

Bayesian Networks

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

Inference Tasks

- Simple queries
 - compute the posterior marginal distribution for a variable
- Conjunctive queries
 - compute the posterior for a conjunction of variables
$$\mathbf{P}(X_i, X_j | \mathbf{E} = e) = \mathbf{P}(X_i | \mathbf{E} = e) \cdot \mathbf{P}(X_j | X_i, \mathbf{E} = e)$$
- Optimal decisions
 - decision networks include utility information
 - probabilistic inference required for $P(\text{outcome} | \text{action}, \text{evidence})$
- 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?

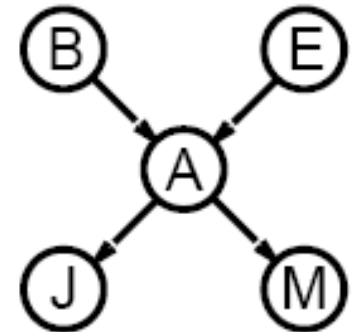
Inference by Enumeration

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

Simple query on the burglary network:

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

Worst case: $O(n d^n)$ time



Rewrite full joint entries using product of CPT entries:

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

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 \mathbf{E}

bn , a Bayesian network with variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$

$Q(X) \leftarrow$ a distribution over X , initially empty

for each value x_i of X **do**

 extend e with value x_i for X

$Q(x_i) \leftarrow$ **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

$Y \leftarrow$ **FIRST**($vars$)

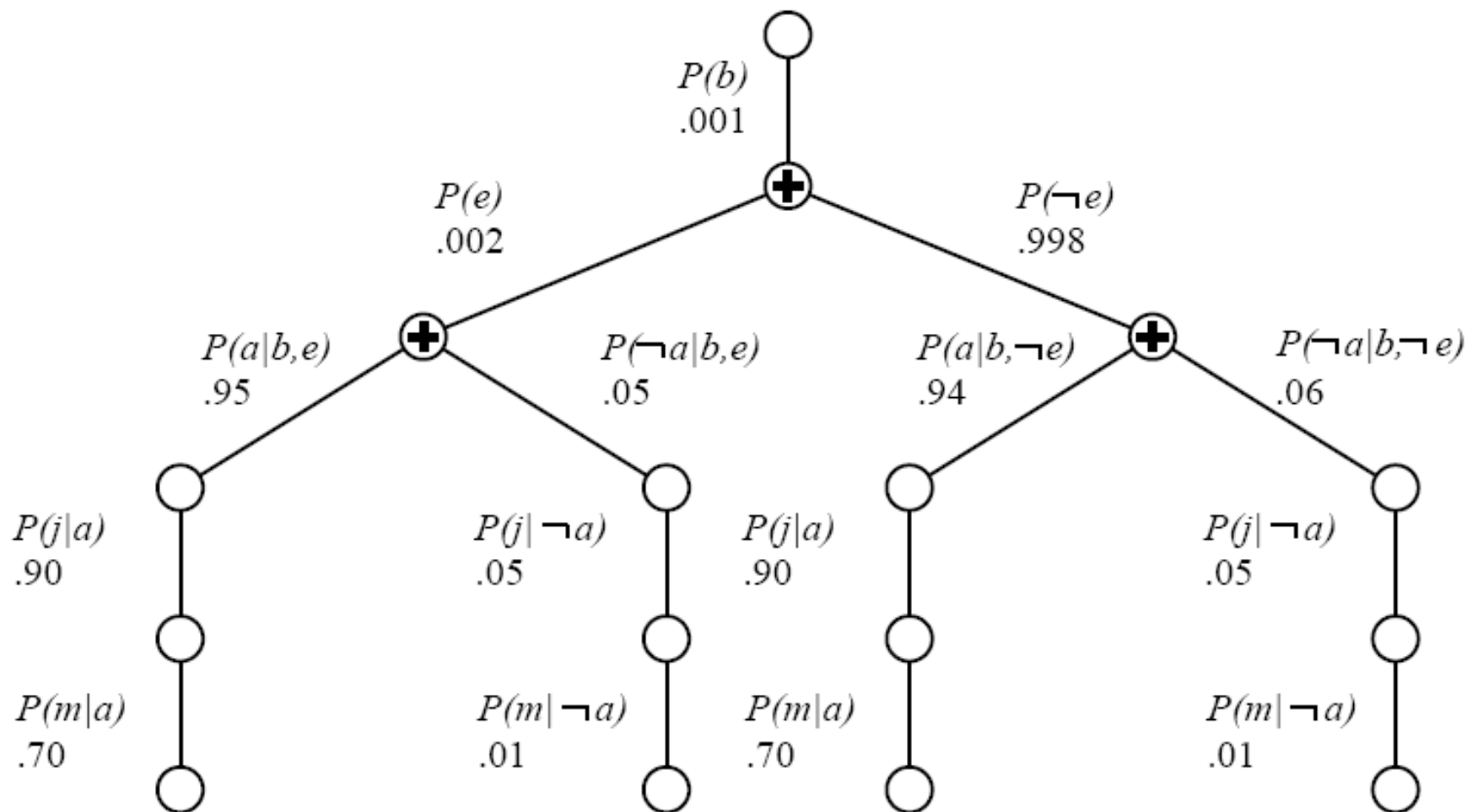
if Y has value y in e

then return $P(y \mid Pa(Y)) \times$ **ENUMERATE-ALL**(**REST**($vars$), e)

else return $\sum_y P(y \mid Pa(Y)) \times$ **ENUMERATE-ALL**(**REST**($vars$), e_y)

 where e_y is e extended with $Y = y$

Evaluation Tree



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

Variable Elimination

- 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: $\mathbf{P}(B|j, m)$

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

Factors

- A factor is a vector / matrix containing all probabilities for all dependent variables
- Examples:

- $\mathbf{f}_M(A) = \begin{pmatrix} P(m | a) \\ P(m | \neg a) \end{pmatrix}$

- The factor $\mathbf{f}_A(A, B, E)$ is a $2 \times 2 \times 2$ matrix

Basic Operations

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

$$\begin{aligned}\sum_x \mathbf{f}_1 \times \dots \times \mathbf{f}_k &= \mathbf{f}_1 \times \dots \times \mathbf{f}_i \times \sum_x \mathbf{f}_{i+1} \times \dots \times \mathbf{f}_k \\ &= \mathbf{f}_1 \times \dots \times \mathbf{f}_i \times \mathbf{f}_{\bar{X}}\end{aligned}$$

assuming $\mathbf{f}_1, \dots, \mathbf{f}_i$ 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: $\mathbf{f}_1(X_1, \dots, X_j, Y_1, \dots, Y_k) \times \mathbf{f}_2(Y_1, \dots, Y_k, Z_1, \dots, Z_l) = \mathbf{f}(X_1, \dots, X_j, Y_1, \dots, Y_k, Z_1, \dots, Z_l)$
 - has 2^{j+k+l} entries (if all variables are binary)

