Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Variable-based models

Special cases:

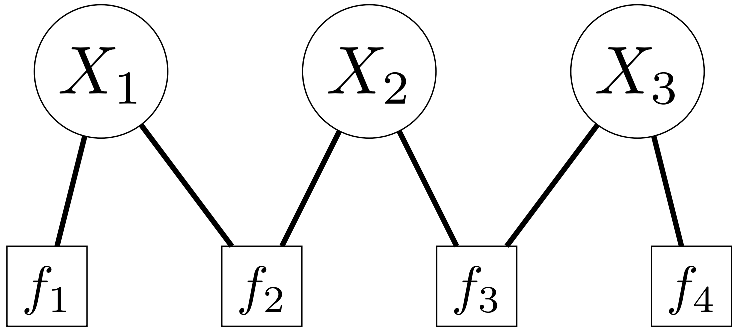
- Constraint satisfaction problems
- Markov networks
- Bayesian networks

Key idea: variables

- Solutions to problems $\Rightarrow$ assignments to variables (modeling).
- Decisions about variable ordering, etc. chosen by inference.

Higher-level modeling language than state-based models

Factor graph



Definition: factor graph

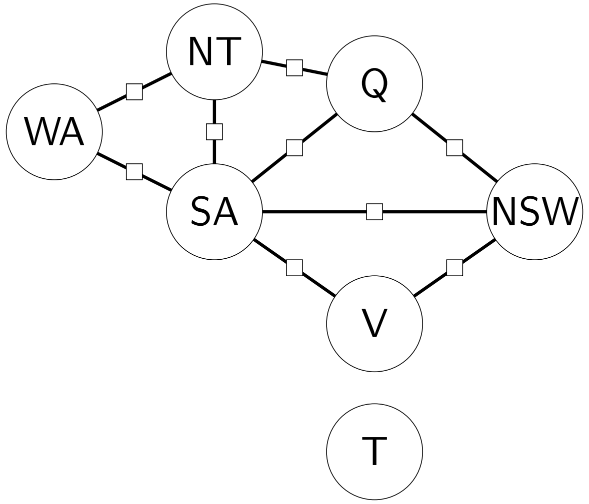
Variables:

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

Factors:

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

Example: map coloring



Variables:

$$X = (\mathsf{WA}, \mathsf{NT}, \mathsf{SA}, \mathsf{Q}, \mathsf{NSW}, \mathsf{V}, \mathsf{T})$$

$$\mathsf{Domain}_i \in \{\mathsf{R}, \mathsf{G}, \mathsf{B}\}$$

Factors:

$$f_1(X) = [\mathsf{WA} \neq \mathsf{NT}]$$

$$f_2(X) = [\mathsf{NT} \neq \mathsf{Q}]$$

...

Factors

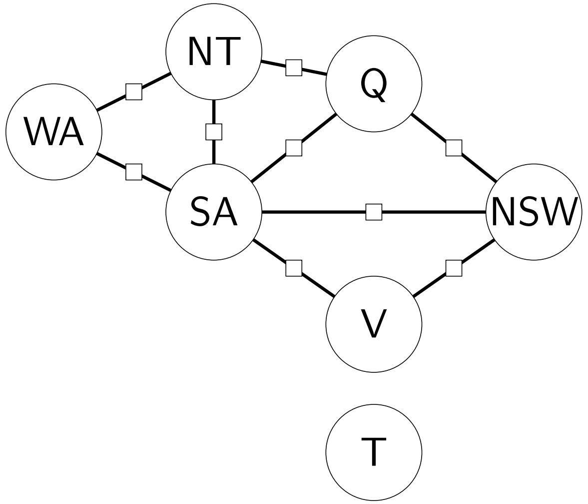
Definition: scope and arity

Scope of a factor f_j: set of variables it depends on.

Arity of f_j is the number of variables in the scope.

Unary factors (arity 1); **Binary** factors (arity 2).

Constraints are factors that return 0 or 1.



Example: map coloring

Scope of $f_1(X) = [\text{WA} \neq \text{NT}]$ is $\{\text{WA}, \text{NT}\}$

f_1 is a binary constraint

Assignment weights

Definition: assignment weight

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

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

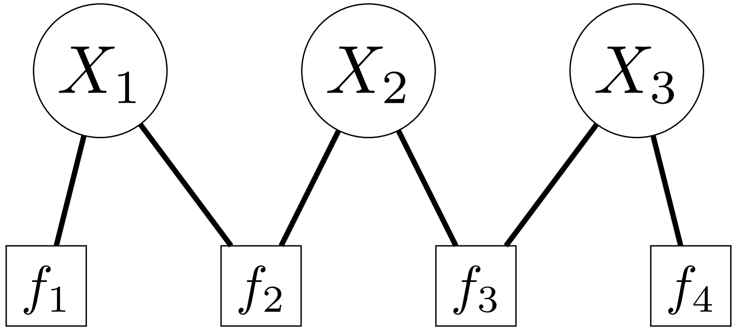
An assignment is **consistent** if $\text{Weight}(x) > 0$.

Objective: find the maximum weight assignment

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

A CSP is **satisfiable** if $\max_x \text{Weight}(x) > 0$.

Summary

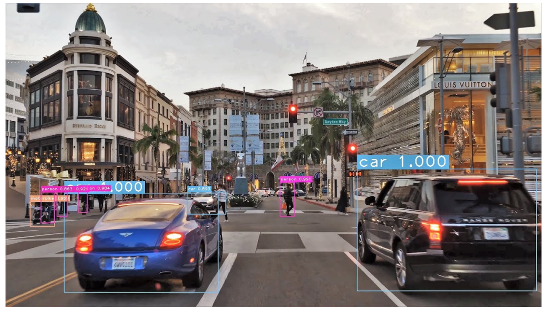


Variables, factors: specify locally

$$\text{Weight}(\{X_1 : \text{B}, X_2 : \text{B}, X_3 : \text{R}\}) = 1 \cdot 1 \cdot 2 \cdot 2 = 4$$

Assignments, weights: optimize globally

Example: object tracking

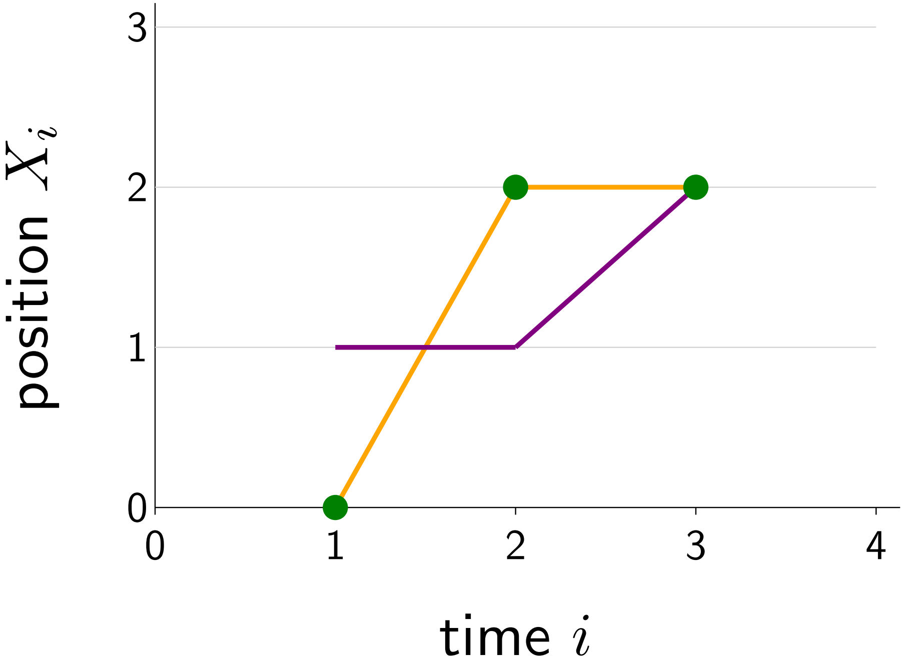


Problem: object tracking

(O) Noisy sensors report positions: $0, 2, 2$.

