CSC384: Lecture 12

- Last time
  - Variable elimination, Intro to decision theory
- Today
  - sequential decision problems; decision trees
- Readings:
  - Today: 10.4 (decision trees, decision networks)
  - Next week: wrap up
- Announcements:
  - none

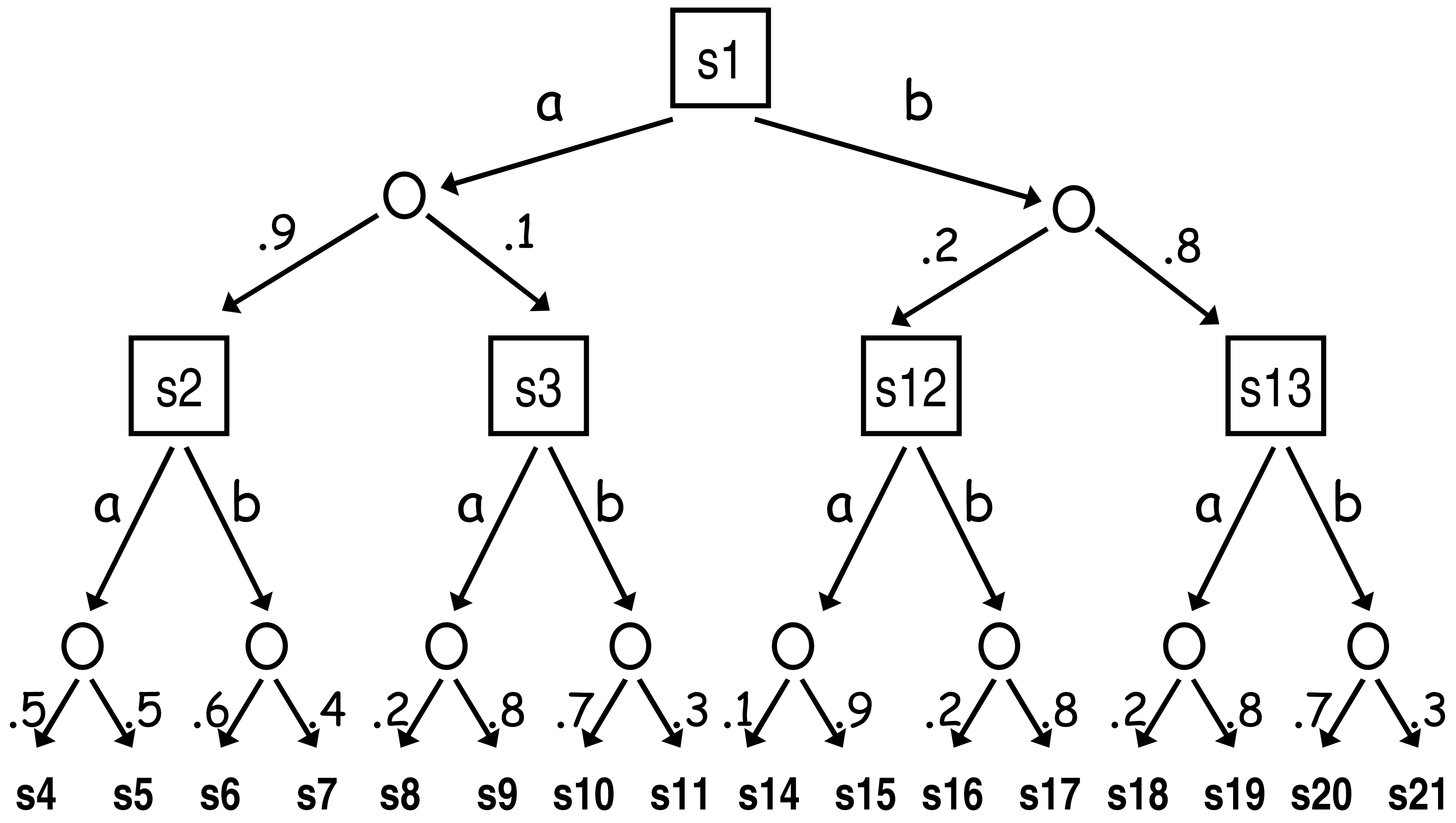
Decision Making under Uncertainty

- We saw expected utility can be quite useful
  - allows one to tradeoff outcome probabilities with their relative desirability to help make decisions
  - but formulation so far for "single shot" decisions
- Decision space is often quite large
  - they involve sequential choices (like plans)
  - if we treat each plan as a distinct decision, decision space is too large to handle directly
  - Soln: use dynamic programming methods to *construct* optimal plans (actually generalizations of plans, called policies… like in game trees)

