preferences: chc, ¬mess ≻ ¬chc,¬mess ≻ ¬chc, mess
- What should robot do?
  - decision *getcoffee* leads to a good outcome and a bad outcome with some probability
  - decision *donothing* leads to a medium outcome for sure
- Should robot be optimistic? pessimistic?
- Odds of success should influence decision
  - but how?

Utilities

- Rather than just ranking outcomes, we must *quantify our degree of preference*
  - e.g., how much more important is having coffee than having tea?
- A *utility function* **U**: S $\rightarrow \mathbb{R}$ associates a real-valued *utility* with each outcome (state).
  - U(s) quantifies our degree of preference for s
- Note: U induces a preference ordering $\succcurlyeq_U$ over the states S defined as: $s \succcurlyeq_U t$ iff $U(s) \geq U(t)$
  - $\succcurlyeq_U$ is reflexive, transitive, connected

Expected Utility

- With utilities we can compute **expected utilities!**
- In decision making under uncertainty, each decision *d* induces a distribution $\Pr_d$ over possible outcomes
  - $\Pr_d(s)$ is probability of outcome *s* under decision *d*

- The *expected utility* of decision *d* is defined

$$EU(d) = \sum_{s \in S} \Pr_d(s) U(s)$$

Expected Utility

- Say U(chc,¬mess) = 10, U(¬chc,¬mess) = 5, U(¬chc,mess) = 0,
- Then
  - EU(getcoffee) = 8
  - EU(donothing) = 5

Maximum Expected Utility (MEU)

- If U(chc,¬mess) = 10, U(¬chc,¬mess) = 9, U(¬chc,ms) = 0,
  - EU(getcoffee) = 8
  - EU(donothing) = 9

Maximum Expected Utility (MEU)

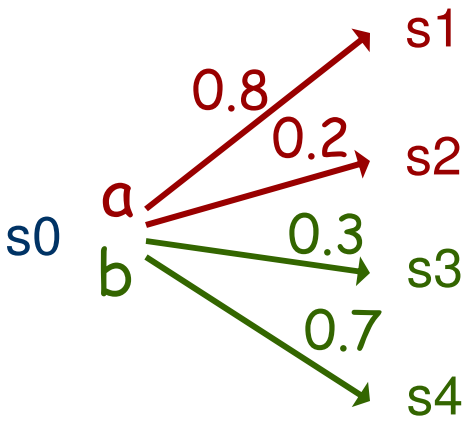
The MEU Principle

- *Principle of maximum expected utility (MEU)* states that the optimal decision under conditions of uncertainty is the decision that has greatest expected utility.

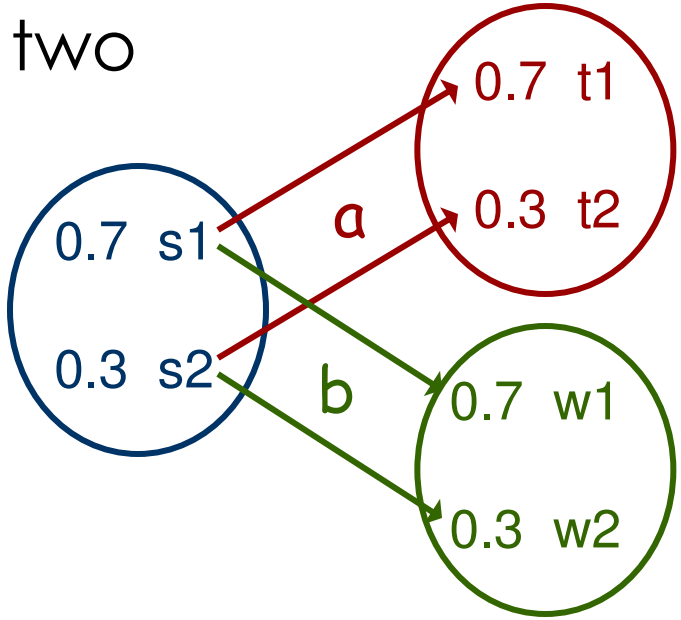
- In our example
  - if my utility function is the first one, my robot should get coffee
  - if your utility function is the second one, your robot should do nothing

Expected Utility: Notes

- Note that this viewpoint accounts for both:
  - uncertainty in action outcomes
  - uncertainty in state of knowledge
  - any combination of the two



Stochastic actions



Uncertain knowledge

Expected Utility: Notes

- Why MEU?
  - underlying foundations of utility theory tightly couple utility with action/choice
- Where do utilities come from?
  - a utility function can be determined by asking someone about their preferences for actions in specific scenarios (or "lotteries" over outcomes)
  - much work on how to do this efficiently

Decision Problems: Uncertainty

- A *decision problem under uncertainty* is:
  - a set of *decisions* D
  - a set of *outcomes* or states S
  - an *outcome function* $Pr : D \rightarrow \Delta(S)$
    - $\Delta(S)$ is the set of distributions over S (e.g., Pr_d)
  - a *utility* function U over S

- A *solution* to a decision problem under uncertainty is any $d^* \in D$ such that $EU(d^*) \geq EU(d)$ for all $d \in D$

