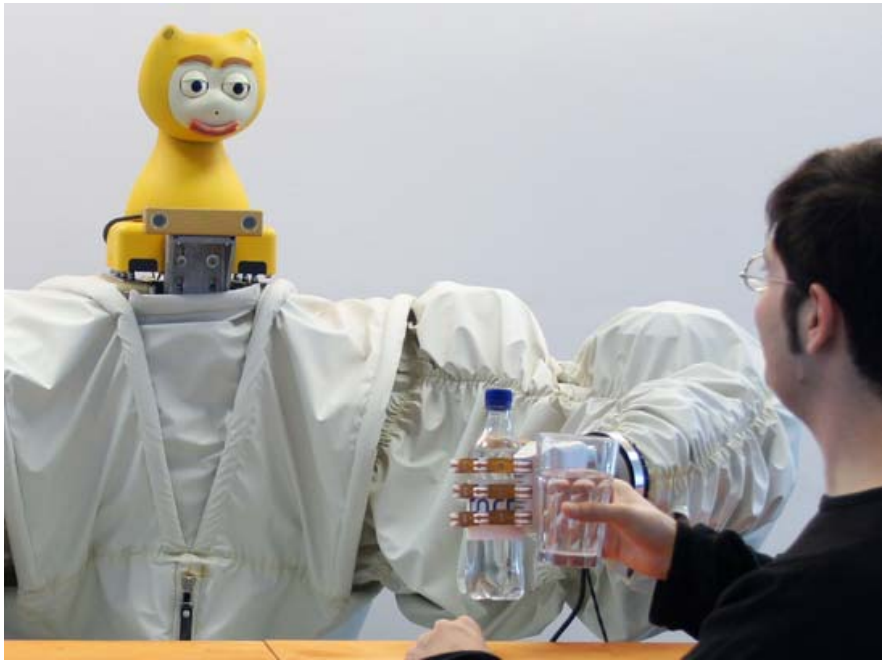
# Vorlesung

# Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055



Wintersemester 2012/13 25.1.2013

# Chapter 14 (cont‘d) + 15

# Probabilistic Reasoning / time

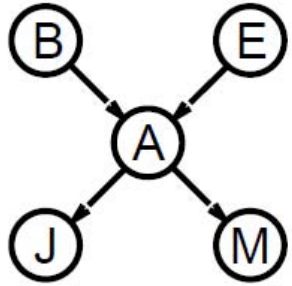
**with material from Russel/Norvig original slides and Michael Beetz**

# Bayesian networks - inference

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:

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

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

# Exact inference - algorithm

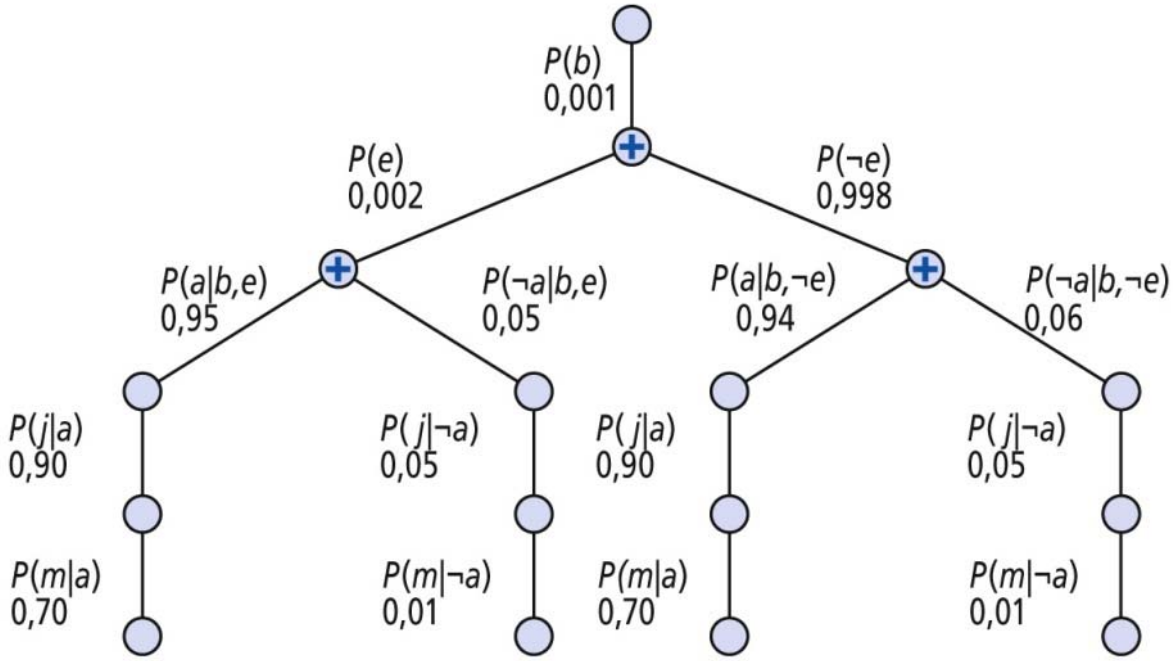
**function** ENUMERATION-ASK($X, \mathbf{e}, bn$) **returns** a distribution over $X$
  **inputs**: $X$, the query variable
    $\mathbf{e}$, observed values for variables $\mathbf{E}$
    $bn$, a Bayesian network with variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$