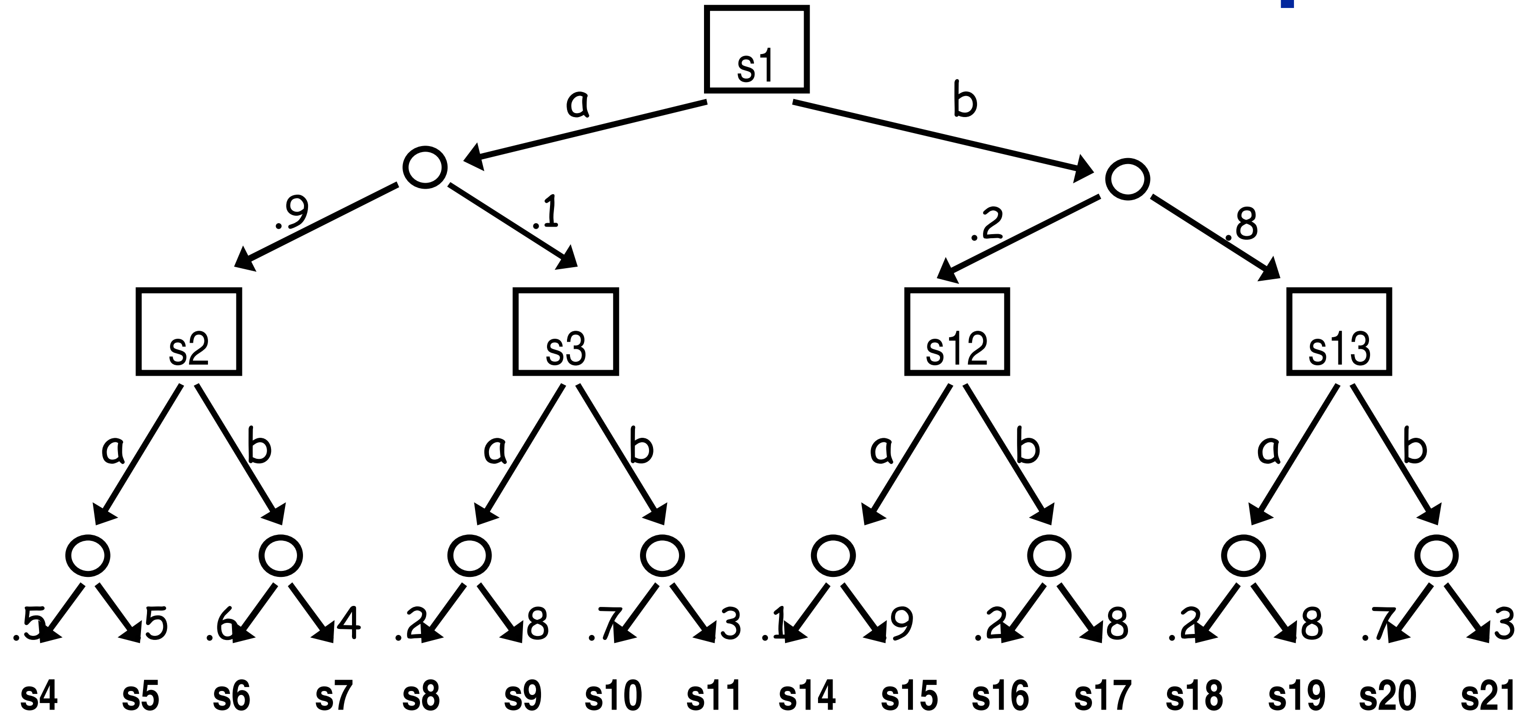
An Simple Example

- 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

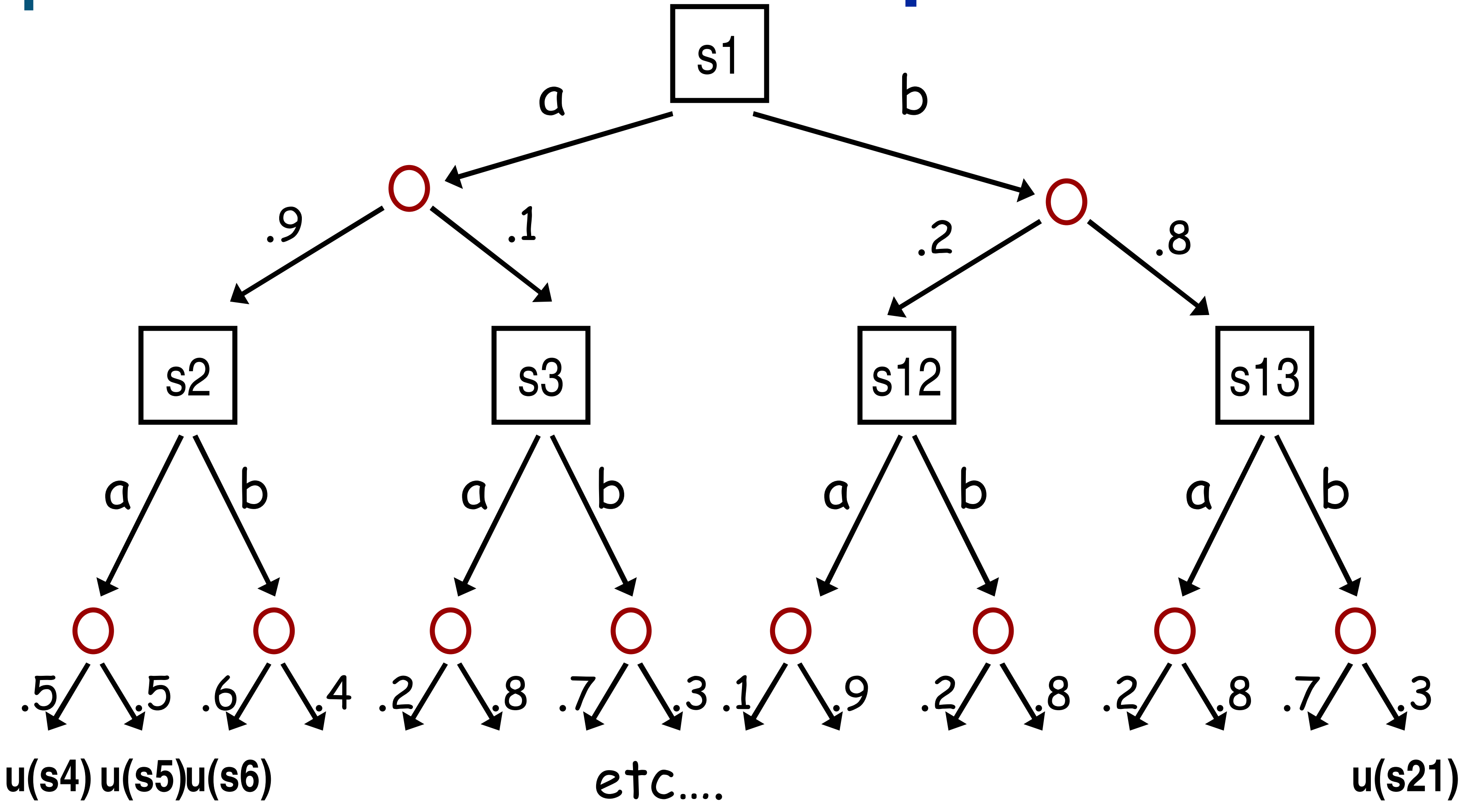


- Sequence [a,a] gives distribution over "final states"
  - Pr(s4) = .45, Pr(s5) = .45, Pr(s8) = .02, Pr(s9) = .08
- Similarly:
  - [a,b]: Pr(s6) = .54, Pr(s7) = .36, Pr(s10) = .07, Pr(s11) = .03
  - and similar distributions for sequences [b,a] and [b,b]

How Good is a Sequence?

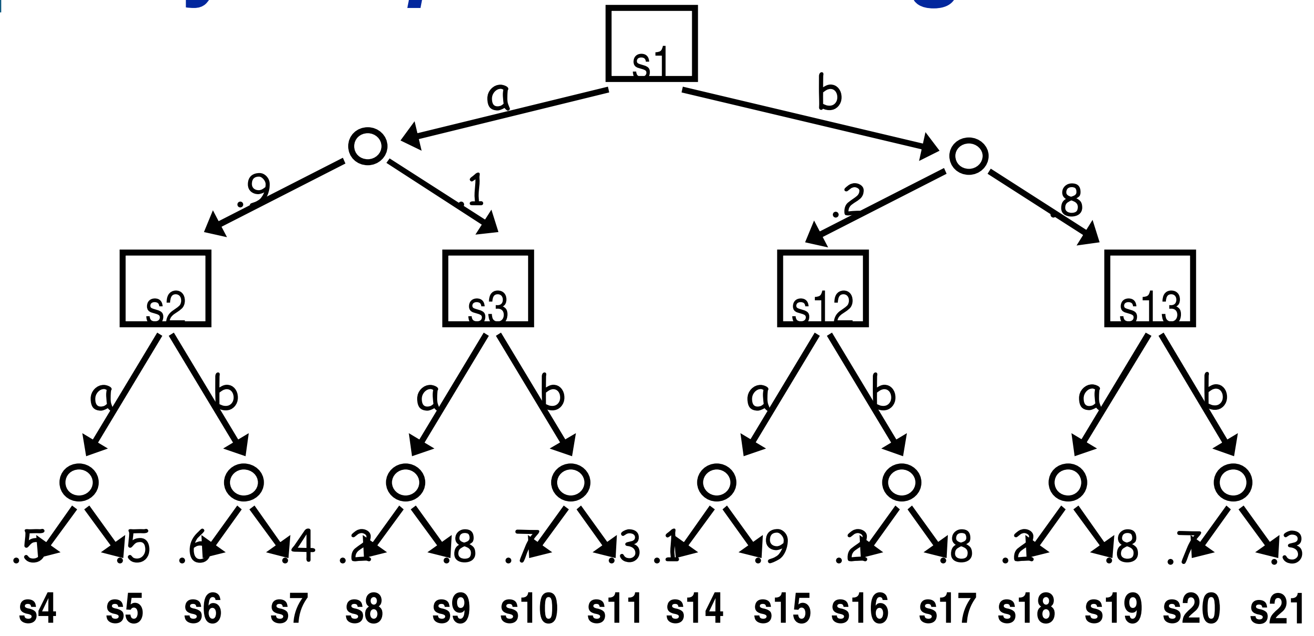
- We associate *utilities with the “final” outcomes*
  - how good is it to end up at s4, s5, s6, …
  - note: we could assign utilities to the intermediate states s2, s3, s12, and s13 also. We ignore this for now. Technically, think if utility u(s4) as utility of entire *trajectory* or sequence of states we pass through.
- 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)

Why *Sequences* might be bad



- 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 we want to consider *policies*, **not** 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]

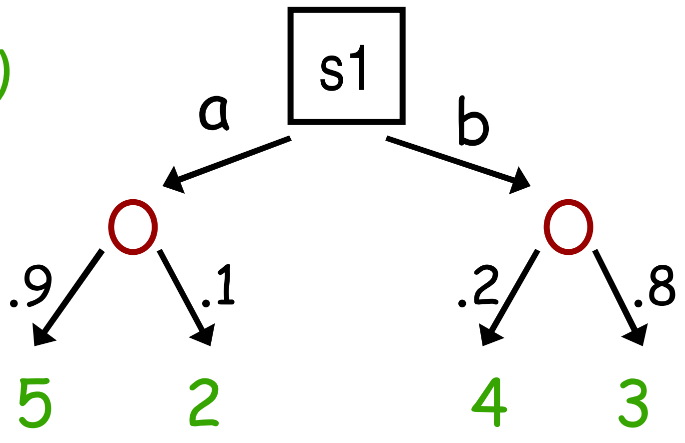
- Contrast this with four "plans"
  - [a; a], [a; b], [b; a], [b; b]
  - note: each plan corresponds to a policy, so we can only *gain* by allowing decision maker to use policies

