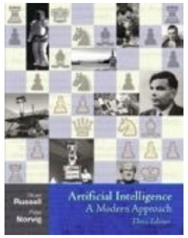
Bayesian Networks

- Syntax
- Semantics
- Parametrized Distributions
- Inference in Bayesian Networks
  - Exact Inference
    - enumeration
    - variable elimination
  - Approximate Inference
    - stochastic simulation
    - Markov Chain Monte Carlo (MCMC)



Many slides based on Russell & Norvig's slides [Artificial Intelligence: A Modern Approach]

Inference Tasks

- **Simple queries**
  - compute the posterior marginal distribution for a variable
- **Conjunctive queries**
  - compute the posterior for a conjunction of variables

$$\mathbf{P}(X_i, X_j \mid \mathbf{E}=\mathrm{e}) = \mathbf{P}(X_i \mid \mathbf{E}=\mathrm{e}) \cdot \mathbf{P}(X_j \mid X_i, \mathbf{E}=\mathrm{e})$$

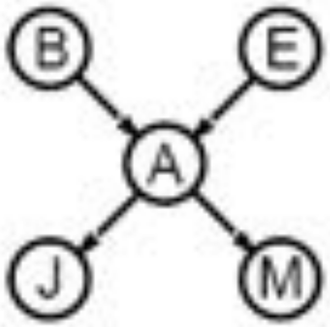
- **Optimal decisions**
  - decision networks include utility information
  - probabilistic inference required for $P(outcome \mid action, evidence)$
- **Value of Information**
  - Which evidence to seek next?
- **Sensitivity Analysis**
  - Which probability values are most critical?
- **Explanation**
  - Why do I need a new starter motor?

Inference by Enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

$\mathbf{P}(B|j,m)$
$= \mathbf{P}(B,j,m)/P(j,m)$
$= \alpha\mathbf{P}(B,j,m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B,e,a,j,m)$



Rewrite full joint entries using product of CPT entries:

$\mathbf{P}(B|j,m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B)P(e)\mathbf{P}(a|B,e)P(j|a)P(m|a)$
$= \alpha\mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a|B,e)P(j|a)P(m|a)$

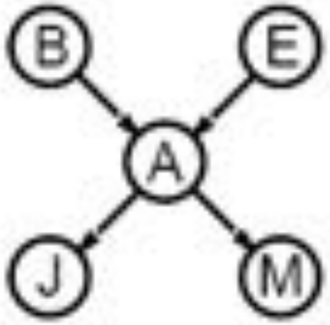
Inference by Enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

$\mathbf{P}(B|j,m)$
$= \mathbf{P}(B,j,m)/P(j,m)$
$= \alpha\mathbf{P}(B,j,m)$
$= \alpha\sum_e\sum_a \mathbf{P}(B,e,a,j,m)$

Worst case: $O(n\ d^n)$ time
$O(d^n)$ terms, each consisting of a product of $O(n)$ probabilities



Rewrite full joint entries using product of CPT entries:

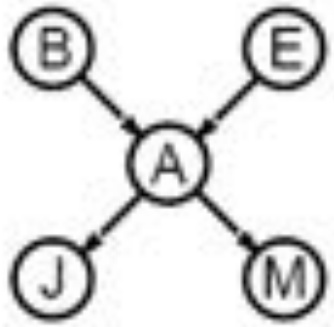
$\mathbf{P}(B|j,m)$
$= \alpha\sum_e\sum_a \mathbf{P}(B)P(e)\mathbf{P}(a|B,e)P(j|a)P(m|a)$
$= \alpha\mathbf{P}(B)\sum_e P(e)\sum_a \mathbf{P}(a|B,e)P(j|a)P(m|a)$

Inference by Enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

$\mathbf{P}(B|j, m)$
$= \mathbf{P}(B, j, m)/P(j, m)$
$= \alpha \mathbf{P}(B, j, m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B, e, a, j, m)$



Worst case: $O(n\ d^n)$ time
$O(d^n)$ terms, each consisting of a product of $O(n)$ probabilities

Rewrite full joint entries using product of CPT entries:

$\mathbf{P}(B|j, m)$
$= \alpha \sum_e \sum_a \mathbf{P}(B)P(e)\mathbf{P}(a|B, e)P(j|a)P(m|a)$
$= \alpha \mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a|B, e)P(j|a)P(m|a)$

Recursive depth-first enumeration: $O(n)$ space, $O(d^n)$ time

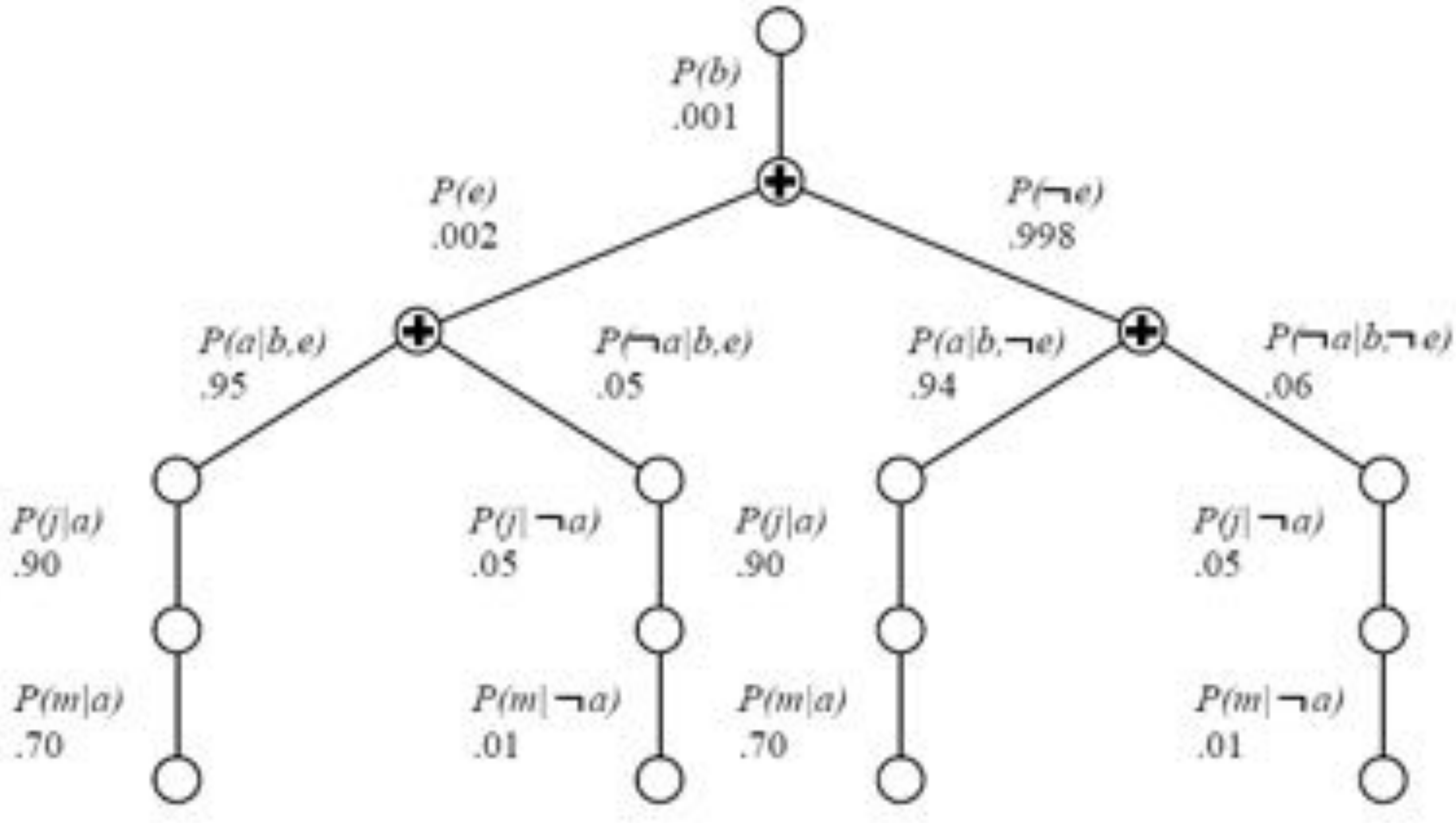
where n is the number of variables and d is the number of values per variable

Enumeration Algorithm

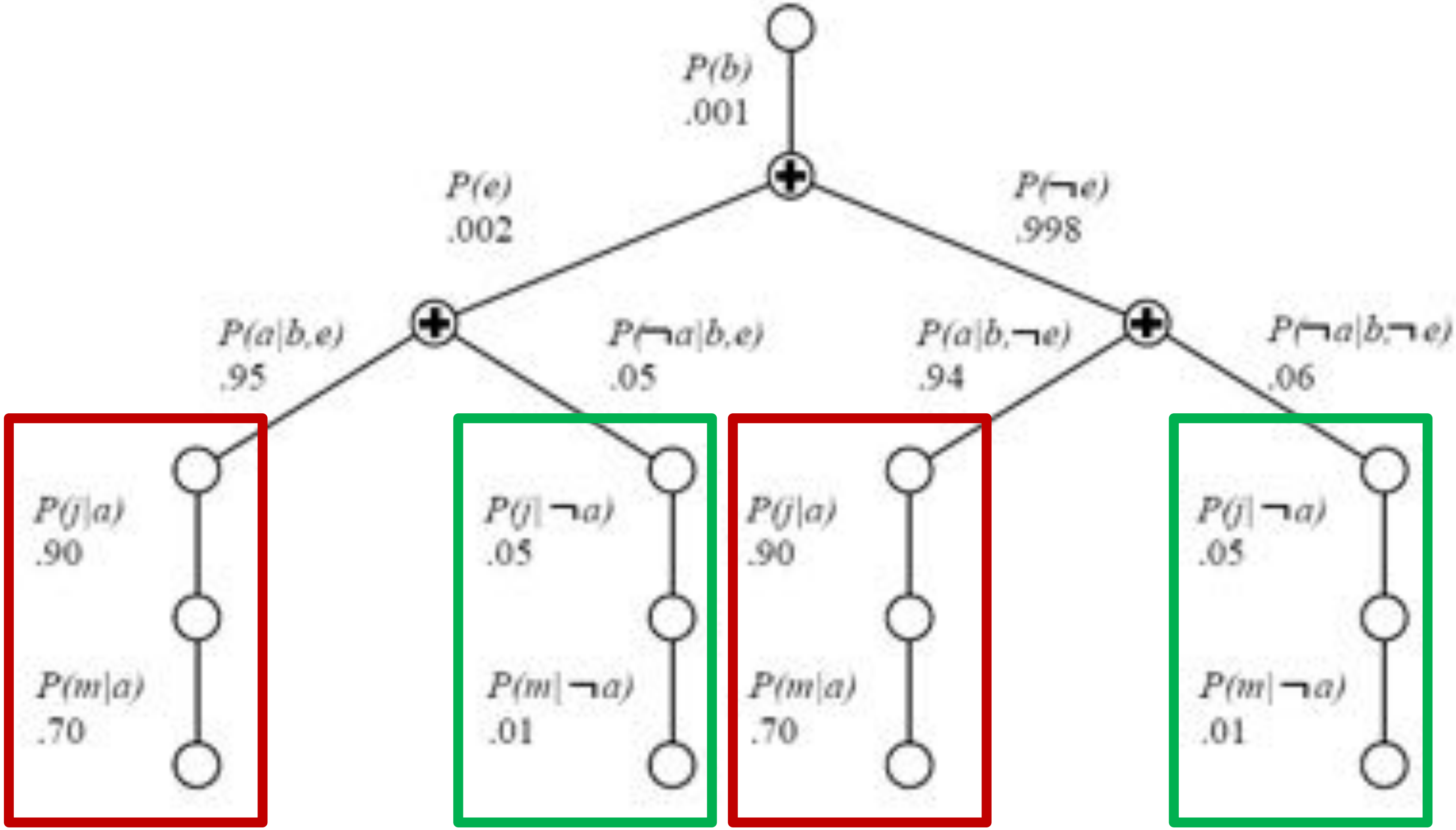
```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
   inputs: X, the query variable
           e, observed values for variables E
           bn, a Bayesian network with variables {X} ∪ E ∪ Y

   Q(X) ← a distribution over X, initially empty
   for each value x_i of X do
       extend e with value x_i for X
       Q(x_i) ← ENUMERATE-ALL(VARS[bn], e)
   return NORMALIZE(Q(X))
```

```
function ENUMERATE-ALL(vars, e) returns a real number
   if EMPTY?(vars) then return 1.0
   Y ← FIRST(vars)
   if Y has value y in e
       then return P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e)
       else return Σ_y P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e_y)
           where e_y is e extended with Y = y
```

Evaluation Tree



Evaluation Tree



Enumeration is inefficient: repeated computation
e.g., computes $P(j|a)P(m|a)$ for each value of e

Variable Elimination

Move the sums into the products

- Key idea:
  - Do not multiply left-to-right but right-to-left.
  - Thus, terms that appear inside sums are evaluated first
    - intermediate results are stored as so-called **factors**
    - factors can be re-used several times in the same computation
  - is a form of **dynamic programming**

