CS221 Final Exam Review

Week 10

Stanford University

Exam topics

- CSPs
- Markov networks
- Bayesian networks
- Logic

Questions encountered

- What is the difference between a Markov net, Bayesian net, HMM and Markov model?
- Why Gibbs sampling? How to compute $P(x_i \mid X_{-i})$?
- What is the FB algorithm? What does F and B mean? Why is it prob? Why do we only we use it for HMMs?

Outline

- Markov networks vs Bayesian networks vs Markov models vs HMMs
- Markov networks
  - Gibbs sampling
- Bayesian networks
  - Forward backward algorithm

What are we leaving out?

- More about Bayesian networks - PS week 8
- Logic - PS week 9

Outline

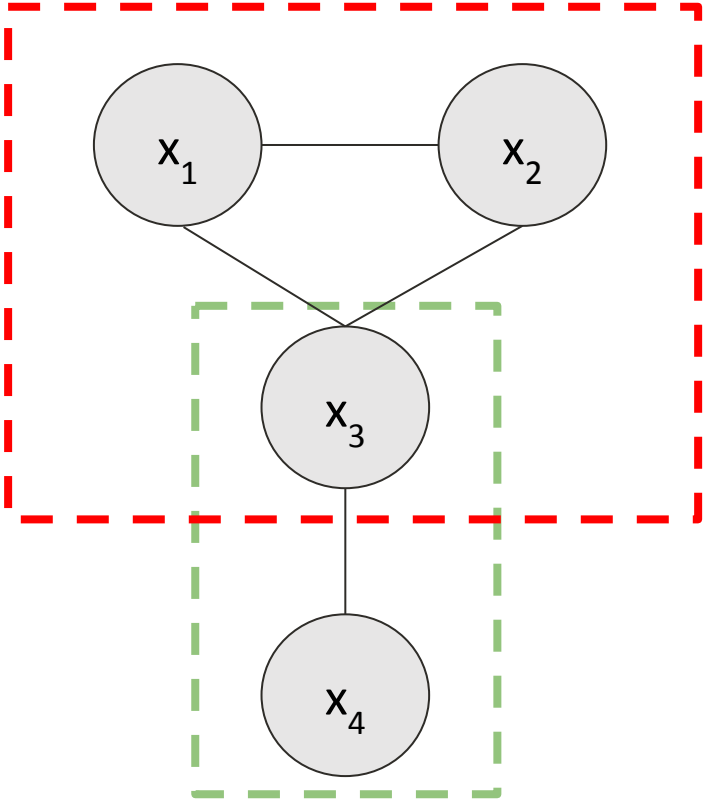
- **Markov networks vs Bayesian networks vs Markov models vs HMMs**
- Markov networks
  - Gibbs sampling
- Bayesian networks
  - Forward backward algorithm

All the different nets

$g(x_1,x_2,x_3,x_4) = \mathbf{f_1}(x_1,x_2,x_3) \times \mathbf{f_2}(x_3,x_4)$

Markov networks: g = P when normalized, f_i's >= 0

Bayesian networks: g = P, and f_i's are conditional Ps, hence directed and Z = 1



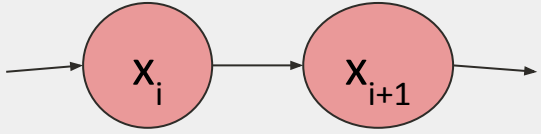
All the different nets

$g(x_1,x_2,x_3,x_4) = \mathbf{f_1}(x_1,x_2,x_3) \times \mathbf{f_2}(x_3,x_4)$

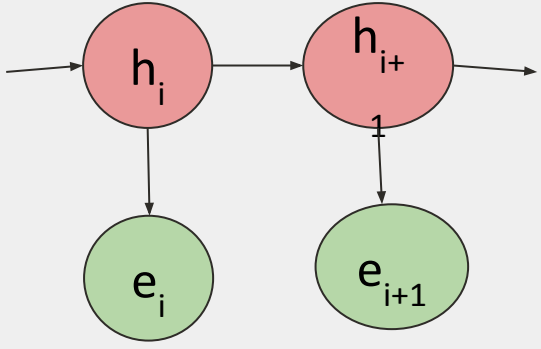
Markov networks: g = P when normalized, f_i's >= 0

Bayesian networks: g = P, and f_i's are conditional Ps, hence directed and Z = 1

Markov models:



HMMs:

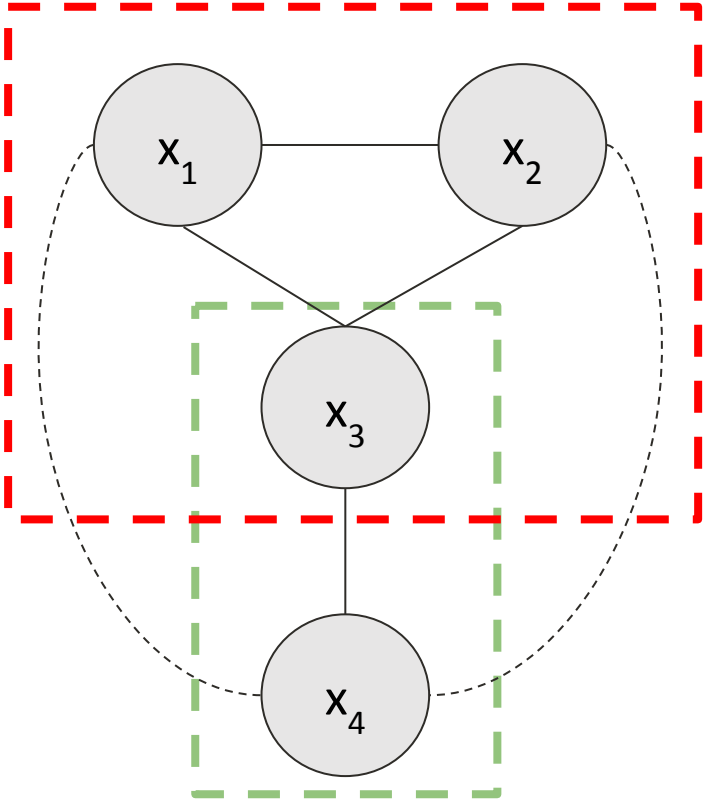


All the different nets

$g(x_1,x_2,x_3,x_4) = \mathbf{f_1}(x_1,x_2,x_3) \times \mathbf{f_2}(x_3,x_4)$

Why factorize?