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 much larger
  - if we have $n=|A|$ actions and $m=|O|$ outcomes per action, then we have $(nm)^k$ 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 "trajectory" (branch) to decision maker

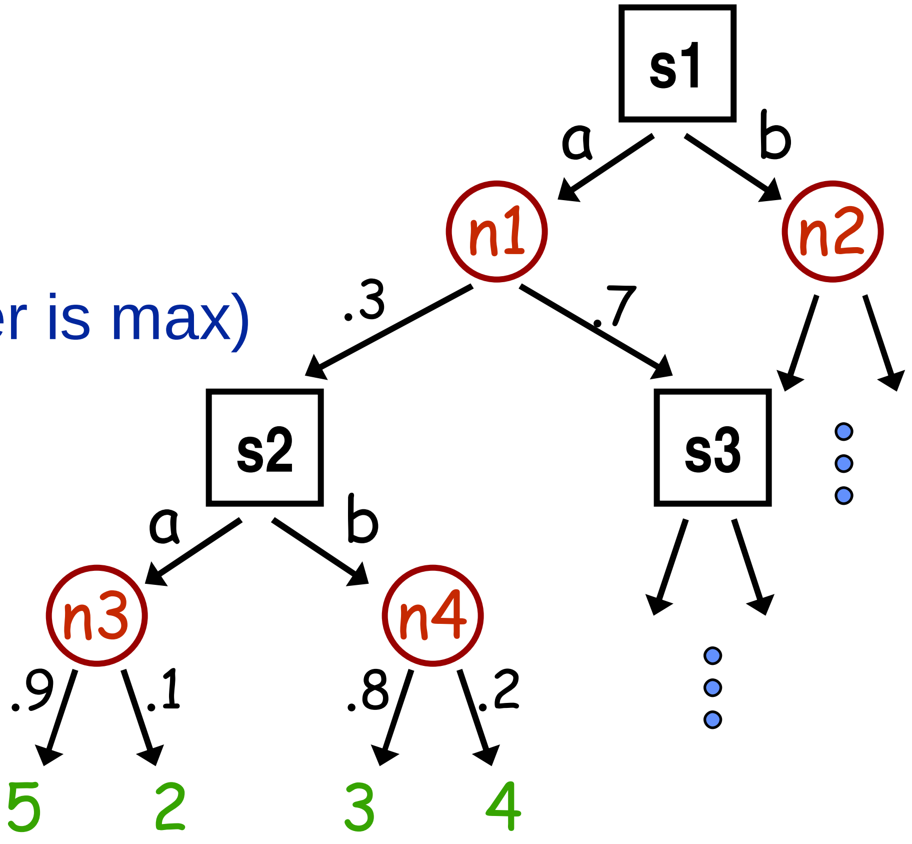


Evaluating Decision Trees

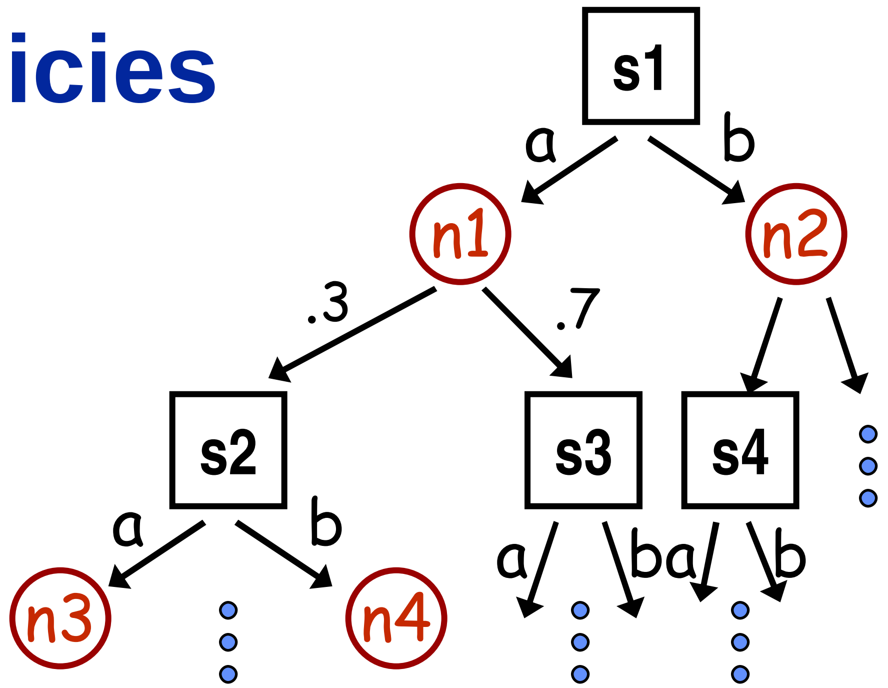
- Procedure is exactly like game trees, except…
  - key difference: the "opponent" is "nature" who simply chooses outcomes at chance nodes with specified probability: so we average instead on minimizing
- Back values *up* the tree
  - *U(t)* is defined for all terminals (part of input)
  - *U(n)* = avg {*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*

Evaluating a Decision Tree

- U(n3) = .9*5 + .1*2
- U(n4) = .8*3 + .2*4
- U(s2) = max{U(n3), U(n4)}
  - decision a or b (whichever is max)
- U(n1) = .3U(s2) + .7U(s3)
- U(s1) = max{U(n1), U(n2)}
  - decision: max of a, b



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 tree

- Some policies can't be distinguished in terms of there expected values
  - e.g., if policy chooses a at node s1, choice at s4 doesn't matter because it won't be reached
  - Two policies are *implementationally indistinguishable* if they disagree only at unreachable decision nodes
    - reachability is determined by policy themselves

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 <u>any chance node that is followed by a decision node</u>*
  - 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 ations)
- Note: we don't need to resolve the uncertainty at a chance node if no decision follows it
  - no future choice depends on outcome (only utility)

Computational Issues

- Savings compared to explicit policy evaluation is substantial
- Evaluate only $O((nm)^d)$ nodes in tree of depth d
  - total computational cost is thus $O((nm)^d)$
- Note that this is how many ***policies*** there are
  - but evaluating a single policy explicitly requires substantial computation: $O(nm^d)$
  - total computation for explicity evaluating each policy would be $O(n^d m^{2d})$!!!
- Tremendous value to dynamic programming solution