Variable Elimination

Move the sums into the products

- Key idea:
  - Do not multiply left-to-right but right-to-left.
  - Thus, terms that appear inside sums are evaluated first
    - intermediate results are stored as so-called **factors**
    - factors can be re-used several times in the same computation
  - is a form of **dynamic programming**
- Example: $\mathbf{P}(B|j,m)$

$$
\begin{aligned}
&= \alpha \underbrace{\mathbf{P}(B)}_{B} \textstyle\sum_e \underbrace{P(e)}_{E} \textstyle\sum_a \underbrace{\mathbf{P}(a|B,e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M} \\
&= \alpha \mathbf{P}(B) \textstyle\sum_e P(e) \textstyle\sum_a \mathbf{P}(a|B,e) P(j|a) f_M(a) \\
&= \alpha \mathbf{P}(B) \textstyle\sum_e P(e) \textstyle\sum_a \mathbf{P}(a|B,e) f_J(a) f_M(a) \\
&= \alpha \mathbf{P}(B) \textstyle\sum_e P(e) \textstyle\sum_a f_A(a,b,e) f_J(a) f_M(a) \\
&= \alpha \mathbf{P}(B) \textstyle\sum_e P(e) f_{\bar{A}JM}(b,e) \text{ (sum out } A\text{)} \\
&= \alpha \mathbf{P}(B) f_{E\bar{A}JM}(b) \text{ (sum out } E\text{)} \\
&= \alpha f_B(b) \times f_{E\bar{A}JM}(b)
\end{aligned}
$$

Factors

- A **factor** is a vector / matrix containing all probabilities for all dependent variables
- Examples:
  - $\mathbf{f}_M(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix}$
  - The factor $\mathbf{f}_A(A,B,E)$ is a 2 x 2 x 2 matrix

Basic Operations

- **Summing Out** a variable from a product of factors
  - move all constant factors outside of the summation
  - add up submatrices in pointwise product of remaining factors

$$\sum_x \mathbf{f}_1 \times \ldots \times \mathbf{f}_k = \mathbf{f}_1 \times \ldots \times \mathbf{f}_i \times \sum_x \mathbf{f}_{i\;1} \times \ldots \times \mathbf{f}_k$$

$$= \mathbf{f}_1 \times \ldots \times \mathbf{f}_i \times \mathbf{f}_X$$

assuming $\mathbf{f}_1, \ldots, \mathbf{f}_i$ do not depend on X

Basic Operations

- **Summing Out** a variable from a product of factors
  - move all constant factors outside of the summation
  - add up submatrices in pointwise product of remaining factors

$$\sum_x \mathbf{f}_1 \times \ldots \times \mathbf{f}_k = \mathbf{f}_1 \times \ldots \times \mathbf{f}_i \times \sum_x \mathbf{f}_{i+1} \times \ldots \times \mathbf{f}_k$$
$$= \mathbf{f}_1 \times \ldots \times \mathbf{f}_i \times \mathbf{f}_X$$

assuming $\mathbf{f}_1, \ldots, \mathbf{f}_i$ do not depend on X

- **Pointwise Product** of factors $\mathbf{f}_1$ and $\mathbf{f}_2$
  - for example: $\mathbf{f}_1(A,B) \times \mathbf{f}_2(B,C) = \mathbf{f}(A,B,C)$
  - in general: $\mathbf{f}_1(X_1,\ldots,X_j,Y_1,\ldots,Y_k) \times \mathbf{f}_2(Y_1,\ldots,Y_k,Z_1,\ldots,Z_l) = \mathbf{f}(X_1,\ldots,X_j,Y_1,\ldots,Y_k,Z_1,\ldots,Z_l)$
    - has 2^{j+k+l} entries (if all variables are binary)

Example: Pointwise Product

p	q	$f_1(p,q)$
T	T	0.1
T	F	0.3
F	T	0.5
F	F	0.7

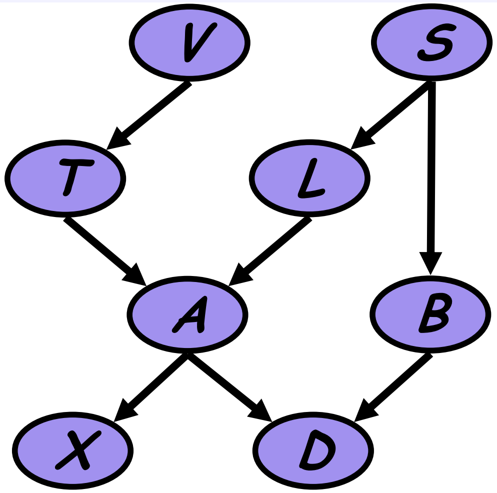
q	r	$f_2(q,r)$
T	T	0.2
T	F	0.4
F	T	0.6
F	F	0.8

×

p	q	r	$f_1(p, q)$	$f_2(q, r)$	pointwise product
T	T	T	0.1	0.2	0.1 x 0.2
T	T	F	0.1	0.4	0.1 x 0.4
T	F	T	0.3	0.6	0.3 x 0.6
T	F	F	0.3	0.8	0.3 x 0.8
F	T	T	0.5	0.2	0.5 x 0.2
F	T	F	0.5	0.4	0.5 x 0.4
F	F	T	0.7	0.6	0.7 x 0.6
F	F	F	0.7	0.8	0.7 x 0.8

Variable Elimination Algorithm

```
function ELIMINATION-ASK(X, e, bn) returns a distribution over X
    inputs: X, the query variable
            e, evidence specified as an event
            bn, a belief network specifying joint distribution P(X_1, ..., X_n)
    factors ← [ ]; vars ← REVERSE(VARS[bn])
    for each var in vars do
        factors ← [MAKE-FACTOR(var, e)|factors]
        if var is a hidden variable then factors ← SUM-OUT(var, factors)
    return NORMALIZE(POINTWISE-PRODUCT(factors))
```



We want to compute *P(d)*
Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

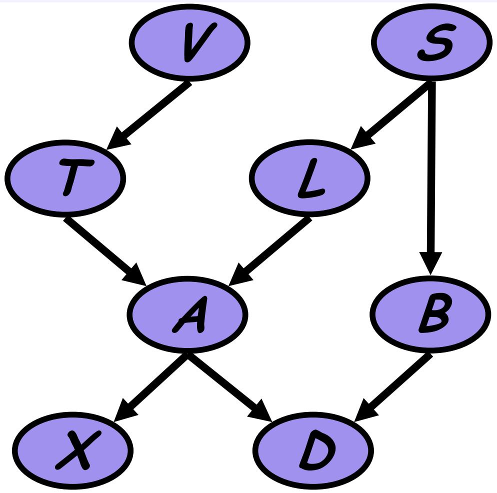
$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

Eliminate: *v*

Compute: $f_v(t) = \sum_v P(v)P(t|v)$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

Note: $f_v(t) = P(t)$
In general, result of elimination is not necessarily a probability term



We want to compute *P(d)*

Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

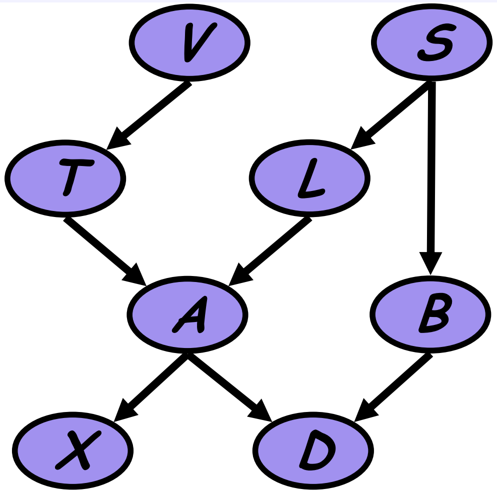
Eliminate: *s*

Compute: $f_s(b,l) = \sum_s P(s)P(b|s)P(l|s)$

$$\Rightarrow f_v(t)f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

Summing on *s* results in a factor with two arguments $f_s(b,l)$

In general, result of elimination may be a function of several variables



We want to compute *P(d)*

Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

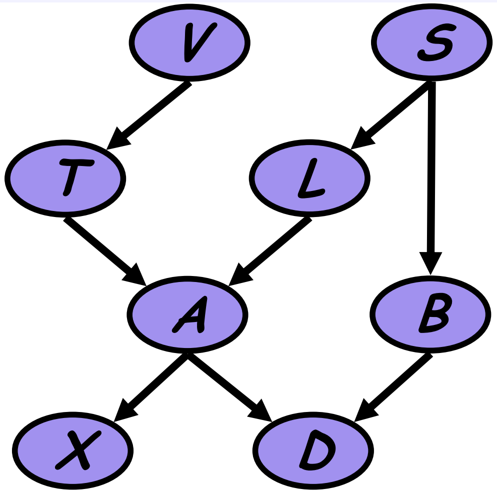
$$\Rightarrow f_v(t)f_s(b,l)P(a|t,l)\underline{P(x|a)}P(d|a,b)$$

Eliminate: *x*

Compute: $f_x(a) = \sum_x P(x|a)$

$$\Rightarrow f_v(t)f_s(b,l)\underline{f_x(a)}P(a|t,l)P(d|a,b)$$

Note: $f_x(a) = 1$ for all values of *a* !!



We want to compute *P(d)*

Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

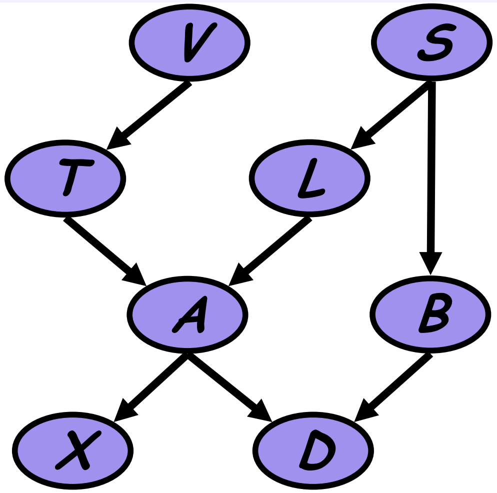
$$\Rightarrow f_v(t)f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow \underline{f_v(t)}f_s(b,l)f_x(a)\underline{P(a|t,l)}P(d|a,b)$$

Eliminate: *t*

Compute: $f_t(a,l) = \sum_t f_v(t)P(a|t,l)$

$$\Rightarrow f_s(b,l)f_x(a)\underline{f_t(a,l)}P(d|a,b)$$



We want to compute *P(d)*

Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

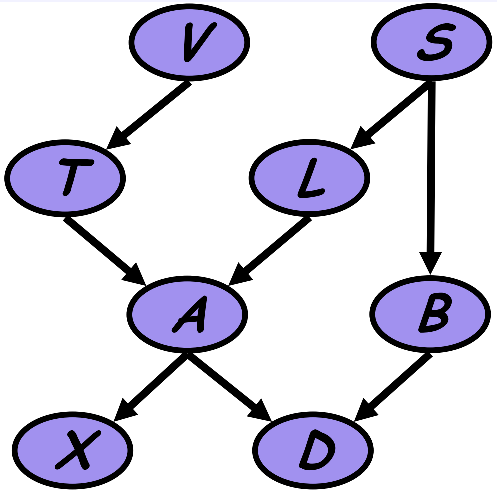
$$\Rightarrow f_v(t)f_s(b,l)f_x(a)P(a|t,l)P(d|a,b)$$

$$\Rightarrow \underline{f_s(b,l)}f_x(a)\underline{f_t(a,l)}P(d|a,b)$$

Eliminate: *l*

Compute: $f_l(a,b) = \sum_l f_s(b,l) f_t(a,l)$

$$\Rightarrow \underline{f_l(a,b)}f_x(a)P(d|a,b)$$



We want to compute *P(d)*

Need to eliminate: *v,s,x,t,l,a,b*

Initial factors

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

$$\Rightarrow f_v(t)f_s(b,l)f_x(a)P(a|t,l)P(d|a,b)$$

$$\Rightarrow f_s(b,l)f_x(a)f_t(a,l)P(d|a,b)$$

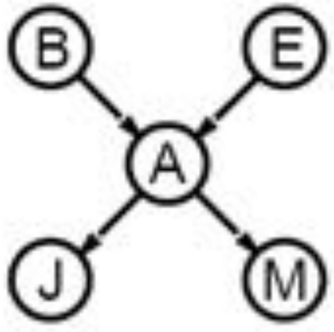
$$\Rightarrow f_l(a,b)f_x(a)P(d|a,b) \Rightarrow f_a(b,d) \Rightarrow f_b(d)$$

Eliminate: *a,b*

Compute: $f_a(b,d) = \sum_a f_l(a,b) f_x(a) p(d|a,b) \quad f_b(d) = \sum_b f_a(b,d)$

Irrelevant Variables

Consider the query $P(JohnCalls|Burglary = true)$



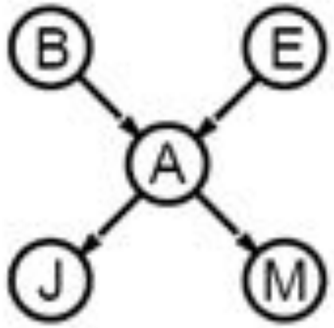
$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

Irrelevant Variables

Consider the query $P(JohnCalls|Burglary = true)$

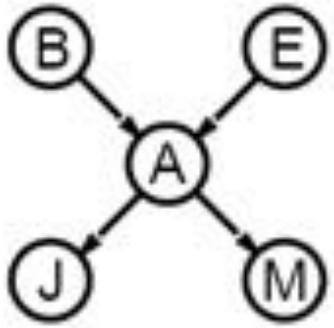
$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

Sum over m is identically 1; M is **irrelevant** to the query



Irrelevant Variables

Consider the query $P(JohnCalls|Burglary = true)$



$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

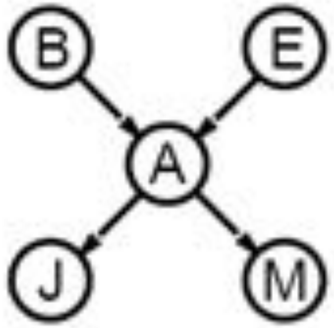
Sum over m is identically 1; M is **irrelevant** to the query