Computational Issues

- At some level, solution to a decision problem is trivial
  - complexity lies in the fact that the decisions and outcome function are rarely specified explicitly
  - e.g., in planning or search problem, you *construct* the set of decisions by constructing paths or exploring search paths. Then we have to evaluate the expected utility of each. Computationally hard!
  - e.g., we find a plan achieving some expected utility e
    - Can we stop searching?
    - Must convince ourselves no better plan exists
    - Generally requires searching entire plan space, unless we have some clever tricks

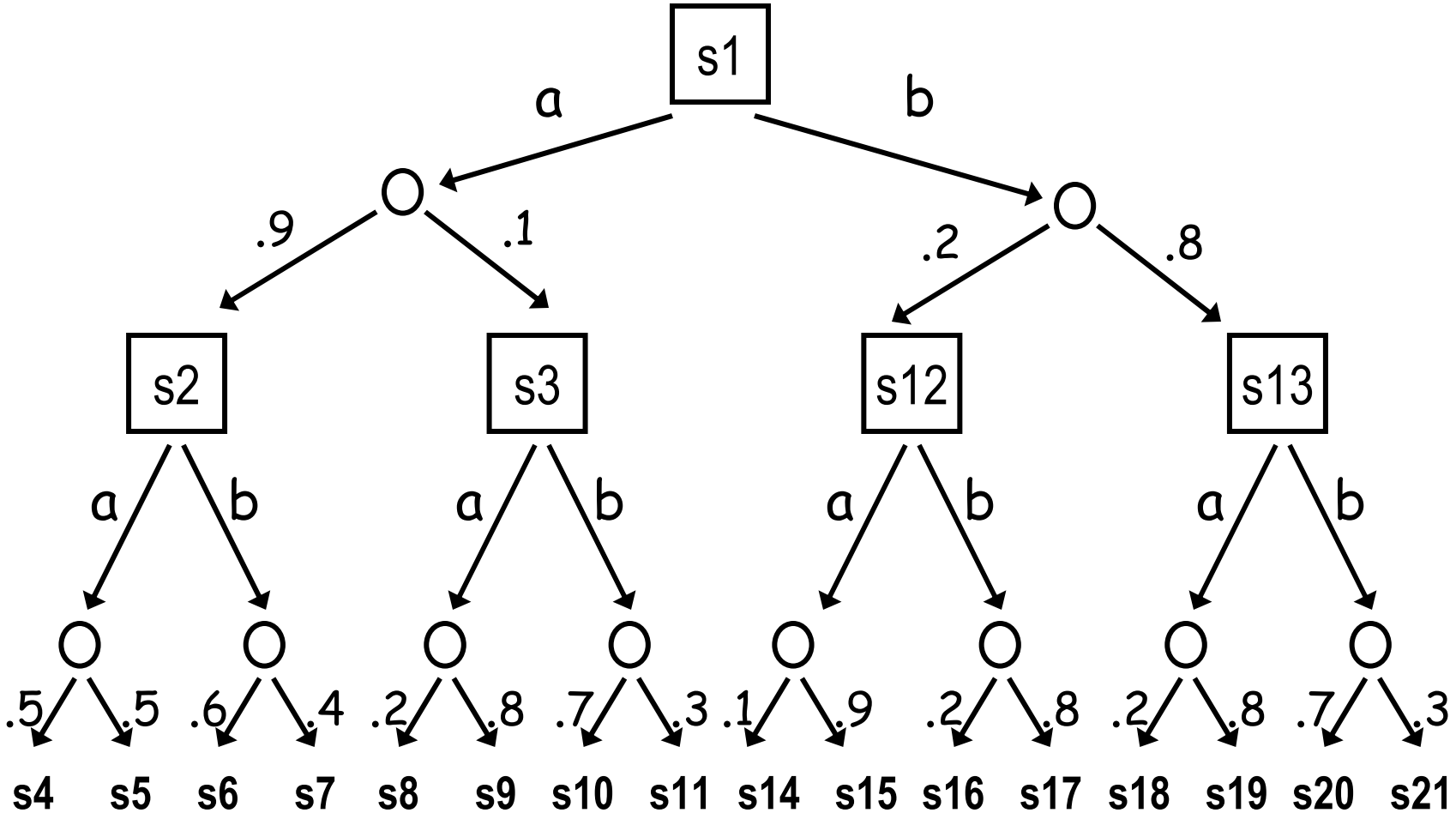
Computational Issues

- Outcome space is large
  - Like all of our problems, states spaces can be huge
  - Don't want to spell out distributions like Pr_d explicitly
  - Solution: Bayes nets (or related: *influence diagrams*)
- Decision space is large
  - Usually our decisions are not one-shot actions
  - Rather they involve sequential choices (like plans)
  - If we treat each plan as a distinct decision, decision space is too large to handle directly
  - Solution: Use dynamic programming methods to *construct* optimal plans (actually generalizations of plans, called policies… like in game trees)