  $\mathbf{Q}(X) \leftarrow$ a distribution over $X$, initially empty
  **for each** value $x_i$ of $X$ **do**
    extend $\mathbf{e}$ with value $x_i$ for $X$
    $\mathbf{Q}(x_i) \leftarrow$ ENUMERATE-ALL(VARS[$bn$], $\mathbf{e}$)
  **return** NORMALIZE($\mathbf{Q}(X)$)

---

**function** ENUMERATE-ALL($vars, \mathbf{e}$) **returns** a real number
  **if** EMPTY?($vars$) **then return** 1.0
  $Y \leftarrow$ FIRST($vars$)
  **if** $Y$ has value $y$ in $\mathbf{e}$
    **then return** $P(y \mid Pa(Y)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}$)
    **else return** $\Sigma_y \; P(y \mid Pa(Y)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_y$)
      where $\mathbf{e}_y$ is $\mathbf{e}$ 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

# Inference by variable elimination

Variable elimination: carry out summations right-to-left, storing intermediate results (factors) to avoid recomputation

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

# Variable elimination: 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

$$\sum_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \sum_x f_{i+1} \times \cdots \times f_k = f_1 \times \cdots \times f_i \times f_{\bar{X}}$$

assuming $f_1, \ldots, f_i$ do not depend on $X$

Pointwise product of factors $f_1$ and $f_2$:

$$f_1(x_1, \ldots, x_j, y_1, \ldots, y_k) \times f_2(y_1, \ldots, y_k, z_1, \ldots, z_l)$$
$$= f(x_1, \ldots, x_j, y_1, \ldots, y_k, z_1, \ldots, z_l)$$

E.g., $f_1(a, b) \times f_2(b, c) = f(a, b, c)$

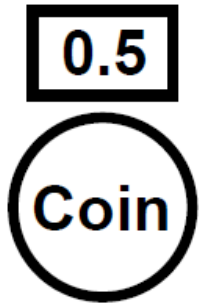
# Approximate inference

- In general, inference in Bayesian networks is NP-hard
- For polytrees, exact inference has linear time and space complexity.
- For all other network topologies, approximate algorithms are needed