Thm 1: Y is irrelevant unless $Y \in Ancestors(\{X\} \cup \mathbf{E})$

Here, $X = JohnCalls$, $\mathbf{E} = \{Burglary\}$, and
$Ancestors(\{X\} \cup \mathbf{E}) = \{Alarm, Earthquake\}$
so $MaryCalls$ is irrelevant

Irrelevant Variables

Consider the query $P(JohnCalls|Burglary = true)$



$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

Sum over m is identically 1; M is **irrelevant** to the query

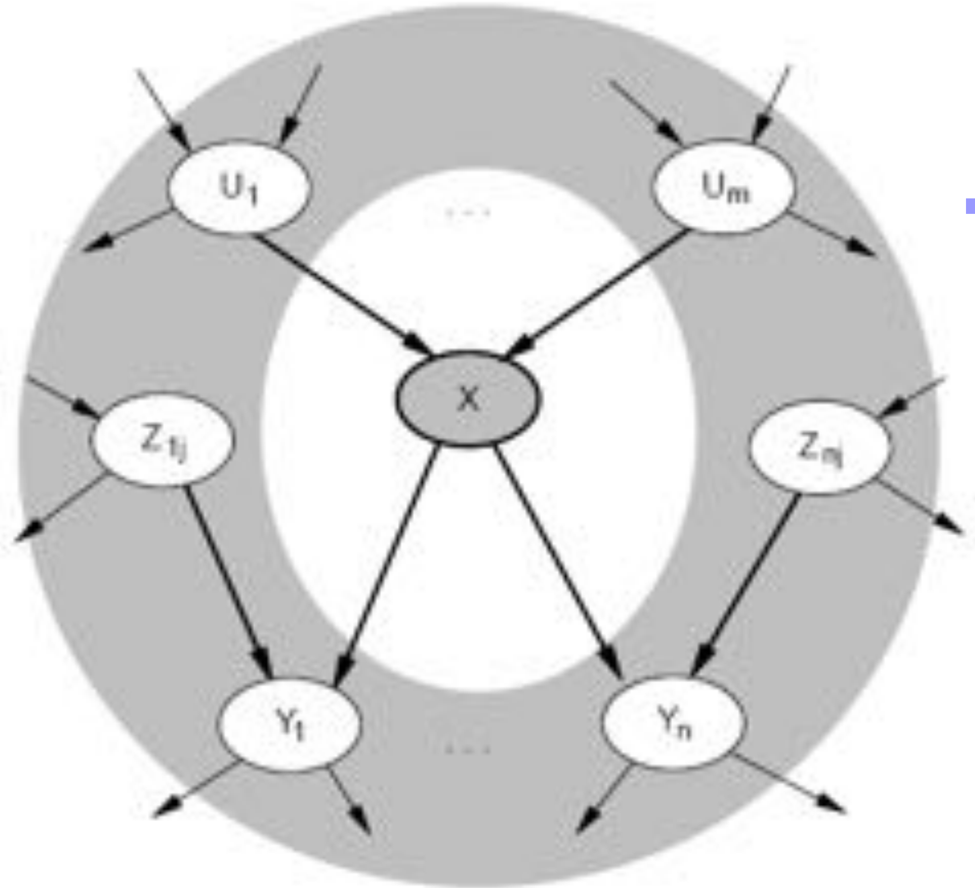
Thm 1: Y is irrelevant unless $Y \in Ancestors(\{X\} \cup \mathbf{E})$

Here, $X = JohnCalls$, $\mathbf{E} = \{Burglary\}$, and
$Ancestors(\{X\} \cup \mathbf{E}) = \{Alarm, Earthquake\}$
so $MaryCalls$ is irrelevant

Note: This is similar to backward chaining from a query in Prolog

(Directed) Markov Blanket

- **Markov Blanket:**
  - parents + children + children's parents

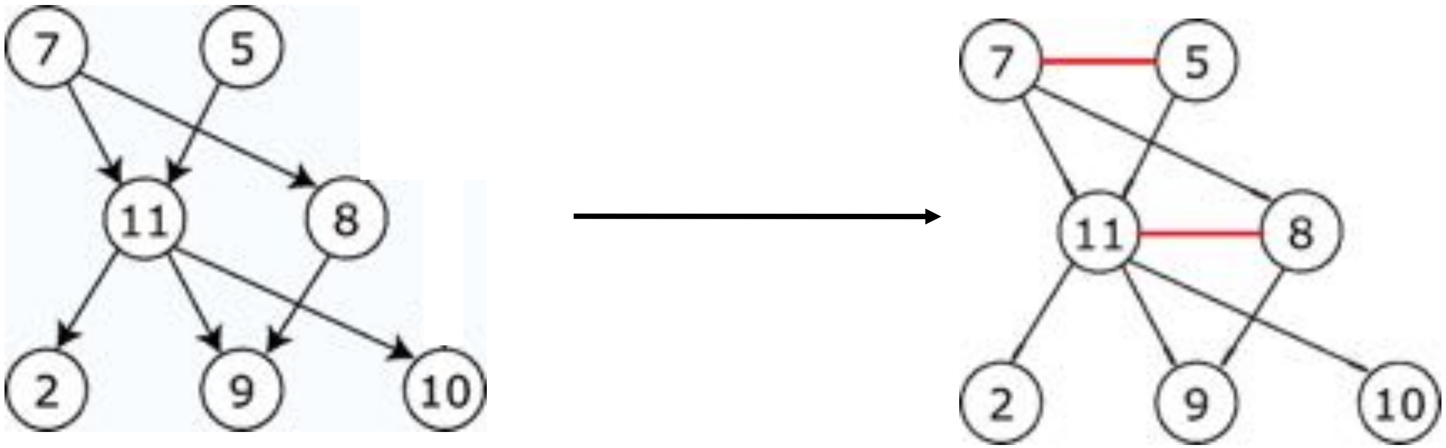


- Each node is conditionally independent of all other nodes given its markov blanket

$$\mathbf{P}(X \mid U_{1,}\ldots, U_m, Y_{1,}\ldots, Y_n, Z_{1j}, \ldots, Z_{nj}) = \\ = \mathbf{P}(X \mid all\ variables)$$

Moral Graph

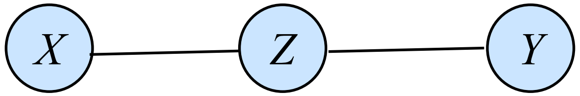
- The moral graph is an undirected graph that is obtained as follows:
  - connect all parents of all nodes
  - make all directed links undirected
- Note:
  - the moral graph connects each node to all nodes of its Markov blanket
    - it is already connected to parents and children
    - now it is also connected to the parents of its children



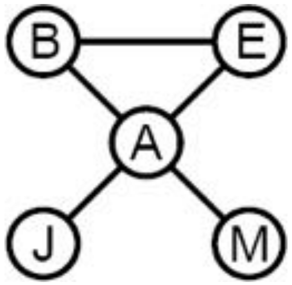
 Graphs taken from Wikipedia

Moral Graph and Irrelevant Variables

- m-separation:
  - variable X is m-separated from Y by Z iff it is separated by Z in the moral graph

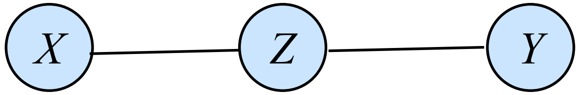


- Example:
  - J is m-separated from E by A

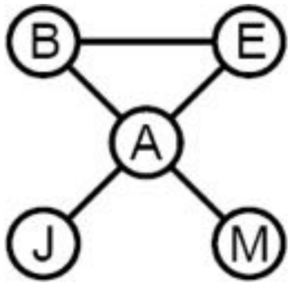


Moral Graph and Irrelevant Variables

- m-separation:
  - variable X is m-separated from Y by Z iff it is separated by Z in the moral graph



- Example:
  - J is m-separated from E by A



Theorem 2: Y is irrelevant if it is m-separated from X by Z

- Example:

For $P(JohnCalls|Alarm = true)$, both $Burglary$ and $Earthquake$ are irrelevant

Complexity of Exact Inference

Singly connected networks (or polytrees):

- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^k n)$

Complexity of Exact Inference

Singly connected networks (or polytrees):

- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^k n)$

Multiply connected networks:

Complexity of Exact Inference

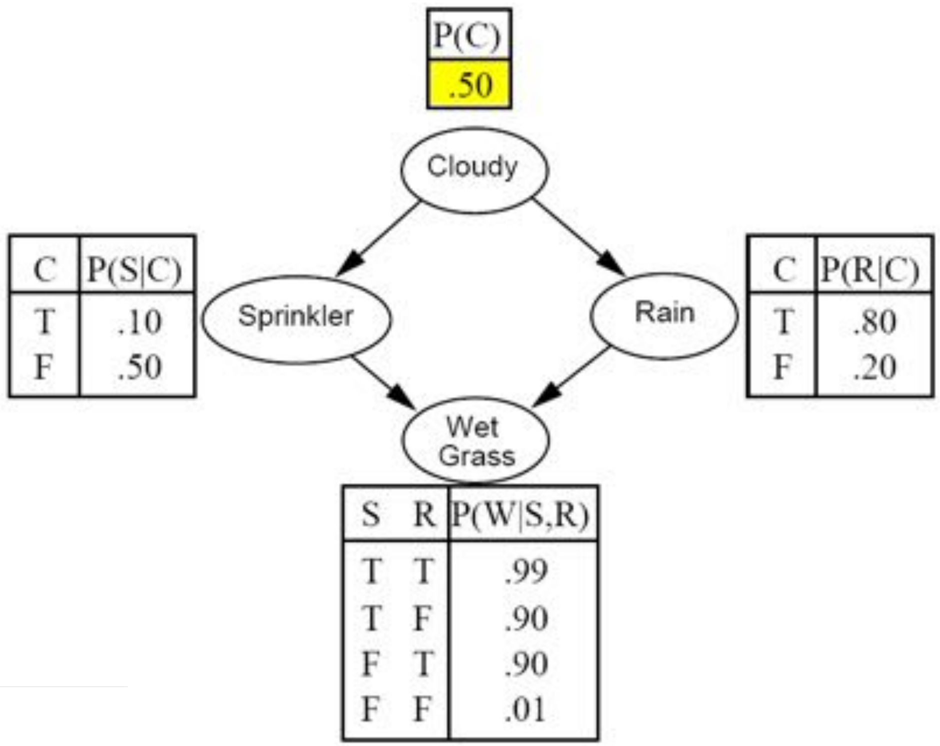
Singly connected networks (or polytrees):

- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^k n)$

Multiply connected networks:

- can reduce 3SAT to exact inference $\Rightarrow$ NP-hard

Example:
Two paths from
Cloudy to Wet Grass



Complexity of Exact Inference

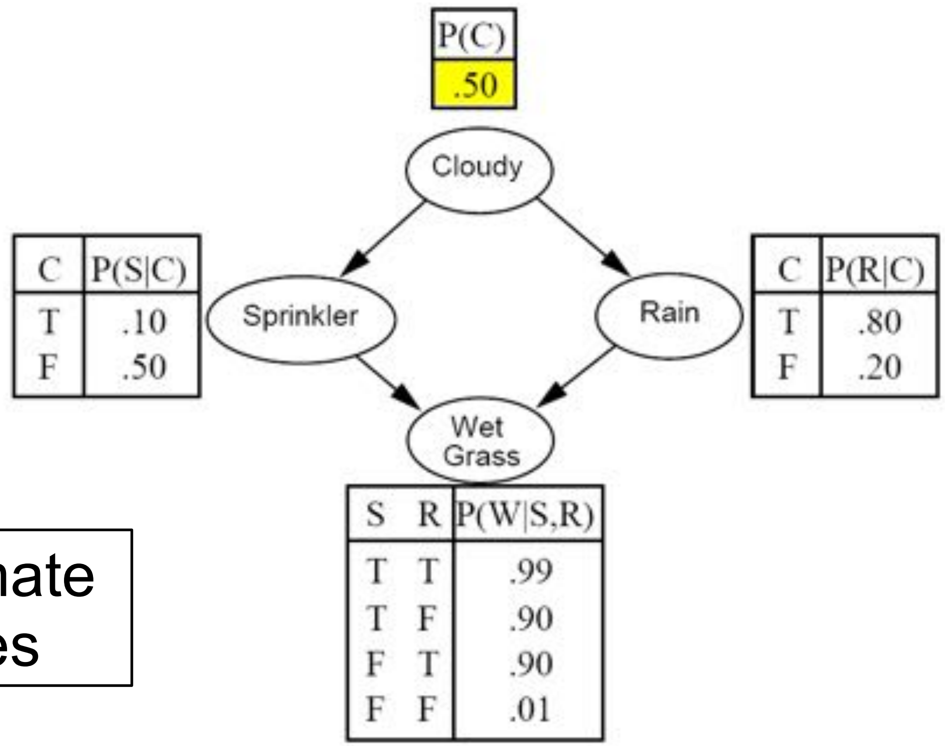
Singly connected networks (or polytrees):

- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^k n)$

Multiply connected networks:

- can reduce 3SAT to exact inference $\Rightarrow$ NP-hard

Example:
Two paths from
Cloudy to Wet Grass



→ we “need” approximate inference techniques

Complexity of Inference

Theorem:

Inference in Bayesian networks (even approximate, without proof) is NP-hard

Inference by Stochastic Simulation
(Sampling from a Bayesian Network)

Basic idea:

1) Draw N samples from a sampling distribution S
2) Compute an approximate posterior probability $\hat{P}$
3) Show this converges to the true probability P

Outline:

- Sampling from an empty network
- Rejection sampling: reject samples disagreeing with evidence
- Likelihood weighting: use evidence to weight samples
- Markov chain Monte Carlo (MCMC): sample from a stochastic process whose stationary distribution is the true posterior

How to draw a sample ?