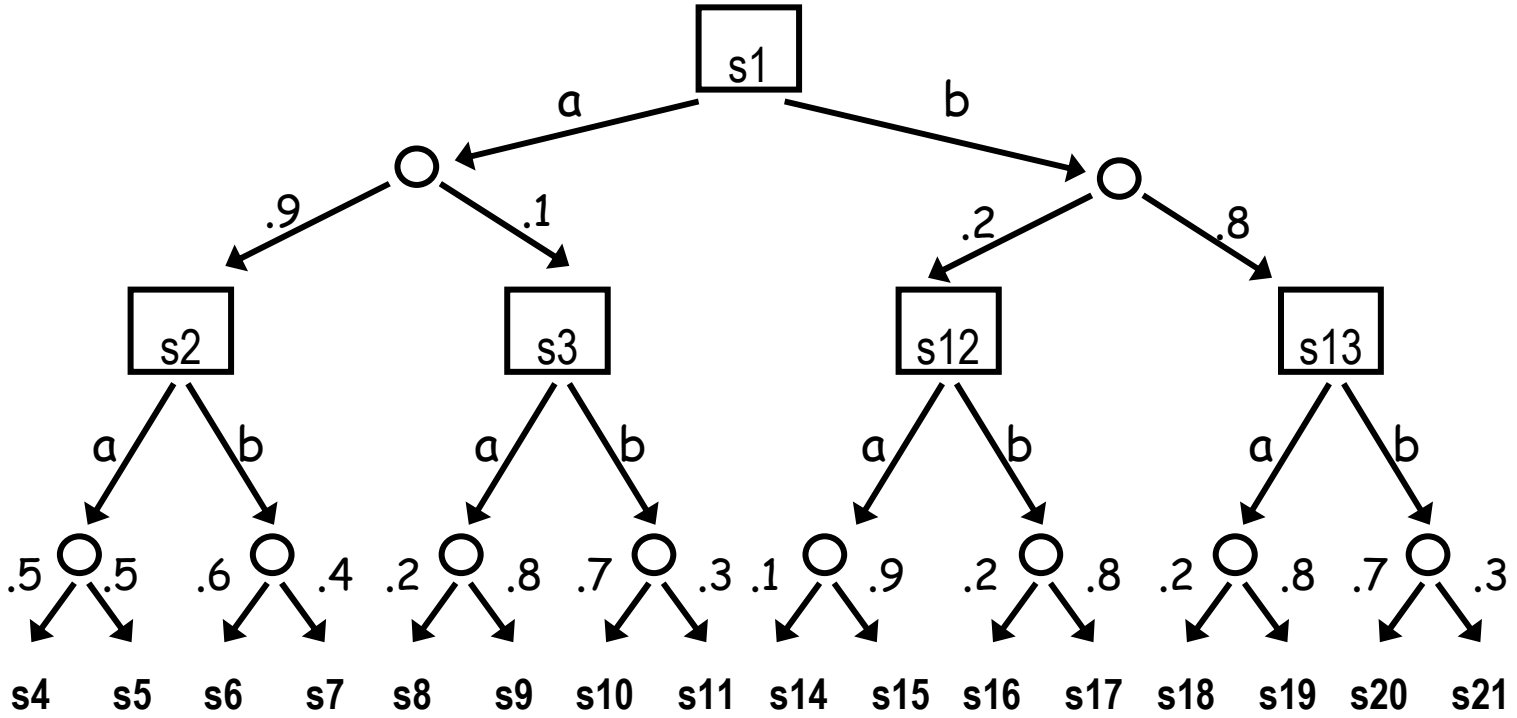
A Simple Example

- Suppose we have two actions: a, b
- We have time to execute *two* actions in sequence
- This means we can do either:
  - [a,a], [a,b], [b,a], [b,b]
- Actions are stochastic: action *a* induces distribution $Pr_a(s_i \mid s_j)$ over states
  - e.g., $Pr_a(s_2 \mid s_1) = .9$ means prob. of moving to state s_2 when a is performed at s_1 is .9
  - similar distribution for action b
- How good is a particular sequence of actions?