- Simplifies g
- Reduced # params
- Table size: $O(|\text{domain}|^4)$ reduced to $O(|\text{domain}|^3)$

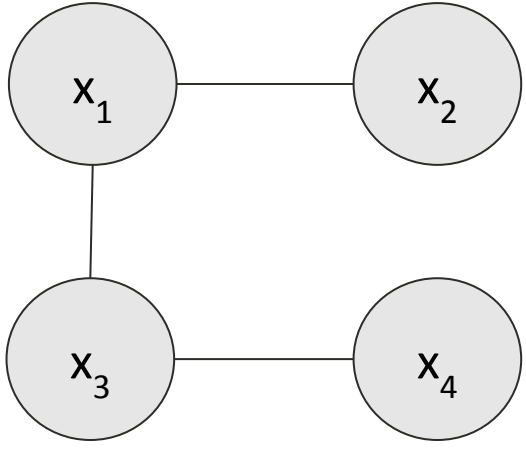


Outline

- Markov networks vs Bayesian networks vs Markov models vs HMMs
- **Markov networks**
  - Gibbs sampling
- Bayesian networks
  - Forward backward algorithm

Markov nets

x_1	x_3	weight (x_1, x_3)
0	0	2
0	1	1
1	0	1
1	1	2



x_1	x_2	weight (x_1, x_2)
0	0	1
0	1	2
1	0	2
1	1	1

x_3	x_4	weight (x_3, x_4)
0	0	1
0	1	2
1	0	2
1	1	1

Markov nets

x_1	x_3	weight(x_1, x_3)
0	0	2
0	1	1
1	0	1
1	1	2

x_1	x_2	weight(x_1, x_2)
0	0	1
0	1	2
1	0	2
1	1	1

x_3	x_4	weight(x_3, x_4)
0	0	1
0	1	2
1	0	2
1	1	1

What is $P(x_1, x_2, x_3, x_4)$?

$$\frac{1}{Z} \text{weight}(x_1, x_2) \text{weight}(x_1, x_3) \text{weight}(x_3, x_4)$$

Markov nets

x_1	x_3	weight(x_1, x_3)
0	0	2
0	1	1
1	0	1
1	1	2

x_1	x_2	weight(x_1, x_2)
0	0	1
0	1	2
1	0	2
1	1	1

x_3	x_4	weight(x_3, x_4)
0	0	1
0	1	2
1	0	2
1	1	1

what is $P(x_1)$?

$$\sum_{x_2, x_3, x_4} P(x_1, x_2, x_3, x_4)$$

Expensive !!

Markov nets

x_1	x_3	weight(x_1, x_3)
0	0	2
0	1	1
1	0	1
1	1	2

x_1	x_2	weight(x_1, x_2)
0	0	1
0	1	2
1	0	2
1	1	1

x_3	x_4	weight(x_3, x_4)
0	0	1
0	1	2
1	0	2
1	1	1

What is $P(x_1 \mid x_2 = 0, x_3 = 1, x_4 = 0)$?

$\therefore P(x_1 = 0 \mid 0, 1, 0) = \frac{1}{1+4}$

$P(x_1 = 1 \mid 0, 1, 0) = \frac{4}{1+4}$

$x_1 = 0$ | 1×1

$x_1 = 1$ | 2×2

Cheap!

Sum over x_1 only!!

Markov nets: Why Gibbs?

- Compute $P(x_i)$ using $P(x_i \mid X_{-i})$ instead of summing over X_{-i}

Algorithm: Gibbs sampling

Initialize x to a random complete assignment

Loop through $i = 1, \ldots, n$ until convergence:

Set $x_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$

(X_{-i} denotes all variables except X_i)

Increment $\text{count}_i(x_i)$

Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$

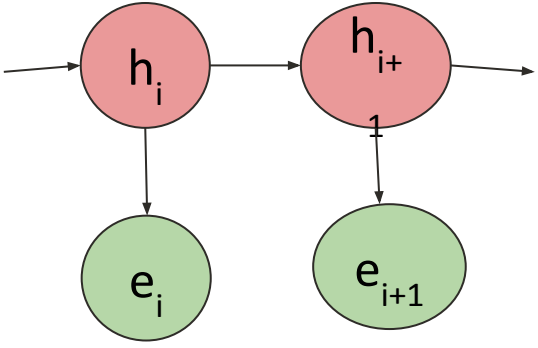
For more practice on Gibbs sampling: refer PS week 7

Outline

- Markov networks vs Bayesian networks vs Markov models vs HMMs
- Markov networks
  - Gibbs sampling
- **Bayesian networks**
  - Forward backward algorithm

Forward backward algorithm

- Compute $P(h_i \mid e's)$
- Applicable only to HMMs or similar
- What special about markov models?
  - One parent for each node



Intuition:

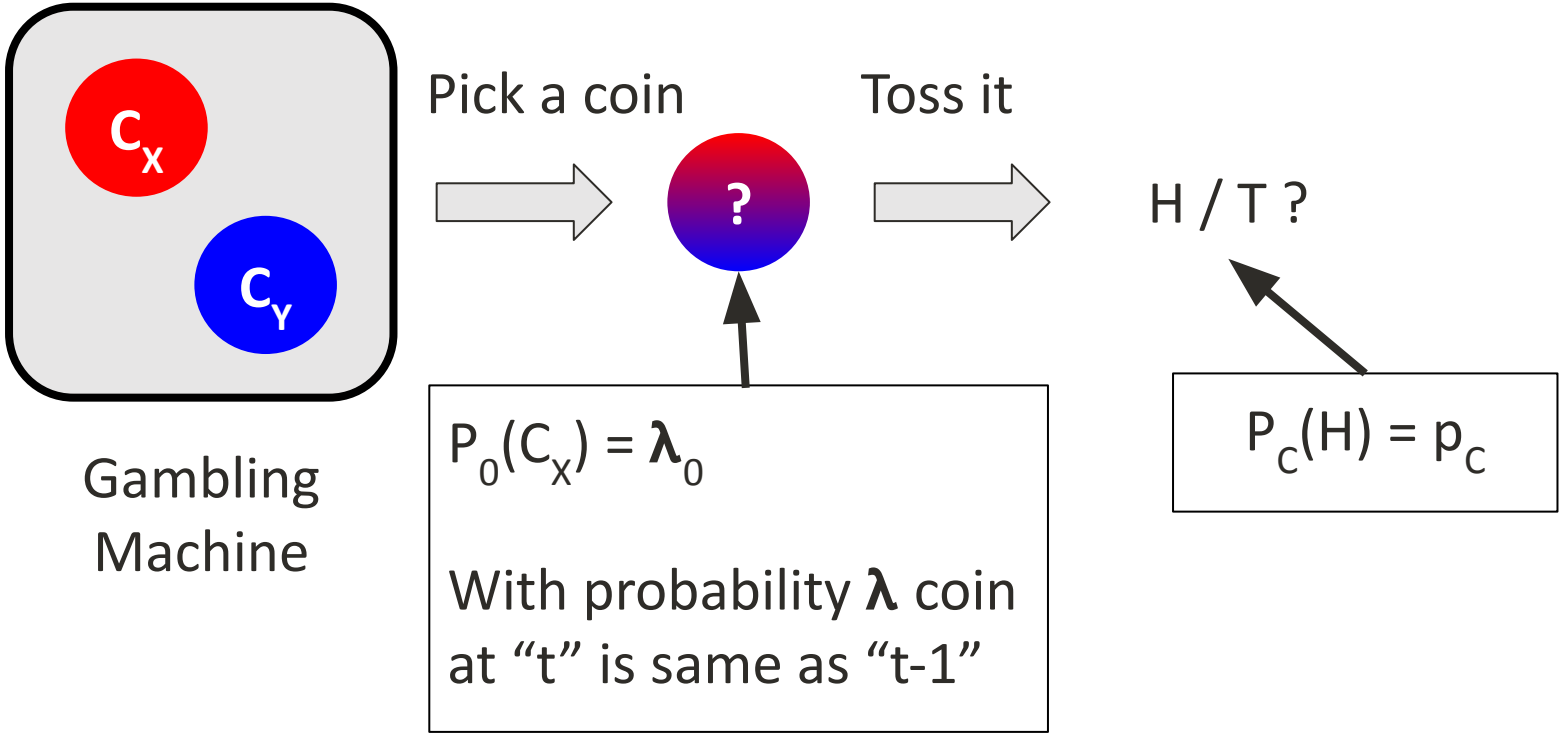
$F(h_i) = P(h_i, e_i \mid e_{<i})$

$B(h_i) = P(e_{>i} \mid h_i)$

$S(h_i) \propto P(h_i, e_i \mid e_{<i}) \times P(e_{>i} \mid h_i) \propto P(h_i \mid e's)$