Given random variable X, D(X)={0, 1}
Given P(X) = {0.3, 0.7}

How to draw a sample ?

Given random variable X, D(X)={0, 1}
Given P(X) = {0.3, 0.7}

Sample X ← P (X)
- **draw random number r ∈ [0, 1]**
- **If (r < 0.3) then set X=0**
- **Else set X=1**

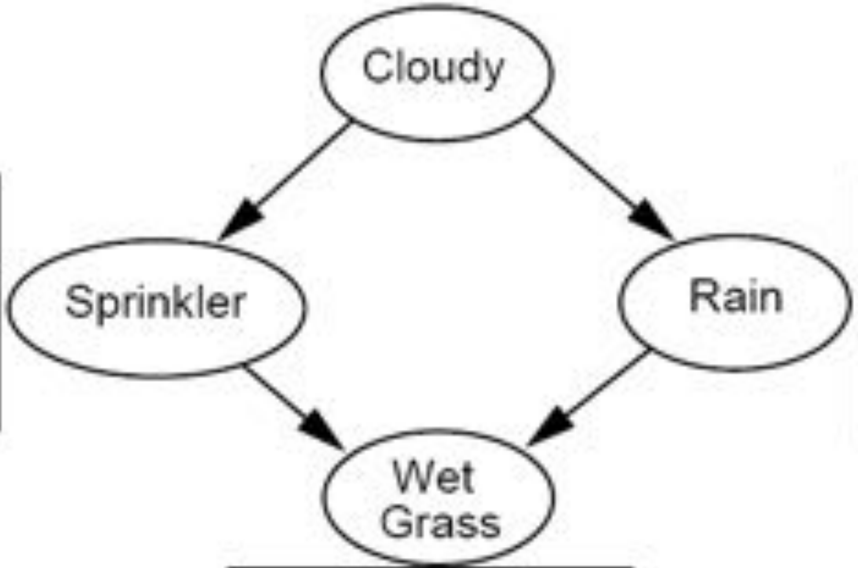
Can generalize for any domain size

Sampling from an "Empty" Network

- Generating samples from a network that has no evidence associated with it (*empty* network)
- Basic idea
  - sample a value for each variable in topological order
  - using the specified conditional probabilities

```
function PRIOR-SAMPLE(bn) returns an event sampled from bn
   inputs: bn, a belief network specifying joint distribution P(X_1, ..., X_n)
   x ← an event with n elements
   for i = 1 to n do
        x_i ← a random sample from P(X_i | parents(X_i))
             given the values of Parents(X_i) in x
   return x
```

Example



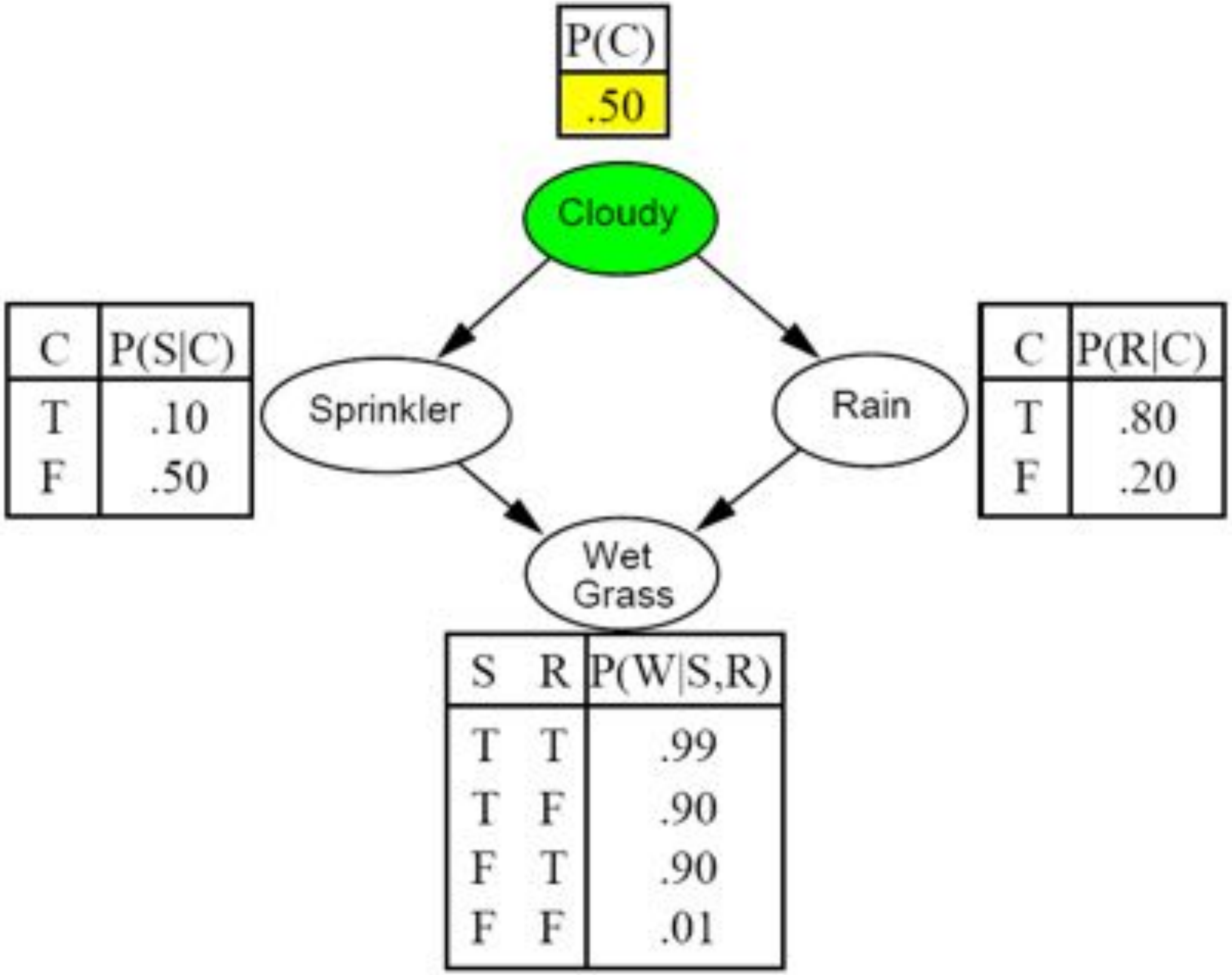
P(C)
.50

C	P(S|C)
T	.10
F	.50

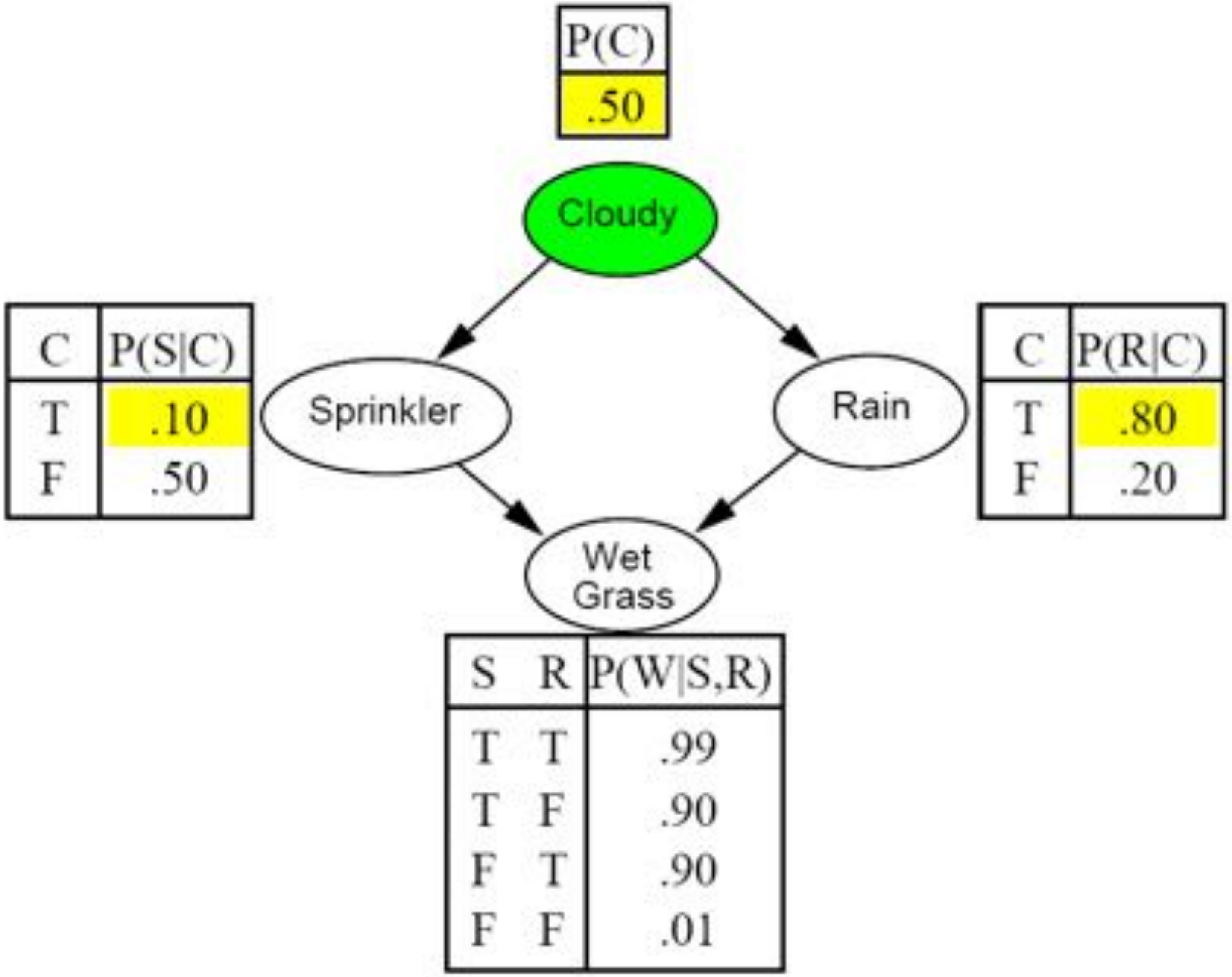
C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