Distributions for Action Sequences



Distributions for Action Sequences



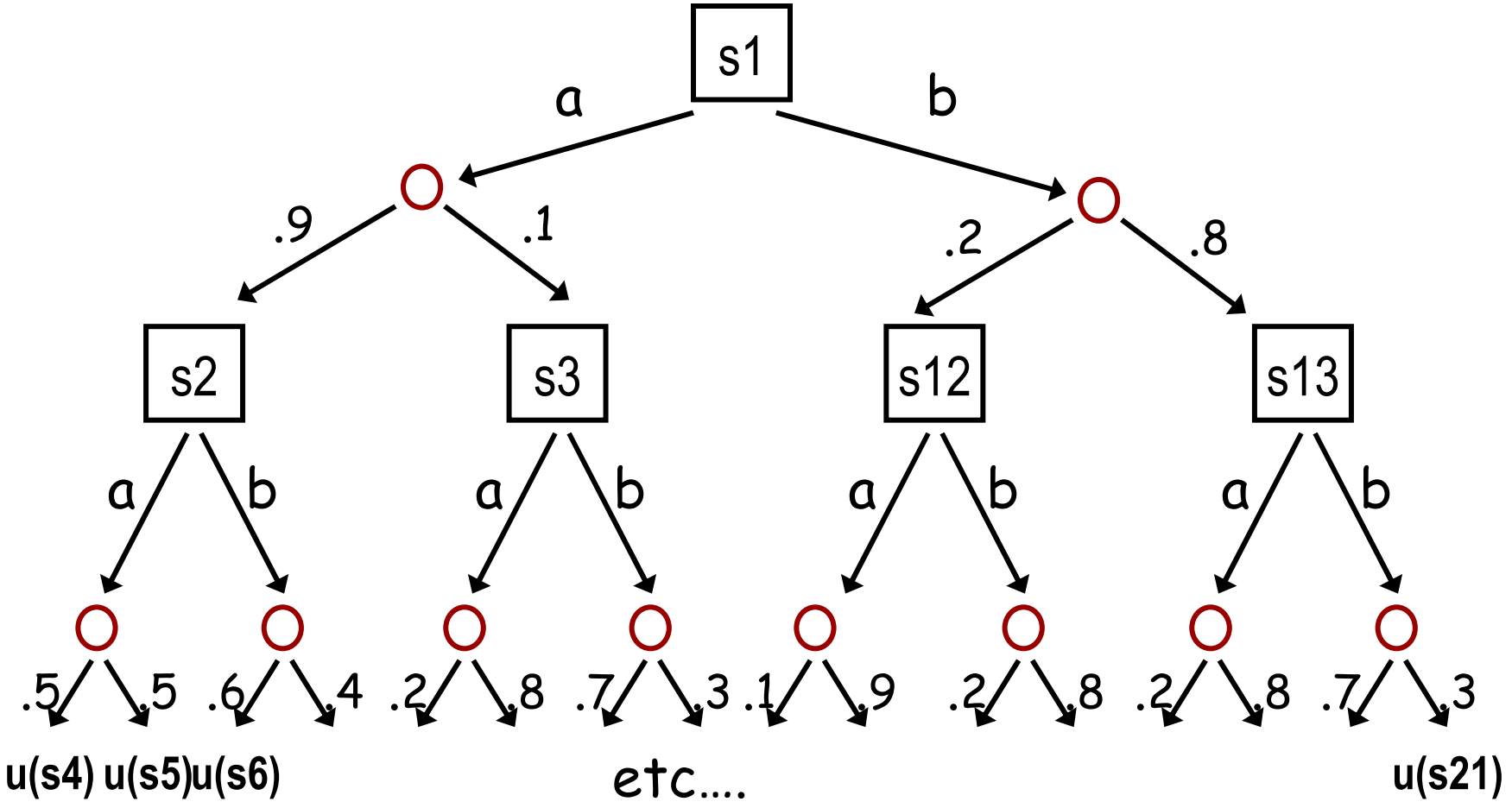
- Sequence [a,a] gives distribution over "final states"
  - Pr(s4) = .45, Pr(s5) = .45, Pr(s8) = .02, Pr(s9) = .08
- Similarly:
  - [a,b]: Pr(s6) = .54, Pr(s7) = .36, Pr(s10) = .07, Pr(s11) = .03
  - and similar distributions for sequences [b,a] and [b,b]

How Good is a Sequence?

- We associate *utilities with the "final" outcomes*
  - how good is it to end up at s4, s5, s6, …

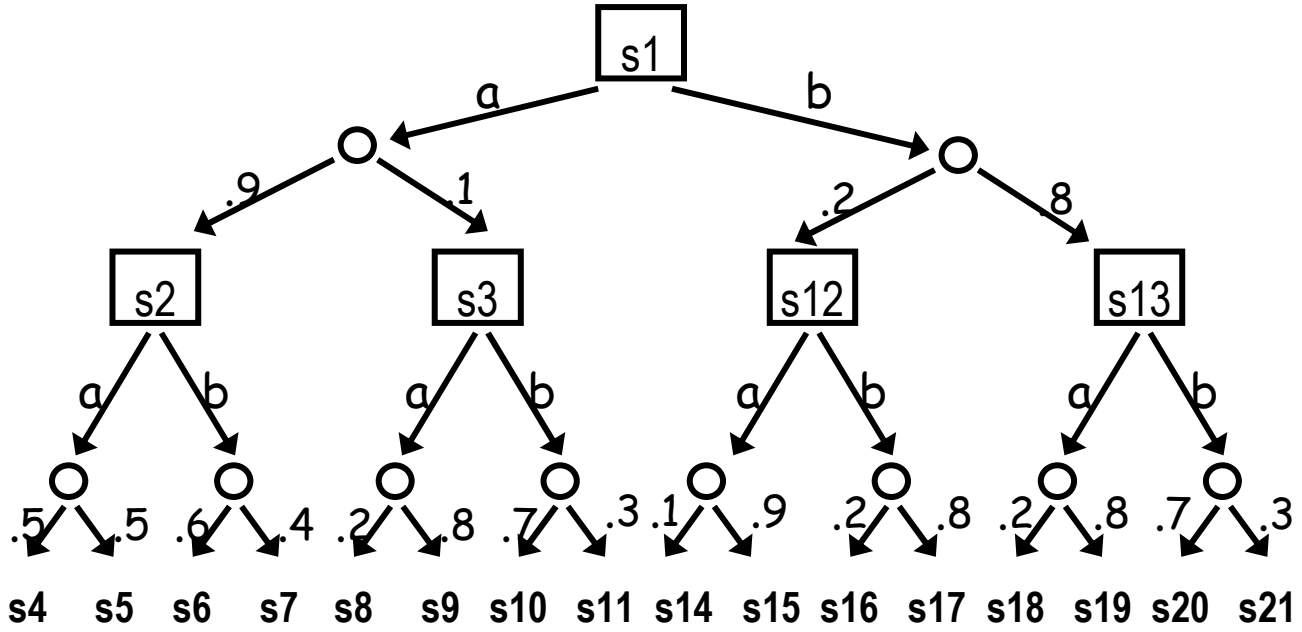
- Now we have:
  - EU(aa) = .45u(s4) + .45u(s5) + .02u(s8) + .08u(s9)
  - EU(ab) = .54u(s6) + .36u(s7) + .07u(s10) + .03u(s11)
  - etc…