For more details refer: wiki

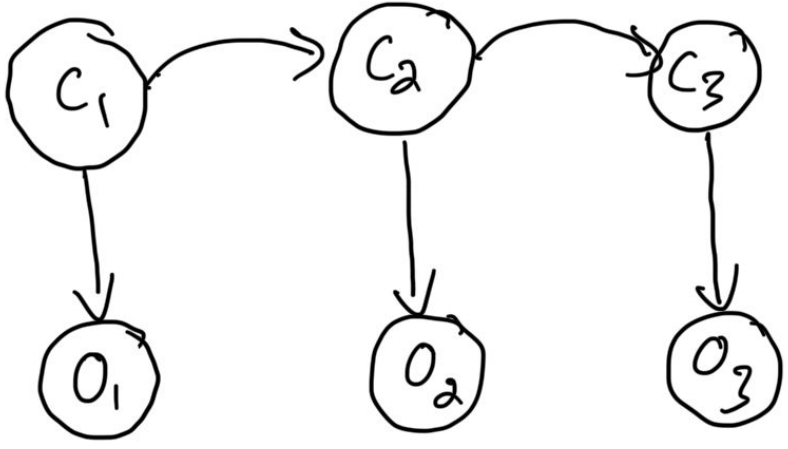
Problem: P2, Winter 2021 Exam 2



How does the bayesian net look?

$P(C_1) = \lambda_0$ if $C_1 = X$, $1-\lambda_0$ else

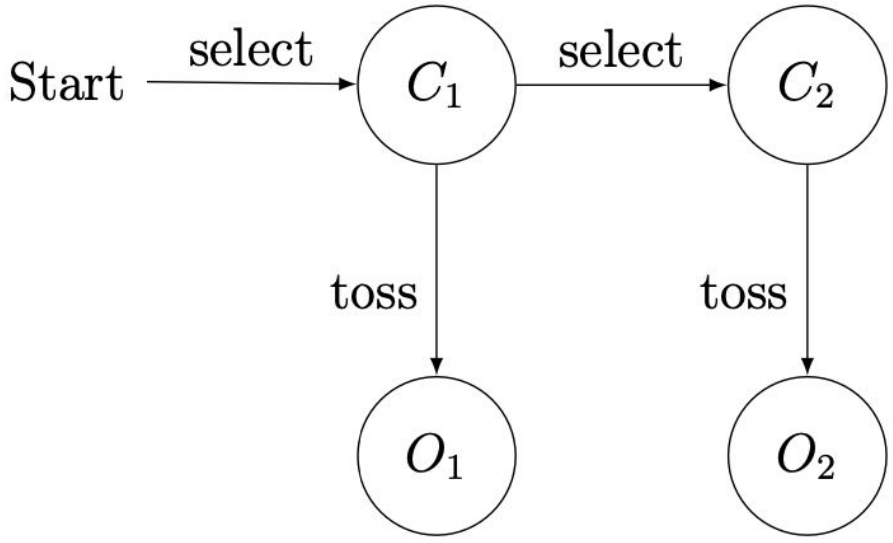
$P(C_{i+1} \mid C_i) = \lambda$ if $C_{i+1} = C_i$, $1-\lambda$ else