Example



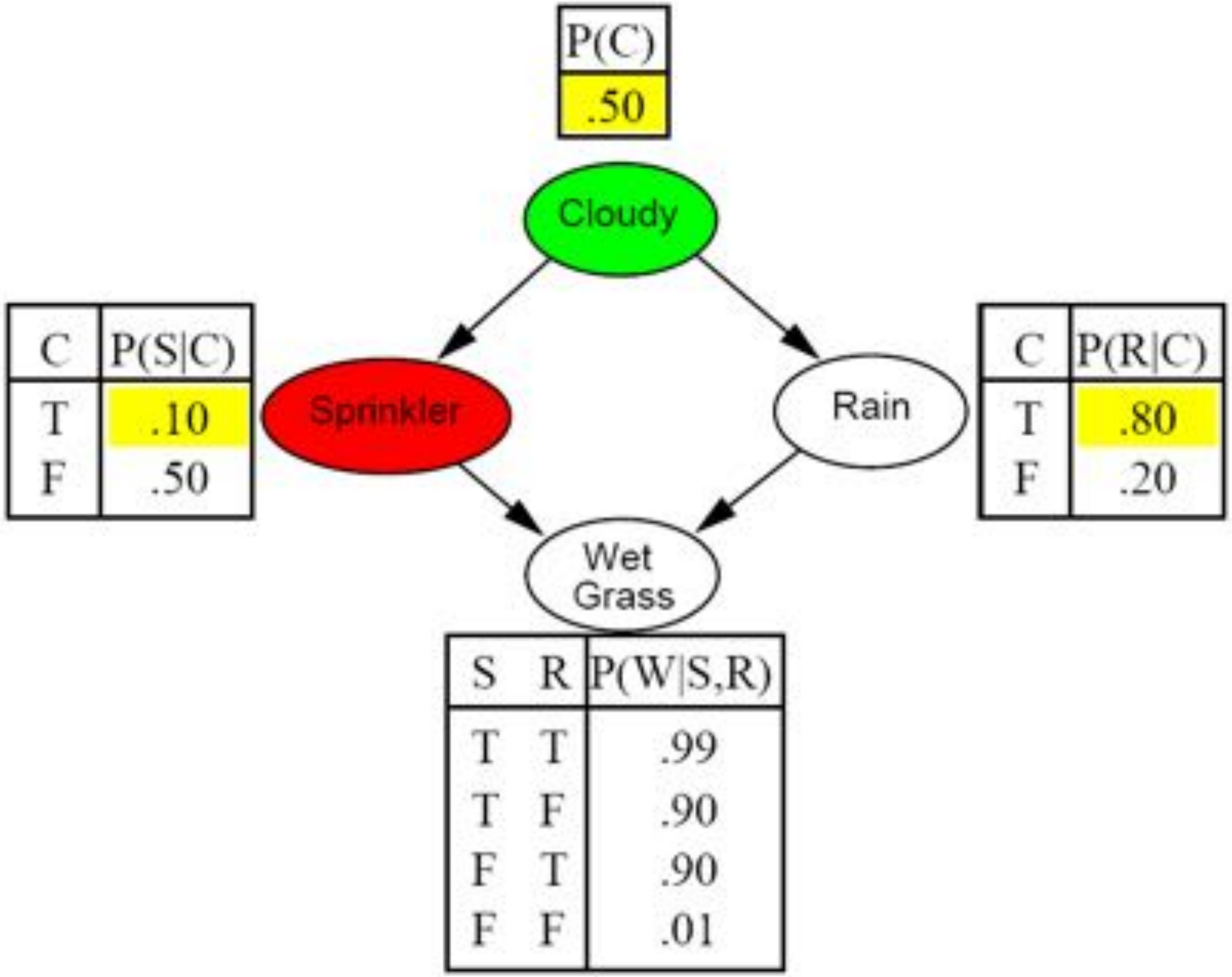
P(C)
.50

C	P(S|C)
T	.10
F	.50

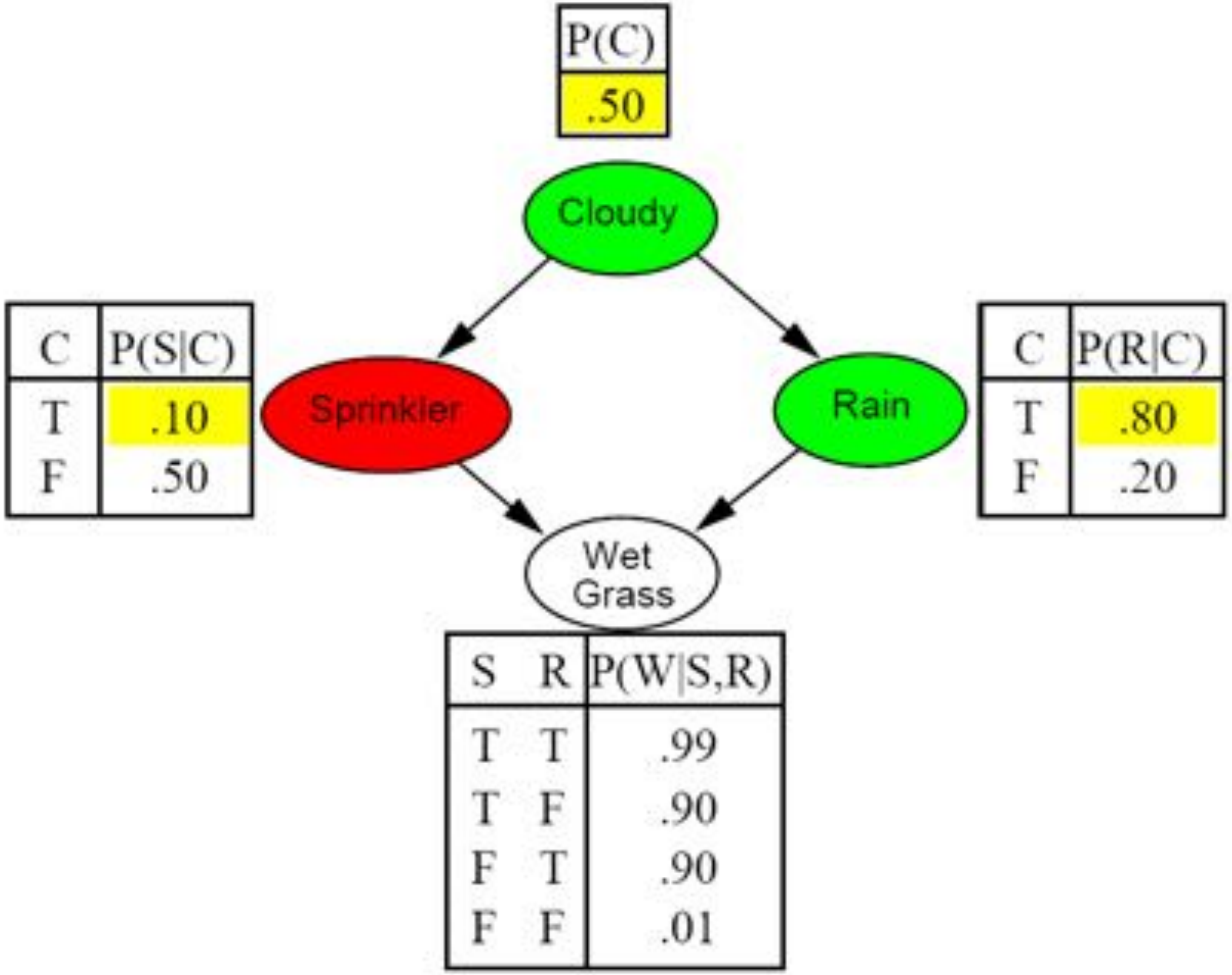
C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

Example



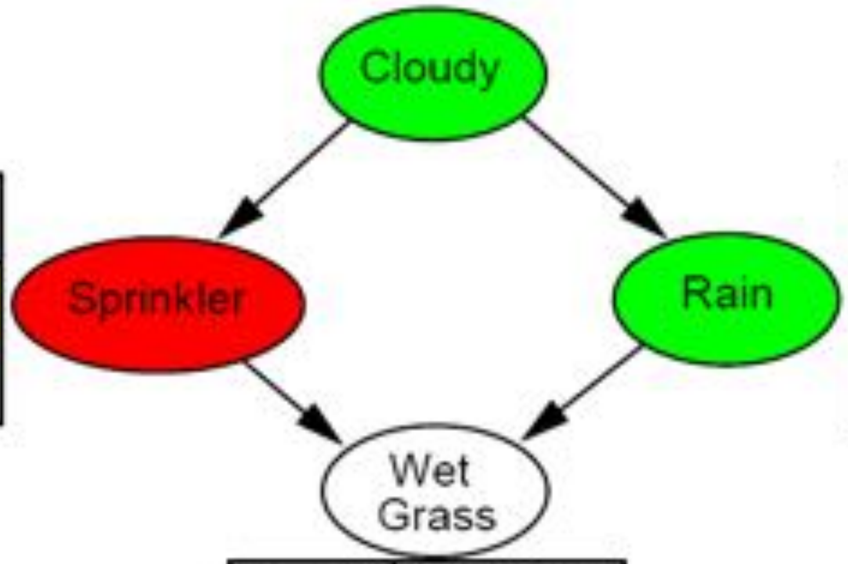
P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Example



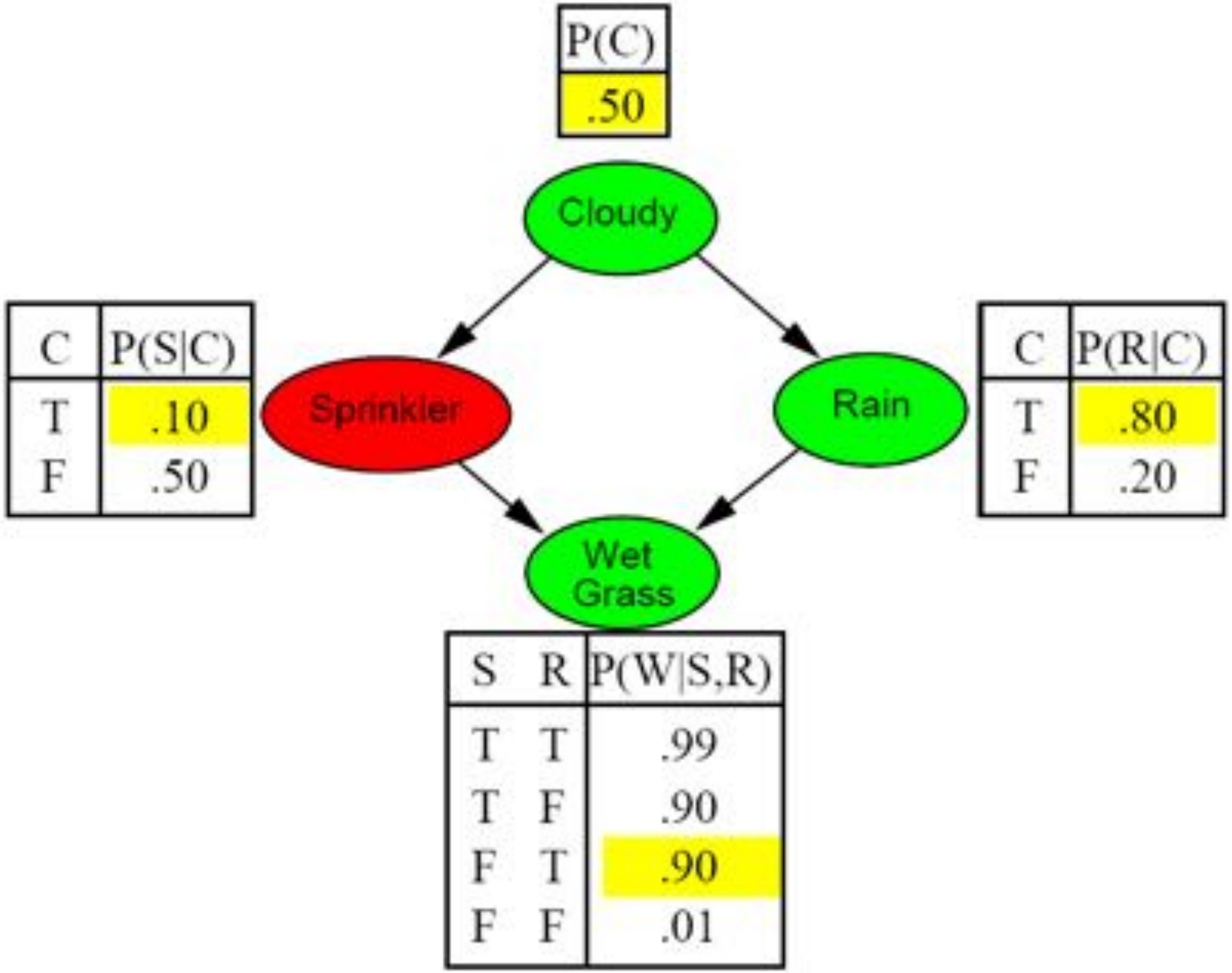
P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

Probability Estimation using Sampling

- sample many points using the above algorithm
- count how often each possible combination $x_{1,} x_2, \ldots, x_n$ appears
  - increment counters $N_{PS}(x_1 \ldots x_n)$
- estimate the probability by the observed percentages

$$\hat{P}_{PS}(x_1 \ldots x_n) = N_{PS}(x_1 \ldots x_n) / N$$

Probability Estimation using Sampling

- sample many points using the above algorithm
- count how often each possible combination $x_{1,} x_2, \ldots, x_n$ appears
  - increment counters $N_{PS}(x_1 \ldots x_n)$
- estimate the probability by the observed percentages

$$\hat{P}_{PS}(x_1 \ldots x_n) = N_{PS}(x_1 \ldots x_n) / N$$

Does this converge towards the joint probability function?

Convergence of Sampling from an Empty Network

Probability that PRIORSAMPLE generates a particular event

$$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i)) = P(x_1 \ldots x_n)$$

i.e., the true prior probability

E.g., $S_{PS}(t, f, t, t) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 = P(t, f, t, t)$

Let $N_{PS}(x_1 \ldots x_n)$ be the number of samples generated for event $x_1, \ldots, x_n$

Then we have

$$\begin{aligned}\lim_{N\to\infty} \hat{P}(x_1, \ldots, x_n) &= \lim_{N\to\infty} N_{PS}(x_1, \ldots, x_n)/N \\ &= S_{PS}(x_1, \ldots, x_n) \\ &= P(x_1 \ldots x_n)\end{aligned}$$

That is, estimates derived from PRIORSAMPLE are consistent

Shorthand: $\hat{P}(x_1, \ldots, x_n) \approx P(x_1 \ldots x_n)$

Rejection Sampling

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

function REJECTION-SAMPLING(X, **e**, bn, N) **returns** an estimate of $P(X|\mathbf{e})$
  local variables: **N**, a vector of counts over X, initially zero

  for $j = 1$ to N **do**
    x ← PRIOR-SAMPLE(bn)
    if **x** is consistent with **e** **then**
      N[x] ← **N**[x]+1 where x is the value of X in **x**
  return NORMALIZE(**N**[X])

Rejection Sampling

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

```
function REJECTION-SAMPLING(X, e, bn, N) returns an estimate of P(X|e)
    local variables: N, a vector of counts over X, initially zero

    for j = 1 to N do
        x ← PRIOR-SAMPLE(bn)
        if x is consistent with e then
            N[x] ← N[x]+1 where x is the value of X in x
    return NORMALIZE(N[X])
```

E.g., estimate $\mathbf{P}(Rain|Sprinkler = true)$ using 100 samples

- 27 samples have $Sprinkler = true$
  - Of these, 8 have $Rain = true$ and 19 have $Rain = false$.