Utilities for Action Sequences



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

Action Sequences are not sufficient



- Suppose we do *a* first; we could reach s2 or s3:
  - At s2, assume: EU(a) = .5u(s4) + .5u(s5) > EU(b) = .6u(s6) + .4u(s7)
  - At s3: 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 s2*, but we want to do *b* second *if we reach s3*

Policies

- This suggests that when dealing with uncertainty we want to consider *policies*, **not** just sequences of actions (plans)
- We have eight policies for this 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, if s3 a]	[b; if s12 b, if s13 a]
[a; if s2 b, if s3 b]	[b; if s12 b, if s13 b]

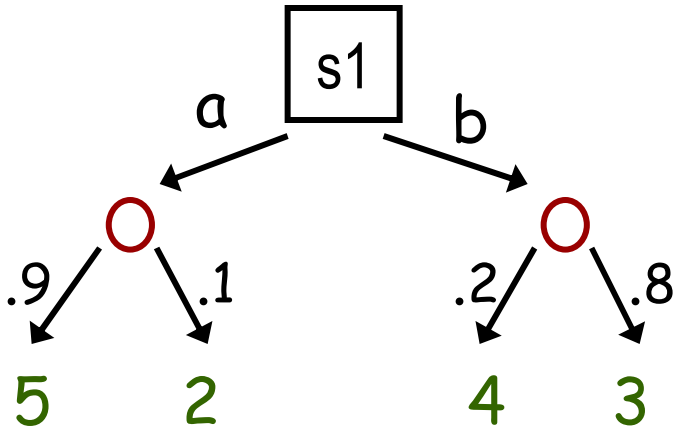
- Contrast this with four "plans"
  - [a; a], [a; b], [b; a], [b; b]
  - note: each plan corresponds to a policy, so we can only *gain* by allowing decision maker to use policies

Evaluating Policies

- Number of plans (sequences) of length *k*
  - Exponential in *k*: $|A|^k$ if *A* is our action set
- Number of policies is even larger
  - If A is our action set with max. m outcomes per action, then we have $(m|A|)^k$ policies
- Fortunately, *dynamic programming* can be used
  - e.g., 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
  - these denote action choices by decision maker *(decision nodes)*
- Circles denote *chance* nodes
  - these denote uncertainty regarding action effects
  - "Nature" will choose the child with specified probability



- Terminal nodes labeled with *utilities*
  - denote utility of final state (or it could denote the utility of "trajectory" (branch) to decision maker

Evaluating Decision Trees

- Procedure is exactly like game trees, except…
  - key difference: the "opponent" is "nature" who simply chooses outcomes at chance nodes with specified probability: so we take expectations instead of minimizing
- Back values *up* the tree
  - *U(t)* is defined for all terminals (part of input)
  - *U(n)* = exp {*U(c)* : c a child of *n*} if *n* is a **chance node**
  - *U(n)* = max {*U(c)* : c a child of *n*} if *n* is a **choice node**
- At any choice node (state), the decision maker chooses action that leads to *highest utility child*
- We compute policies: A ***policy*** assigns a *decision* to each choice node in the tree