$P(O_i \mid C_i) = p_{C_i}$ if $O_i = H$, $1-p_{C_i}$ else

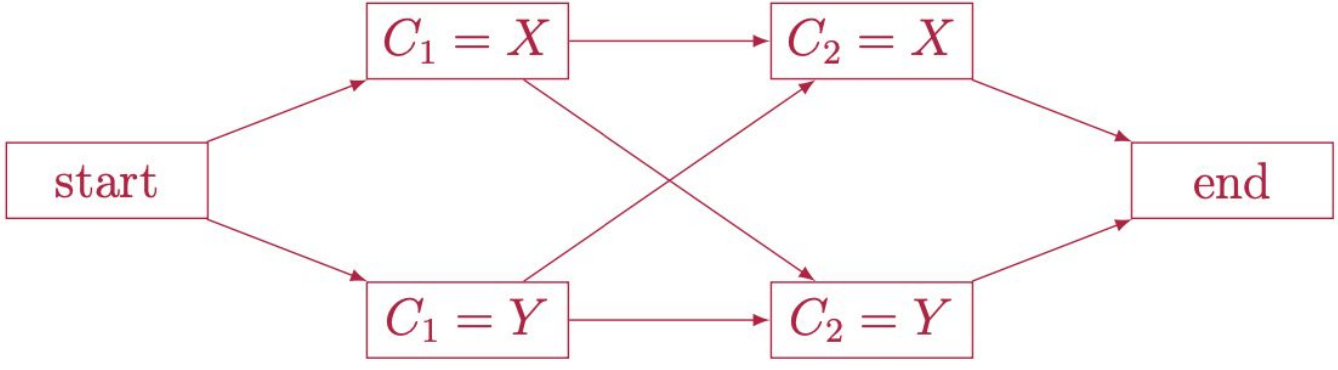
Inference: FB in practice

- Stop after two steps, observe {H,H}



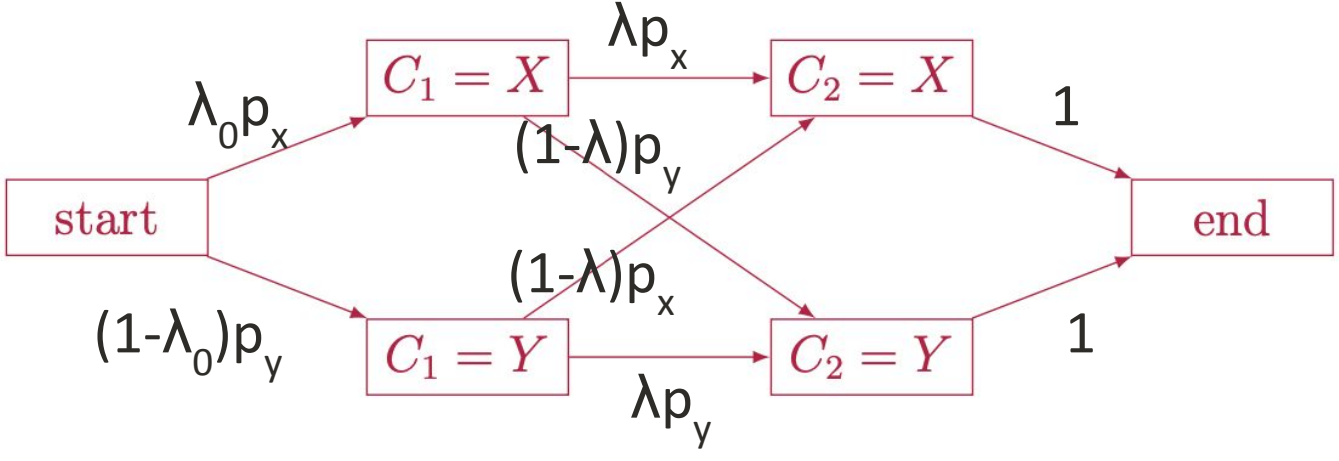
Inference: FB in practice

- Stop after two steps, observe {H,H}
- Draw FB lattice representation



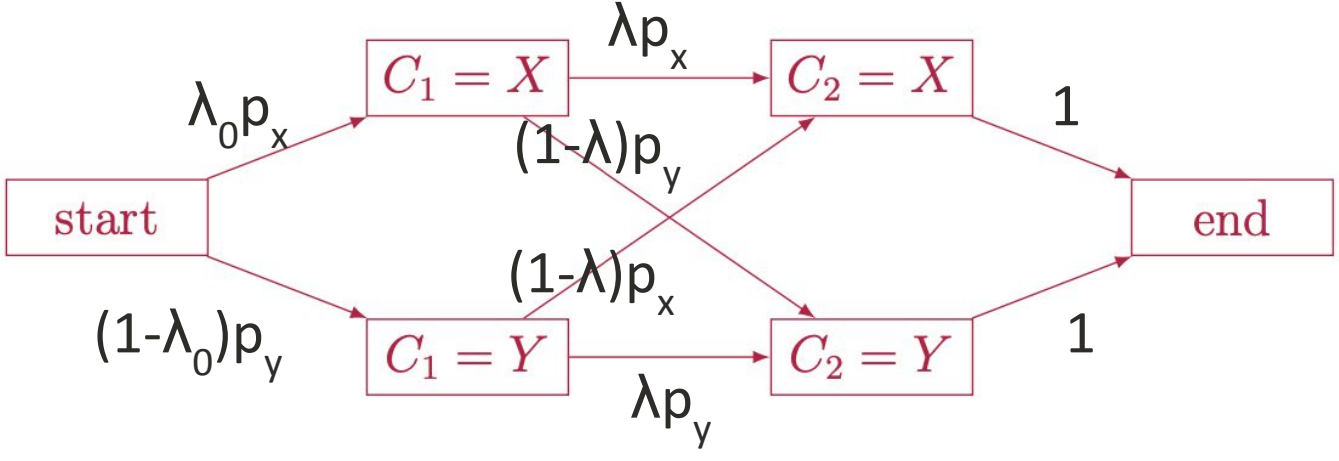
Inference: FB in practice

- Stop after two steps, observe {H,H}
- What are the weights of edges?



Inference: FB in practice

- Stop after two steps, observe {H,H}
- Compute forward passes and backward passes



Inference: FB in practice

- Stop after two steps, observe {H,H}
- Compute forward passes and backward passes

$$F_1(X) = \lambda_0 p_X$$
$$F_1(Y) = (1-\lambda_0) p_Y$$
$$B_2(X) = 1$$
$$B_2(Y) = 1$$

$$F_2(X) = w(X,X) \cdot F_1(X) + w(Y,X) \cdot F_1(Y)$$
$$F_2(Y) = w(X,Y) \cdot F_1(X) + w(Y,Y) \cdot F_1(Y)$$
$$B_1(X) = w(X,X) \cdot B_2(X) + w(X,Y) \cdot B_2(Y)$$
$$B_1(Y) = w(Y,X) \cdot B_2(X) + w(Y,Y) \cdot B_2(Y)$$

For more about Bayesian networks: PS Week 8

Bayesian Networks

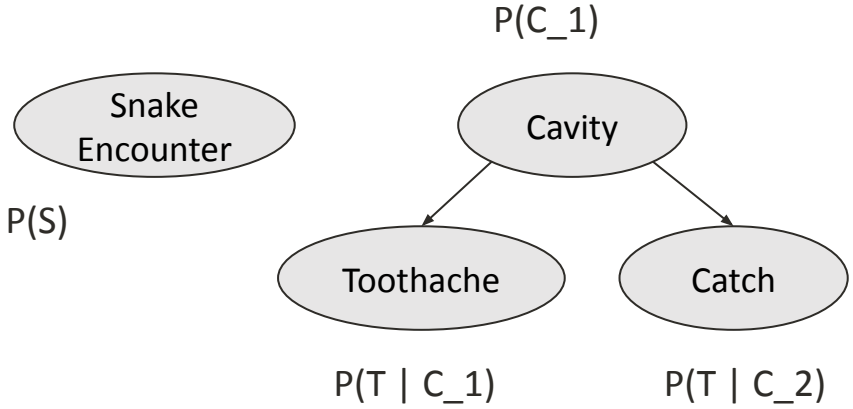
- Key Concepts to review:
  - **Exact Inference**
  - **Sampling**
  - **Probability**: conditional independence, Bayes theorem.

Bayesian Networks Inference

Joint distribution: the chain rule + conditional independence gives us a general way to compute joint distributions:

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

- Example: P(s, c1, t, c2) = P(s) P(c1) P(t | c1)P(c2 | c1)



Gibbs Sampling

all variables except X_i

Algorithm: Gibbs sampling

Initialize x to a random complete assignment

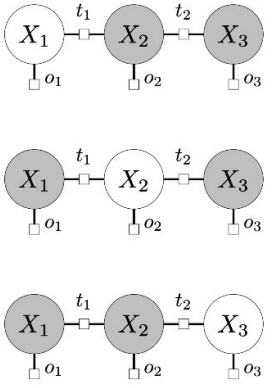
Loop through $i = 1, \ldots, n$ until convergence:

Set $x_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$

(X_{-i} denotes all variables except X_i)

Increment $\text{count}_i(x_i)$

Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$

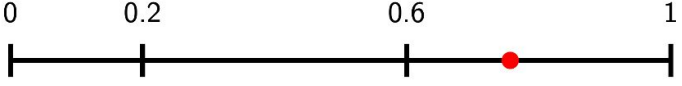


Example: sampling one variable

Weight$(x \cup \{X_2 : 0\}) = 1$ prob. 0.2

Weight$(x \cup \{X_2 : 1\}) = 2$ prob. 0.4

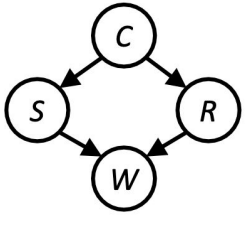
Weight$(x \cup \{X_2 : 2\}) = 2$ prob. 0.4



Markov Blanket and Gibbs Sampling

- **Problem**: How do we sample from $P(Xi \mid$ all other nodes in Bayes Net)?
- We actually only have to worry about a **smaller subset of nodes**
- A random variable is conditionally independent of all other nodes given its **Markov blanket**: parents, children, children's parents
- To sample from $P(Xi \mid mb(Xi))$, compute $P(Xi, mb(Xi))$ and normalize

$$P(X_i|mb(X_i)) \propto P(X_i|parents(X_i)) \prod_{Y_j \in children(X_i)} P(y_j|parents(Y_j))$$



$$P(C \mid mb(C)) = P(C \mid s,r) \propto P(C)P(s|C)P(r|C)$$
$$P(S \mid mb(S)) = P(S \mid c,w) \propto P(S|c)P(w|S,r)$$
$$P(R \mid mb(R)) = P(R|c,w) \propto P(R|c)P(w|s,R)$$
$$P(W \mid mb(W)) = P(W \mid s,r)$$

Bayes Nets Problem

1) **Problem 1: Inferencia**

As the president of the National Trivia Association, you must choose between the Bayesians and the Markovians, the nation's top two rival trivia teams, to represent the US at the World Trivia Olympics. To determine the more popular team, you decide to model the change in monthly TV viewership using a Bayesian network.