$\hat{\mathbf{P}}(Rain|Sprinkler = true) = \text{NORMALIZE}(\langle 8, 19 \rangle) = \langle 0.296, 0.704 \rangle$

Similar to a basic real-world empirical estimation procedure

Analysis of Rejection Sampling

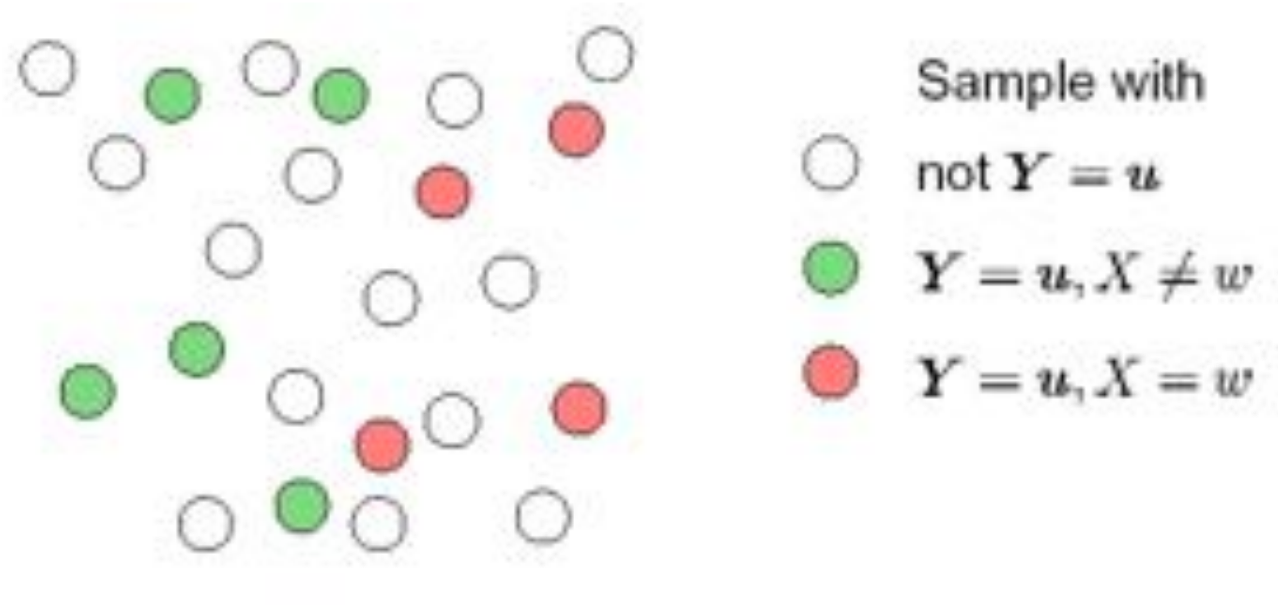
- Rejection sampling generates random samples from an empty network
  - and discards all samples that are inconsistent with the evidence

$$\begin{aligned}
\hat{\mathbf{P}}(X|\mathbf{e}) &= \alpha \mathbf{N}_{PS}(X, \mathbf{e}) && \text{(algorithm defn.)} \\
&= \mathbf{N}_{PS}(X, \mathbf{e}) / N_{PS}(\mathbf{e}) && \text{(normalized by } N_{PS}(\mathbf{e})\text{)} \\
&\approx \mathbf{P}(X, \mathbf{e}) / P(\mathbf{e}) && \text{(property of PRIORSAMPLE)} \\
&= \mathbf{P}(X|\mathbf{e}) && \text{(defn. of conditional probability)}
\end{aligned}$$

Hence rejection sampling returns consistent posterior estimates

Rejection Sampling: Illustration

Let Y be a subset of evidence nodes s.t. Y=u



Approximation for $P^{\mathbf{X}}(X = w \mid \boldsymbol{Y} = \boldsymbol{u})$:

Analysis of Rejection Sampling

- Rejection sampling generates random samples from an empty network
  - and discards all samples that are inconsistent with the evidence

$$\begin{aligned}\hat{\mathbf{P}}(X|\mathbf{e}) &= \alpha \mathbf{N}_{PS}(X,\mathbf{e}) && \text{(algorithm defn.)}\\ &= \mathbf{N}_{PS}(X,\mathbf{e})/N_{PS}(\mathbf{e}) && \text{(normalized by } N_{PS}(\mathbf{e})\text{)}\\ &\approx \mathbf{P}(X,\mathbf{e})/P(\mathbf{e}) && \text{(property of PRIORSAMPLE)}\\ &= \mathbf{P}(X|\mathbf{e}) && \text{(defn. of conditional probability)}\end{aligned}$$

Hence rejection sampling returns consistent posterior estimates

- **Problem**
  - many unnecessary samples will be generated if the probability of observing the evidence e is small
  - $P(\mathbf{e})$ will decrease exponentially with increasing numbers of evidence variables!

Likelihood Weighting

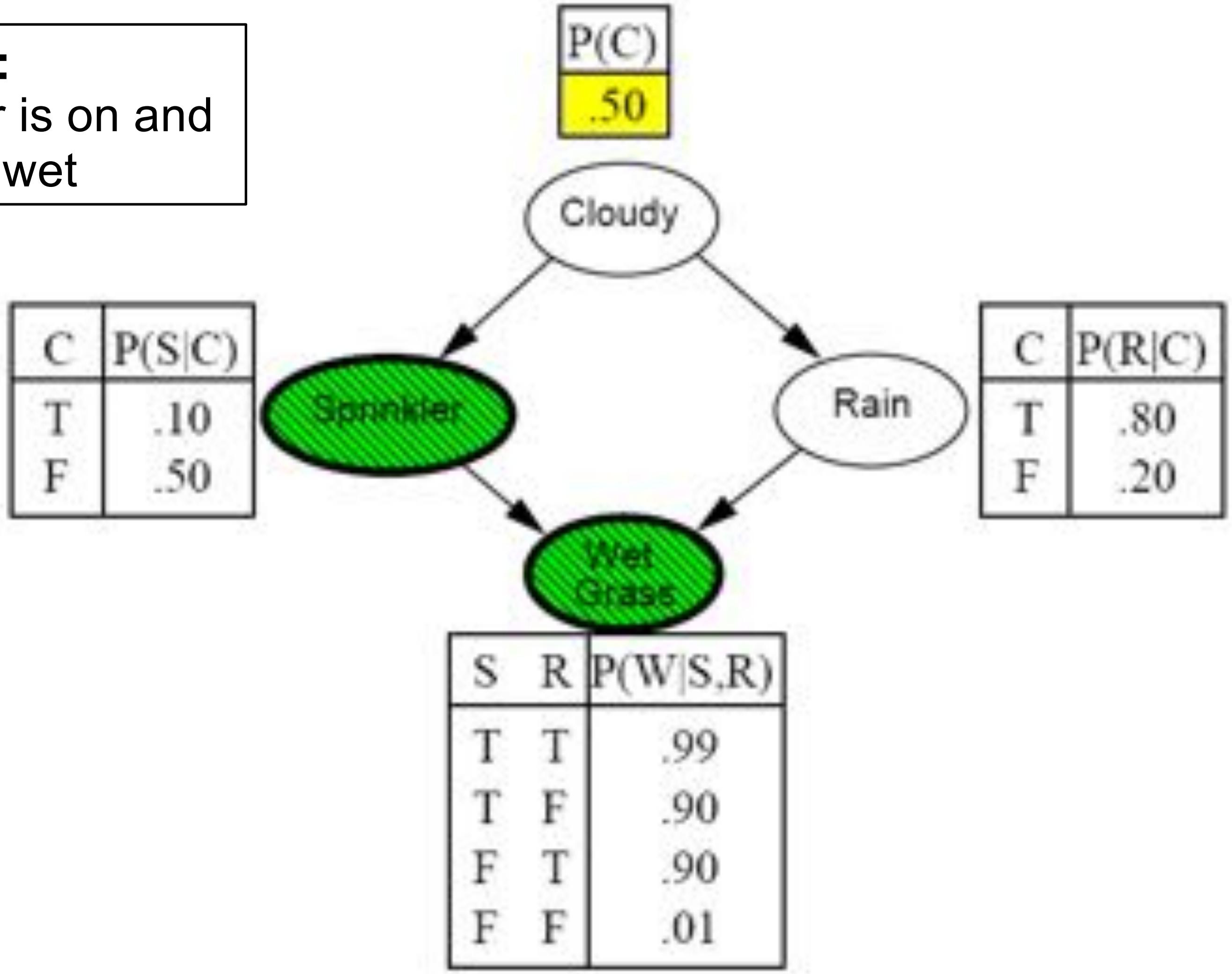
Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

```
function LIKELIHOOD-WEIGHTING(X, e, bn, N) returns an estimate of P(X|e)
   local variables: W, a vector of weighted counts over X, initially zero

   for j = 1 to N do
       x, w ← WEIGHTED-SAMPLE(bn)
       W[x] ← W[x] + w where x is the value of X in x
   return NORMALIZE(W[X])
```

```
function WEIGHTED-SAMPLE(bn, e) returns an event and a weight

   x ← an event with n elements; w ← 1
   for i = 1 to n do
       if X_i has a value x_i in e
           then w ← w × P(X_i = x_i | parents(X_i))
           else x_i ← a random sample from P(X_i | parents(X_i))
   return x, w
```

Example

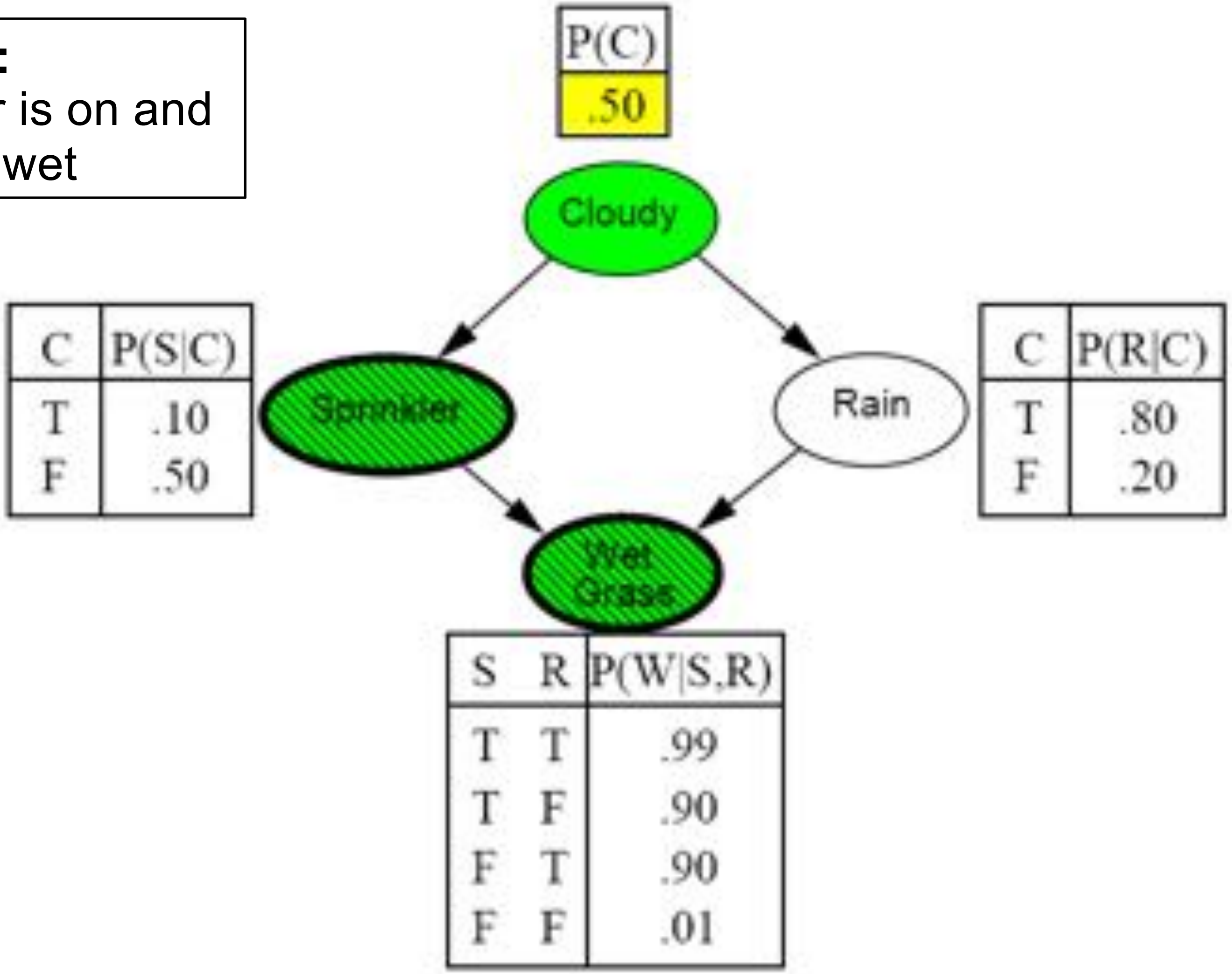
Evidence:
sprinkler is on and
grass is wet



$w = 1.0$

Example

Evidence:
sprinkler is on and
grass is wet



P(C)
.50

C	P(S|C)
T	.10
F	.50

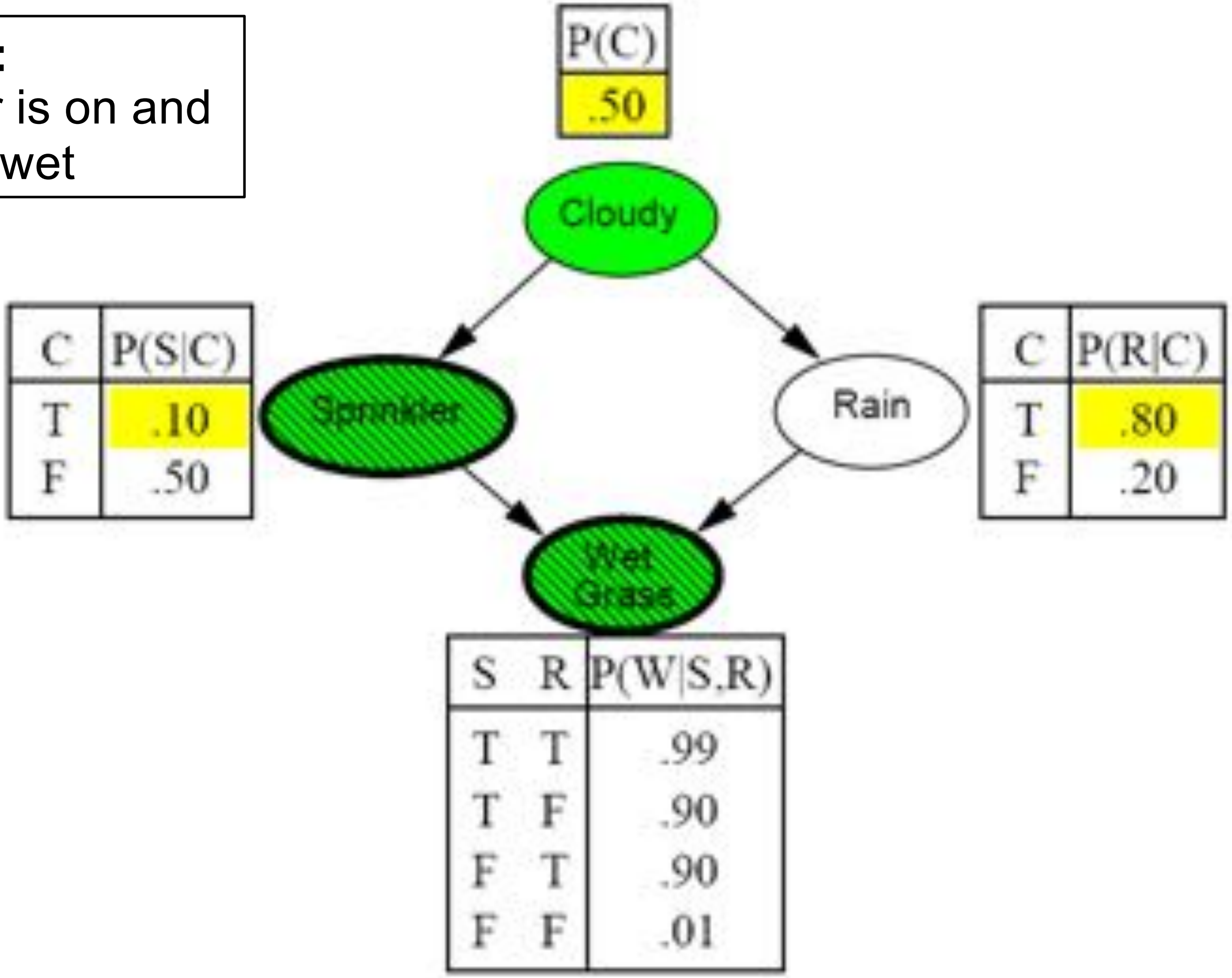
C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