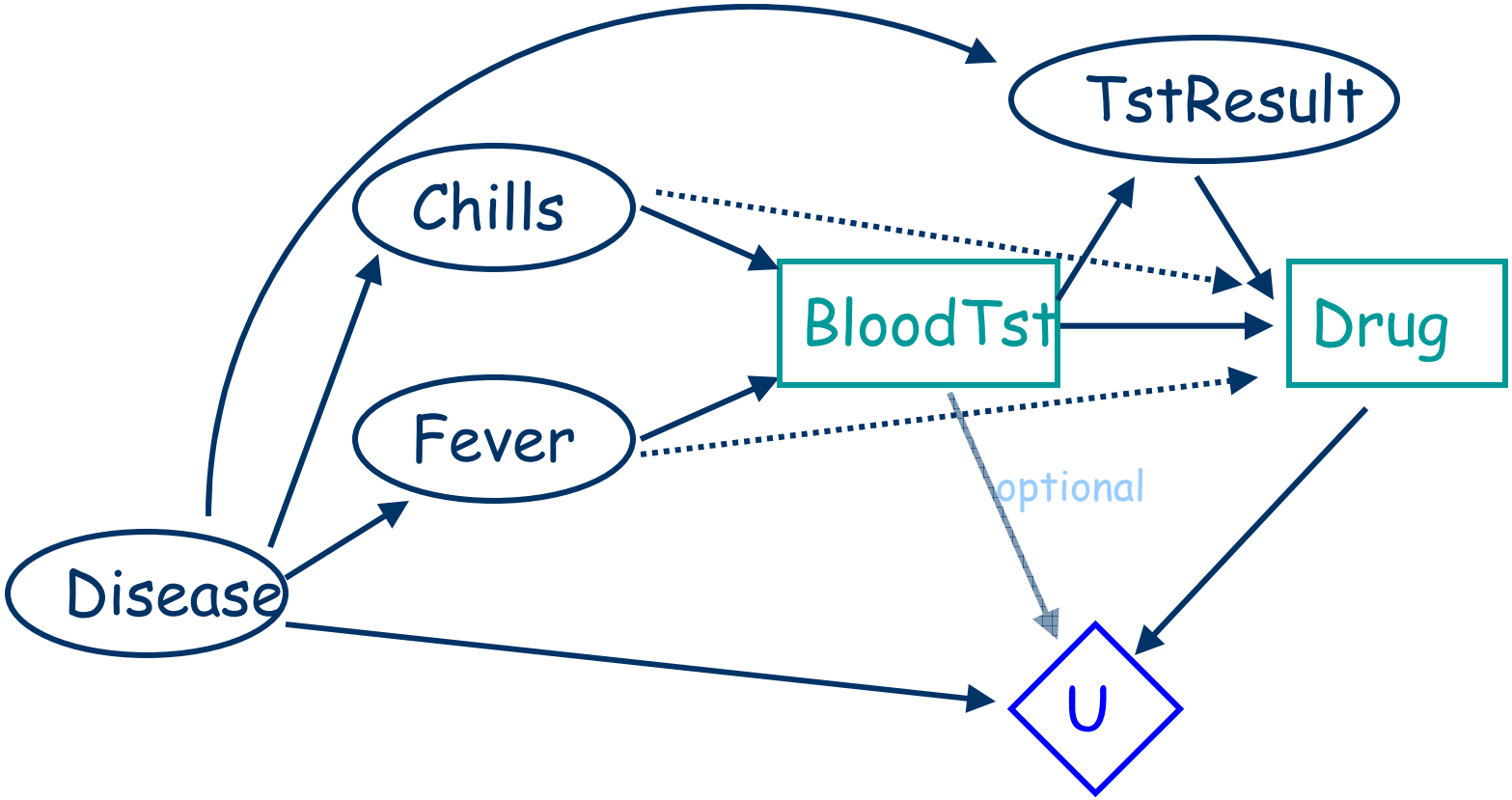
Key Assumption: Observability

- **Full observability:** we must know the initial state and outcome of each action
  - Specifically, to implement the 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 we know what action to do next (note: s2 and s3 may prescribe different actions)