Let B_t, M_t denote the number of TV viewers the Bayesians, Markovians have in month t, which cannot be observed directly. You can observe two other quantities which they influence: let S_t be the number of times internet users searched for the Bayesians in month t. Let A_t be the attendance of the televised friendly match at The Rose & Crown pub between the Bayesians and the Markovians in month t.

The fanbases of the two teams evolve according to the following model, where each month a fan is either gained or lost with equal probability:

$$\Pr(M_{t+1}|M_t) = \begin{cases} \frac{1}{2} & \text{if } M_{t+1} = M_t - 1 \\ \frac{1}{2} & \text{if } M_{t+1} = M_t + 1 \\ 0 & \text{otherwise} \end{cases} \qquad \Pr(B_{t+1}|B_t) = \begin{cases} \frac{1}{2} & \text{if } B_{t+1} = B_t - 1 \\ \frac{1}{2} & \text{if } B_{t+1} = B_t + 1 \\ 0 & \text{otherwise} \end{cases}$$

The Bayesian fans like to rewatch their trivia shows by searching the recaps online! We model the fan's size's influence on the number of internet searches by:

$$\Pr(S_t|B_t) = \begin{cases} 0.3 & \text{if } S_t = B_t \\ 0.25 & \text{if } S_t = B_t - 1 \\ 0.2 & \text{if } S_t = B_t - 2 \\ 0.15 & \text{if } S_t = B_t - 3 \\ 0.1 & \text{if } S_t = B_t - 4 \\ 0 & \text{otherwise} \end{cases}$$

Because most TV viewers attend each monthly friendly matches at the pub, we model the influence of the TV viewership number on the friendly match attendance by:

$$\Pr(A_t|B_t, M_t) = \begin{cases} 0.14 & \text{if } A_t = B_t + M_t \\ 0.13 & \text{if } |A_t - (B_t + M_t)| = 1 \\ 0.11 & \text{if } |A_t - (B_t + M_t)| = 2 \\ 0.09 & \text{if } |A_t - (B_t + M_t)| = 3 \\ 0.06 & \text{if } |A_t - (B_t + M_t)| = 4 \\ 0.04 & \text{if } |A_t - (B_t + M_t)| = 5 \\ 0 & \text{otherwise} \end{cases}$$

Bayes Nets Problem

Note: The only assumptions you may make in a given part are those which are explicitly stated in that part's description.

(a) Model the changing fanbases as a Bayesian network. You should create 8 nodes: B_t, B_{t+1}, M_t, M_{t+1}, A_t, A_{t+1}, S_t, and S_{t+1}. Indicate which nodes correspond to latent/hidden fanbase counts and which correspond to the observable emissions.

Bayes Nets Problem

(b) **Domain Consistencies**

As a first step, we will not concern ourselves with which fanbase counts are *probable*, but instead which counts are even *possible*. Suppose that we observe, in our first month of collecting data, that $S_1 = 75$ and $A_1 = 100$. Give the domains for M_1 and B_1 that are consistent with these observations. You need only give the consistent domains (using either set notation or inequality notation).

Bayes Nets Problem

(c) **Inference**

Suppose the Bayesian's trivia team captain took a nationwide poll in month t that concluded they had exactly 75 TV viewers. Suppose additionally that in month $t+2$, the search engine reported 73 people search for the Bayesians online. What is the probability that in month $t+1$, 72 people searched for the Bayesians online?

$$\Pr(S_{t+1} = 72 | B_t = 75) =$$

Bayes Nets Problem

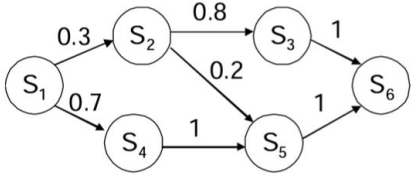
(d) **Gibbs Sampling**

You decide that you'd be satisfied with simply being able to draw samples from distributions rather than specifying them exactly. In particular, we want to estimate $P(B_t)$. You decide to implement Gibbs sampling for this purpose. Assume that all necessary information is provided to you. Provide the set of nodes such that if those nodes are observed, when performing Gibbs sampling, the random variable B_t will be conditionally independent from all other nodes.

Problem 2

3) Problem 3: Hidden Markov Models

Let's consider the following HMM, with the state transition provided in the figure above. For example $P(X_{t+1} = s_3 | X_t = s_2) = 0.8$. Missing edges correspond to zero transition probability. The table below provides the observation probabilities. For example, $P(O_t = c | X_t = s_2) = 0.5$.



For each of the items below, insert $>, <, =$ into the parenthesis between the expressions. We will use the following abbreviation: $P(O = abc, X_2 = s_1) = P(O_1 = a, O_2 = b, O_3 = c, X_2 = s_1)$ (Hint: the answers can be found without doing much computation.) Justify your work.

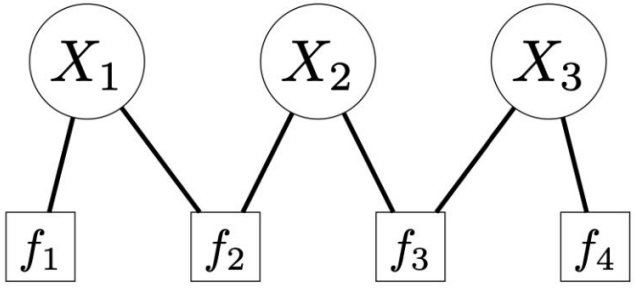
- $P(O = abca, X_1 = s_1, X_4 = s_6)$ ( ) $P(O = abca \,|X_1 = s_1, X_4 = s_6)$

1

- $P(O = acdb)$ ( ) $P(O = acdb \,|X_2 = s4, X_3 = s_5)$

Constraint Satisfaction Problems

Review: CSPs



Definition: factor graph

Variables:

$X = (X_1, \ldots, X_n)$, where $X_i \in \text{Domain}_i$

Factors:

$f_1, \ldots, f_m$, with each $f_j(X) \geq 0$

Definition: assignment weight

Each **assignment** $x = (x_1, \ldots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

Objective:

$$\arg\max_x \text{Weight}(x)$$

Algorithm: backtracking search

Backtrack($x, w,$ Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i
- Order **VALUES** Domain$_i$ of chosen X_i
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$' \leftarrow$ Domains via **LOOKAHEAD**
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

Algorithm: backtracking search

Backtrack($x, w,$ Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i
- Order **VALUES** Domain$_i$ of chosen X_i
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$' \leftarrow$ Domains via **LOOKAHEAD**
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

Pick an ordering

Figure out how good your variable assignment is

Prune domains based on selection

Recurse

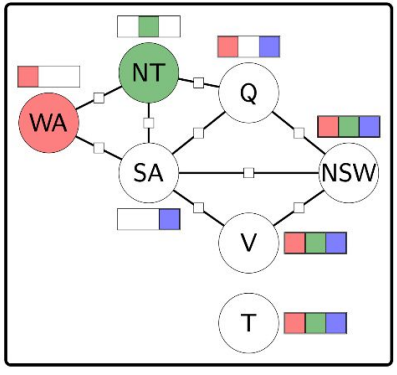
Algorithm: backtracking search

Backtrack($x, w,$ Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i (1) How do we pick next variable?
- Order **VALUES** Domain$_i$ of chosen X_i (2) How do we pick next value?
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$' \leftarrow$ Domains via **LOOKAHEAD** (3) How do we lookahead?
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

Most Constrained Variable

(1) How do we pick next variable?