Computational Issues

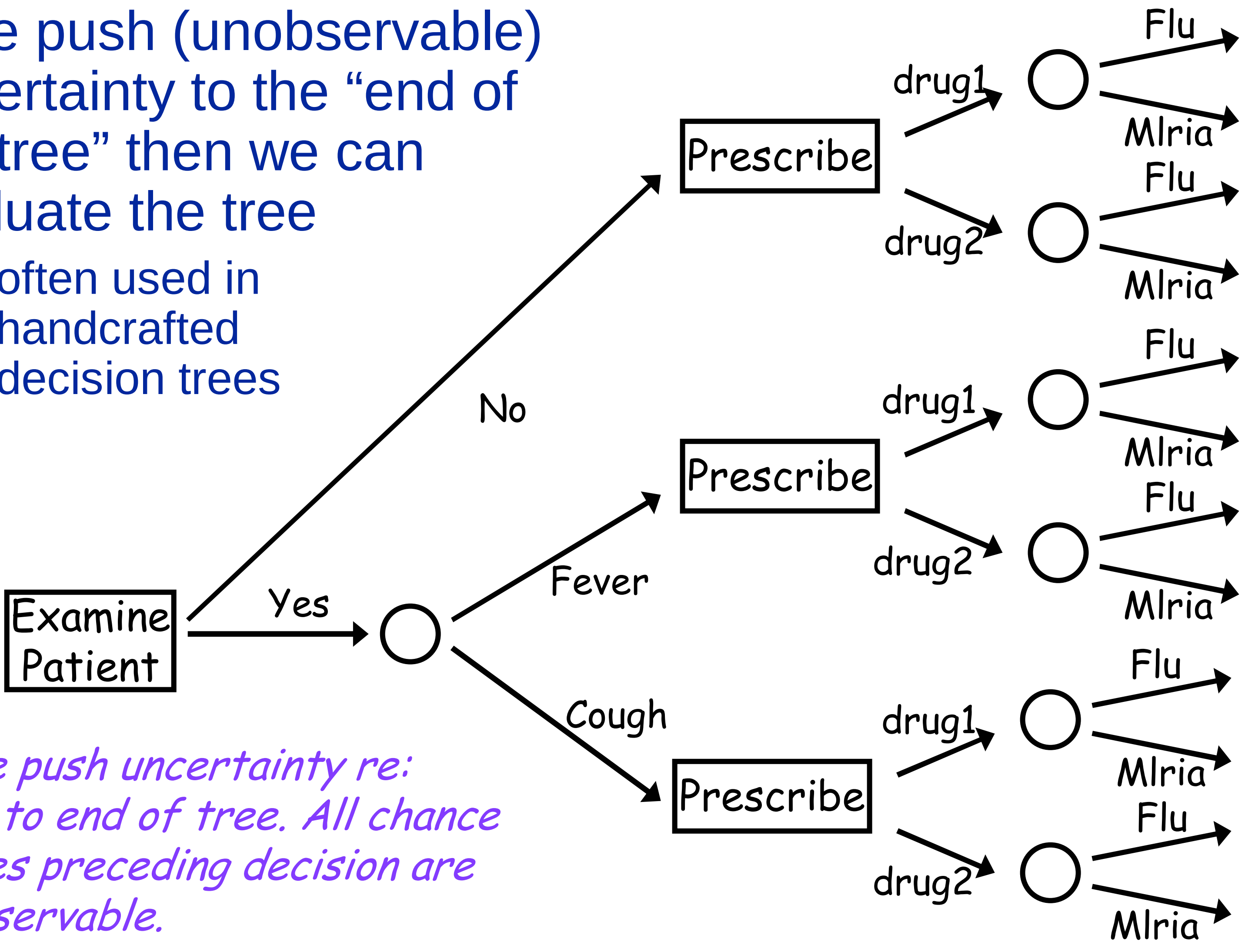
- **Tree size:** grows exponentially with depth
- Possible solutions:
  - bounded lookahead with heuristics (like game trees)
  - heuristic search procedures (like A*)
- **Full observability:** we must know the initial state and outcome of each action
- Possible solutions:
  - handcrafted decision trees for certain initial state uncertainty
  - more general policies based on *observations*

Other Issues

- **Specification:** suppose each state is an assignment to variables; then representing action probability distributions is complex (and branching factor could be immense)
- Possible solutions:
  - represent distribution using Bayes nets
  - solve problems using *decision networks* (or influence diagrams)

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.

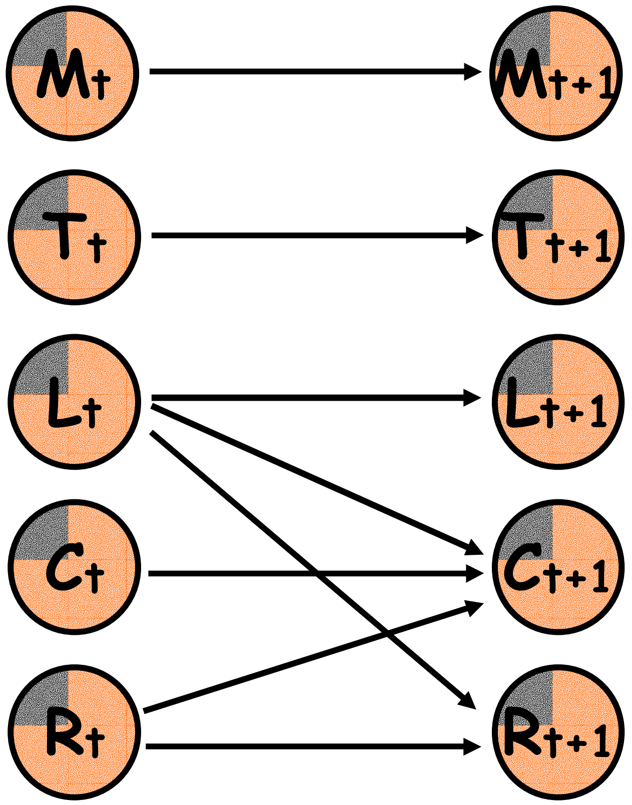
Large State Spaces (Variables)

- To represent outcomes of actions or decisions, we need to specify distributions
  - Pr(s|d) : probability of outcome s given decision d
  - Pr(s|a,s'): prob. of state s given that action a performed in state s'
- But state space exponential in # of variables
  - spelling out distributions explicitly is intractable
- Bayes nets can be used to represent actions
  - this is just a joint distribution over variables, conditioned on action/decision and previous state

Example Action using Dynamic BN

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

Deliver Coffee action



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

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

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

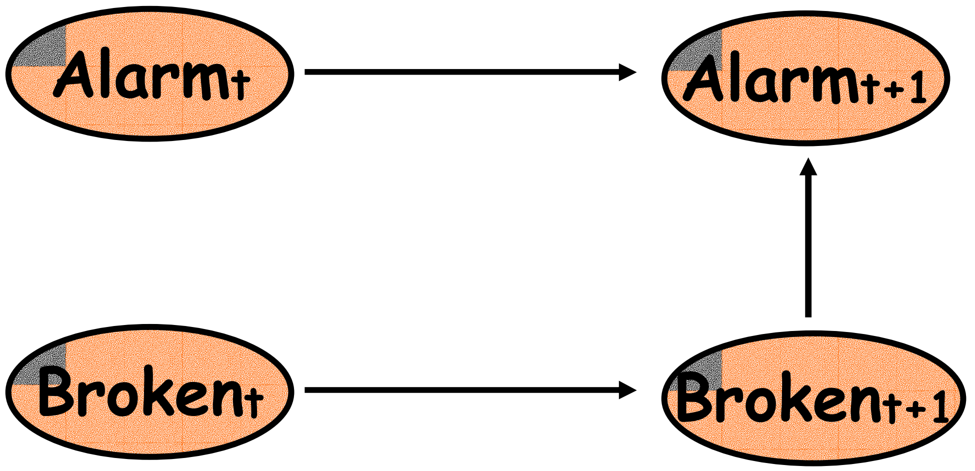
$f_R(L_t, R_t, C_t, C_{t+1})$

Dynamic BN Action Representation

- Dynamic Bayesian networks (DBNs):
  - a way to use BNs to represent *specific* actions
  - 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
    - these can include time t and time t+1 variables
    - network must be acyclic though
  - specify CPT for each time t+1 variable
- Note: generally *no prior given* for time t variables
  - we're (generally) interested in *conditional* distribution over post-action states given pre-action state
  - so time t vars are instantiated as "evidence" when using a DBN (generally)

Example of Dependence within Slice

Throw rock at window action



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

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

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.