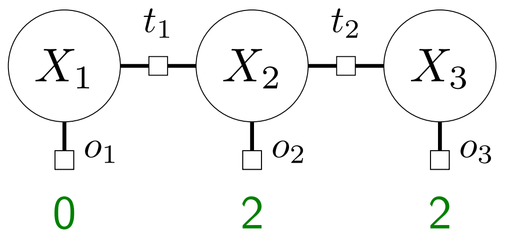
(T) Objects can't teleport.

What trajectory did the object take?



Example: object tracking CSP

Factor graph:



x_1	$o_1(x_1)$
0	2
1	1
2	0

x_2	$o_2(x_2)$
0	0
1	1
2	2

x_3	$o_3(x_3)$
0	0
1	1
2	2

$\lvert x_i - x_{i+1} \rvert$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

[demo]

- Variables $X_i \in \{0, 1, 2\}$: position of object at time i
- Observation factors $o_i(x_i)$: noisy information compatible with position
- Transition factors $t_i(x_i, x_{i+1})$: object positions can't change too much

Summary

- Decide on variables and domains

- Translate each desideratum into a set of factors

- Try to keep CSP small (variables, factors, domains, arities)

- When implementing each factor, think in terms of checking a solution rather than computing the solution

Backtracking search

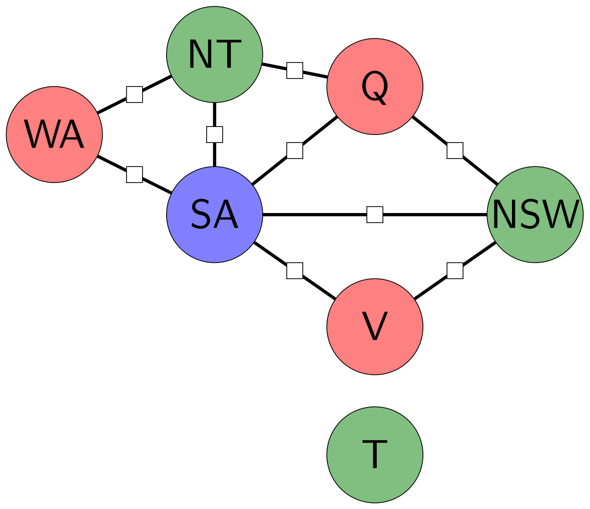
Algorithm: backtracking search

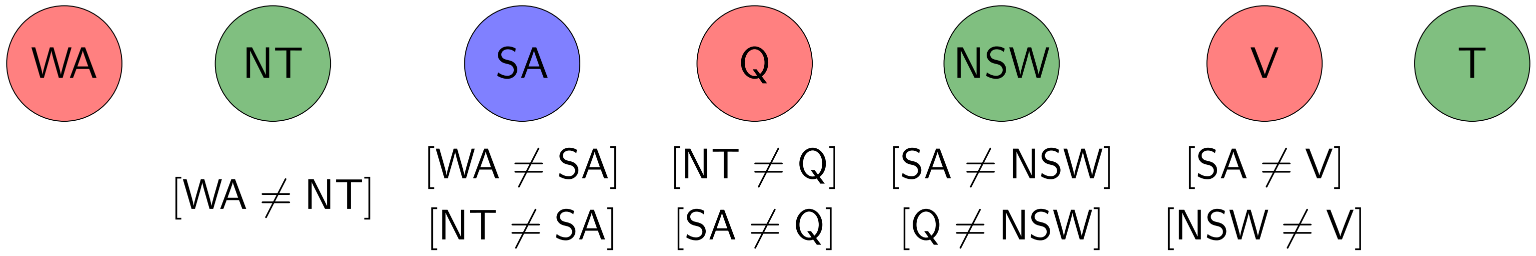
Backtrack(x, w, Domains):

- 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$'$)

Partial assignment weights

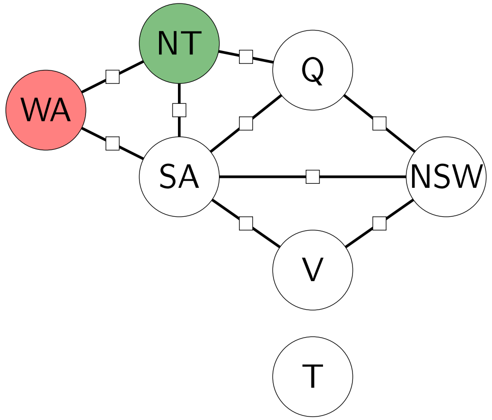
Idea: compute weight of partial assignment as we go

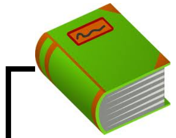




Dependent factors

- Partial assignment (e.g., $x = \{\text{WA} : \text{R}, \text{NT} : \text{G}\}$)





Definition: dependent factors

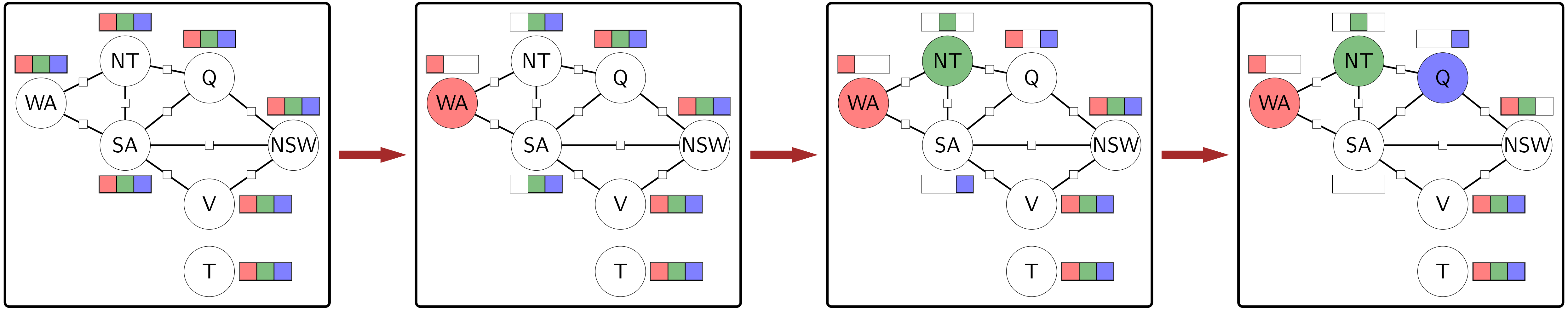
Let $D(x, X_i)$ be set of factors depending on X_i and x but not on unassigned variables.

$$D(\{\text{WA} : \text{R}, \text{NT} : \text{G}\}, \text{SA}) = \{[\text{WA} \neq \text{SA}], [\text{NT} \neq \text{SA}]\}$$

Lookahead: forward checking

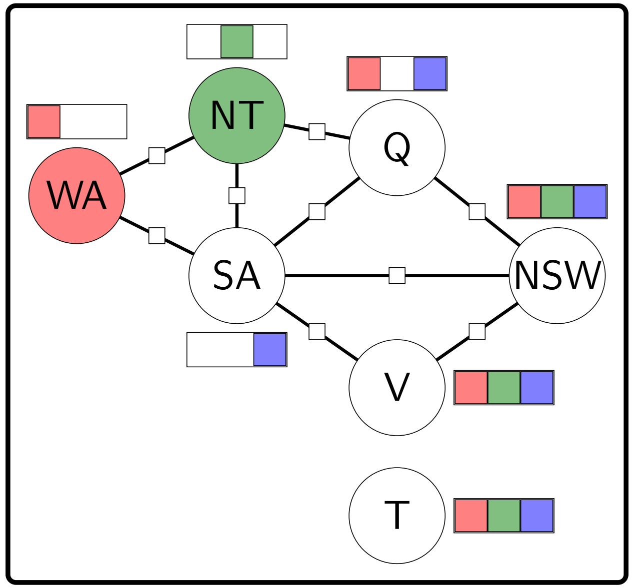
Key idea: forward checking (one-step lookahead)

- After assigning a variable X_i, eliminate inconsistent values from the domains of X_i's neighbors.
- If any domain becomes empty, return.