Which variable to assign next?

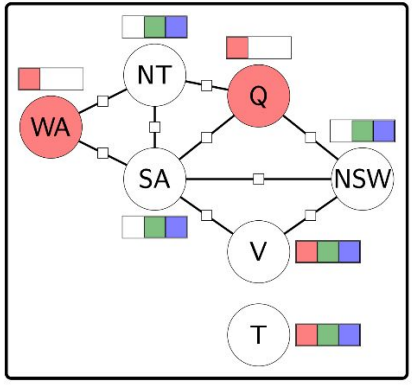
Key idea: most constrained variable

Choose variable that has the smallest domain.

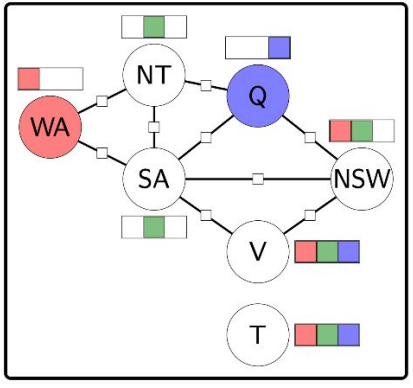
Ordering Values of a Variable

(2) How do we pick next value?

What values to try for Q?



$2 + 2 + 2 = 6$ consistent values



$1 + 1 + 2 = 4$ consistent values

Key idea: least constrained value

Order values of selected X_i by decreasing number of consistent values of neighboring variables.

CSP Heuristics

Most constrained variable (MCV):

- Must assign every variable
- If going to fail, fail early ⇒ more pruning
- Most useful when *some* factors are constraints

(1) How do we pick next variable?

Recall: constraints

A **constraint** is a factor where some setting of the variables will cause it to take on a value of **zero**!

CSP Heuristics

Most constrained variable (MCV):

- Must assign every variable
- If going to fail, fail early ⇒ more pruning
- Most useful when *some* factors are constraints

(1) How do we pick next variable?

Least constrained value (LCV):

- Need to choose some value
- Choose value that is most likely to lead to solution
- Most useful when *all* factors are constraints

(2) How do we pick next value?

Recall: constraints

A **constraint** is a factor where some setting of the variables will cause it to take on a value of **zero**!

CSP Heuristics

Most constrained variable (MCV):

- Must assign every variable
- If going to fail, fail early ⇒ more pruning
- Most useful when *some* factors are constraints

(1) How do we pick next variable?

Least constrained value (LCV):

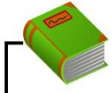
- Need to choose some value
- Choose value that is most likely to lead to solution
- Most useful when *all* factors are constraints

(2) How do we pick next value?

Note: These heuristics are most useful in general when we don't need to find the optimal solution, such as when factors do not have weights (thus any final solution is optimal)

Arc Consistency

(3) How do we lookahead?



Definition: arc consistency

A variable X_i is **arc consistent** with respect to X_j if for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that $f(\{X_i : x_i, X_j : x_j\}) \neq 0$ for all factors f whose scope contains X_i and X_j.

Algorithm: enforce arc consistency

EnforceArcConsistency(X_i, X_j): Remove values from Domain_i to make X_i arc consistent with respect to X_j.

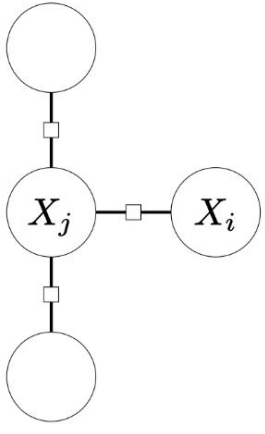
(3) How do we lookahead?

Forward checking: when assign $X_j : x_j$, set $\text{Domain}_j = \{x_j\}$ and enforce arc consistency on all neighbors X_i with respect to X_j

AC-3: repeatedly enforce arc consistency on all variables

Algorithm: AC-3

$S \leftarrow \{X_j\}$.
While S is non-empty:
    Remove any X_j from S.
    For all neighbors X_i of X_j:
        Enforce arc consistency on X_i w.r.t. X_j.
        If Domain_i changed, add X_i to S.



Algorithm: backtracking search

Backtrack($x, w,$ Domains):

- If x is complete assignment: update best and return
- Choose unassigned **VARIABLE** X_i (MCV)
- Order **VALUES** Domain$_i$ of chosen X_i (LCV)
- For each value v in that order:
  - $\delta \leftarrow \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
  - If $\delta = 0$: continue
  - Domains$'$ $\leftarrow$ Domains via **LOOKAHEAD** (forward checking or AC3)
  - If any Domains$'_i$ is empty: continue
  - Backtrack($x \cup \{X_i : v\}, w\delta,$ Domains$'$)

Alternatives to Backtracking: Beam Search

Idea: keep $\leq K$ **candidate list** C of partial assignments

Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \ldots, n$:

Extend:

$C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$

Prune:

$C \leftarrow K$ elements of C' with highest weights

Not guaranteed to find maximum weight assignment!

Alternatives to Backtracking: Beam Search

Idea: keep $\leq K$ **candidate list** C of partial assignments

Note: we still have a choice of variable ordering!

Algorithm: beam search

Initialize $C \leftarrow [\{\}]$

For each $i = 1, \ldots, n$:

Extend:

$C' \leftarrow \{x \cup \{X_i : v\} : x \in C, v \in \text{Domain}_i\}$

Prune:

$C \leftarrow K$ elements of C' with highest weights

Not guaranteed to find maximum weight assignment!

Alternatives to Backtracking: ICM

- ICM: start with a random complete assignment. Repeatedly loop through all the variables X_i.
- On variable X_i, we consider all possible ways of re-assigning it Xi : v for v ∈ Domaini, and choose the new assignment that has the highest weight.
- We represent each step of the algorithm by having shaded nodes for the variables which are fixed and unshaded for the single variable which is being re-assigned.

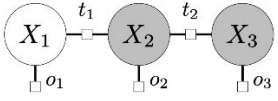
Algorithm: iterated conditional modes (ICM)

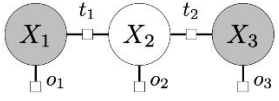
Initialize x to a random complete assignment

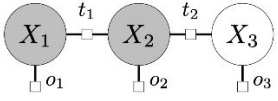
Loop through $i = 1, \ldots, n$ until convergence:

Compute weight of $x_v = x \cup \{X_i : v\}$ for each v

$x \leftarrow x_v$ with highest weight







CSP Problem

4) Problem 4: CA Assignment

Every quarter, the Stanford computer science department assigns graduate students as course assistants (CAs). Students who wish to serve as CAs fill out an application in which they can list the classes they'd like to CA for. After the application due date, the department matches applicants to courses, taking into account student preferences as well as how many course assistants each class needs. Here's the formal CA-assignment problem setup:

- There are n students $S_1, \ldots, S_n$ who apply for CAships.
- There are m courses $C_1, \ldots, C_m$ that have CA openings.
- Each student S_i specifies arbitrary non-negative preferences $P_1^{(i)}, \ldots, P_m^{(i)} \geq 0$ for each of the m classes. A large preference value $P_j^{(i)}$ means student S_i really wants to CA for class C_j, and a preference value of 0 for $P_j^{(i)}$ means student S_i does not want to CA for class C_j.