# 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

**function** PRIOR-SAMPLE($bn$) **returns** an event sampled from $bn$

**inputs**: $bn$, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

$\mathbf{x} \leftarrow$ an event with $n$ elements

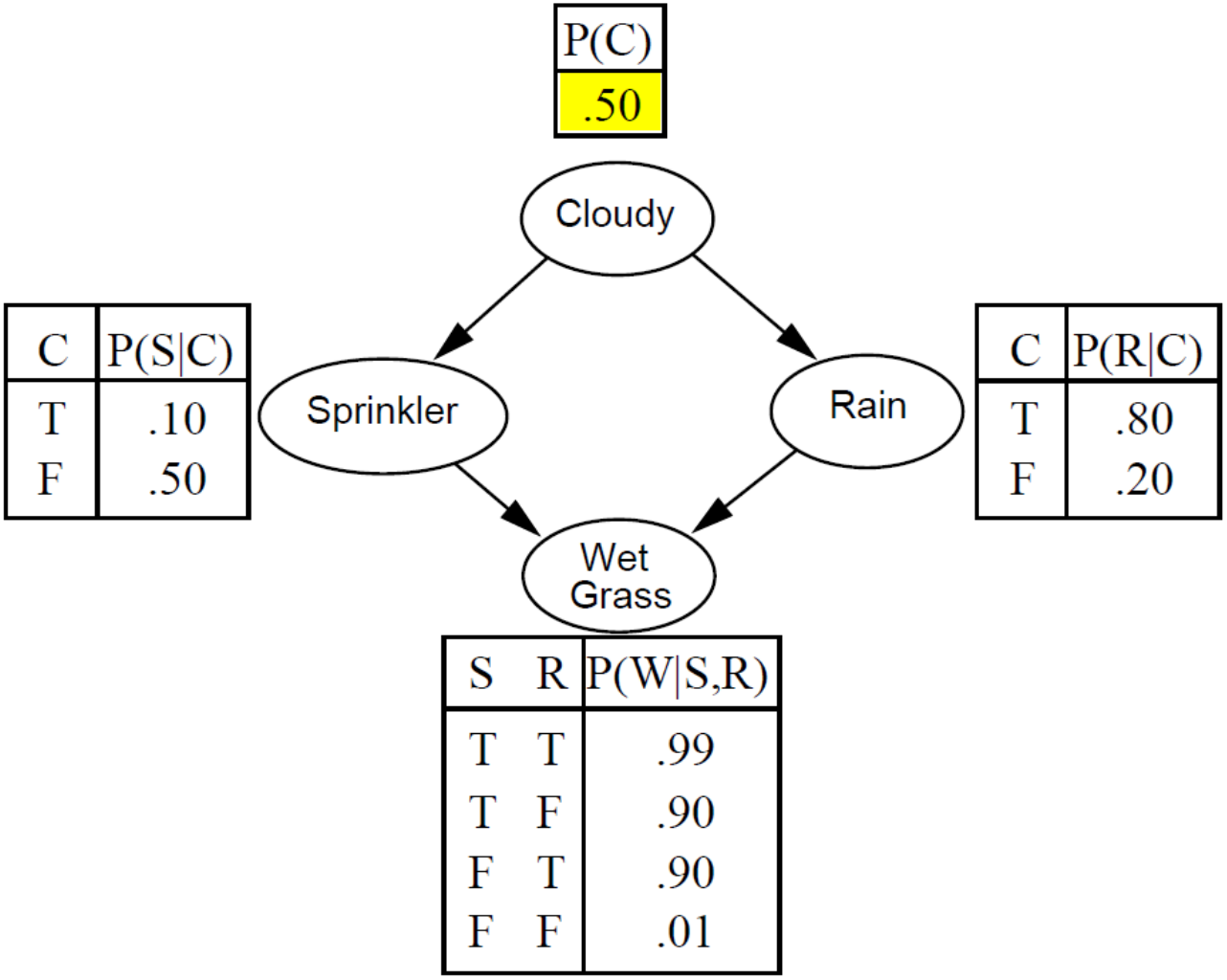
**for** $i = 1$ **to** $n$ **do**

$x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$

given the values of $Parents(X_i)$ in $\mathbf{x}$

**return** $\mathbf{x}$

# 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

Idea: Use Bayesian Network as simulation of the world and count frequencies with which events occur