Inconsistent!

Choosing an unassigned variable



Which variable to assign next?

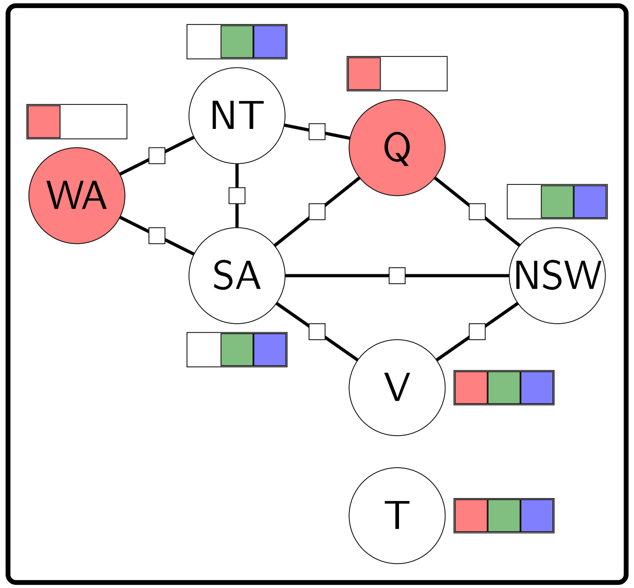
Key idea: most constrained variable

Choose variable that has the smallest domain.

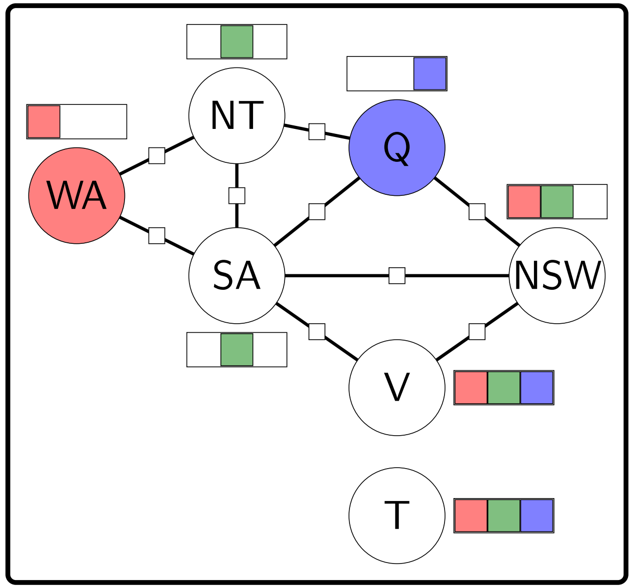
This example: SA (has only one value)

Ordering values of a selected variable

What values to try for Q?



$2 + 2 + 2 = 6$ consistent values

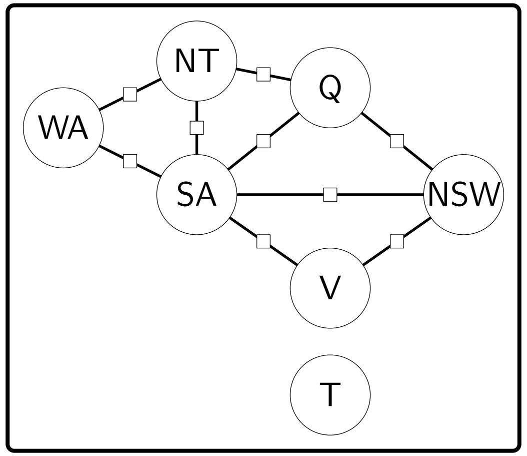


$1 + 1 + 2 = 4$ consistent values

Key idea: least constrained value

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

When to fail?



Most constrained variable (MCV):

- Must assign **every** variable
- If going to fail, fail early $\Rightarrow$ more pruning

Least constrained value (LCV):

- Need to choose **some** value
- Choose value that is most likely to lead to solution

When do these heuristics help?

- Most constrained variable: useful when **some** factors are constraints (can prune assignments with weight 0)

$$[x_1 = x_2] \qquad\qquad [x_2 \neq x_3] + 2$$

- Least constrained value: useful when **all** factors are constraints (all assignment weights are 1 or 0)

$$[x_1 = x_2] \qquad\qquad [x_2 \neq x_3]$$

- Forward checking: needed to prune domains to make heuristics useful!

Arc consistency

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.

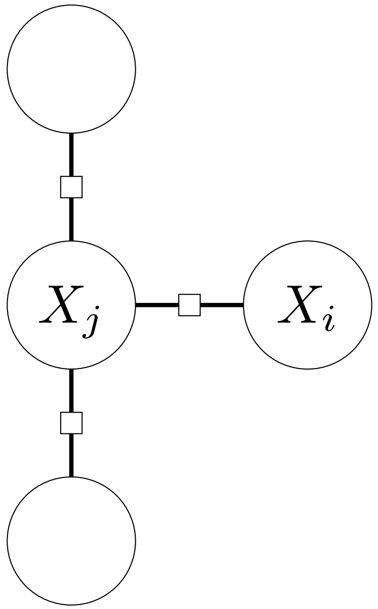
AC-3

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

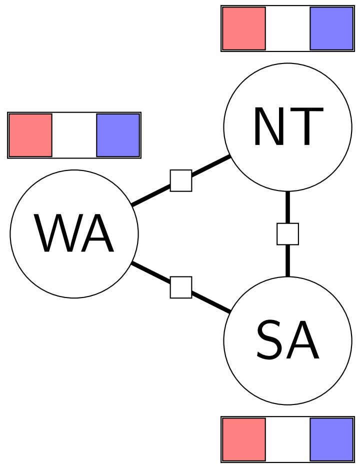
Algorithm: AC-3

$S \leftarrow \{X_j\}$.
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.



Limitations of AC-3

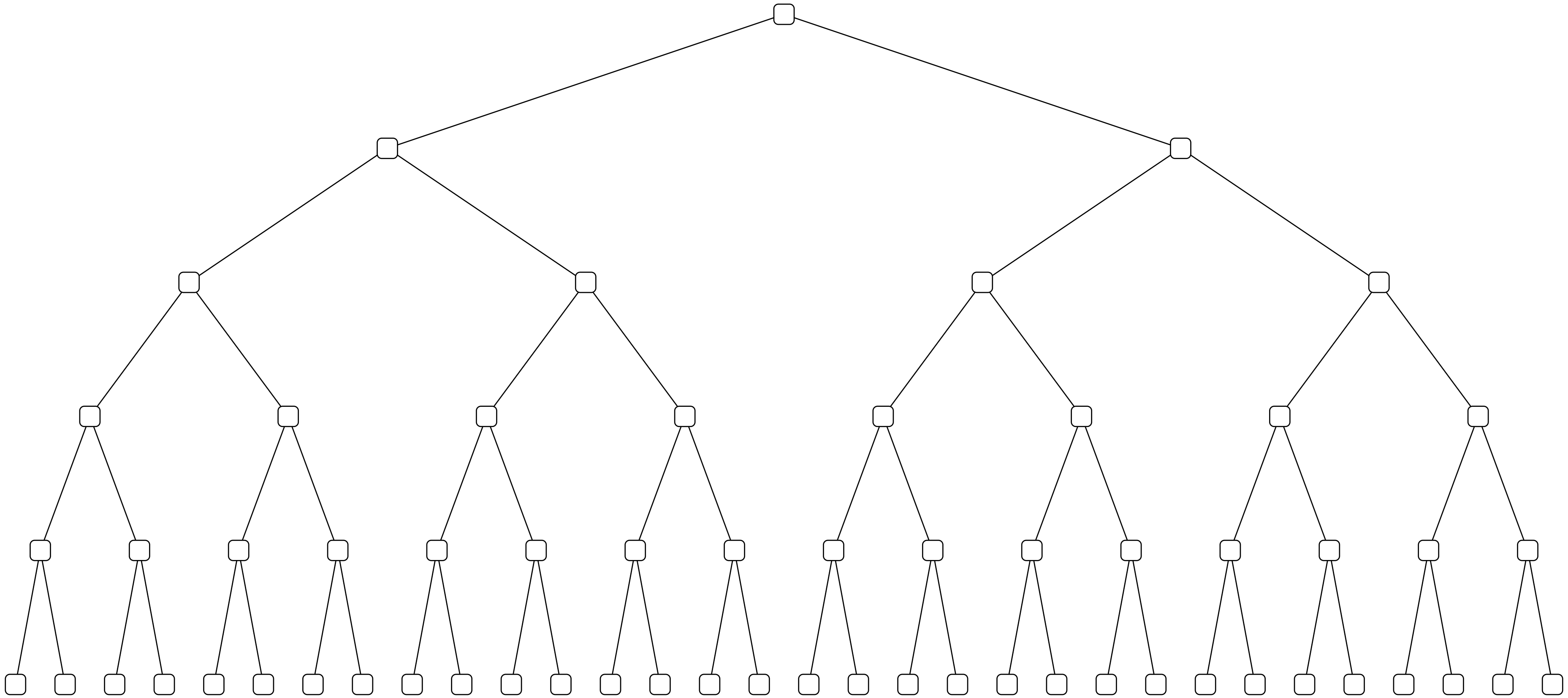
- AC-3 isn't always effective:



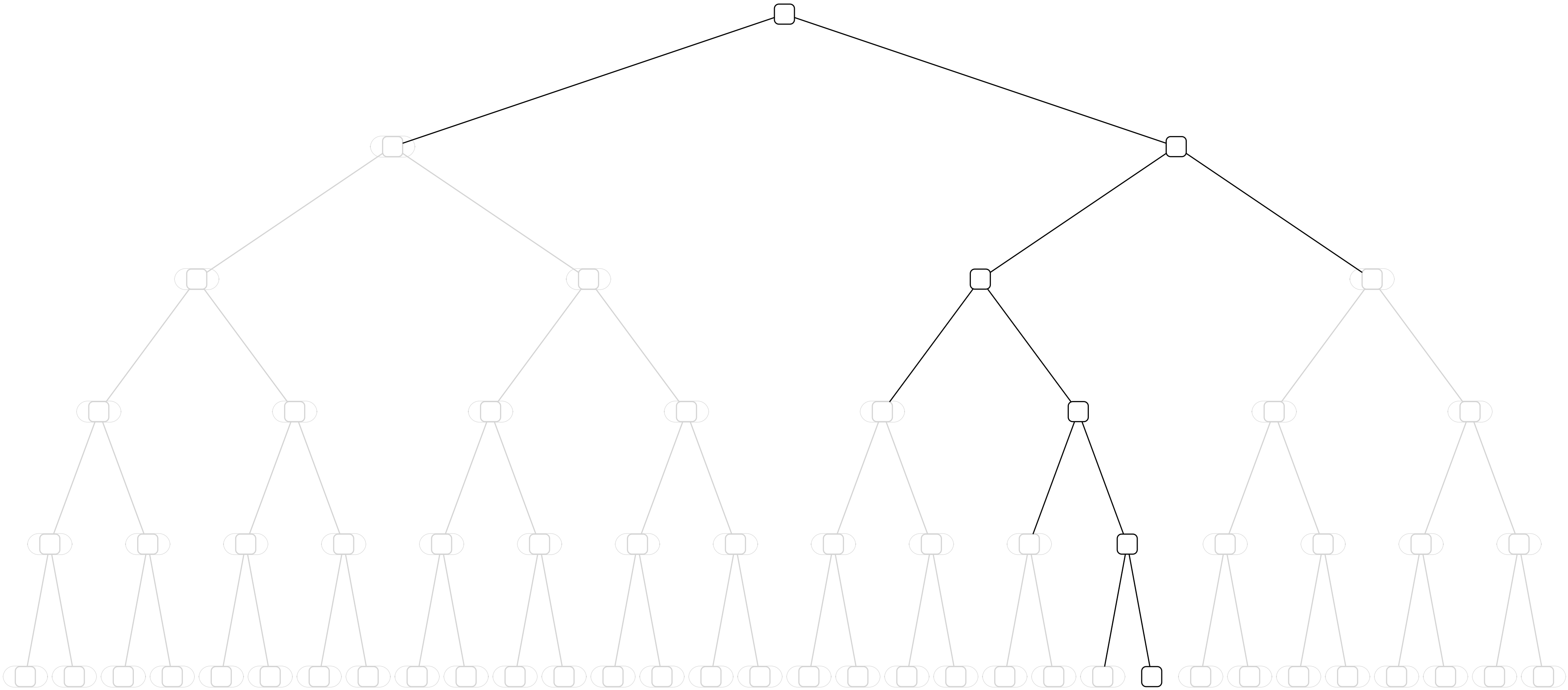
- No consistent assignments, but AC-3 doesn't detect a problem!

- Intuition: if we look locally at the graph, nothing blatantly wrong...

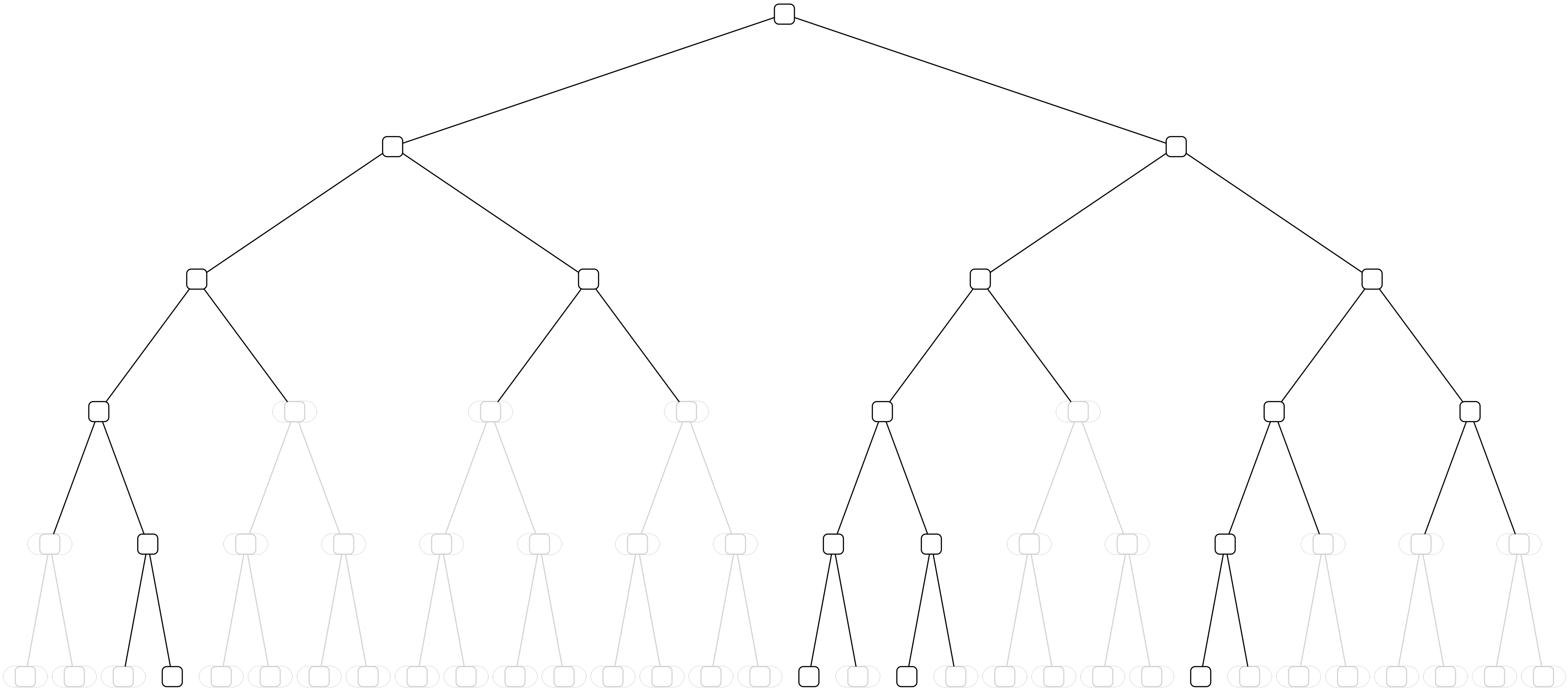
Backtracking search



Greedy search



Beam search



Beam size $K = 4$

Beam search

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

Algorithm: beam search

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

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

Extend:

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

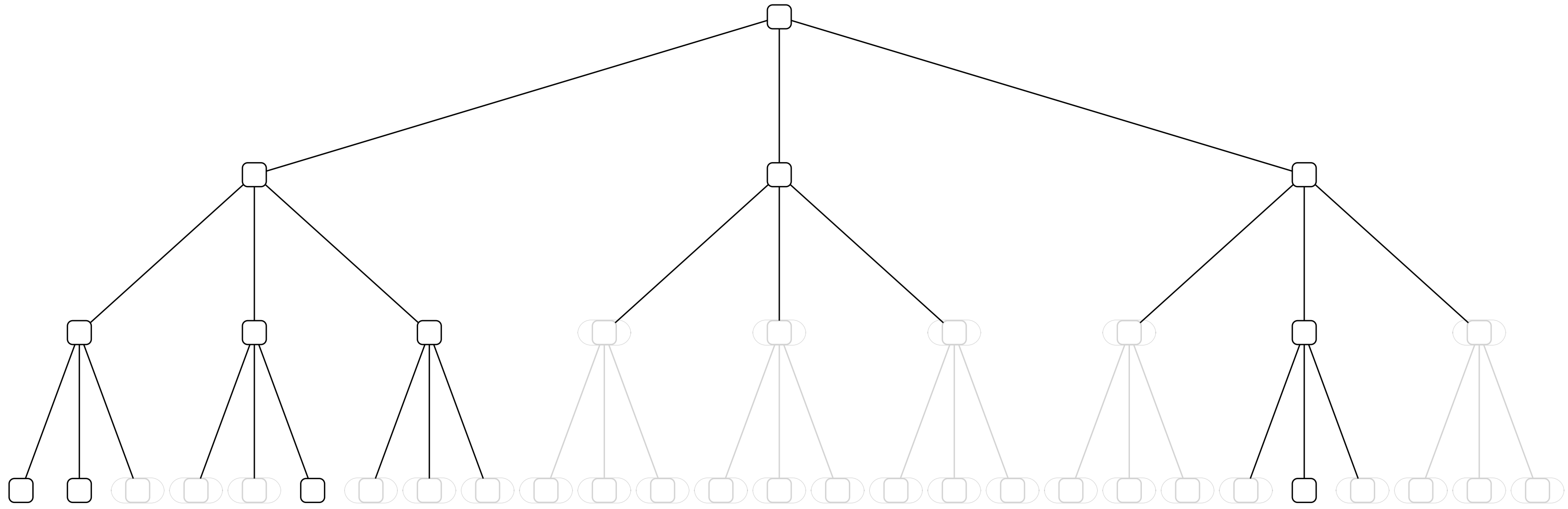
Prune:

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

Not guaranteed to find maximum weight assignment!

[demo: `beamSearch({K:3})`]

Summary



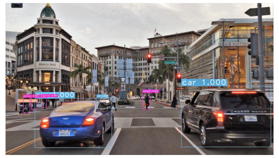
- Beam size K controls tradeoff between efficiency and accuracy
  - $K = 1$ is greedy search ($O(nb)$ time)
  - $K = \infty$ is BFS ($O(b^n)$ time)

Backtracking search : DFS :: beam search : pruned BFS

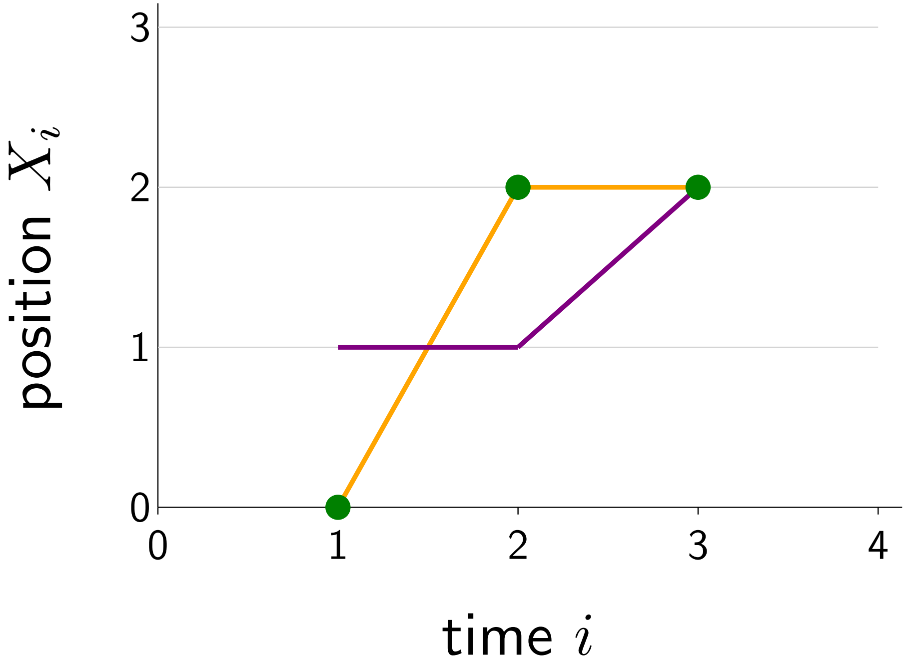
Search strategies

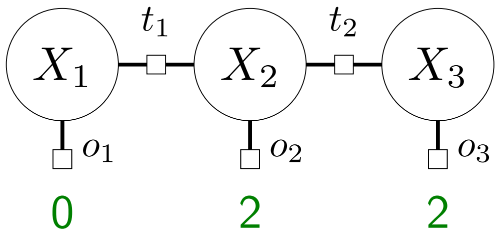
Backtracking/beam search: extend partial assignments

Local search: modify complete assignments



Example: object tracking





x_1	$o_1(x_1)$
0	2
1	1
2	0

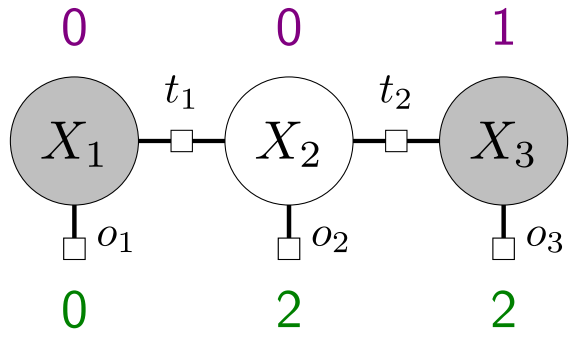
x_2	$o_2(x_2)$
0	0
1	1
2	2

x_3	$o_3(x_3)$
0	0
1	1
2	2

$\lvert x_i - x_{i+1} \rvert$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

[demo]

One small step

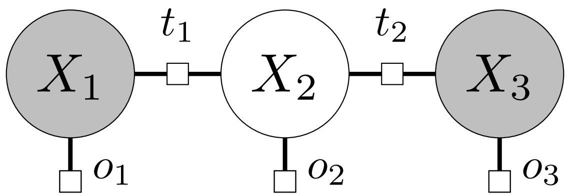


Old assignment: $(0, 0, 1)$; how to improve?