Example: Pointwise Product

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

Variable Elimination Algorithm

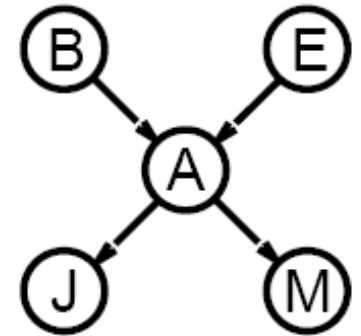
```
function ELIMINATION-ASK( $X, \mathbf{e}, bn$ ) returns a distribution over  $X$   
inputs:  $X$ , the query variable  
          $\mathbf{e}$ , evidence specified as an event  
          $bn$ , a belief network specifying joint distribution  $\mathbf{P}(X_1, \dots, X_n)$   
 $factors \leftarrow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

Irrelevant Variables

Consider the query $P(\text{JohnCalls} | \text{Burglary} = \text{true})$

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

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



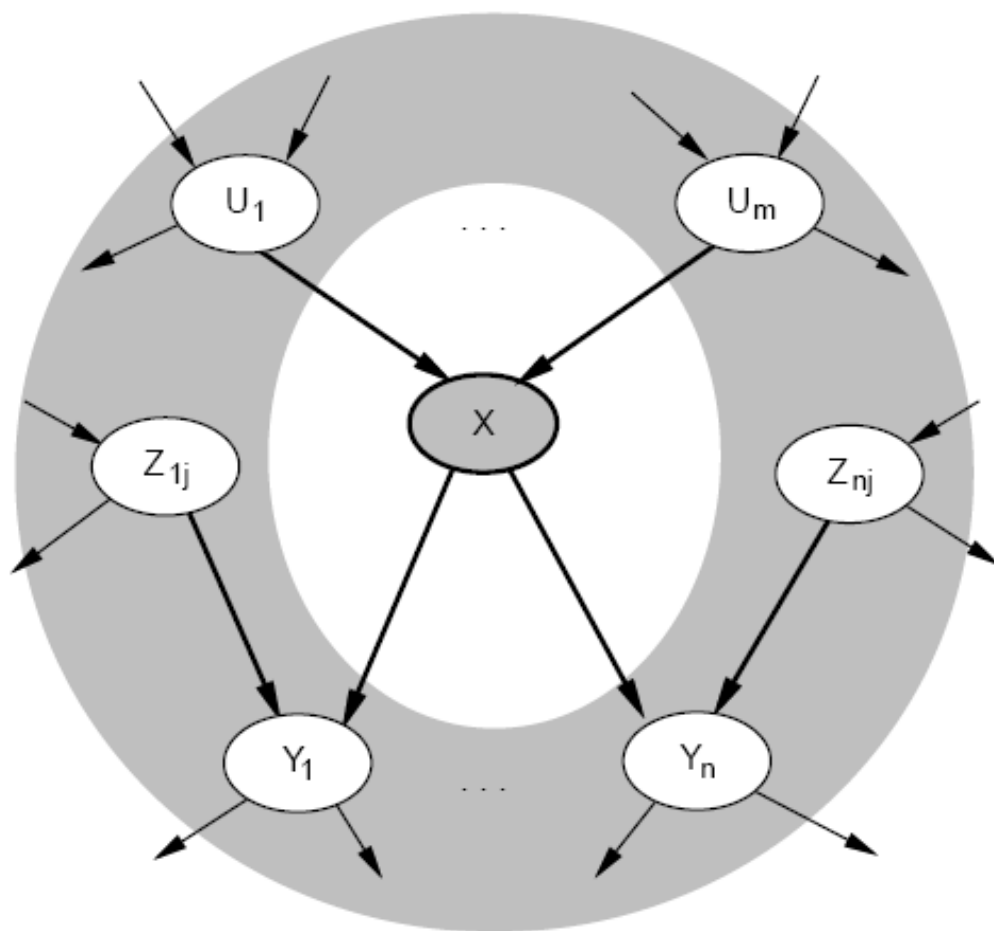
Thm 1: Y is irrelevant unless $Y \in \text{Ancestors}(\{X\} \cup \mathbf{E})$

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

(Compare this to backward chaining from the query in Horn clause KBs)

Markov Blanket

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

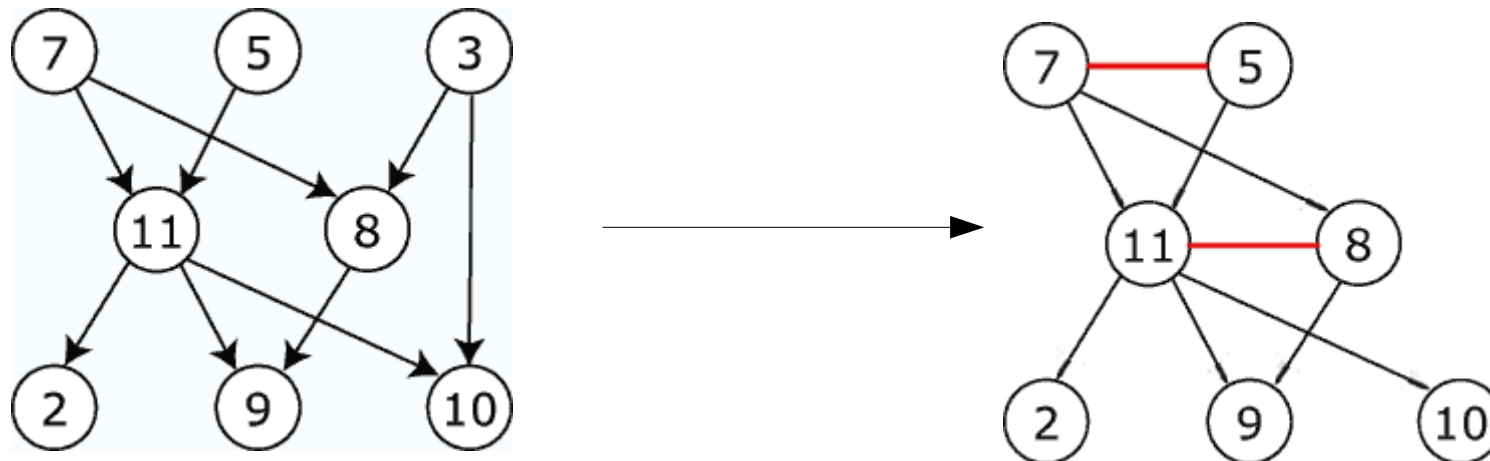


- Each node is conditionally independent of all other nodes given its markov blanket

$$\begin{aligned} \mathbf{P}(X \mid U_1, \dots, U_m, Y_1, \dots, Y_n, Z_{1j}, \dots, Z_{nj}) = \\ = \mathbf{P}(X \mid \textit{all variables}) \end{aligned}$$

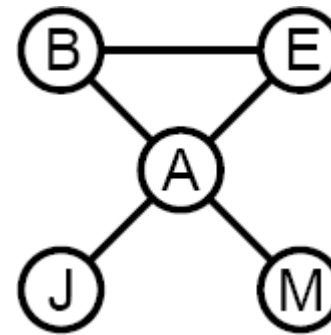
Moral Graph

- 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



Moral Graph and Irrelevant Variables

- m-separation:
 - A is **m-separated** from B by C iff it is separated by C 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 E

- Example:

For $P(\text{JohnCalls} | \text{Alarm} = \text{true})$, both *Burglary* and *Earthquake* are irrelevant

Complexity of Exact Inference

Singly connected networks (or polytrees):

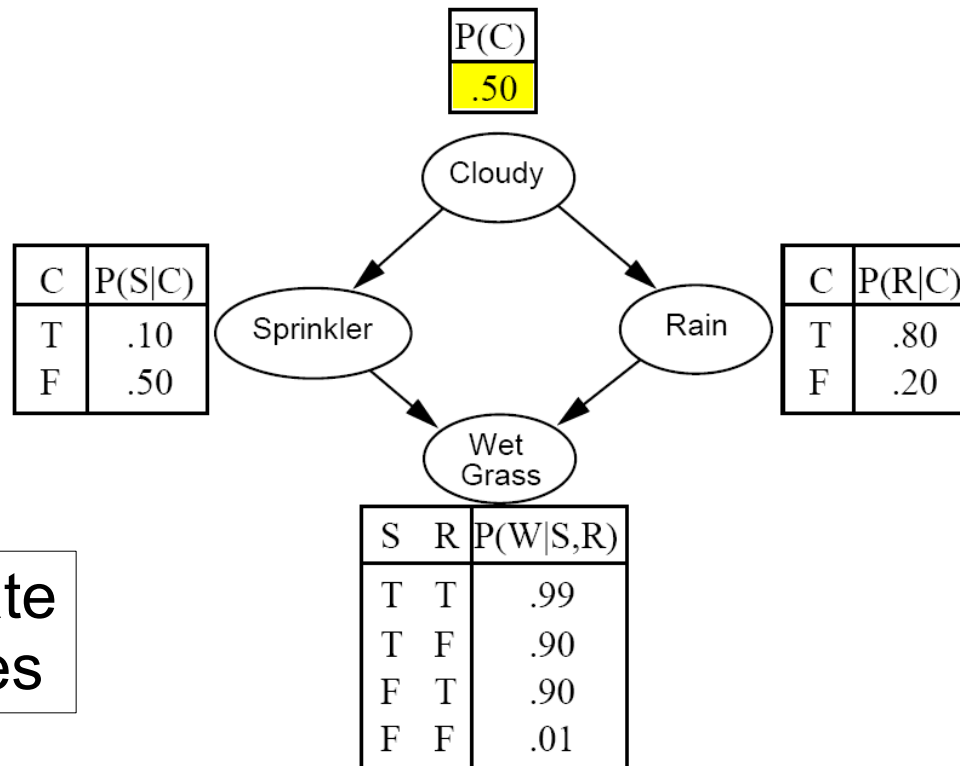
- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^k n)$

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

Inference by Stochastic Simulation

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:

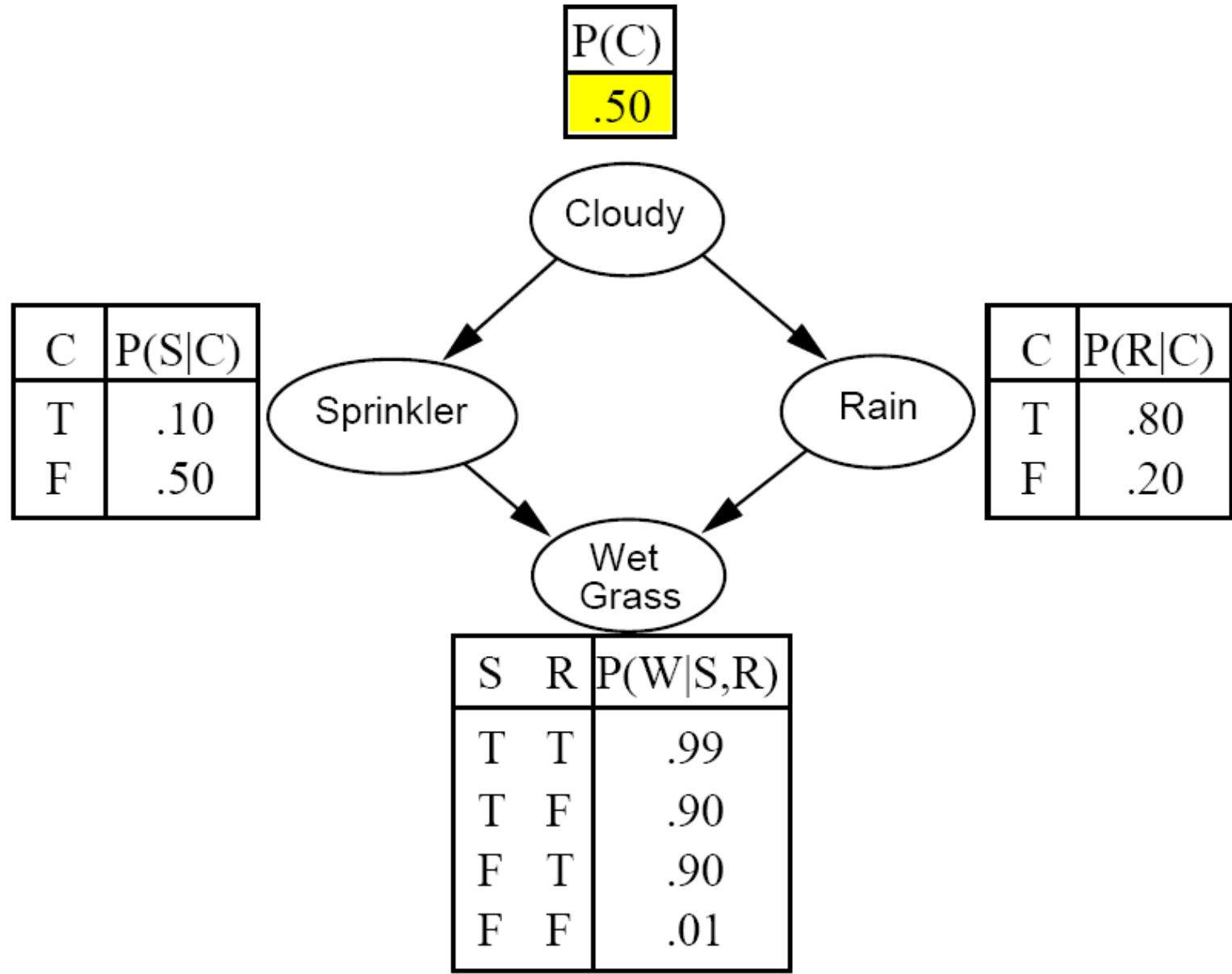
- Sampling from an empty network
- Rejection sampling: reject samples disagreeing with evidence
- Likelihood weighting: use evidence to weight samples
- Markov chain Monte Carlo (MCMC): sample from a stochastic process whose stationary distribution is the true posterior

Sampling from an Empty Network

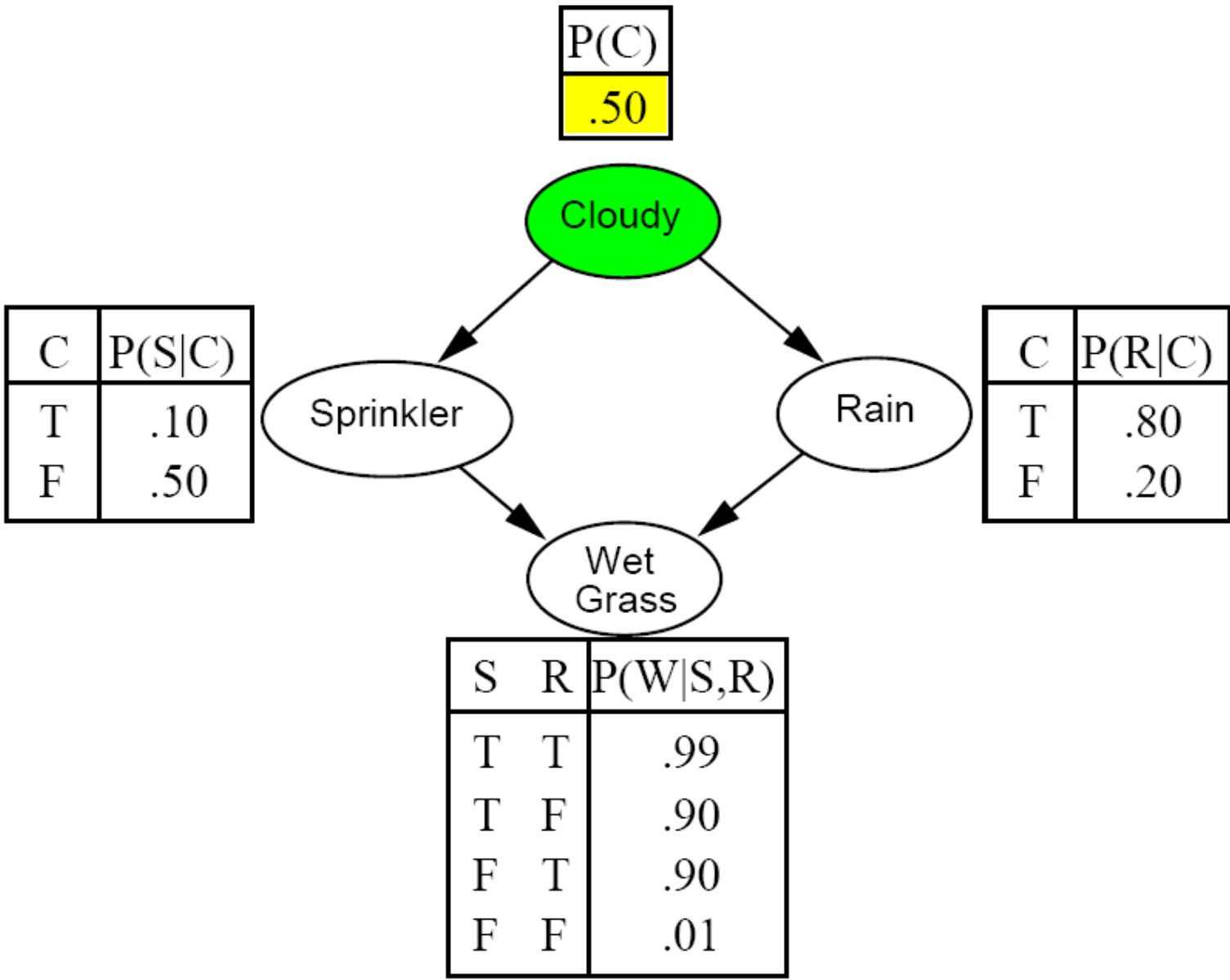
- Generating samples from a network that has no evidence associated with it (*empty* network)
- Basic idea
 - sample a value for each variable in topological order
 - using the specified conditional probabilities