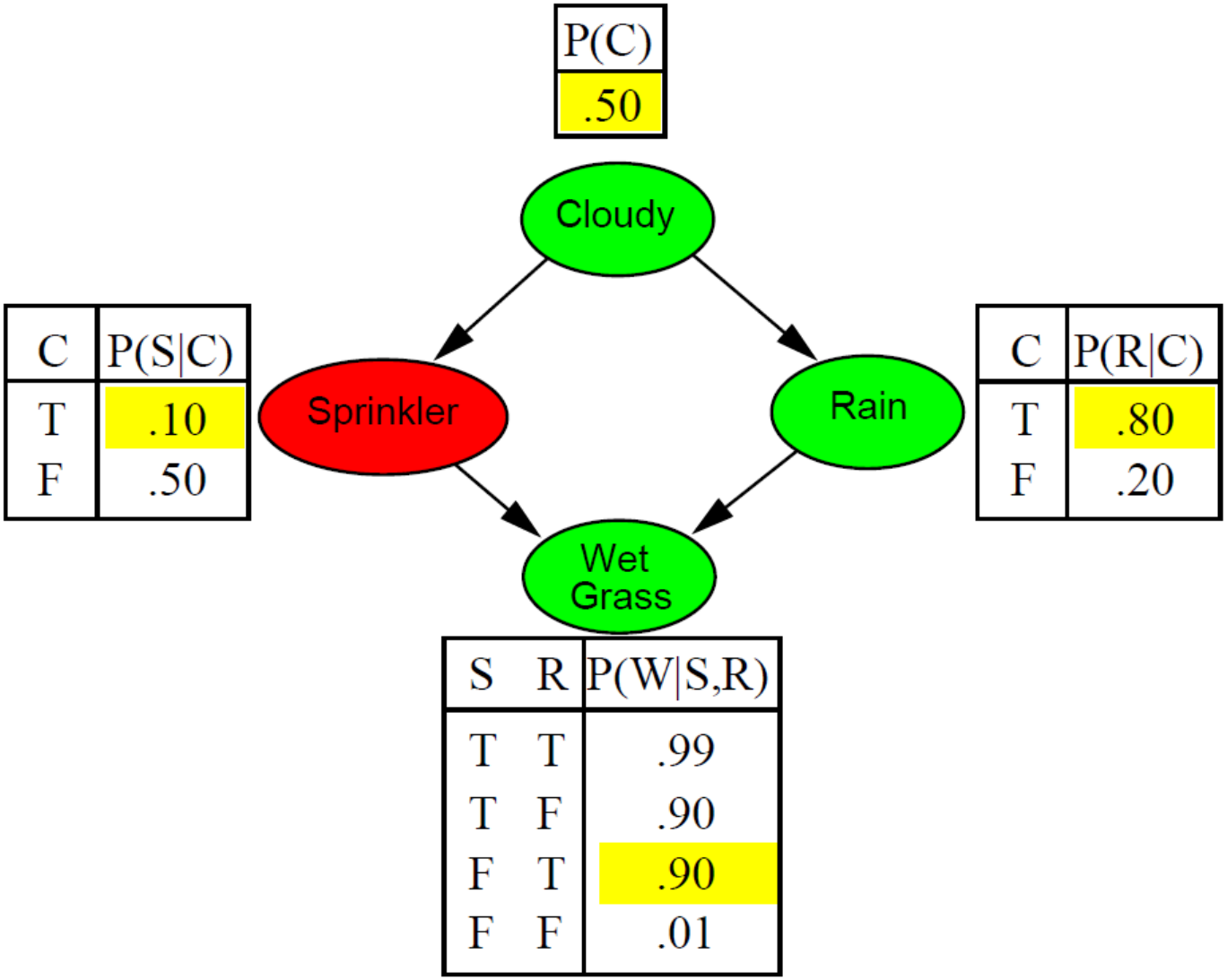
## Example

Topological order of nodes: [Cloudy,Sprinkler,Rain,WetGrass]

- Sample from $P(C) = \langle 0.5, 0.5 \rangle$ true
- Sample from $P(S|C = t) = \langle 0.1, 0.9 \rangle$ false
- Sample from $P(R|C = t) = \langle 0.8, 0.2 \rangle$ true
- Sample from $P(W|S = f, R = t) = \langle 0.9, 0.1 \rangle$ true

$\Rightarrow$ [true,false,true,true]

# 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

# Sampling from an empty network (cont'd)

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

## Algorithm

Find $P(Q|E)$

1. generate samples, rejecting those where $E$ is false
2. calculate $\frac{N(Q \wedge E)}{N(E)}$

($N(X)$: number of samples where X is true)

- converges to the correct value with increasing number of samples
- small number of samples for rare events (fraction of useful runs decreases exponentially with number of evidence variables)

# Likelihood weighting

- generate only "useful" samples
- weight samples according to their probability of occurence

## Example

Find sample for $P(Rain|Sprinkler = true, WetGrass = true)$
Set weight sample $w$ to 1.0

- Sample from $P(C) = \langle 0.5, 0.5 \rangle$ true
- *Sprinkler* is evidence variable with value true true
  $w \leftarrow w \cdot P(S = t|C = t) = 0.1$
- Sample from $P(R|C = t) = \langle 0.8, 0.2 \rangle$ true
- *WetGrass* is evidence variable with value true true
  $w \leftarrow w \cdot P(W = t|S = t, R = t) = 0.099$

$\Rightarrow$ [true,true,true,true] with weight 0.099

# Likelihood weighting

## Algorithm

Find $P(Q|E)$

1. generate samples $s_i$ with weights $w_i$
2. calculate $\frac{W(Q \wedge E)}{W(E)}$

($W(X)$: sum of weights from all samples where X is true)

- converges faster than rejection sampling
- performance still degrades with many evidence variables

# Summary Bayesian Networks

- Bayesian networks allow a compact representation of the joint probability distribution using independence assumptions
- They support different forms of inference: causal, diagnostic, …
- Inference means here the calculation of the distribution of a set of variables given evidences
- The complexity of inference depends of the network structure
- In general, inference in Bayesian networks is NP-hard
- Approximate algorithms needed for complex networks
- Sampling can be used

# Chapter 15:
# Inference in temporal models

The world changes; we need to track and predict it

Diabetes management vs vehicle diagnosis

Basic idea: copy state and evidence variables for each time step

$\mathbf{X}_t$ = set of unobservable state variables at time $t$
e.g., $BloodSugar_t$, $StomachContents_t$, etc.