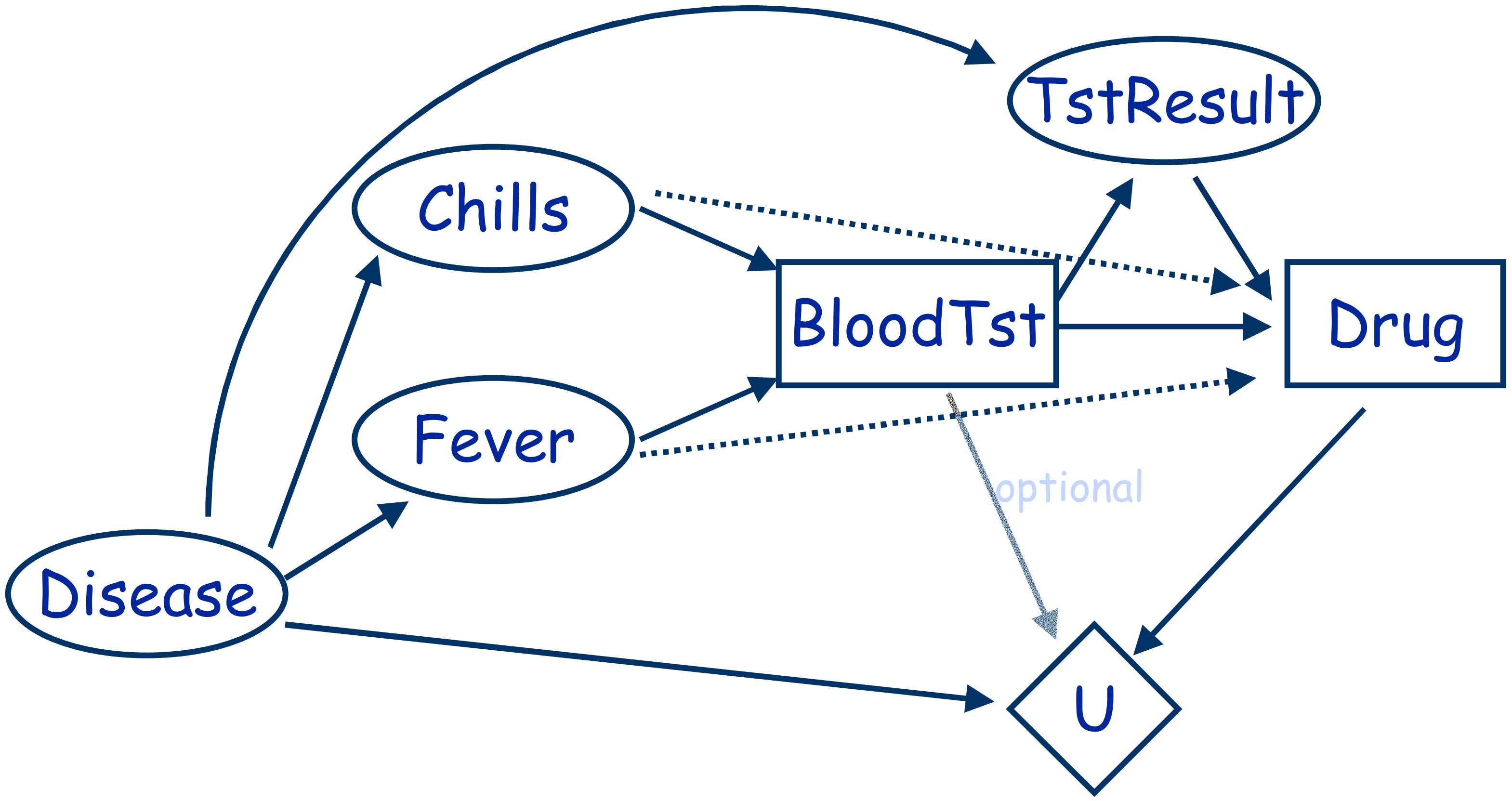
Use of BN Action Reprsnt’n

- DBNs: actions concisely,naturally specified
  - These look a bit like STRIPS and the situtation calculus, but allow for probabilistic effects
- How to use:
  - use to generate “expectimax” search tree to solve decision problems
  - use directly in stochastic decision making algorithms
- First use doesn’t buy us much computationally when solving decision problems. But second use allows us to compute expected utilities without enumerating the outcome space (tree)
  - well see something like this with *decision networks*

Decision Networks

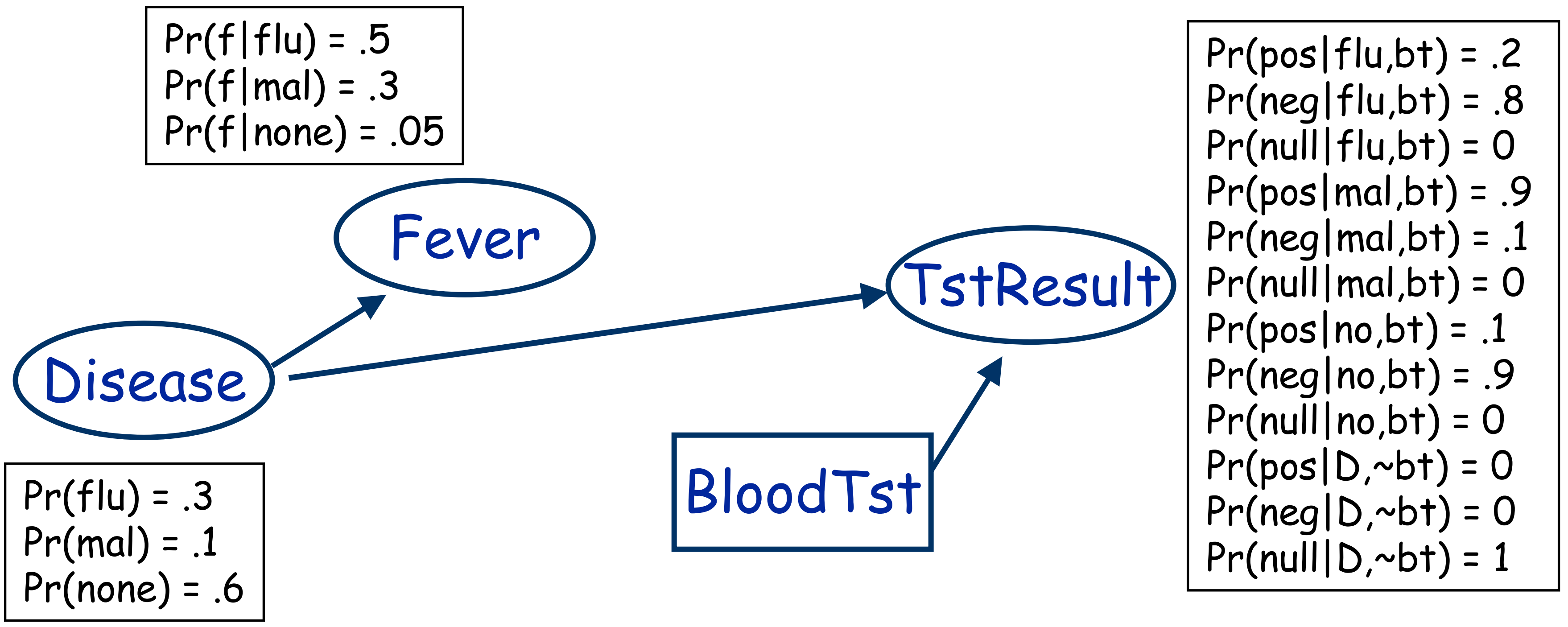
- *Decision networks* (more commonly known as *influence diagrams*) provide a way of representing sequential decision problems
  - basic idea: represent the variables in the problem as you would in a BN
  - add decision variables – variables that you “control”
  - add utility variables – how good different states are

Sample Decision Network



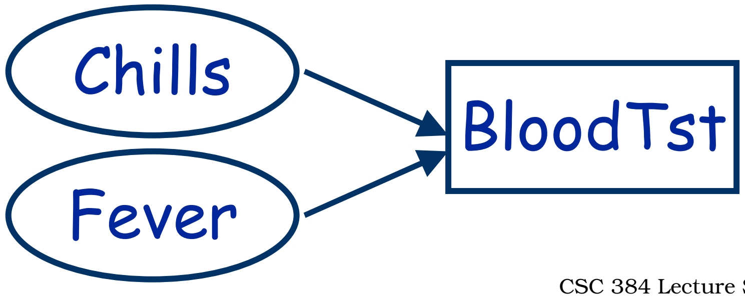
Decision Networks: Chance Nodes

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



Decision Networks: Decision Nodes

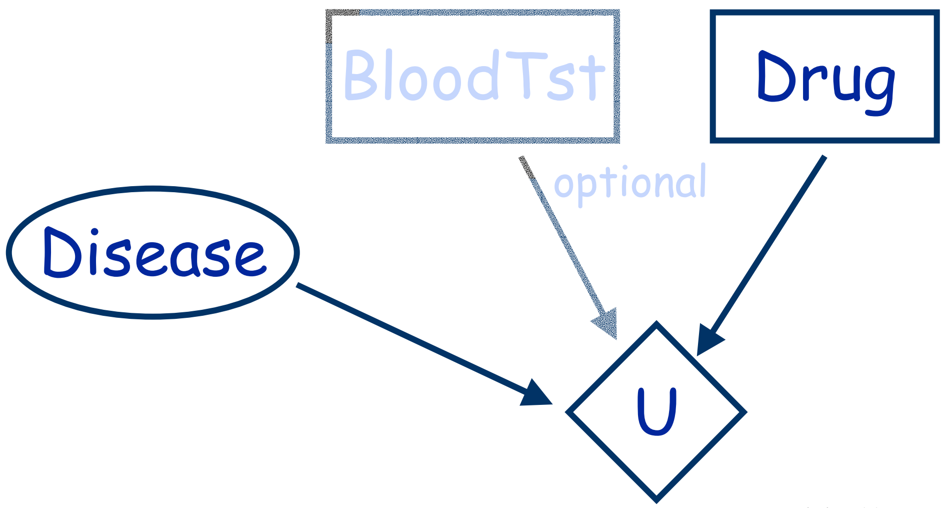
- **Decision nodes**
  - variables decision maker sets, denoted by squares
  - parents reflect *information available* at time decision is to be made
- In example decision node: the actual values of Ch and Fev will be observed before the decision to take test must be made
  - agent can make *different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