```
function PRIOR-SAMPLE(bn) returns an event sampled from bn
  inputs: bn, a belief network specifying joint distribution  $\mathbf{P}(X_1, \dots, X_n)$ 
   $\mathbf{x} \leftarrow$  an event with  $n$  elements
  for  $i = 1$  to  $n$  do
     $x_i \leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$ 
      given the values of  $\text{Parents}(X_i)$  in  $\mathbf{x}$ 
  return  $\mathbf{x}$ 
```

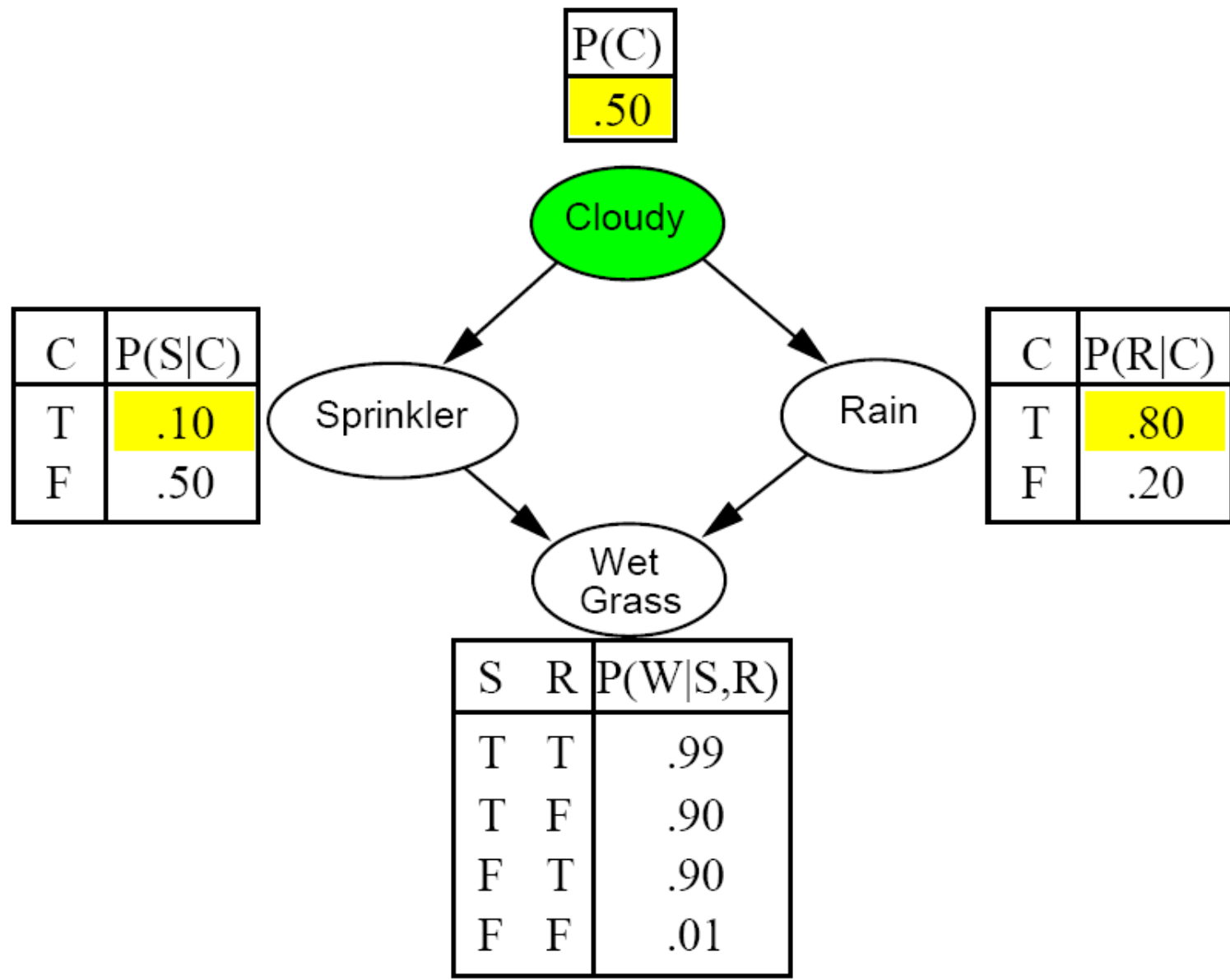
Example



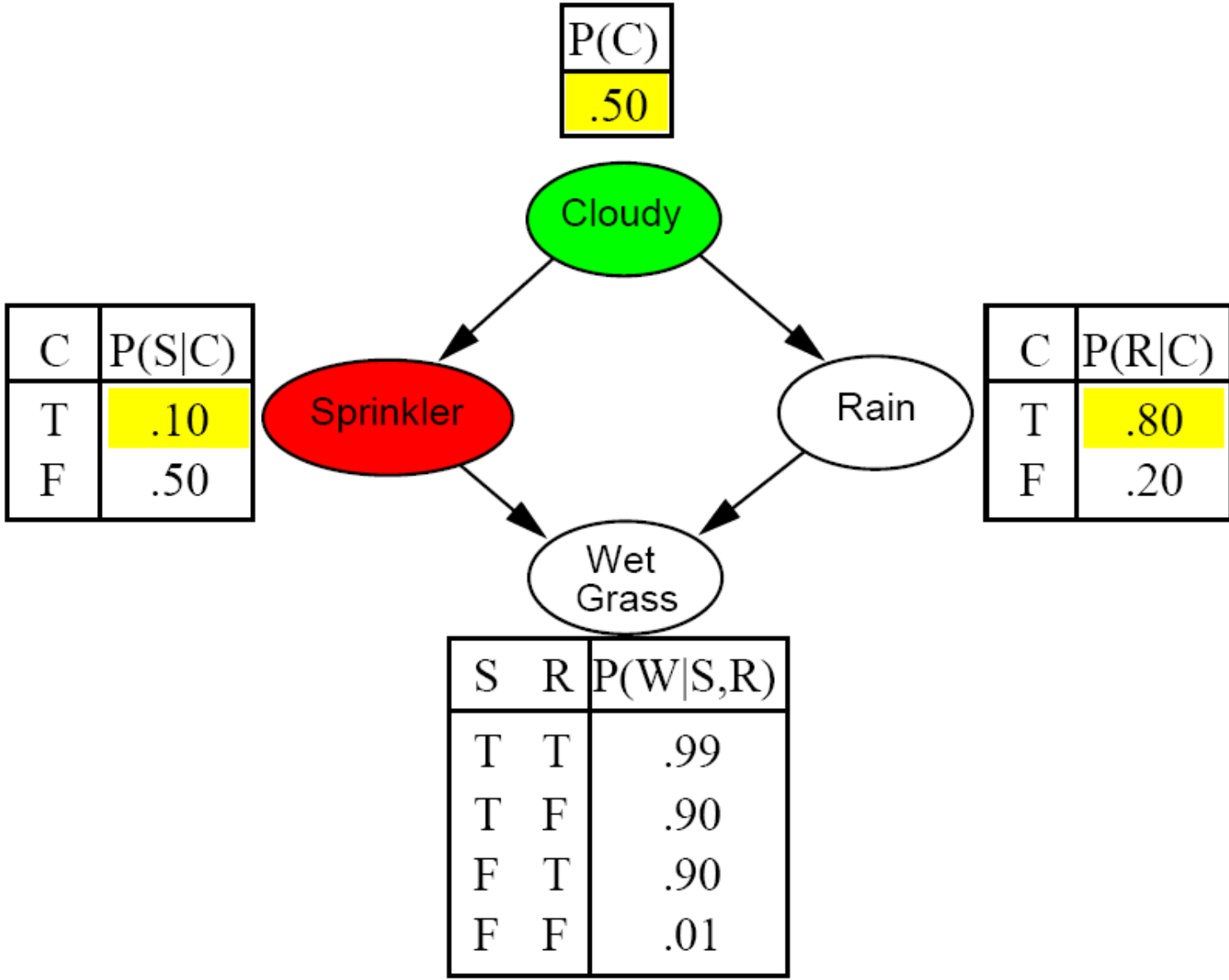
Example



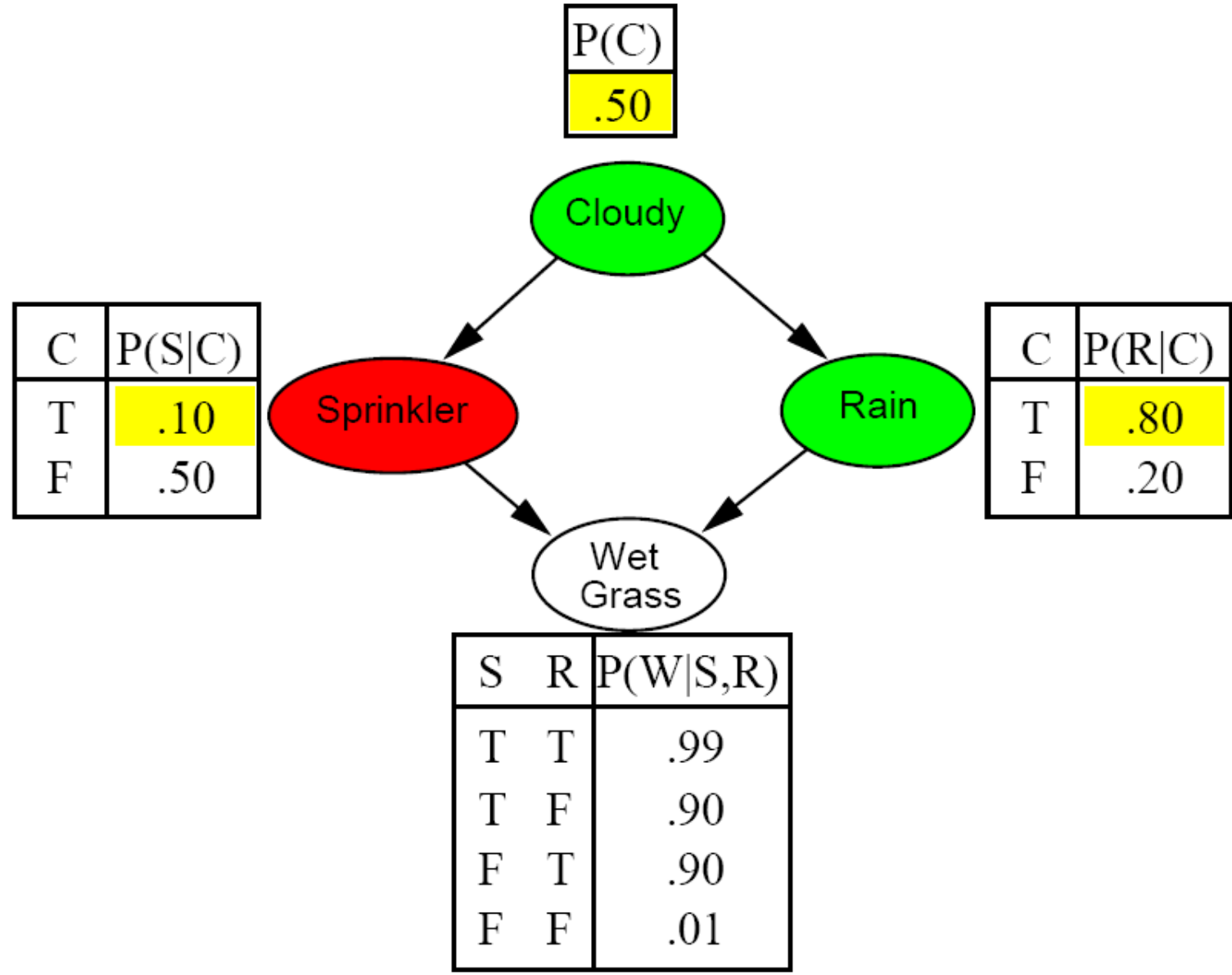
Example



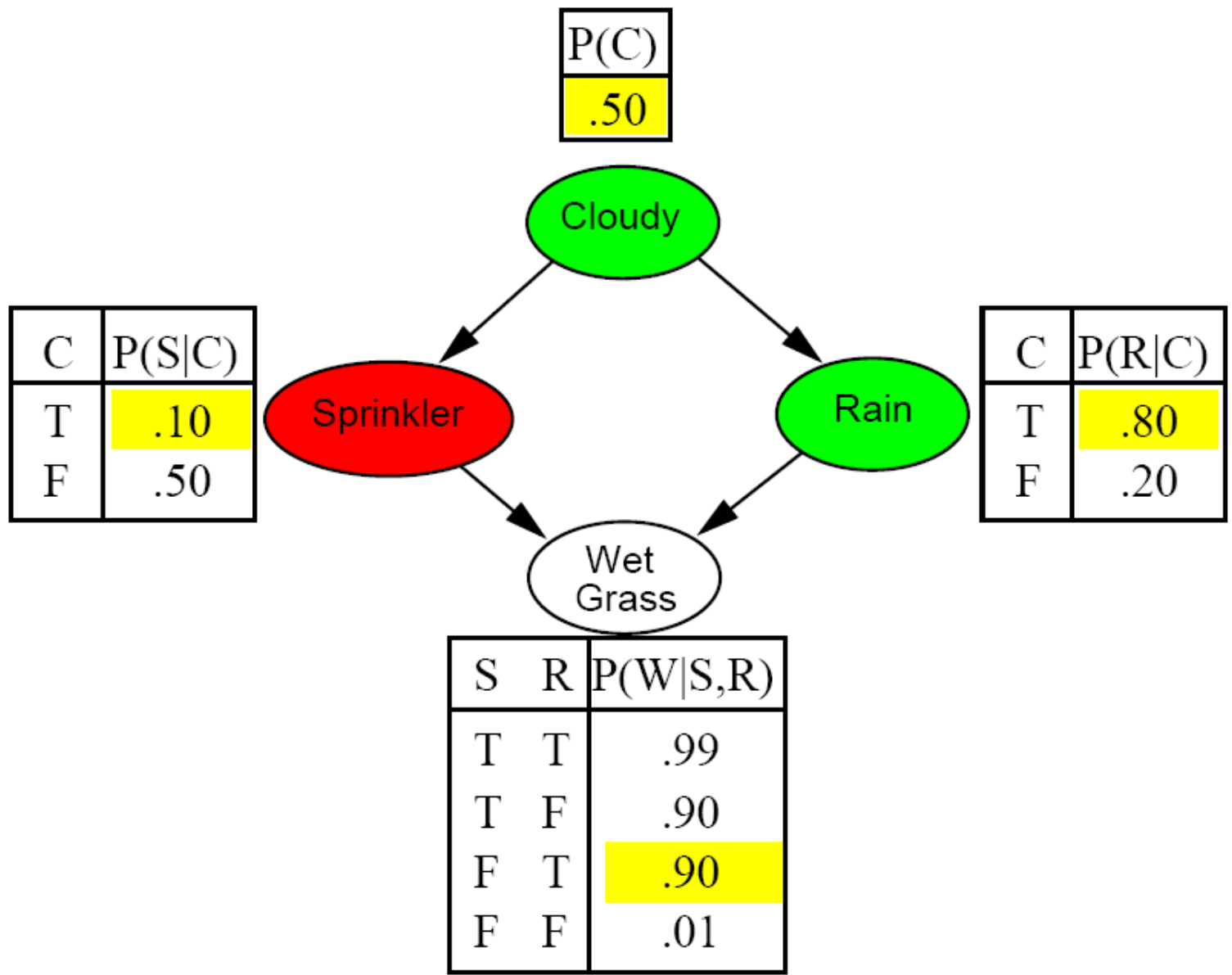