$w = 1.0$

Example

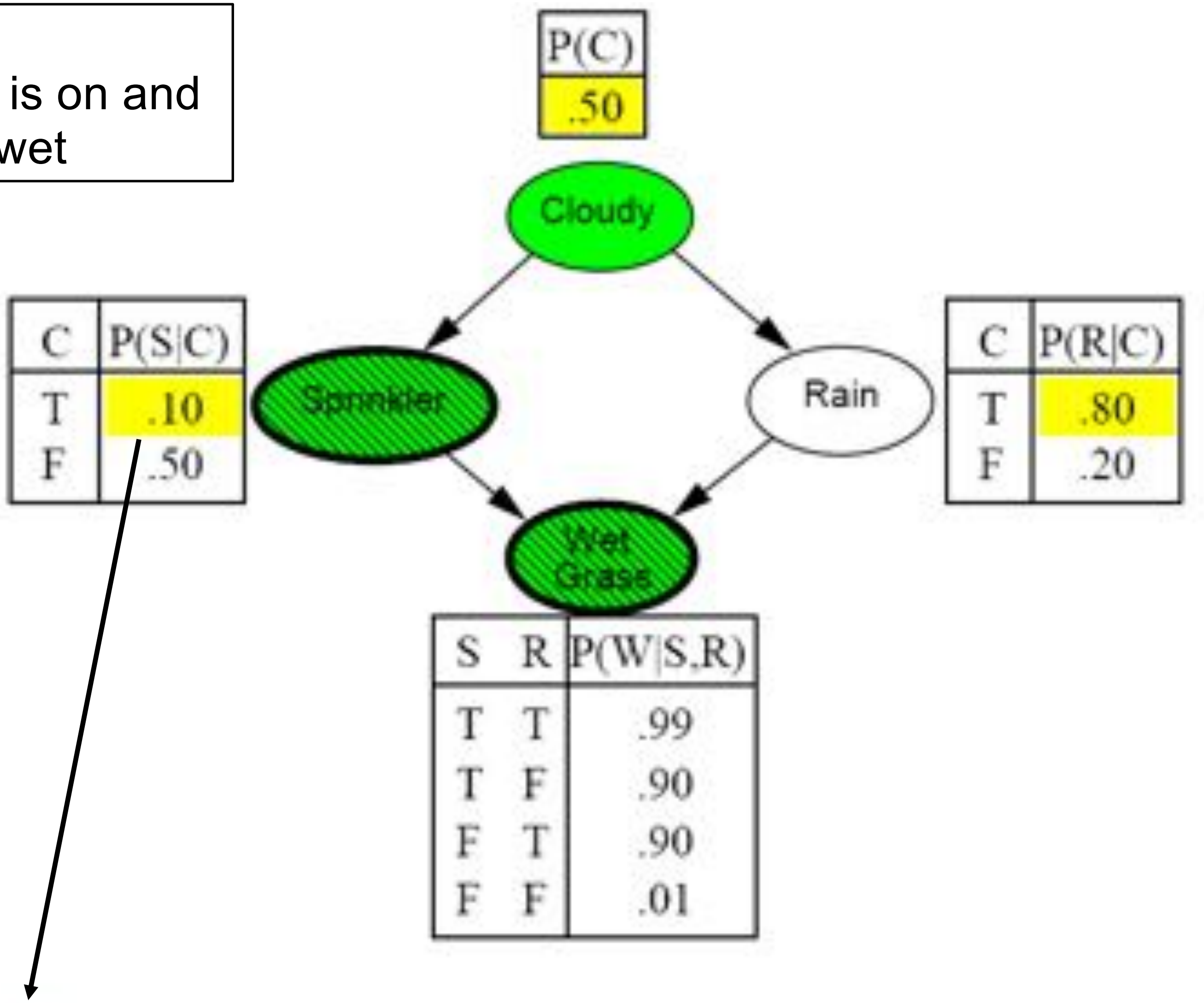
Evidence:
sprinkler is on and
grass is wet



$w = 1.0$

Example

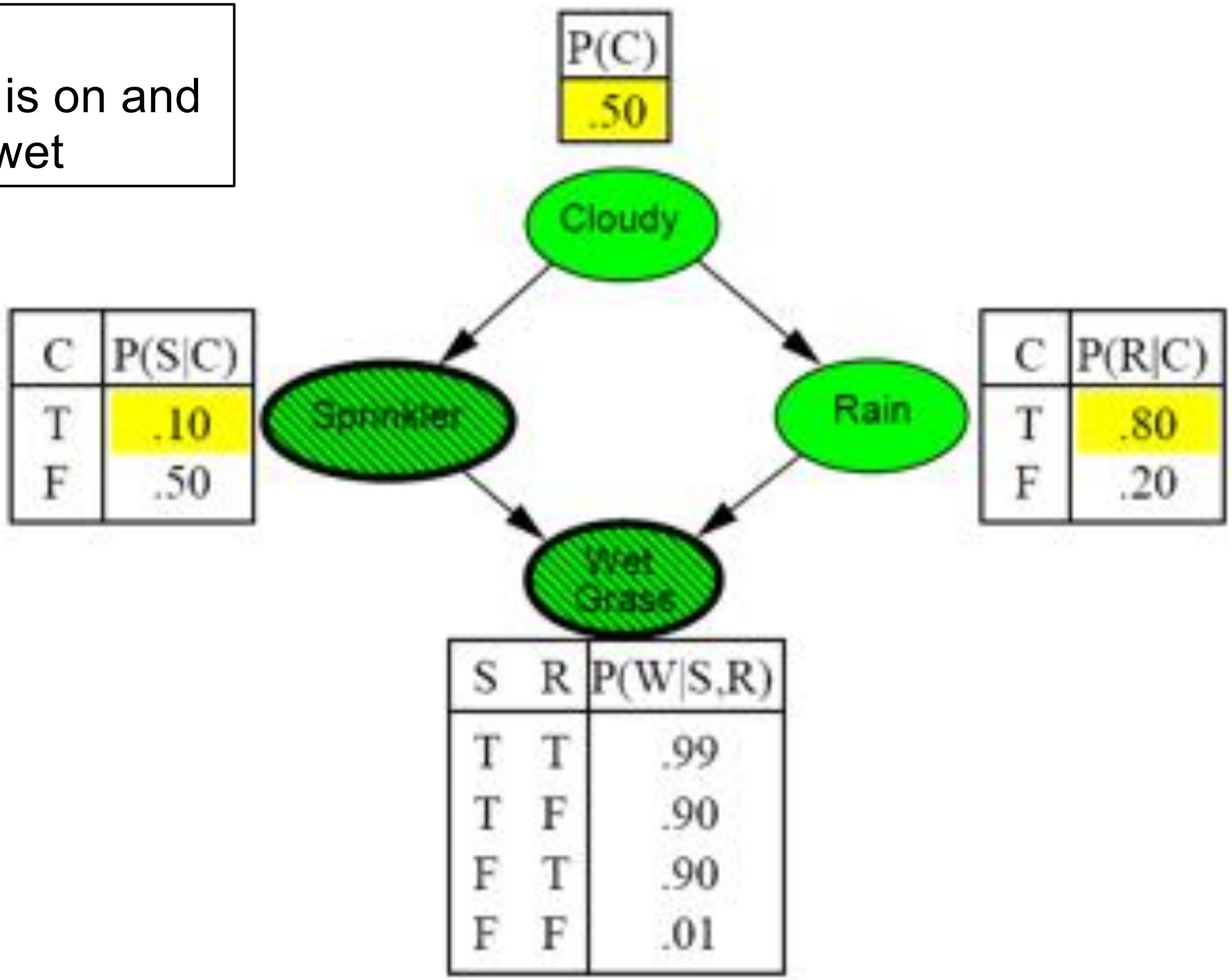
Evidence:
sprinkler is on and
grass is wet



$w = 1.0 \times 0.1$

Example

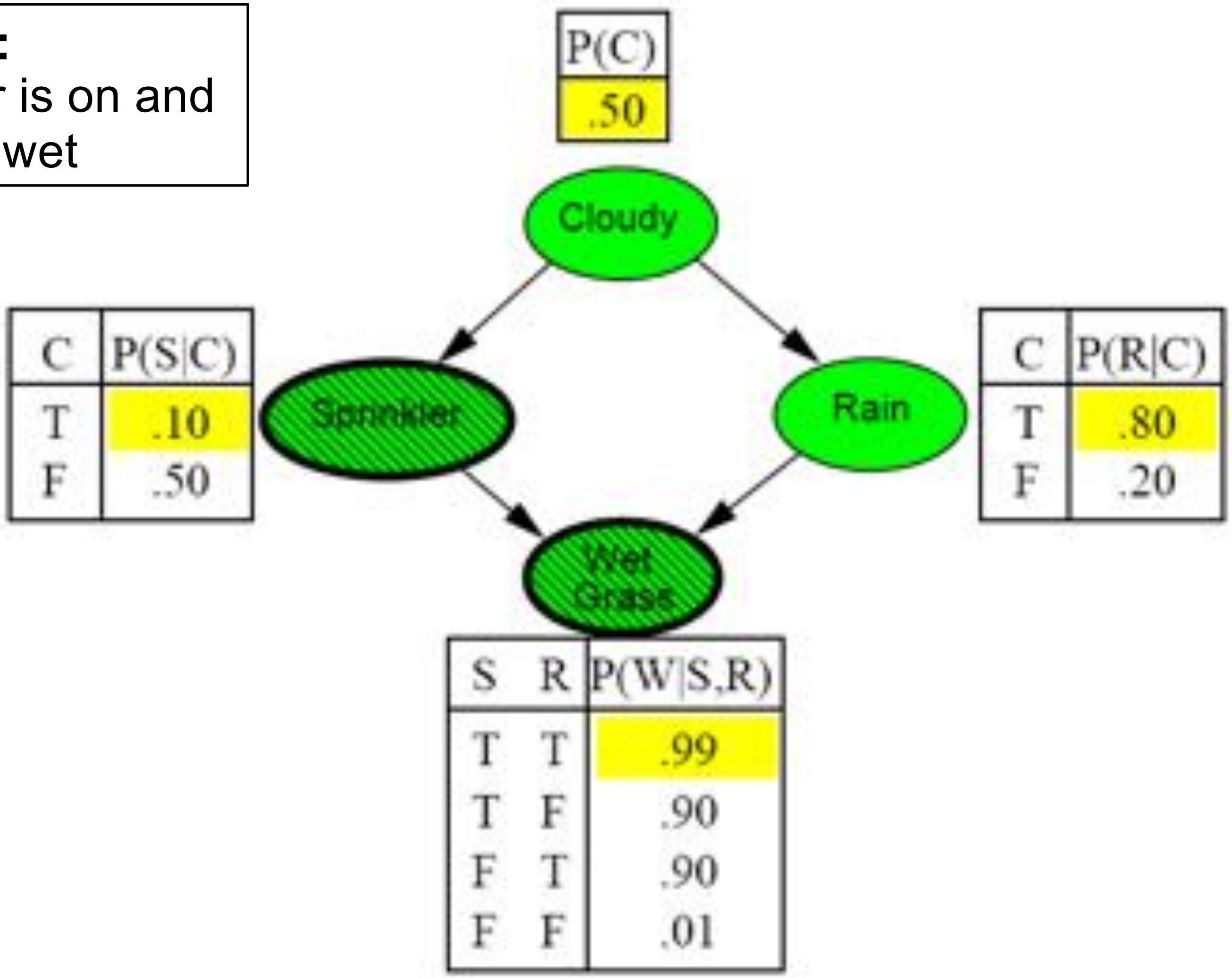
Evidence:
sprinkler is on and
grass is wet