Decision Networks

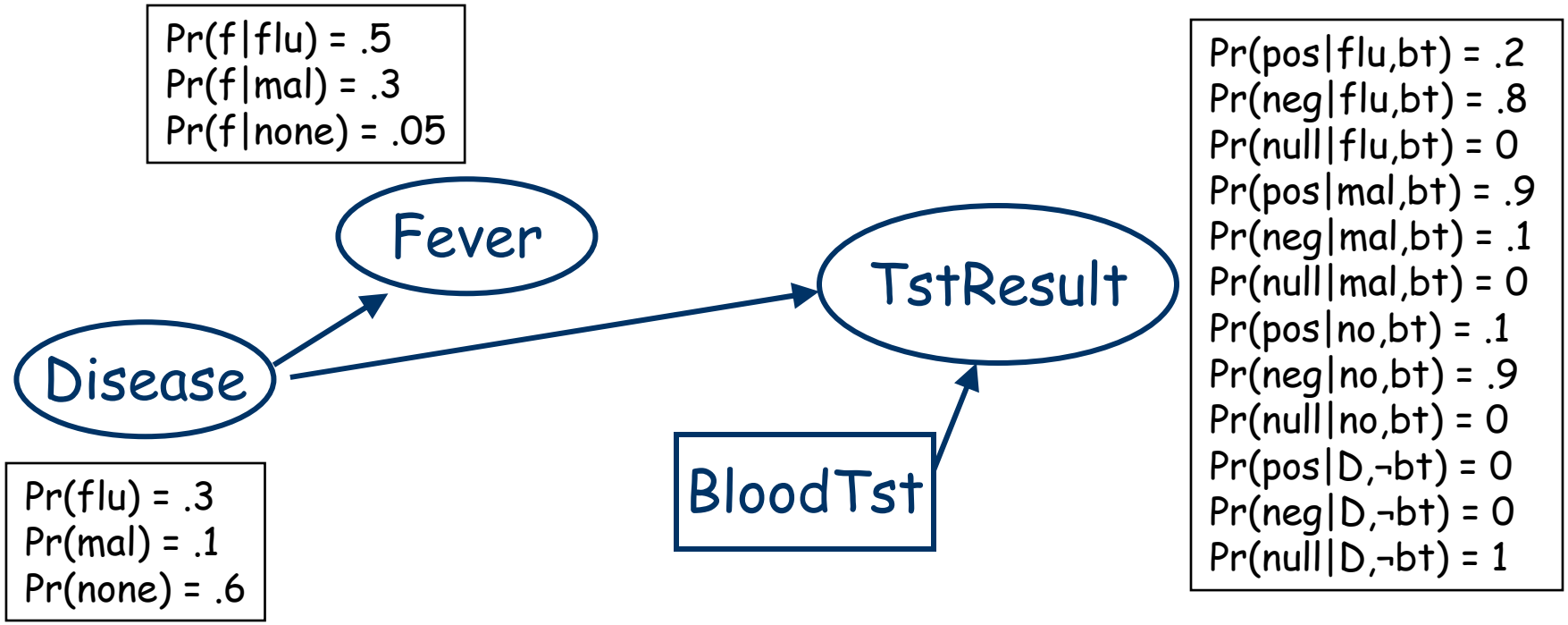
- *Decision networks* provide a way of representing sequential decision problems
  - Basic idea: represent the variables in the problem as you would in a BN
  - Add decision variables
    - These are variables that you "control"
  - Add utility variables
    - These are variable for how good different states are
- Also known as *influence diagrams*

Sample Decision Network



Decision Networks: Chance Nodes

- **Chance nodes**
  - Random variables, denoted by circles
  - as in a BN, probabilistic dependence on parents

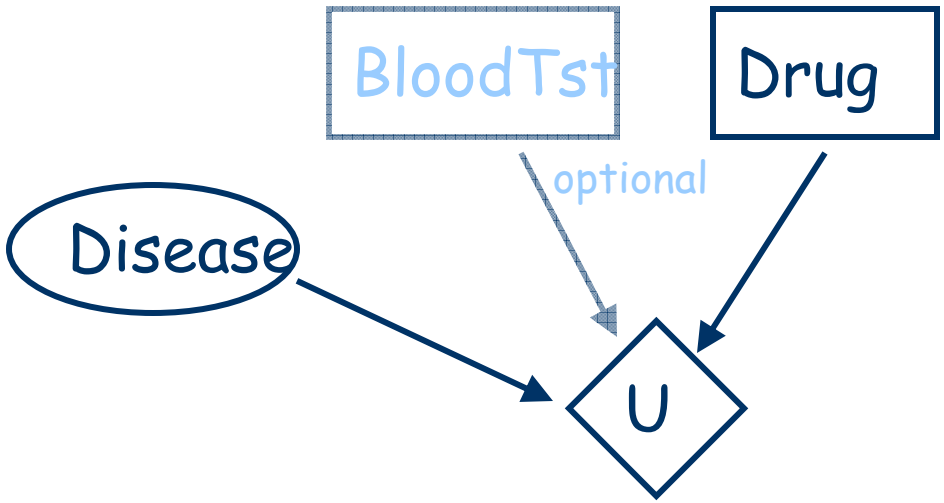


Decision Networks: Decision Nodes

- **Decision nodes**
  - Variables decision maker sets, denoted by squares
  - Parents reflect *information available* at time decision is to be made
- In example decision node: the actual values of Ch and Fev will be observed before the decision to take test must be made
  - Agent can make *different decisions* for each instantiation of parents

Decision Networks: Value Node

- **Value node**
  - specifies utility of a state, denoted by a diamond
  - utility depends *only on state of parents* of value node
  - generally: only one value node in a decision network
- Utility depends only on disease and drug



U(fludrug, flu) = 20
U(fludrug, mal) = -300
U(fludrug, none) = -5
U(maldrug, flu) = -30
U(maldrug, mal) = 10
U(maldrug, none) = -20
U(no drug, flu) = -10
U(no drug, mal) = -285
U(no drug, none) = 30