Example



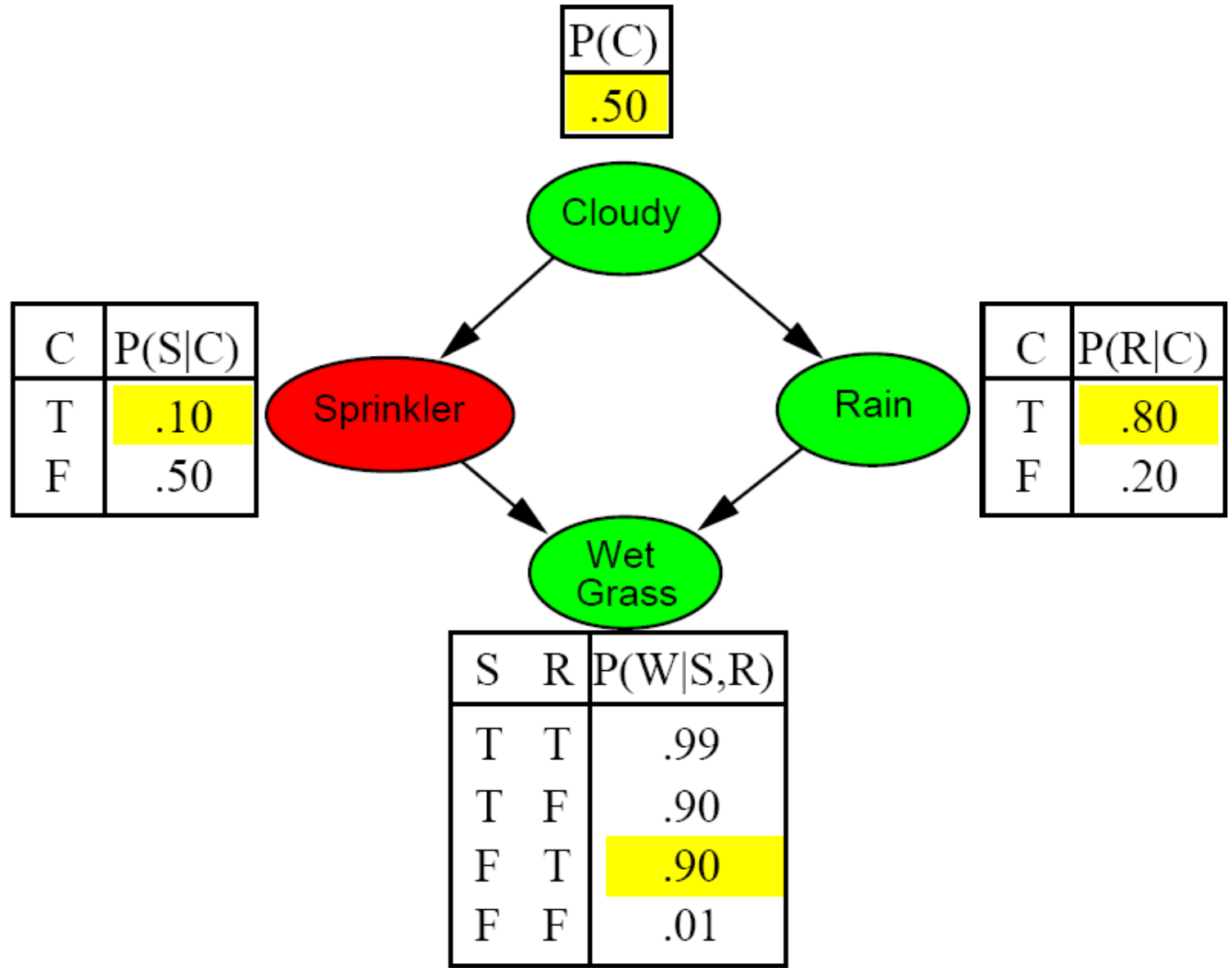
Example



Example



Example



Convergence of Sampling from an Empty Network

Probability that PRIORSAMPLE generates a particular event

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

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

Then we have

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

That is, estimates derived from PRIORSAMPLE are **consistent**

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

Rejection Sampling

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