$BT \in \{bt, \sim bt\}$

Decision Networks: Value Node

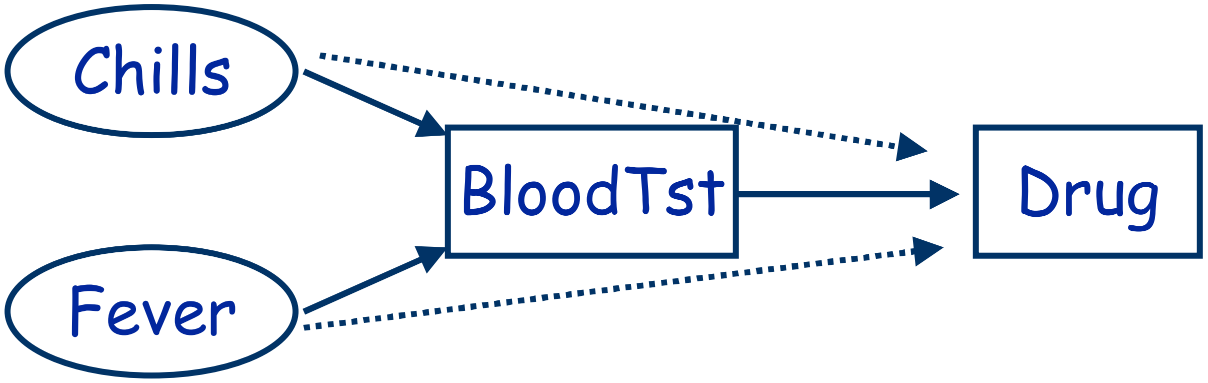
- **Value node**
  - specifies utility of a state, denoted by a diamond
  - utility depends *only on state of parents* of value node
  - generally: only one value node in a decision network
- Utility depends only on disease and drug



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

Decision Networks: Assumptions

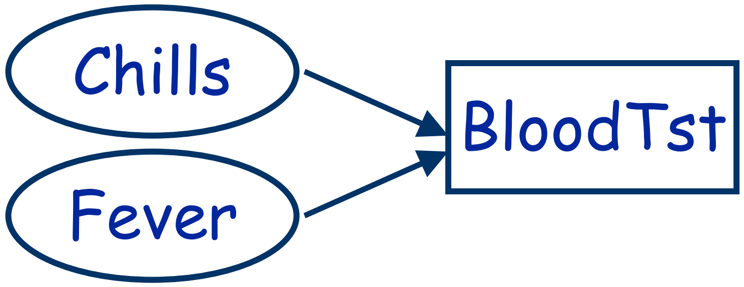
- Decision nodes are totally ordered
  - decision variables $D_1, D_2, \ldots, D_n$
  - decisions are made in sequence
  - e.g., BloodTst (yes,no) decided before Drug (fd,md,no)
- *No-forgetting property*
  - any information available when decision D_i is made is available when decision D_j is made (for $i < j$)
  - thus all parents of D_i are parents of D_j



Dashed arcs ensure the no-forgetting property

Policies

- Let *Par(D$_i$)* be the parents of decision node *D$_i$*
  - *Dom(Par(D$_i$))* is the set of assignments to parents
- A policy δ is a set of mappings δ_i, one for each decision node *D$_i$*
  - $\delta_i : Dom(Par(D_i)) \rightarrow Dom(D_i)$
  - δ_i associates a decision with each parent asst for *D$_i$*
- For example, a policy for BT might be:
  - $\delta_{BT}(c,f) = bt$
  - $\delta_{BT}(c,\sim f) = \sim bt$
  - $\delta_{BT}(\sim c,f) = bt$
  - $\delta_{BT}(\sim c,\sim f) = \sim bt$



Value of a Policy

- *Value of a policy δ* is the expected utility given that decision nodes are executed according to δ
- Given asst **x** to the set **X** of all chance variables, let δ(**x**) denote the asst to decision variables dictated by δ
  - e.g., asst to D_1 determined by it's parents' asst in **x**
  - e.g., asst to D_2 determined by it's parents' asst in **x** along with whatever was assigned to D_1
  - etc.
- Value of δ :

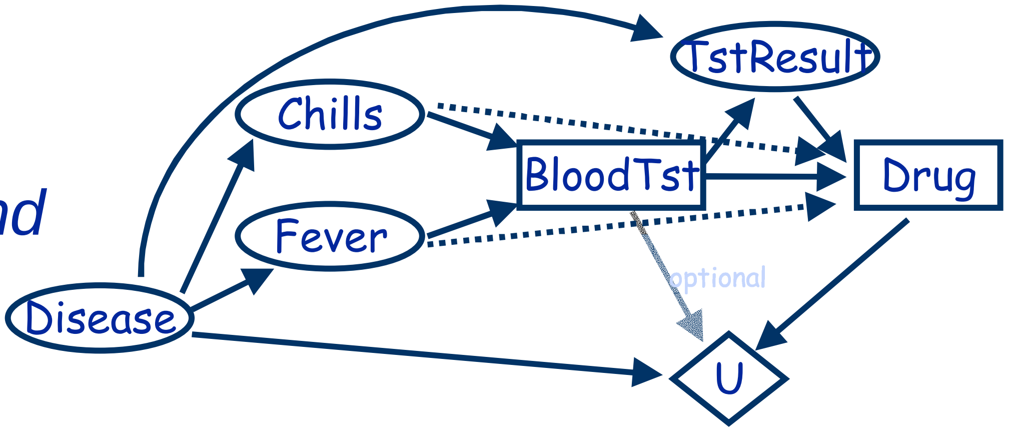
$$EU(\delta) = \Sigma_{\mathbf{X}}\, P(\mathbf{X}, \delta(\mathbf{X}))\, U(\mathbf{X}, \delta(\mathbf{X}))$$

Optimal Policies

- An *optimal policy* is a policy δ^* such that
  $EU(\delta^*) \geq EU(\delta)$ for all policies δ
- We can use the dynamic programming principle yet again to avoid enumerating all policies
- We can also use the structure of the decision network to use variable elimination to aid in the computation

Computing the Best Policy

- We can work backwards as follows
- First compute optimal policy for Drug (last dec'n)
  - for each asst to parents (C,F,BT,TR) and for each decision value (D = md,fd,none), *compute the expected value* of choosing that value of D
  - set policy choice for each value of parents to be the value of D that has max value
  - eg: $\delta_D(c,f,bt,pos) = md$

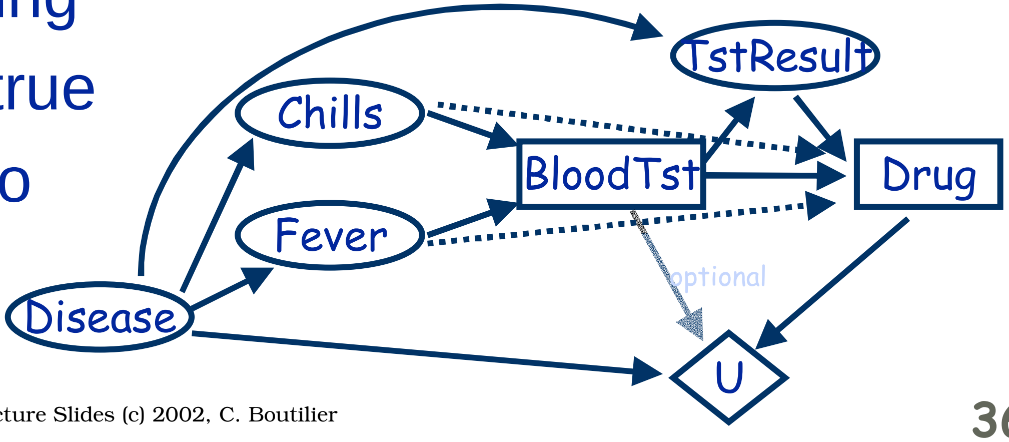


Computing the Best Policy

- Next compute policy for BT given policy $\delta_D(C,F,BT,TR)$ just determined for Drug
  - since $\delta_D(C,F,BT,TR)$ is fixed, we can treat Drug as a normal random variable with deterministic probabilities
  - i.e., for any instantiation of parents, value of Drug is fixed by policy δ_D
  - this means we can solve for optimal policy for BT just as before
  - only uninstantiated vars are random vars (once we fix *its* parents)