The CA-matching process must adhere to the following requirements:

- Each course C_i can have a maximum of M_i course assistants.
- Every student must be matched to exactly one class for which they have specified a positive preference (assume each student has at least one such preference).

Model the CA-matching process with a CSP with n variables, one for each student $S_1, \ldots, S_n$. Our CSP should find the *maximum weight assignment*, where the weights are determined by student preferences.

(a) What is the domain of each variable and what is the cardinality?

CSP Problem

(b) What are the factors? State the arity of each.

CSP Problem

(c) We imagine a small setting of this problem for 3 students S_1, S_2, S_3 and 3 courses C_1, C_2, C_3. The student preferences are given by the following table:

	C_1	C_2	C_3
S_1	3	0	0
S_2	2	1	3
S_3	5	3	0

Additionally, classes C_1 and C_2 can have a maximum of 1 CA each, and class C_3 can have at most 2 CAs.

Apply the CSP you designed to this small setting and enforce arc-consistency amongst its variables. In particular, write out each variable and its domain after arc-consistency has been enforced. For example, if you have a variable X_i with a domain $\{a, b, c\}$ after enforcing arc-consistency, you should write

$$X_i : \{a, b, c\}$$

CSP Problem

(d) True or False, with justification.

i. The least constrained value (LCV) heuristic would be a useful optimization for our CA-assignment CSP.

ii. The most constrained variable (MCV) heuristic would be a useful optimzation for our CA-assignment CSP.

iii. If we use the ICM algorithm to solve our CA-assignment CSP, everytime we modify a single variable assignment our factor recomputation will be on the order of n (recall that n is the number of students applying for a CA assignment).

iv. If we use beam search with different beam sizes k to solve our CA-assignment CSP, our solution's assignment weight will always increase as we increase the beam size k.

CSP Problem

(e) Explain how you would modify your CSP from part a. to allow for the possibility that some students aren't matched to a course. You should encode the (realistic) assumption that not receiving a CAship is the least-preferable assignment for the student. (Note that students can still give a preference of 0 for a class if they do not want to be a CA for that class, and your modification should not prohibit this.)

Logic

Logic

- Key Concepts to review:
  - **Propositional Logic (KB, etc)**
  - **FOL Semantics**
  - **Logical Inference**

Syntax of Propositional Logic

Propositional symbols (atomic formulas): A, B, C

Logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

Build up formulas recursively—if f and g are formulas, so are the following:

- Negation: $\neg f$
- Conjunction: $f \wedge g$
- Disjunction: $f \vee g$
- Implication: $f \rightarrow g$
- Biconditional: $f \leftrightarrow g$

First-Order Logic

- The expressive power of Propositional Logic is limited. For example, it cannot express expressions such as “for all” or “for some”. It is also difficult to express relationships.
- First Order Logic (abbreviated as FOL), also known as predicate logic, combines quantifiers and predicates for a more powerful and compact formalism.
- **You should be comfortable with translate sentences into first-order logic!**

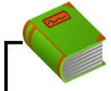
Terms (refer to objects):

- Constant symbol (e.g., arithmetic)
- Variable (e.g., x)
- Function of terms (e.g., $\text{Sum}(3, x)$)

Formulas (refer to truth values):

- Atomic formulas (atoms): predicate applied to terms (e.g., $\text{Knows}(x, \text{arithmetic})$)
- Connectives applied to formulas (e.g., $\text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$)
- Quantifiers applied to formulas (e.g., $\forall x\, \text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$)

Knowledge Base



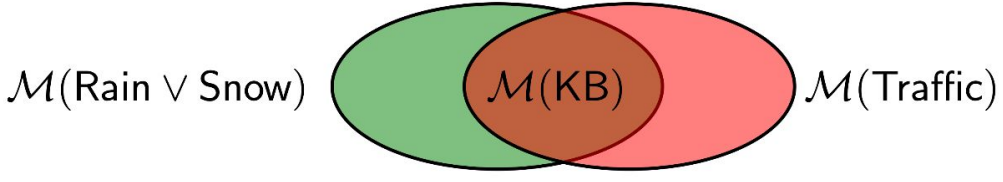
Definition: Knowledge base

A **knowledge base** KB is a set of formulas representing their conjunction / intersection:

$$\mathcal{M}(\text{KB}) = \bigcap_{f \in \text{KB}} \mathcal{M}(f).$$

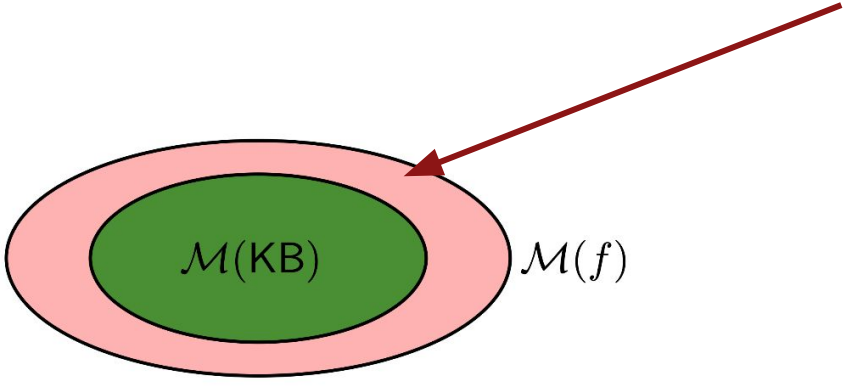
Intuition: KB specifies constraints on the world. $\mathcal{M}(\text{KB})$ is the set of all worlds satisfying those constraints.

Let $\text{KB} = \{\text{Rain} \vee \text{Snow}, \text{Traffic}\}$.



Entailment

M(f) must be the superset!



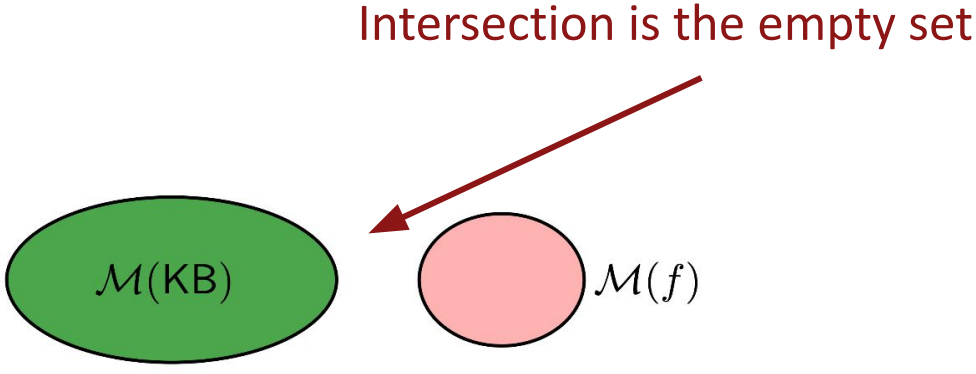
Intuition: f added no information/constraints (it was already known).