```

function REJECTION-SAMPLING( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $P(X|\mathbf{e})$ 
  local variables:  $\mathbf{N}$ , a vector of counts over  $X$ , initially zero
  for  $j = 1$  to  $N$  do
     $\mathbf{x} \leftarrow$  PRIOR-SAMPLE( $bn$ )
    if  $\mathbf{x}$  is consistent with  $\mathbf{e}$  then
       $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$  where  $x$  is the value of  $X$  in  $\mathbf{x}$ 
  return NORMALIZE( $\mathbf{N}[X]$ )
  
```

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

- Problem
 - many unnecessary samples will be generated if the probability of observing the evidence e is small
 - $\mathbf{P}(e)$ 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

```

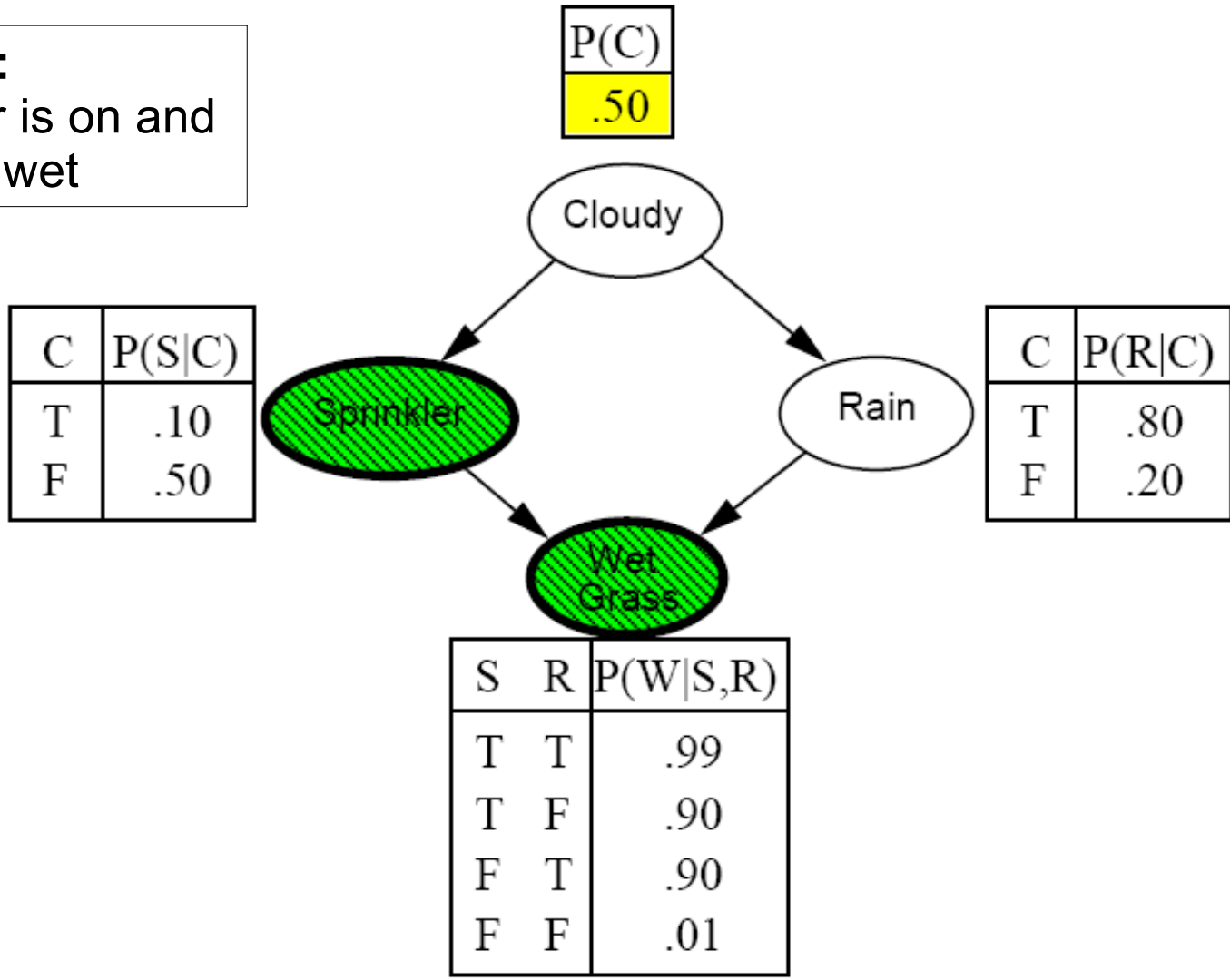
function LIKELIHOOD-WEIGHTING( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $P(X|\mathbf{e})$ 
  local variables:  $\mathbf{W}$ , a vector of weighted counts over  $X$ , initially zero
  for  $j = 1$  to  $N$  do
     $\mathbf{x}, w \leftarrow$  WEIGHTED-SAMPLE( $bn$ )
     $\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$  where  $x$  is the value of  $X$  in  $\mathbf{x}$ 
  return NORMALIZE( $\mathbf{W}[X]$ )
  
```