Computing the Best Policy

- How do we compute these expected values?
  - suppose we have asst <*c,f,bt,pos*> to parents of *Drug*
  - we want to compute EU of deciding to set *Drug = md*
  - we can run variable elimination!
- Treat *C,F,BT,TR,Dr* as evidence
  - this reduces factors (e.g., *U* restricted to *bt,md*: depends on *Dis*)
  - eliminate remaining variables (e.g., only *Disease* left)
  - left with factor: **$U() = \Sigma_{Dis} P(Dis| c,f,bt,pos,md)U(Dis)$**
- We now know EU of doing *Dr=md* when *c,f,bt,pos* true
- Can do same for *fd,no* to decide which is best



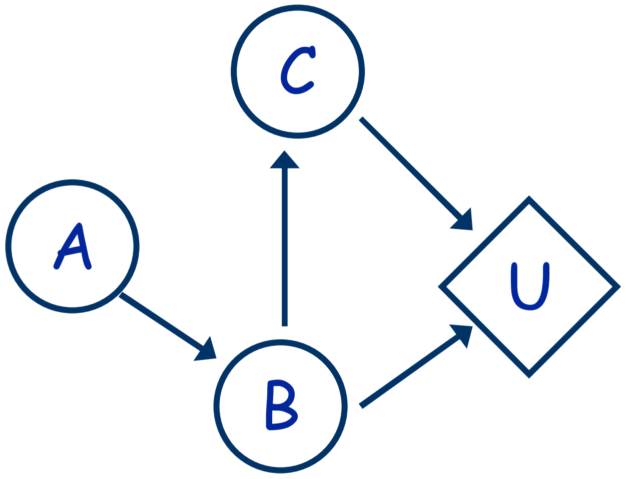
Computing Expected Utilities

- The preceding illustrates a general phenomenon
  - computing expected utilities with BNs is quite easy
  - utility nodes are just factors that can be dealt with using variable elimination

$$EU = \Sigma_{A,B,C} P(A,B,C)\, U(B,C)$$

$$= \Sigma_{A,B,C} P(C|B)\, P(B|A)\, P(A)\, U(B,C)$$

- Just eliminate variables in the usual way



Optimizing Policies: Key Points

- If a decision node D has no decisions that follow it, we can find its policy by instantiating each of its parents and computing the expected utility of each decision for each parent instantiation
  - no-forgetting means that all other decisions are instantiated (they must be parents)
  - its easy to compute the expected utility using VE
  - the number of computations is quite large: we run expected utility calculations (VE) for each parent instantiation together with each possible decision D might allow
  - policy: choose max decision for each parent instant’n

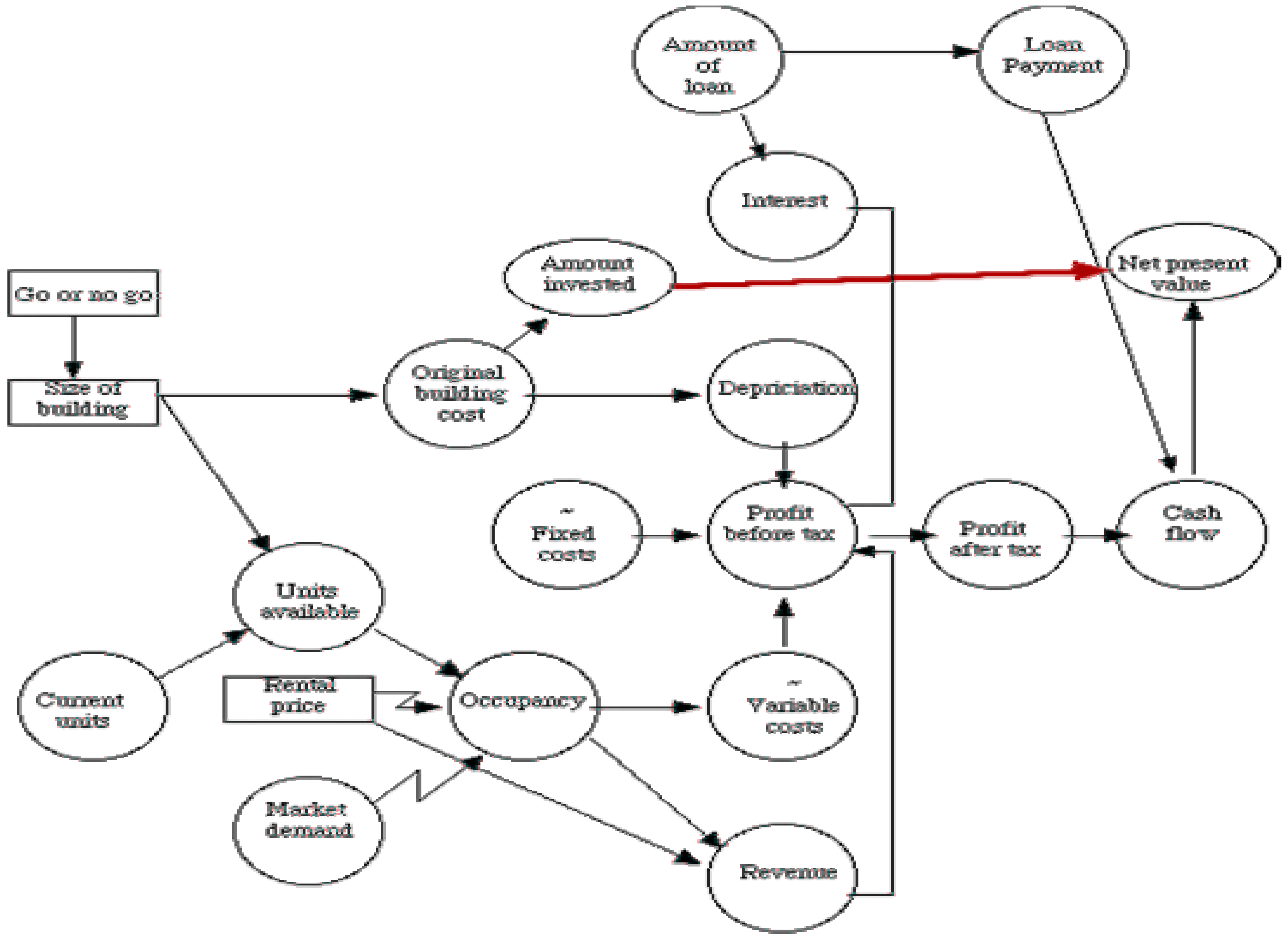
Optimizing Policies: Key Points

- When a decision D node is optimized, it can be treated as a random variable
  - for each instantiation of its parents we now know what value the decision should take
  - just treat policy as a new CPT: for a given parent instantiation **x**, D gets δ(**x**) with probability 1(all other decisions get probability zero)
- If we optimize from last decision to first, at each point we can optimize a specific decision by (a bunch of) simple VE calculations
  - it's successor decisions (optimized) are just normal nodes in the BNs (with CPTs)