$$w = 1.0 \times 0.1$$

Example

Evidence:
sprinkler is on and
grass is wet



P(C)
.50

C	P(S|C)
T	.10
F	.50

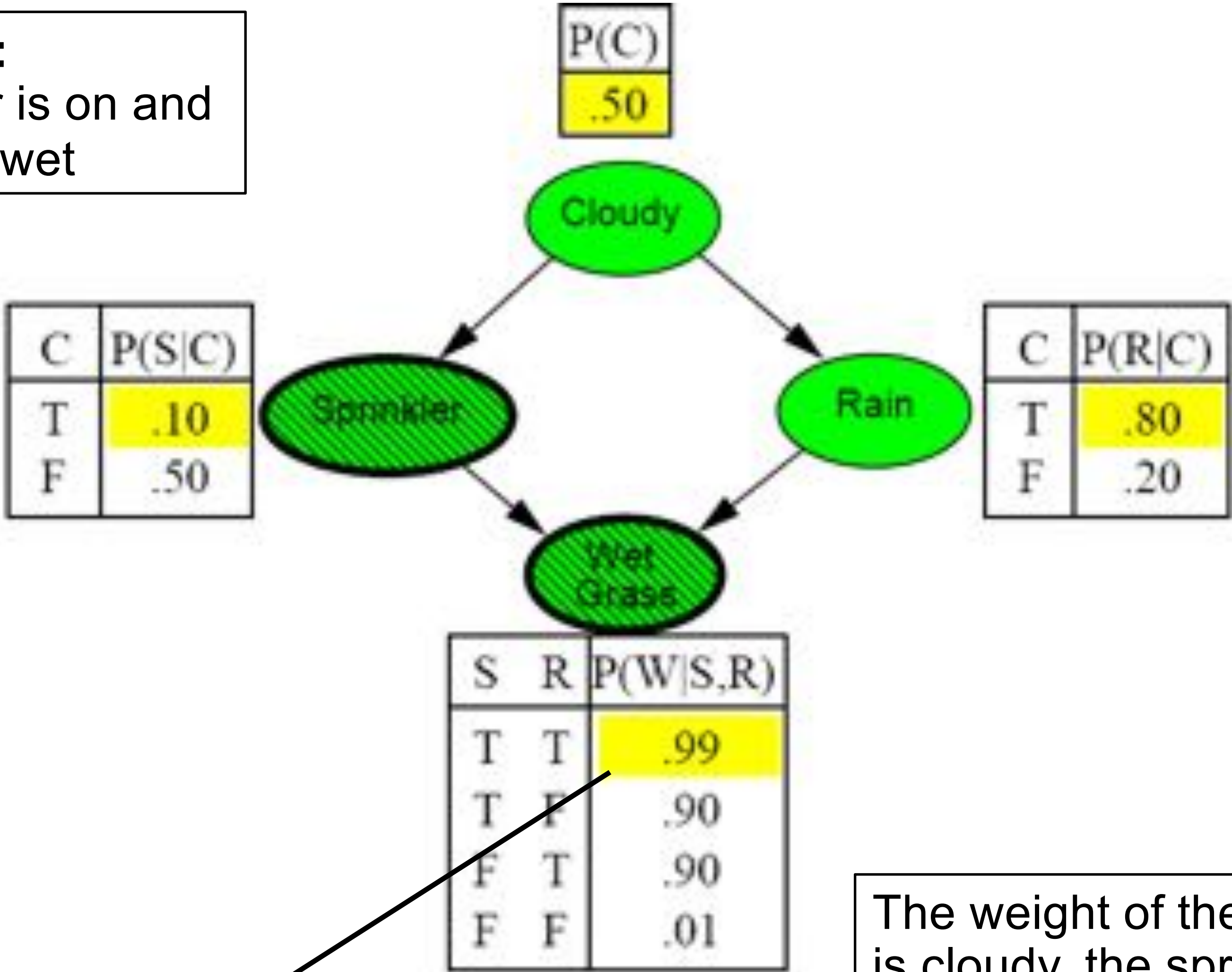
C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

$w = 1.0 \times 0.1$

Example

Evidence:
sprinkler is on and
grass is wet



P(C)
.50

C	P(S|C)
T	.10
F	.50

C	P(R|C)
T	.80
F	.20

S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.01

$$w = 1.0 \times 0.1 \times 0.99 = 0.099$$

The weight of the event that it is cloudy, the sprinkler is on, it is raining, and the grass is wet is 0.099.

Analysis

Sampling probability for WEIGHTEDSAMPLE is

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | parents(Z_i))$$

Note: pays attention to evidence in **ancestors** only

⇒ somewhere "in between" prior and posterior distribution

Weight for a given sample $\mathbf{z}, \mathbf{e}$ is

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | parents(E_i))$$

Weighted sampling probability is

$$S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e})$$
$$= \prod_{i=1}^{l} P(z_i | parents(Z_i)) \ \prod_{i=1}^{m} P(e_i | parents(E_i))$$
$$= P(\mathbf{z}, \mathbf{e}) \text{ (by standard global semantics of network)}$$

Hence likelihood weighting returns consistent estimates but performance still degrades with many evidence variables because a few samples have nearly all the total weight

Markov Chain Monte Carlo (MCMC) Sampling

"State" of network = current assignment to all variables.

Generate next state by sampling one variable given Markov blanket
Sample each variable in turn, keeping evidence fixed

```
function MCMC-Ask(X, e, bn, N) returns an estimate of P(X|e)
   local variables: N[X], a vector of counts over X, initially zero
                    Z, the nonevidence variables in bn
                    x, the current state of the network, initially copied from e

   initialize x with random values for the variables in Y
   for j = 1 to N do
       for each Z_i in Z do
           sample the value of Z_i in x from P(Z_i|mb(Z_i))
               given the values of MB(Z_i) in x
           N[x] ← N[x] + 1 where x is the value of X in x
   return Normalize(N[X])
```

Gibbs Sampling

Can also choose a variable to sample at random each time

Ordered Gibbs Sampler

Generate sample x^{t+1} from x^t :

Process all variables in some order

$$X_1 = x_1^{t+1} \leftarrow P(x_1 \mid x_2^t, x_3^t, \ldots, x_N^t, e)$$

$$X_2 = x_2^{t+1} \leftarrow P(x_2 \mid x_1^{t+1}, x_3^t, \ldots, x_N^t, e)$$

...

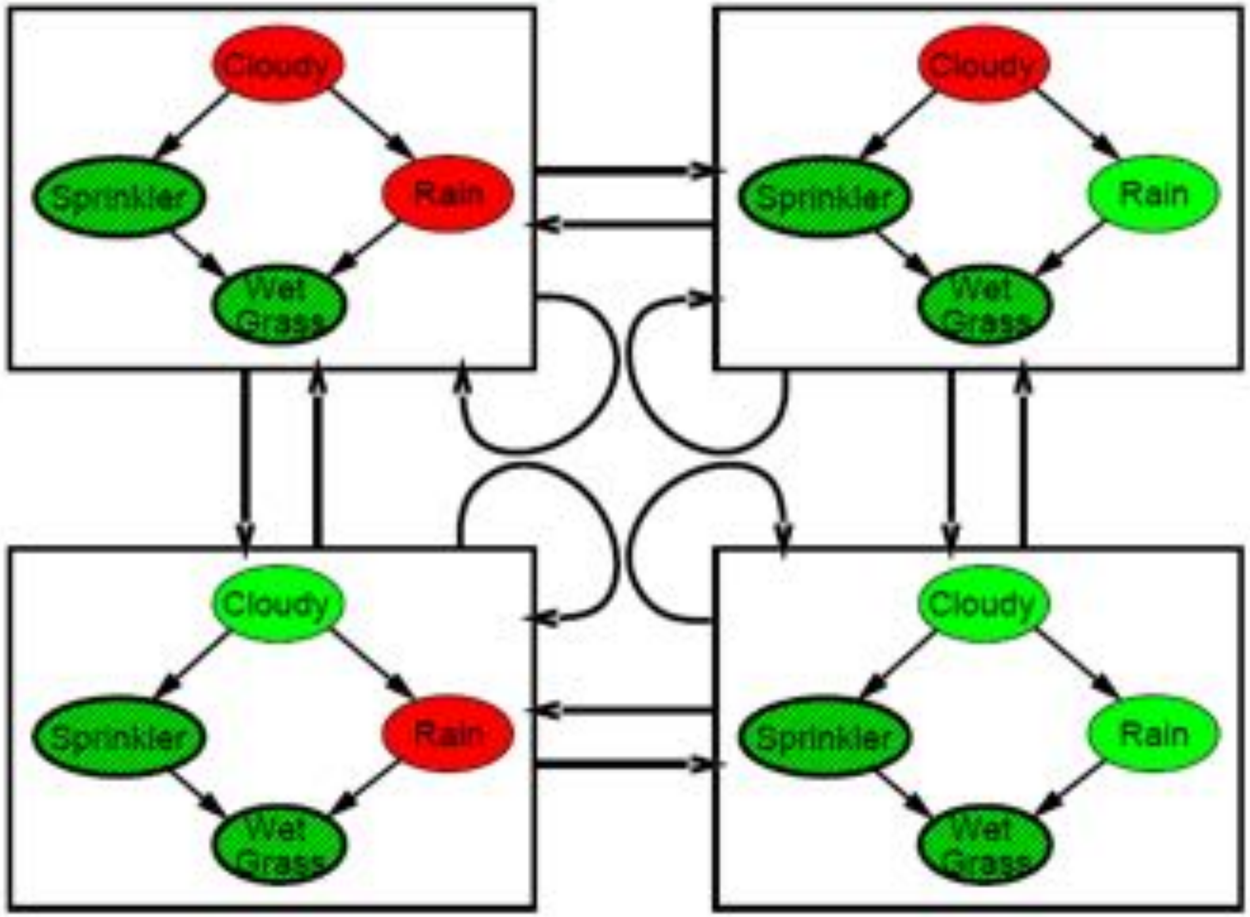
$$X_N = x_N^{t+1} \leftarrow P(x_N \mid x_1^{t+1}, x_2^{t+1}, \ldots, x_{N-1}^{t+1}, e)$$

In short, for i=1 to N:

$$X_i = x_i^{t+1} \leftarrow \textbf{sampled from } P(x_i \mid x^t \setminus x_i, e)$$

The Markov Chain

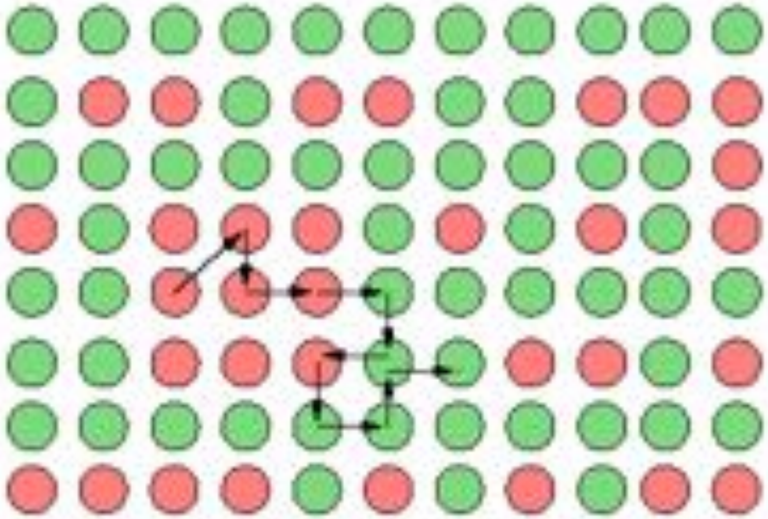
With $Sprinkler = true, WetGrass = true$, there are four states:



Wander about for a while, average what you see

Gibbs Sampling: Illustration

The process of Gibbs sampling can be understood as a *random walk* in the space of all instantiations with $\mathbf{Y} = \mathbf{u}$:



Reachable in one step: instantiations that differ from current one by value assignment to at most one variable (assume randomized choice of variable X_k).

Guaranteed to converge iff chain is :

- irreducible (every state reachable from every other state)
- aperiodic (returns to state i can occur at irregular times)
- ergodic (returns to every state with probability 1)

Example

Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$

Sample $Cloudy$ or $Rain$ given its Markov blanket, repeat.
Count number of times $Rain$ is true and false in the samples.

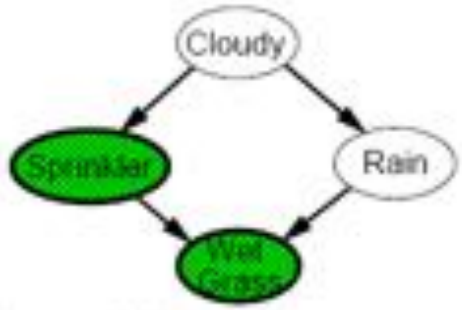
E.g., visit 100 states
31 have $Rain = true$, 69 have $Rain = false$

$\hat{\mathbf{P}}(Rain|Sprinkler = true, WetGrass = true)$
$= \text{NORMALIZE}(\langle 31, 69 \rangle) = \langle 0.31, 0.69 \rangle$

Theorem: chain approaches stationary distribution:
long-run fraction of time spent in each state is exactly proportional to its posterior probability

Markov Blanket Sampling

Markov blanket of $Cloudy$ is
$Sprinkler$ and $Rain$
Markov blanket of $Rain$ is
$Cloudy$, $Sprinkler$, and $WetGrass$



Probability given the Markov blanket is calculated as follows:

$$P(x_i'|mb(X_i)) = P(x_i'|parents(X_i))\Pi_{Z_j\in Children(X_i)}P(z_j|parents(Z_j))$$

What have we learned?

- Exact inference via Variable Elimination (VE)
- Inference in Bayesian networks is **NP-hard**, even when approximating. Still, for many distributions, sampling is the only option

- Forward sampling
- Rejections sample
- MCMC sampling (GIBBS sampling)

- Overall, we now know:
  - Basics of probability theory
  - Arguments why to follow probabilitiy theory
  - Bayesian networks (representation and semantics)
  - Inference in Bayesian networks