Definition: entailment

KB entails f (written KB $\models f$) iff $\mathcal{M}(\text{KB}) \subseteq \mathcal{M}(f)$.

Example: Rain $\wedge$ Snow $\models$ Snow

Contradiction



Intuition: f contradicts what we know (captured in KB).

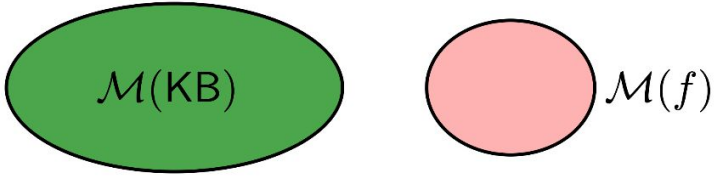
Definition: contradiction

KB contradicts f iff $\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \emptyset$.

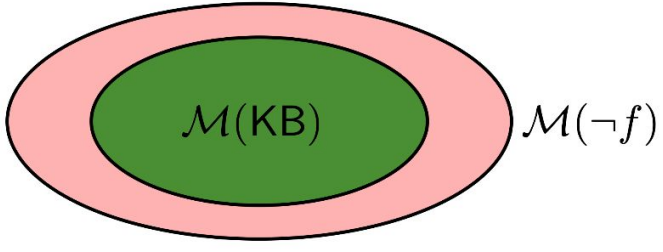
Example: Rain $\wedge$ Snow contradicts $\neg$Snow

Entailment & Contradiction

Contradiction:



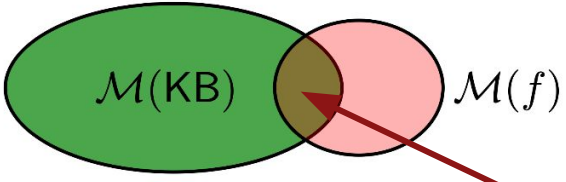
Entailment:



Proposition: contradiction and entailment

KB **contradicts** f iff KB **entails** $\neg f$.

Contingency



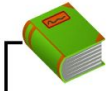
The intersection is not in M(f) nor M(KB)!

Intuition: f adds non-trivial information to KB

$$\emptyset \subsetneq \mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \subsetneq \mathcal{M}(\text{KB})$$

Example: Rain and Snow

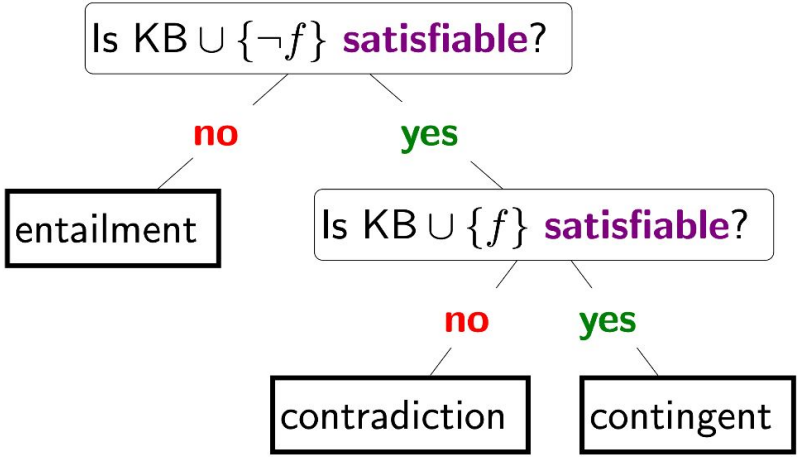
Satisfiability



Definition: satisfiability

A knowledge base KB is **satisfiable** if $\mathcal{M}(\text{KB}) \neq \emptyset$.

Reduce Ask$[f]$ and Tell$[f]$ to satisfiability:



Logic Problem

a. (*12 points*) **Knowledge Base**

Imagine we are building a knowledge base of propositions in first order logic and want to make inferences based on what we know. We will deal with a simple setting, where we only have three objects in the world: Alice, Carol, and Bob. Our predicates are as follows:

- Employee(x): x is an employee.
- Boss(x): x is a boss.
- Works(x): x works.
- Paid(x): x gets paid.

The knowledge base we have constructed consists of the following propositions:

(a) Boss(Carol)

(b) Employee(Bob)

(c) Paid(Carol) $\wedge$ Works(Carol)

(d) Paid(Alice)

(e) $\forall x$ (Employee(x) $\leftrightarrow \neg$ Boss(x))

(f) $\forall x$ (Employee(x) $\rightarrow$ Works(x))

(g) $\forall x$ ((Paid(x) $\wedge \neg$ Works(x)) $\rightarrow$ Boss(x))

Logic Problem

(i) [2 Point] We know from class that one technique we can use to perform inference with our knowledge base is to propositionalize the statements of first-order logic into statements of propositional logic. Practice this by propositionalizing statement (6) from our knowledge base.

$$\forall x \ (\text{Employee(x)} \rightarrow \text{Works(x)})$$

Logic Problem

(i) [2 Point] We know from class that one technique we can use to perform inference with our knowledge base is to propositionalize the statements of first-order logic into statements of propositional logic. Practice this by propositionalizing statement (6) from our knowledge base.

$$\forall x \ (\text{Employee(x)} \rightarrow \text{Works(x)})$$

Logic Problem

(ii) [3 Points] If we translated the statement "Anyone who is not a boss either works or does not get paid" into first-order logic and added it to our knowledge base, how would the size of the set of valid models representing our knowledge base change, and why?

The knowledge base we have constructed consists of the following propositions:

(a) Boss(Carol)

(b) Employee(Bob)

(c) Paid(Carol) $\wedge$ Works(Carol)

(d) Paid(Alice)

(e) $\forall x$ (Employee(x) $\leftrightarrow \neg$ Boss(x))

(f) $\forall x$ (Employee(x) $\rightarrow$ Works(x))

(g) $\forall x$ ((Paid(x) $\wedge \neg$ Works(x)) $\rightarrow$ Boss(x))

Logic Problem

(iii) [7 Points] Using only our original knowledge base (not including the statement from part (ii)), we want to answer the question "Does everyone work?" We first translate the sentence "everyone works" into first order logic as statement f. Determine the answer to our query by considering the following questions of satisfiability:

① [3 points] Is KB $\cup \neg f$ satisfiable? Answer yes/no. If yes, fill in the following table with T for true and F for false to show that there is a satisfying model.

x	Employee(x)	Boss(x)	Works(x)	Paid(x)
Alice				
Bob				
Carol				

Logic Problem

② [3 points] Is KB $\cup$ f satisfiable? Answer yes/no. If yes, fill in the following table with T for true and F for false to show that there is a satisfying model.

x	Employee(x)	Boss(x)	Works(x)	Paid(x)
Alice				
Bob				
Carol				

Logic Problem

③ [1 points] Based on your answers to the previous two parts, does our knowledge base entail f, contradict f, or is f contingent? And what should the answer to our original question "Does everyone work?" be?

Thank you!

Good luck on the exam!

Stanford University