(x_1, v, x_3)	weight
$(0, 0, 1)$	$2 \cdot 2 \cdot 0 \cdot 1 \cdot 1 = 0$
$(0, 1, 1)$	$2 \cdot 1 \cdot 1 \cdot 2 \cdot 1 = 4$
$(0, 2, 1)$	$2 \cdot 0 \cdot 2 \cdot 1 \cdot 1 = 0$

New assignment: $(0, 1, 1)$

Exploiting locality



Weight of new assignment (x_1, v, x_3):

$$o_1(x_1)t_1(x_1, v)o_2(v)t_2(v, x_3)o_3(x_3)$$

Key idea: locality

When evaluating possible re-assignments to X_i, only need to consider the factors that depend on X_i.

Iterated conditional modes (ICM)

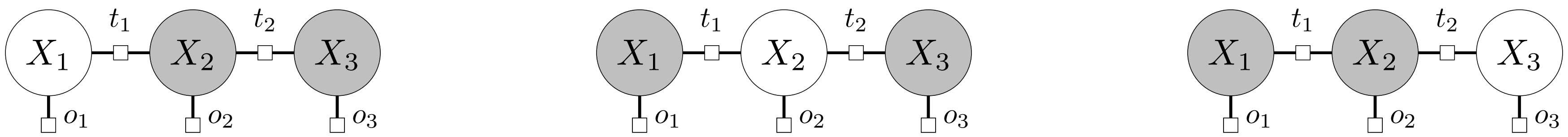
Algorithm: iterated conditional modes (ICM)

Initialize x to a random complete assignment

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

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

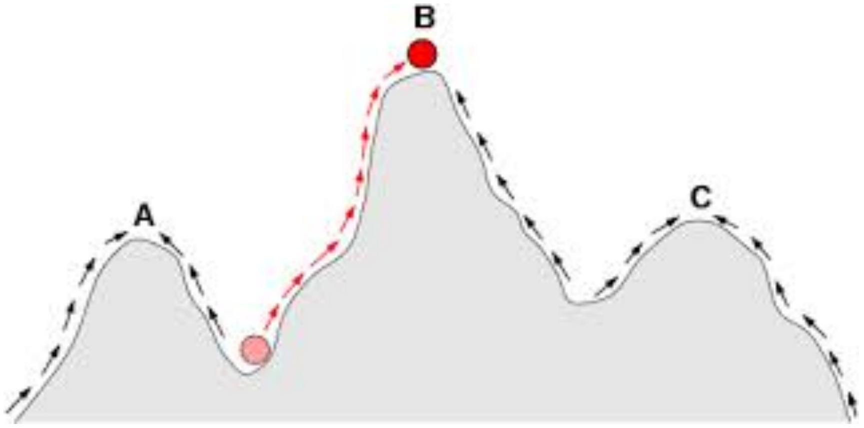
  $x \leftarrow x_v$ with highest weight



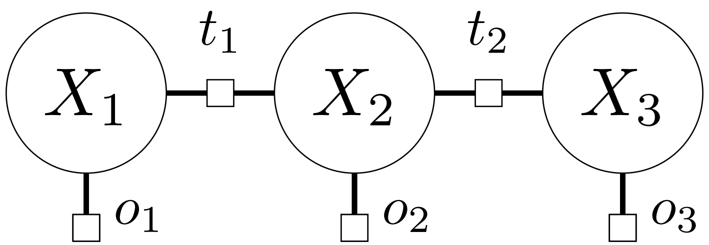
[demo: `iteratedConditionalModes()`]

Convergence properties

- Weight(x) increases or stays the same each iteration
- Converges in a finite number of iterations
- Can get stuck in **local optima**
- Not guaranteed to find optimal assignment!



Summary



Algorithm	Strategy	Optimality	Time complexity
Backtracking search	extend partial assignments	exact	exponential
Beam search	extend partial assignments	approximate	linear
Local search (ICM)	modify complete assignments	approximate	linear

Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Definition

Definition: Markov network

A Markov network is a factor graph which defines a joint distribution over random variables $X = (X_1, \dots, X_n)$:

$$\mathbb{P}(X = x) = \frac{\text{Weight}(x)}{Z}$$

where $Z = \sum_{x'} \text{Weight}(x')$ is the normalization constant.

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$$Z = 4 + 4 + 4 + 4 + 2 + 8 = 26$$

Represents uncertainty!

Marginal probabilities

Example question: where was the object at time step 2 (X_2)?

Definition: Marginal probability

The marginal probability of $X_i = v$ is given by:

$$\mathbb{P}(X_i = v) = \sum_{x : x_i = v} \mathbb{P}(X = x)$$

Object tracking example:

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$$\mathbb{P}(X_2 = 1) = 0.15 + 0.15 + 0.15 + 0.15 = 0.62$$

$$\mathbb{P}(X_2 = 2) = 0.08 + 0.31 = 0.38$$

Note: different than max weight assignment!

Summary

$$\boxed{\text{Markov networks} = \text{factor graphs} + \text{probability}}$$

- Normalize weights to get probablity distribution
- Can compute marginal probabilities to focus on variables

CSPs	Markov networks
variables	random variables
weights	probabilities
maximum weight assignment	marginal probabilities

Gibbs sampling

Algorithm: Gibbs sampling

Initialize x to a random complete assignment

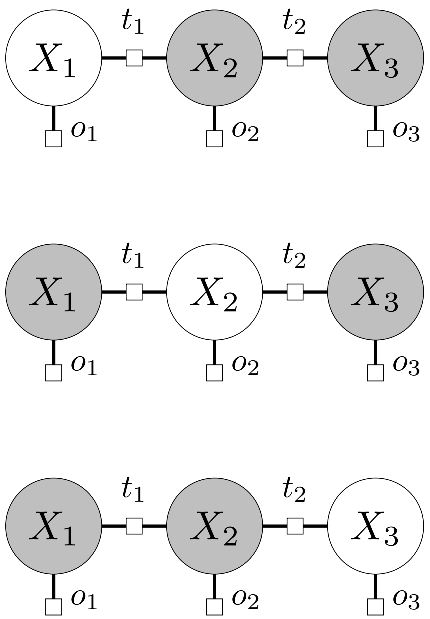
Loop through $i = 1, \dots, n$ until convergence:

  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

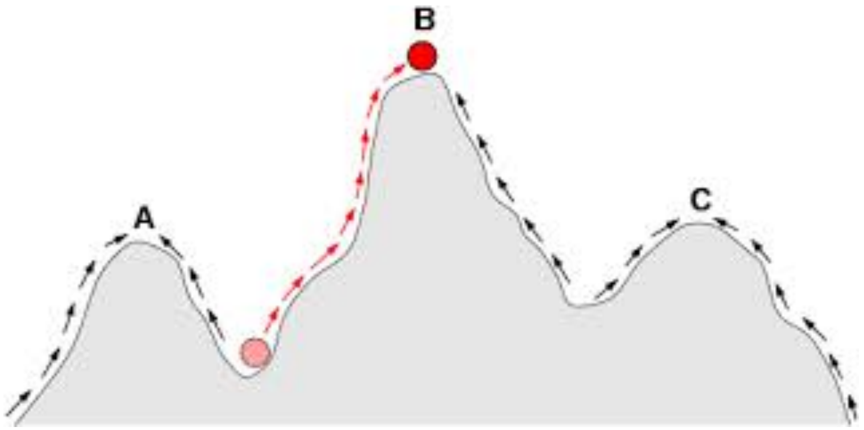
Weight$(x \cup \{X_2 : 2\}) = 2$ prob. 0.4

[demo]