```

function WEIGHTED-SAMPLE( $bn, \mathbf{e}$ ) returns an event and a weight
   $\mathbf{x} \leftarrow$  an event with  $n$  elements;  $w \leftarrow 1$ 
  for  $i = 1$  to  $n$  do
    if  $X_i$  has a value  $x_i$  in  $\mathbf{e}$ 
      then  $w \leftarrow w \times P(X_i = x_i \mid \text{parents}(X_i))$ 
      else  $x_i \leftarrow$  a random sample from  $\mathbf{P}(X_i \mid \text{parents}(X_i))$ 
  return  $\mathbf{x}, w$ 
  
```

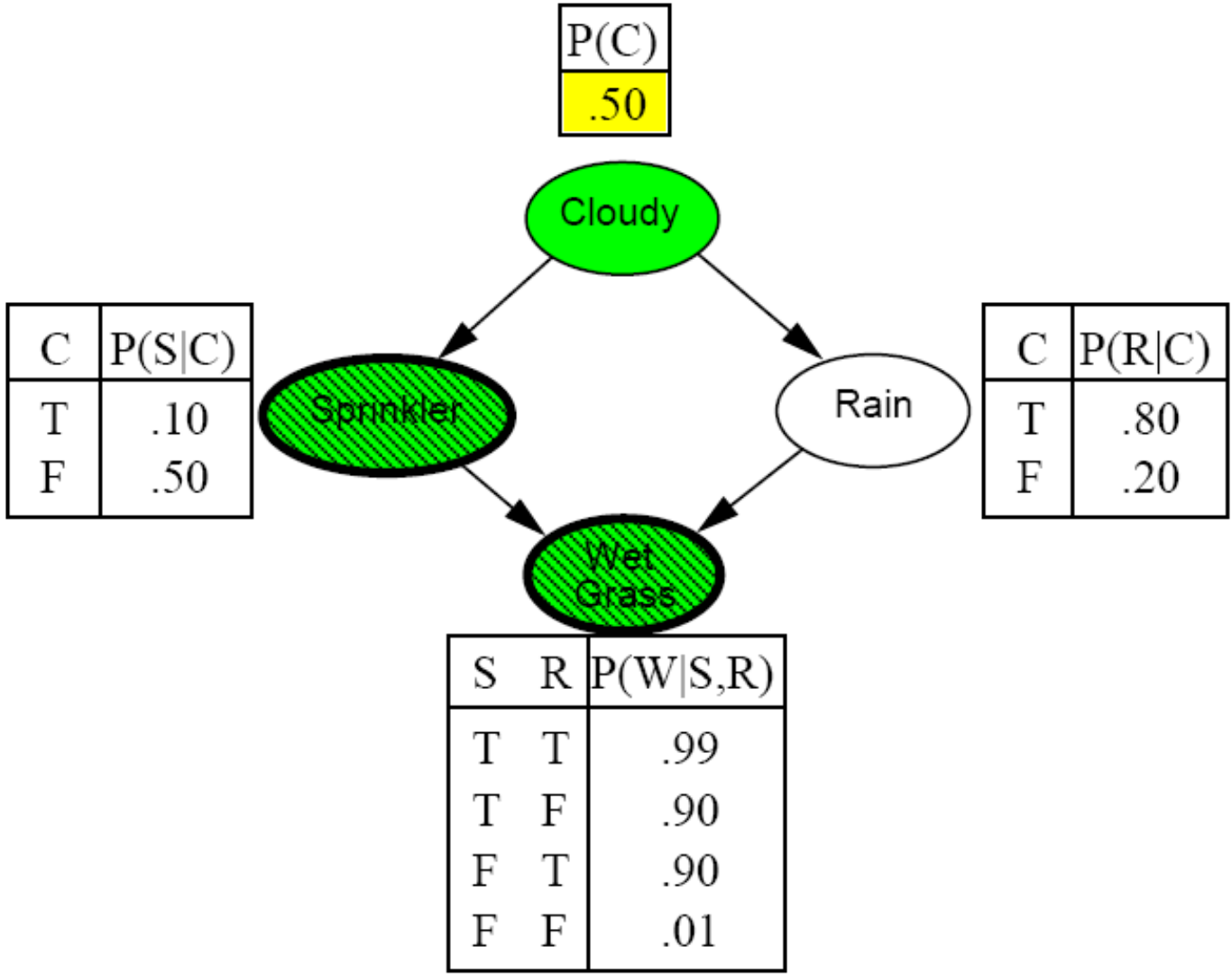
Example

Evidence:
 sprinkler is on and
 grass is wet



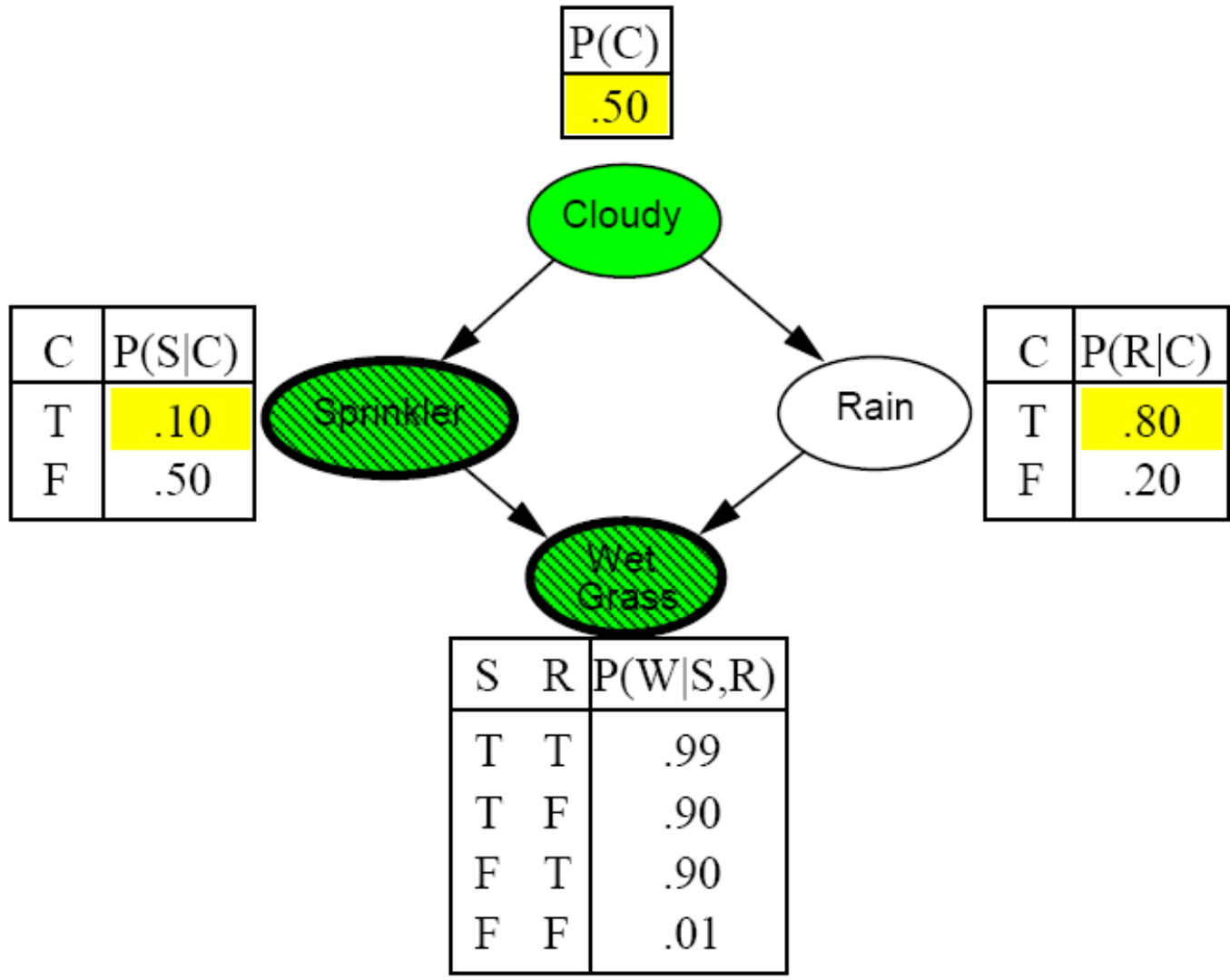
$w = 1.0$

Example



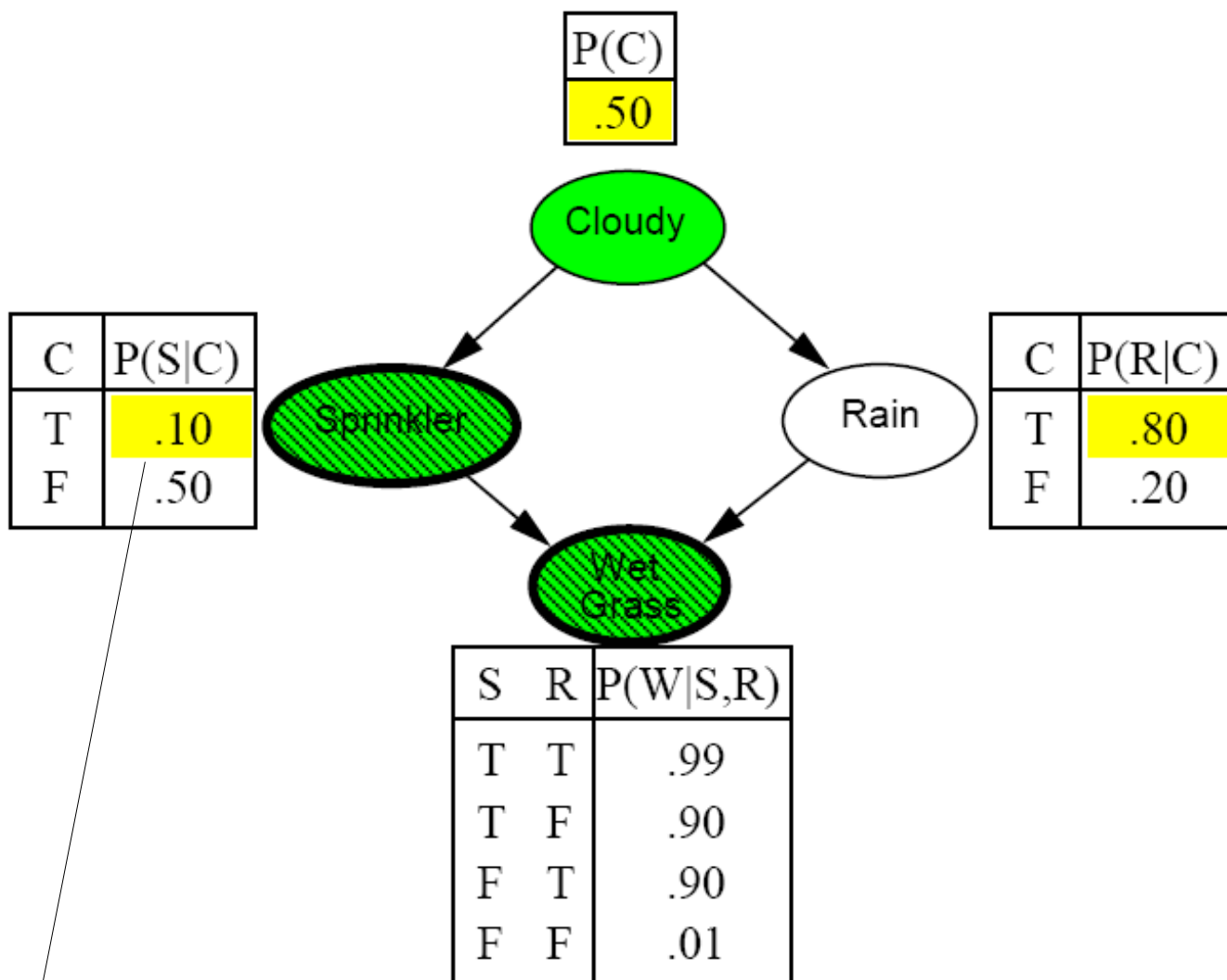
$w = 1.0$

Example



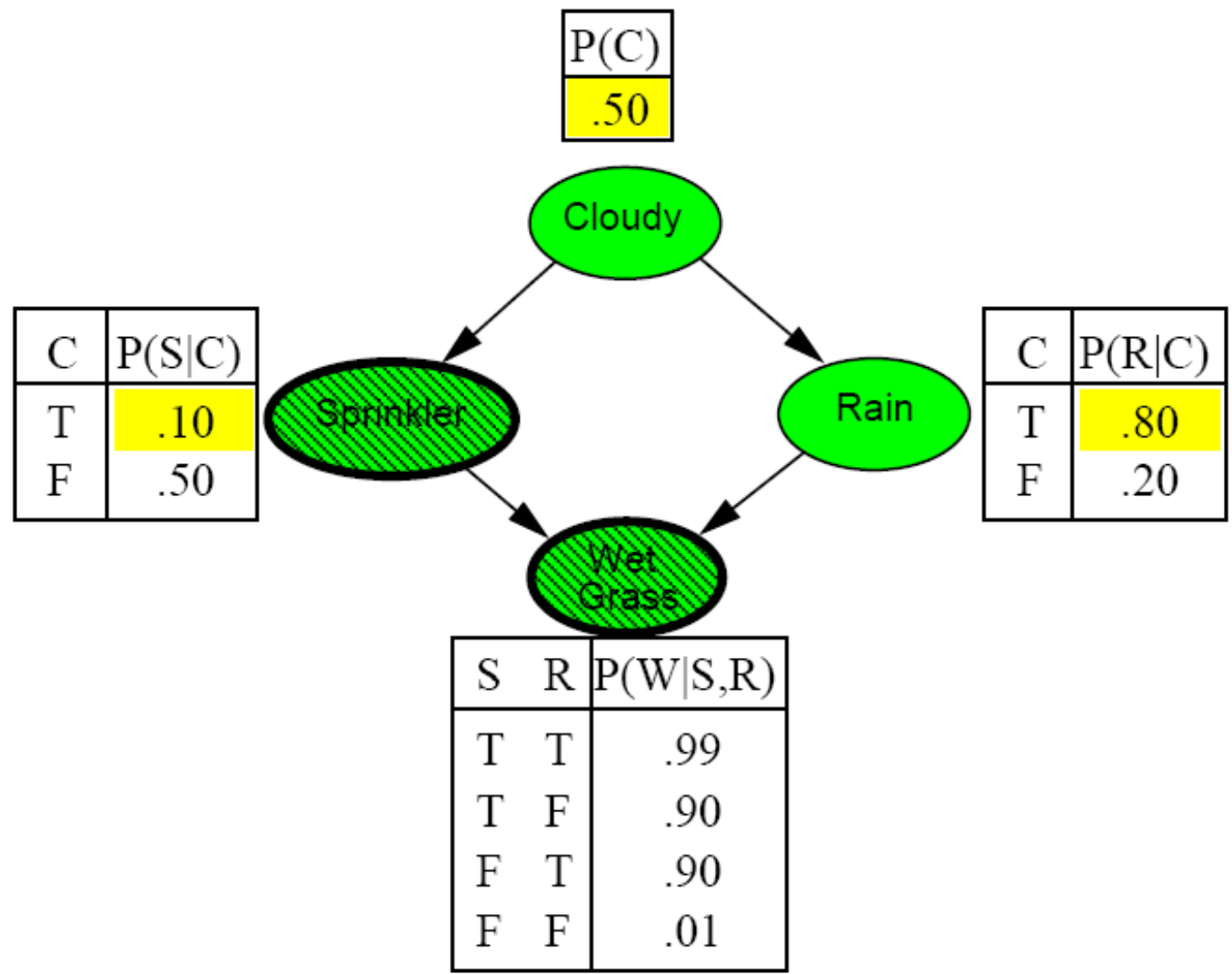
$w = 1.0$

Example



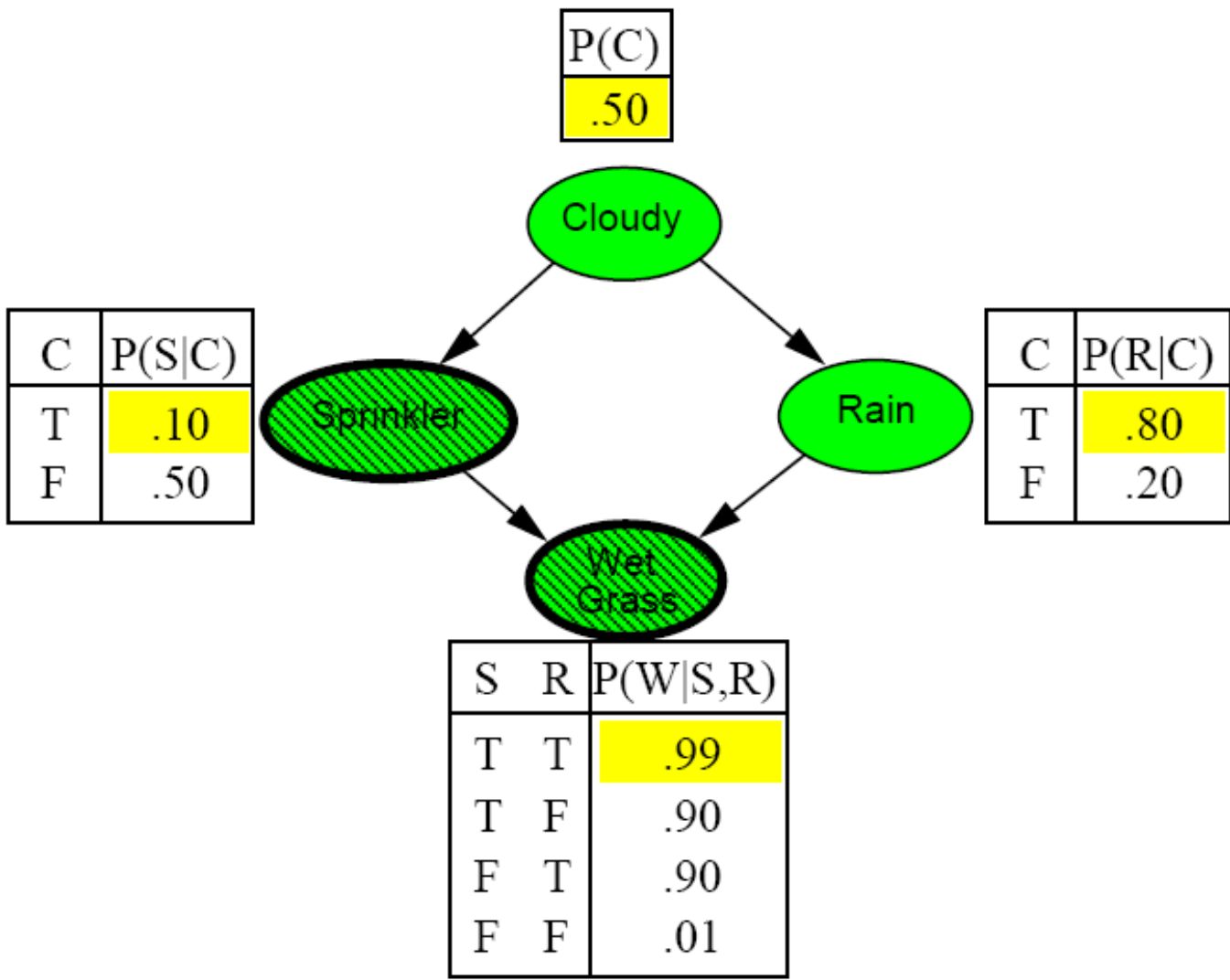
$$w = 1.0 \times 0.1$$

Example



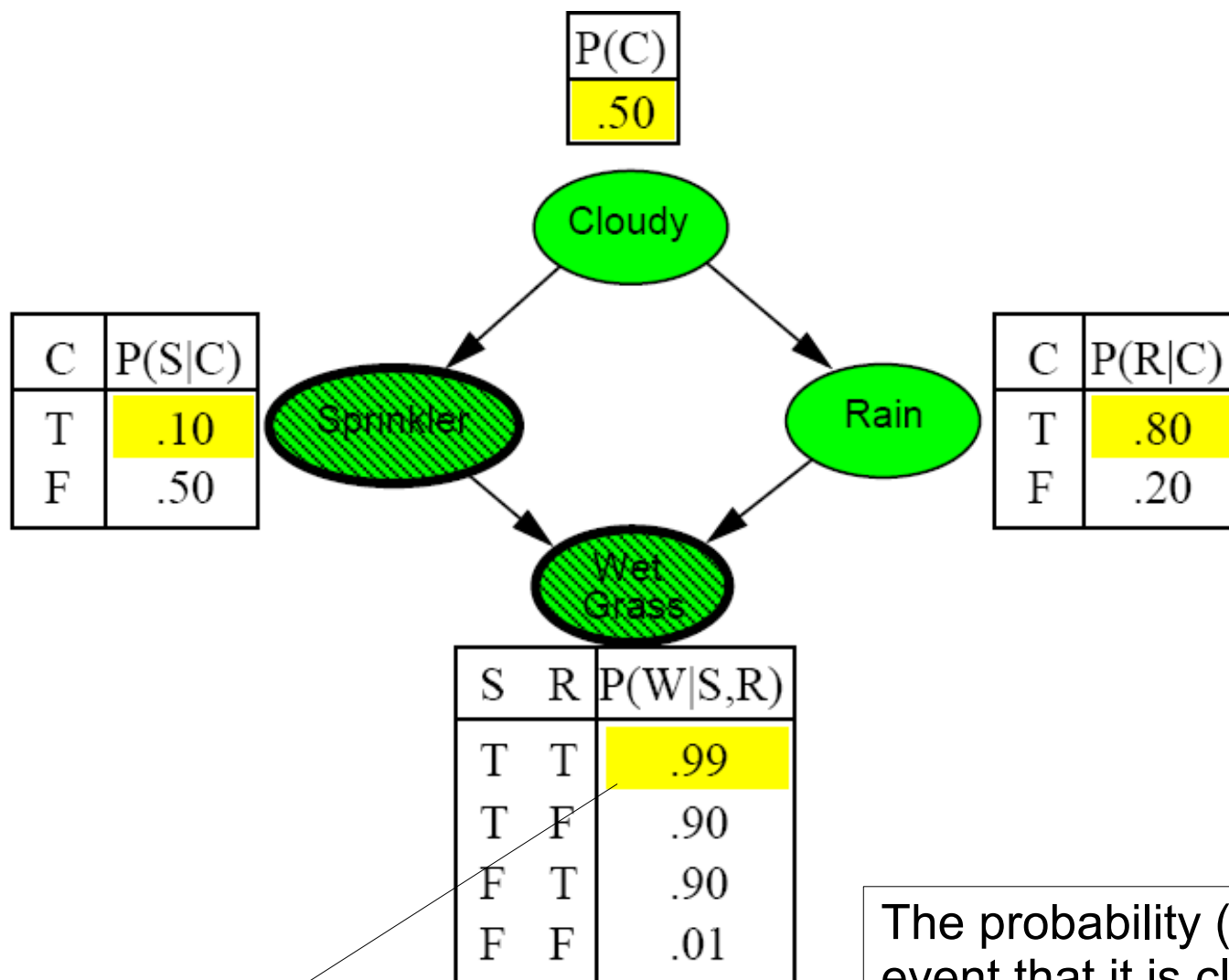
$$w = 1.0 \times 0.1$$

Example



$$w = 1.0 \times 0.1$$

Example



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