$\mathbf{E}_t$ = set of observable evidence variables at time $t$
e.g., $MeasuredBloodSugar_t$, $PulseRate_t$, $FoodEaten_t$

This assumes **discrete time**; step size depends on problem

Notation: $\mathbf{X}_{a:b} = \mathbf{X}_a, \mathbf{X}_{a+1}, \ldots, \mathbf{X}_{b-1}, \mathbf{X}_b$

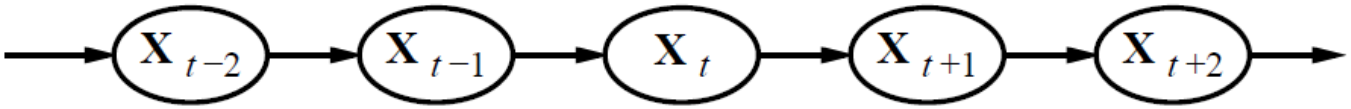
# Markov processes (Markov chains)

Construct a Bayes net from these variables: parents?

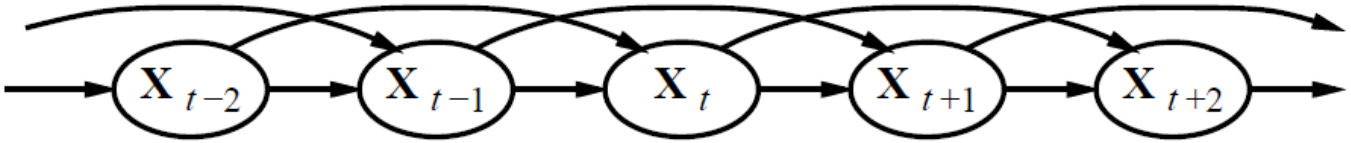
Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$

First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$

First−order



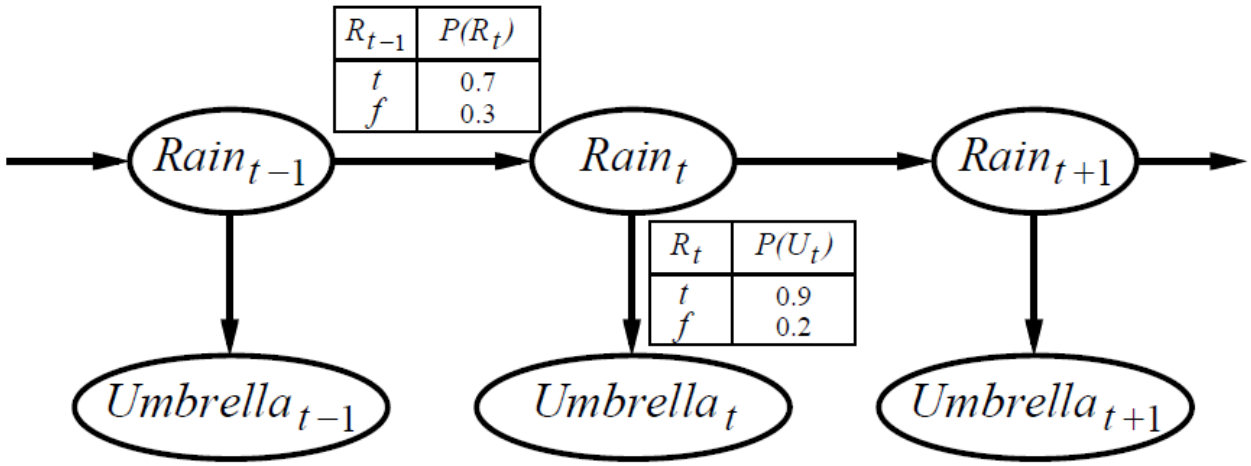

Second−order




Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$

Stationary process: transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and
sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all $t$

# Example




First-order Markov assumption not exactly true in real world!

Possible fixes:

1. **Increase order** of Markov process
2. **Augment state**, e.g., add $Temp_t$, $Pressure_t$

Example: robot motion.

Augment position and velocity with $Battery_t$

# Inference tasks

Filtering: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$

belief state—input to the decision process of a rational agent

Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$

evaluation of possible action sequences;

like filtering without the evidence

Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $0 \leq k < t$

better estimate of past states, essential for learning

Most likely explanation: $\arg\max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t}|\mathbf{e}_{1:t})$

speech recognition, decoding with a noisy channel

# And now a practical example …