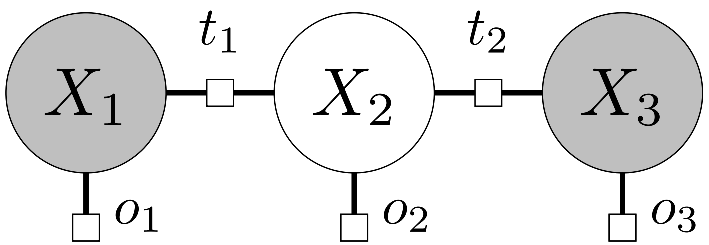
Search versus sampling

Iterated Conditional Modes	Gibbs sampling
maximum weight assignment	marginal probabilities
choose best value	sample a value
converges to local optimum	marginals converge to correct answer*

*under technical conditions (sufficient condition: all weights positive), but could take exponential time



Summary



- Objective: compute marginal probabilities $\mathbb{P}(X_i = x_i)$
- Gibbs sampling: sample one variable at a time, count visitations
- More generally: Markov chain Monte Carlo (MCMC) powerful toolkit of randomized procedures

Course plan

Constraint satisfaction problems

Search problems

Markov networks

Markov decision processes

$\boxed{\text{Bayesian networks}}$

Adversarial games

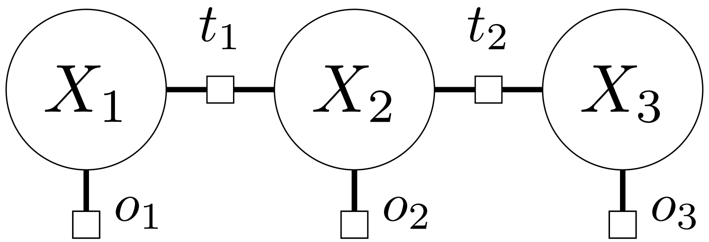
Reflex **States** **Variables** **Logic**

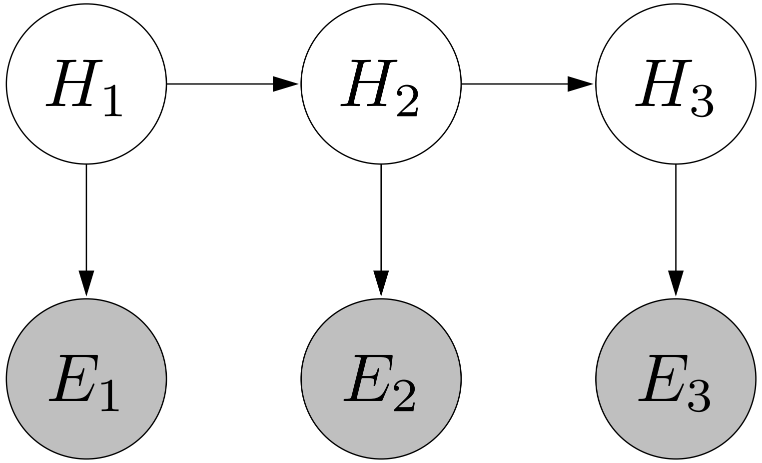
"Low-level intelligence" "High-level intelligence"

Machine learning

Markov networks versus Bayesian networks

Both define a joint probability distribution over assignments





Markov networks	Bayesian networks
arbitrary factors	local conditional probabilities
set of preferences	generative process

Review: probability

Random variables: sunshine $S \in \{0,1\}$, rain $R \in \{0,1\}$

Joint distribution (probabilistic database):

$\mathbb{P}(S,R) =$

s	r	$\mathbb{P}(S=s,R=r)$
0	0	0.20
0	1	0.08
1	0	0.70
1	1	0.02

Marginal distribution:

(aggregate rows)

$\mathbb{P}(S) =$

s	$\mathbb{P}(S=s)$
0	0.28
1	0.72

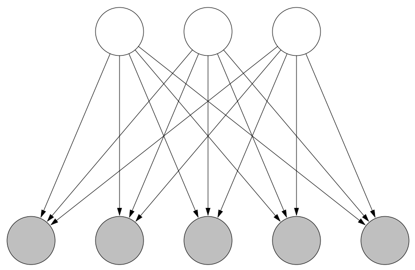
Conditional distribution:

(select rows, normalize)

$\mathbb{P}(S \mid R=1) =$

s	$\mathbb{P}(S=s \mid R=1)$
0	0.8
1	0.2

Bayesian network (definition)



Definition: Bayesian network

Let $X = (X_1, \dots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) \stackrel{\text{def}}{=} \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Probabilistic inference (definition)

Input

Bayesian network: $\mathbb{P}(X_1, \dots, X_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

Query: $Q \subseteq X$ is subset of variables

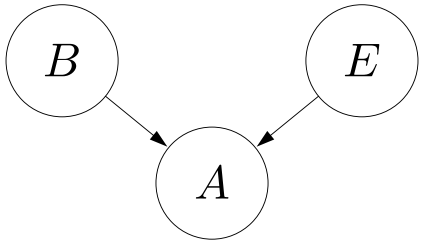
Output

$\mathbb{P}(Q \mid E = e) \longleftrightarrow \mathbb{P}(Q = q \mid E = e)$ for all values q

Example: if coughing and itchy eyes, have a cold?

$$\mathbb{P}(C \mid H = 1, I = 1)$$

Bayesian network (alarm)



b	$p(b)$
1	ϵ
0	$1-\epsilon$

e	$p(e)$
1	ϵ
0	$1-\epsilon$

b	e	a	$p(a \mid b, e)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

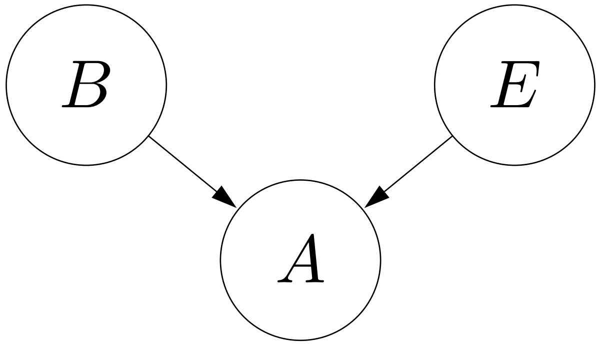
$$p(b) = \epsilon \cdot [b = 1] + (1 - \epsilon) \cdot [b = 0]$$

$$p(e) = \epsilon \cdot [e = 1] + (1 - \epsilon) \cdot [e = 0]$$

$$p(a \mid b, e) = [a = (b \vee e)]$$

$$\mathbb{P}(B = b, E = e, A = a) \stackrel{\text{def}}{=} p(b)p(e)p(a \mid b, e)$$

Explaining away



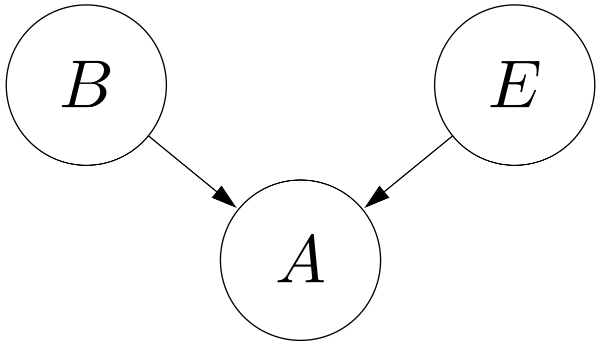
Key idea: explaining away

Suppose two causes positively influence an effect. Conditioned on the effect, further conditioning on one cause reduces the probability of the other cause.

$$\mathbb{P}(B = 1 \mid A = 1, E = 1) < \mathbb{P}(B = 1 \mid A = 1)$$

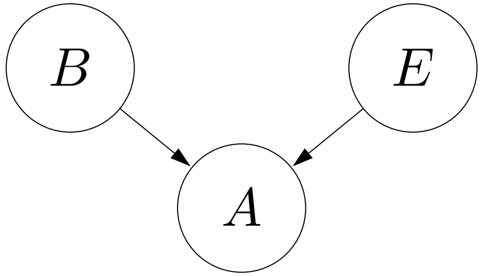
Note: happens even if causes are independent!