The probability (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

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

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

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

Weighted sampling probability is

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

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( $X, \mathbf{e}, bn, N$ ) returns an estimate of  $P(X|\mathbf{e})$ 
  local variables:  $\mathbf{N}[X]$ , a vector of counts over  $X$ , initially zero
                     $\mathbf{Z}$ , the nonevidence variables in  $bn$ 
                     $\mathbf{x}$ , the current state of the network, initially copied from  $\mathbf{e}$ 

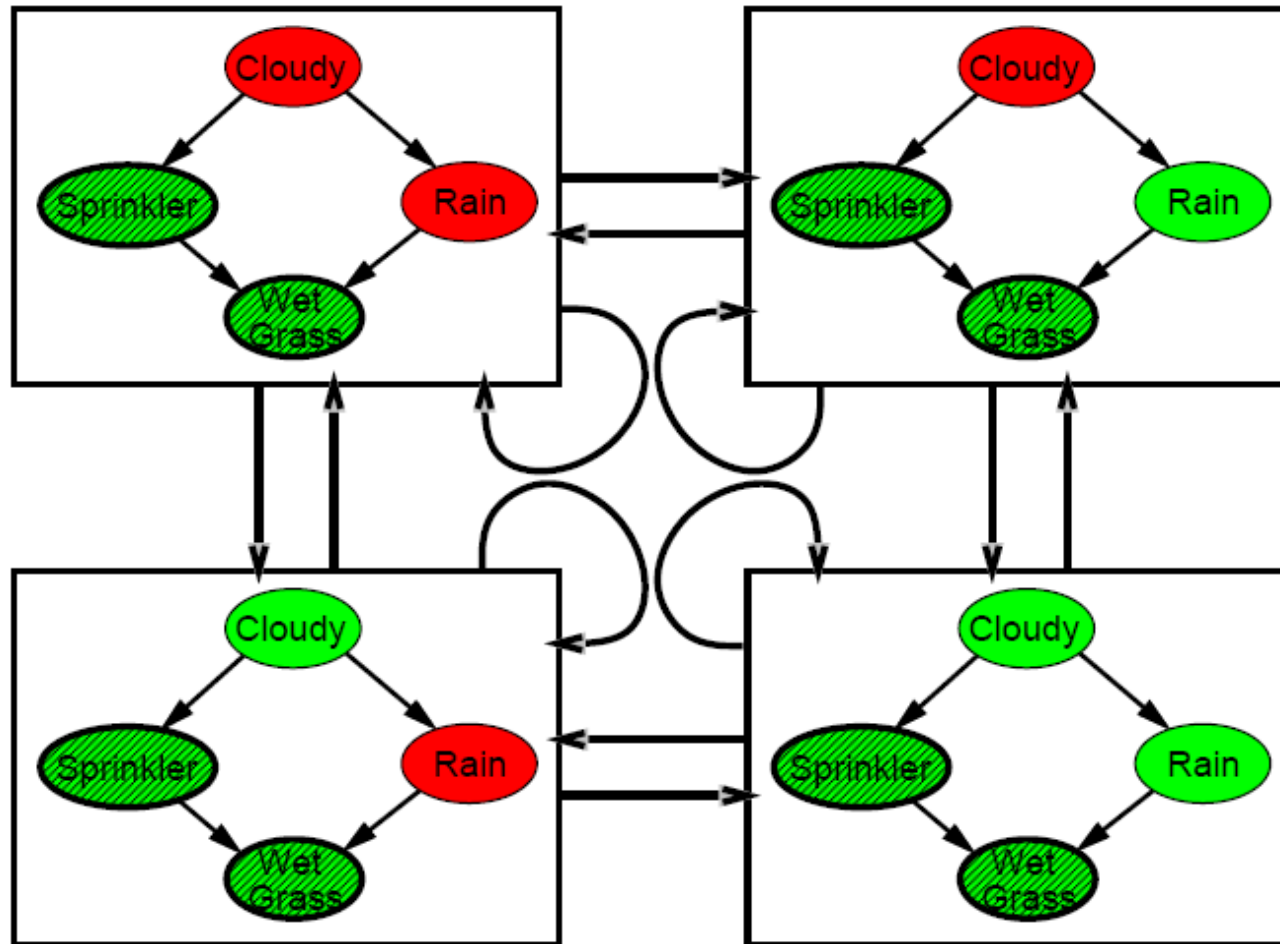
  initialize  $\mathbf{x}$  with random values for the variables in  $\mathbf{Y}$ 
  for  $j = 1$  to  $N$  do
    for each  $Z_i$  in  $\mathbf{Z}$  do