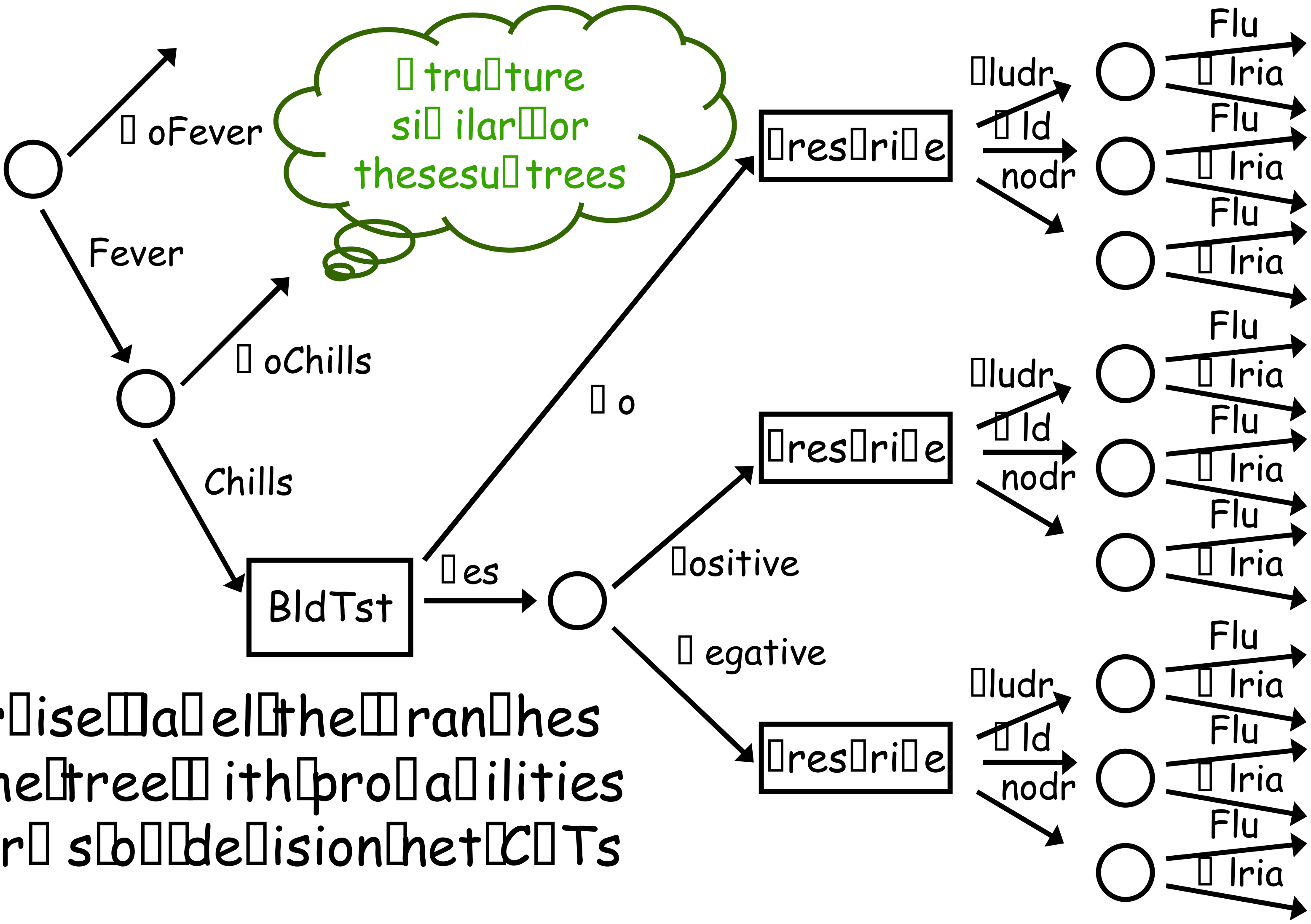
Decision Network Notes

- Decision networks commonly used by decision analysts to help structure decision problems
- Much work put into computationally effective techniques to solve these
  - common trick: replace the decision nodes with random variables at outset and solve a plain Bayes net (a subtle but useful transformation)
- 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

Real Estate Investment



Decision Tree for Medical Network

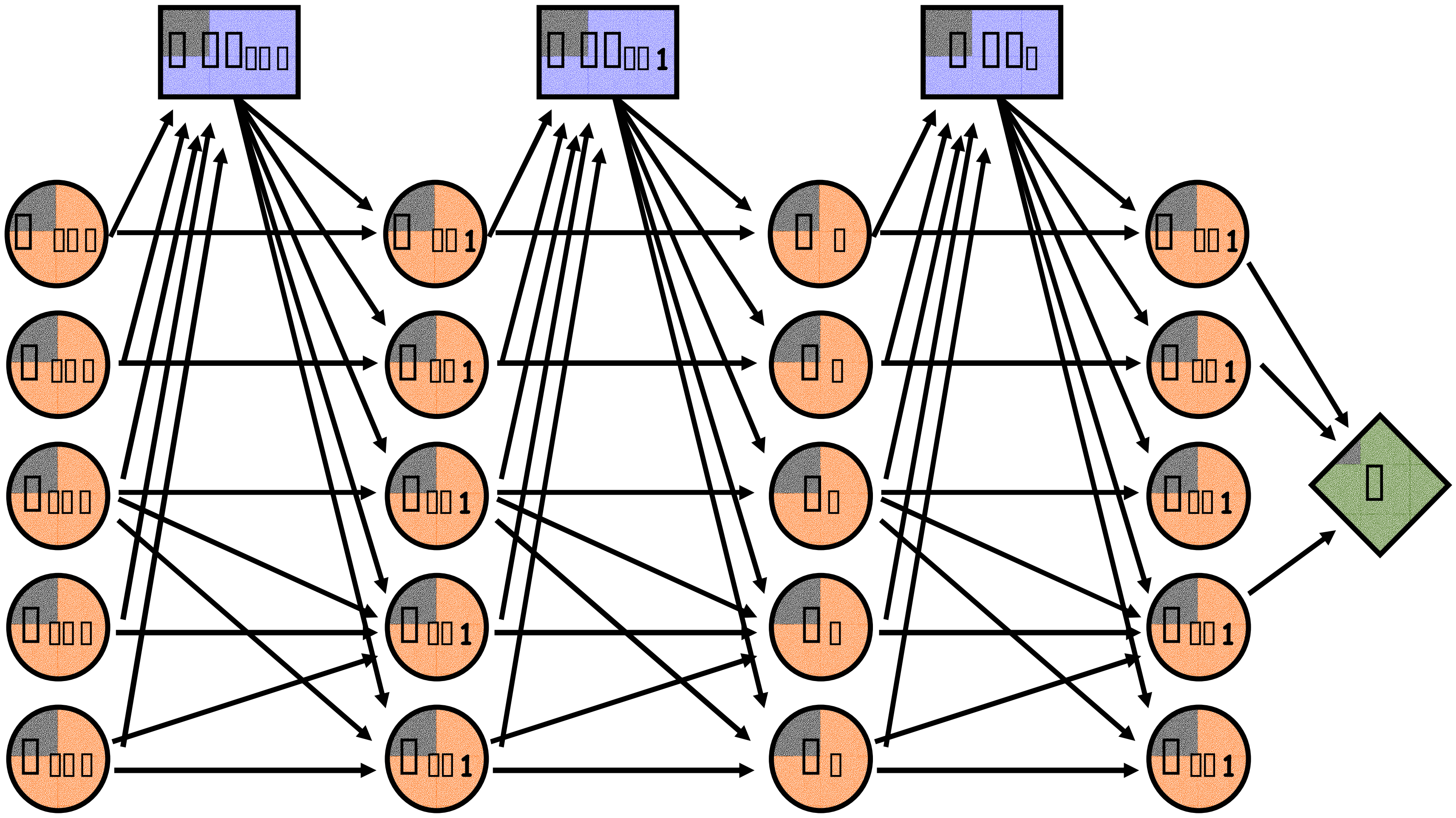


□ □er□ise□la□el□the□ran□hes o□□the□tree□ ith□pro□a□ilities in□ter□ s□b□□de□ision□het□C□Ts

Building Decision Tree from Netw’k

- Structure of decision tree is straightforward
  - order decisions as in the network
  - ensure observed chance nodes are in the tree before the decision that uses them
  - label leaves with utilities dictated by the utility node (using the domain values assigned to the to the utility nodes parents on that branch)
  - assign probabilities to outcomes (chance nodes in the tree) using the conditional probabilities of those outcomes *given* the observed variables and decisions that precede it on that branch of the decision tree

DBN-Decision Nets for Planning



DBN Decision Networks

- In example on previous slide:
  - we assume the state (of the variables at any stage) is fully observable
    - hence all time t vars point to time t decision
  - this means the state at time t d-separates the decision at time t-1 from the decision at time t-2
  - so we ignore "no-forgetting" arcs between decisions
    - once you *know* the state at time t, what you *did* at time t-1 to get there is irrelevant to the decision at time t-1
- If the state were not fully observable, we could not ignore the "no-forgetting" arcs