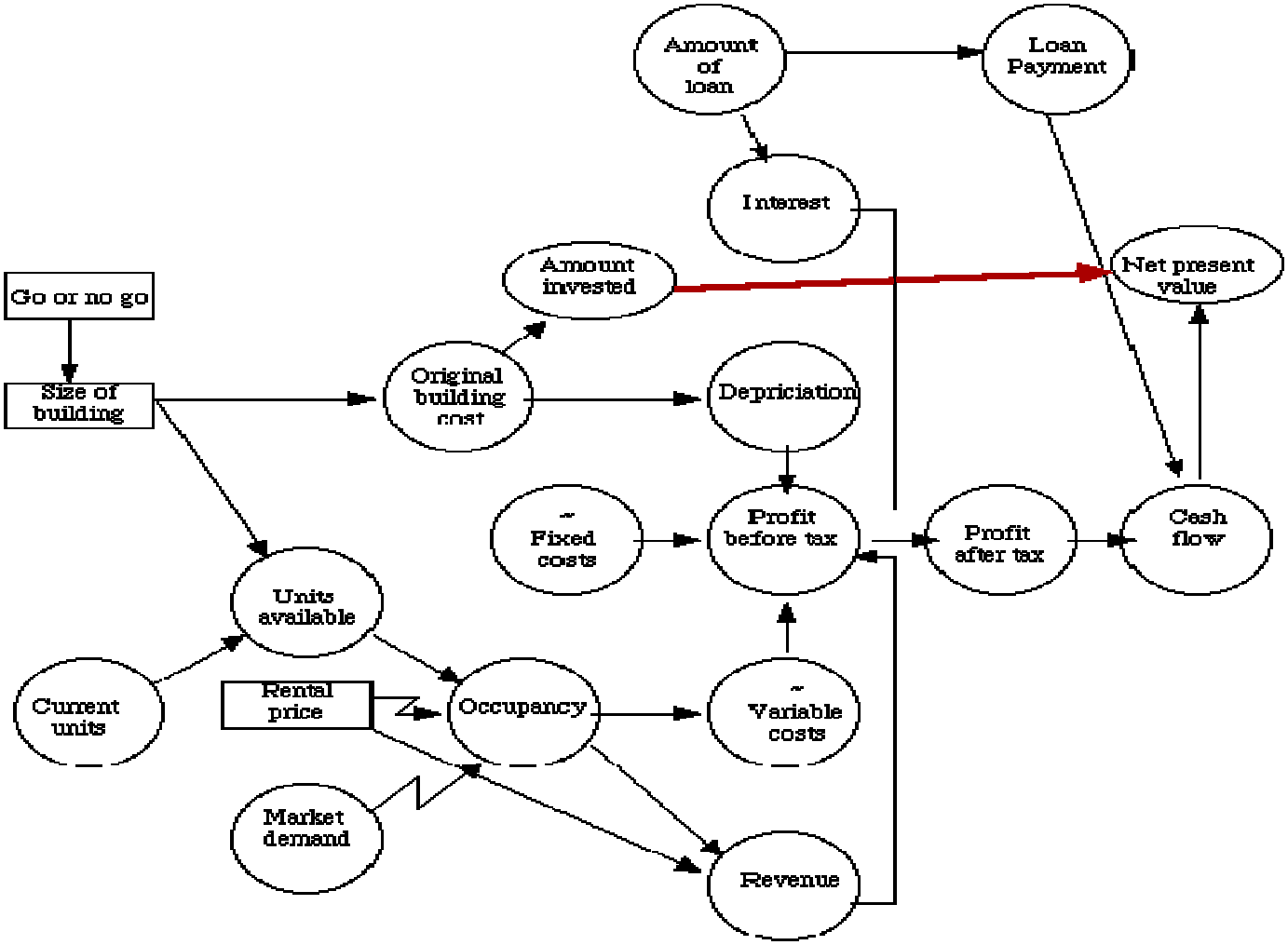
How to Use a Decision Network

- Assumption: Decision nodes are totally ordered
  - Decisions are made in sequence
- Goal: compute best policy δ
  - Set of mappings: a decision for each decision node based on its parents (which we know by then!)
  - *Value of* δ: expected utility (EU) given that decision nodes are executed according to δ
  - *Optimal policy* δ^*: $EU(\delta^*) \geq EU(\delta)$ for all policies δ

Evaluating a Decision Network

- How to compute best policy:
  - Recursively compute policies for decision nodes backwards, for each decision node using variable elimination
  - "*Optimize one decision node at a time*"
  - For each decision node, we have to compute expected utility for every possible instantiation of parent nodes
    → lots of variable eliminations!
- Complexity much greater than BN inference
  - We need to solve a number of BN inference problems
  - One BN problem for each setting of decision node parents and decision node value
  - Much work put into computationally effective techniques to solve these

Real Estate Investment



A Decision Network Example

- Setting: You want to buy a used car, but there's a good chance it is a "lemon" (i.e., prone to breakdown).
  - Before deciding to buy it, you can take it to a mechanic for inspection. They will give you a report on the car, labeling it either "good" or "bad".
  - A good report is positively correlated with the car being sound, while a bad report is positively correlated with the car being a lemon.

- However the report costs $50. So you could risk it, and buy the car without the report.

- Owning a sound car is better than having no car, which is better than owning a lemon.

Value of Information

- Optimal policy is: inspect, buy the car on a good report and don't on a bad report.
  - EU = 205 (Note that the last expected utility calculated takes into account the random outcome of the other decisions.
  - But suppose inspection cost $55: then EU(i) would be equal to 200 which is the same as EU(-i)
  - The *expected value of information* associated with inspection is 55 (it improves expected utility by this amount ignoring cost of inspection). How? Gives opportunity to change decision (¬buy if bad).
  - You should be willing to pay up to $55 for the report