      sample the value of  $Z_i$  in  $\mathbf{x}$  from  $\mathbf{P}(Z_i|mb(Z_i))$ 
      given the values of  $MB(Z_i)$  in  $\mathbf{x}$ 
       $\mathbf{N}[x]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

Gibbs Sampling

Can also choose a variable to sample at random each time

The Markov Chain

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



Wander about for a while, average what you see

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*

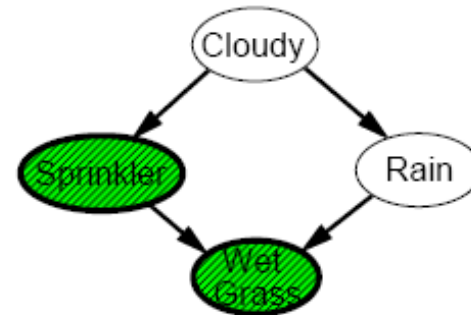
$$\hat{\mathbf{P}}(Rain|Sprinkler = true, WetGrass = true) \\ = \text{NORMALIZE}(\langle 31, 69 \rangle) = \langle 0.31, 0.69 \rangle$$

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:

$$P(x'_i | mb(X_i)) = P(x'_i | parents(X_i)) \prod_{Z_j \in Children(X_i)} P(z_j | parents(Z_j))$$

Main computational problems:

- 1) Difficult to tell if convergence has been achieved
- 2) Can be wasteful if Markov blanket is large:

$P(X_i | mb(X_i))$ won't change much (law of large numbers)