Summary



- Random variables capture state of world
- Directed edges between variables represent dependencies
- Local conditional distributions $\Rightarrow$ joint distribution
- Probabilistic inference: ask questions about world
- Captures reasoning patterns (e.g., explaining away)

Probabilistic programs



Joint distribution:

$$\mathbb{P}(B = b, E = e, A = a) = p(b)p(e)p(a \mid b, e)$$

Probabilistic program: alarm

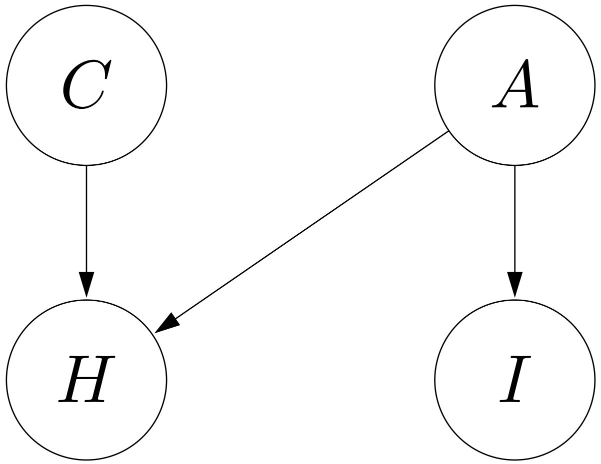
$$B \sim \text{Bernoulli}(\epsilon)$$
$$E \sim \text{Bernoulli}(\epsilon)$$
$$A = B \vee E$$

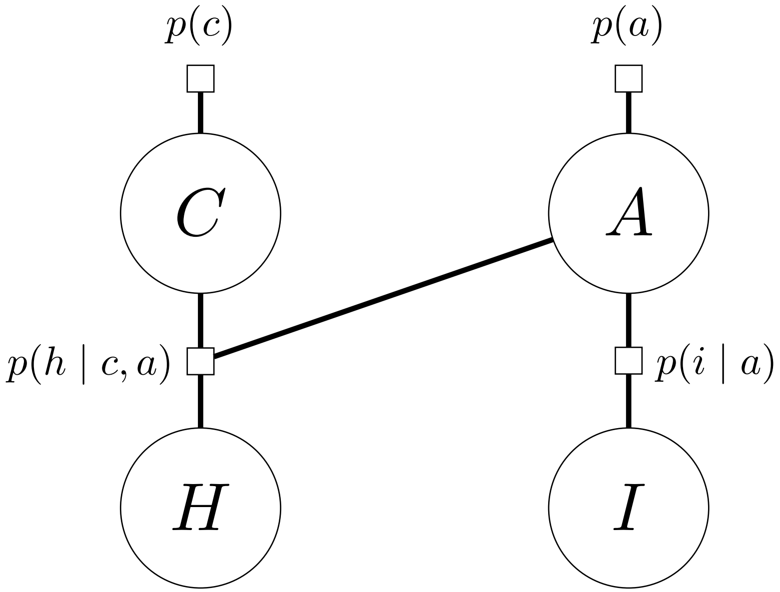
```
def Bernoulli(epsilon):
    return random.random() < epsilon
```

Key idea: probabilistic program

A randomized program that sets the random variables.

Reduction to Markov networks



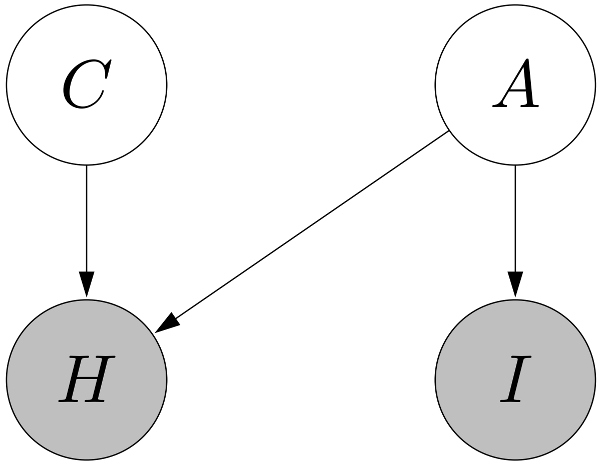


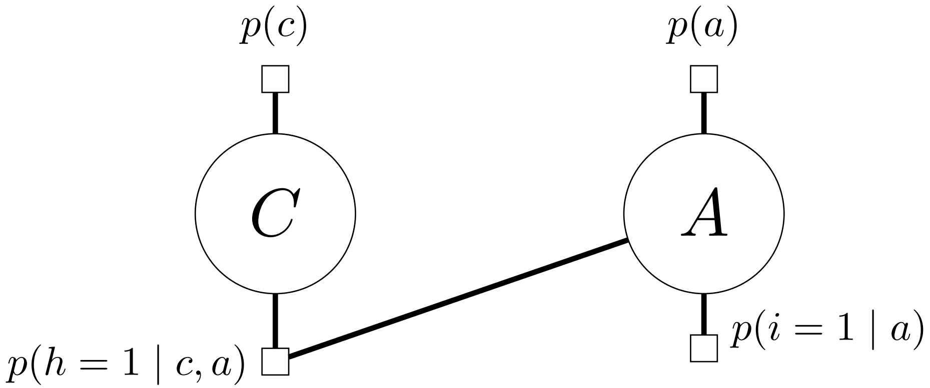
$$\mathbb{P}(C = c, A = a, H = h, I = i) = \frac{1}{Z} p(c) p(a) p(h \mid c, a) p(i \mid a)$$

Bayesian network $=$ Markov network with normalization constant $Z = 1$

Reminder: single factor that connects **all** parents!

Conditioning on evidence





Markov network:

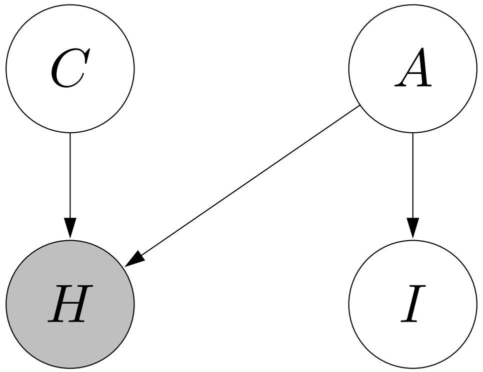
$$\mathbb{P}(C = c, A = a \mid H = 1, I = 1) = \frac{1}{Z} p(c) p(a) p(h = 1 \mid c, a) p(i = 1 \mid a)$$

Bayesian network with evidence = Markov network with $Z = \mathbb{P}(H = 1, I = 1)$

Solution: run any inference algorithm for Markov networks (e.g., Gibbs sampling)!

[demo]

Leveraging additional structure: unobserved leaves



Markov network:

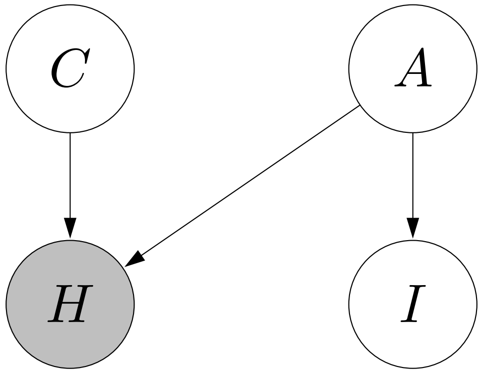
$$\mathbb{P}(C = c, A = a, I = i \mid H = 1) = \frac{1}{Z} p(c) p(a) p(h = 1 \mid c, a) p(i \mid a),$$

where $Z = \mathbb{P}(H = 1)$

Question: $\mathbb{P}(C = 1 \mid H = 1)$

Can we reduce the Markov network before running inference?

Leveraging additional structure: unobserved leaves

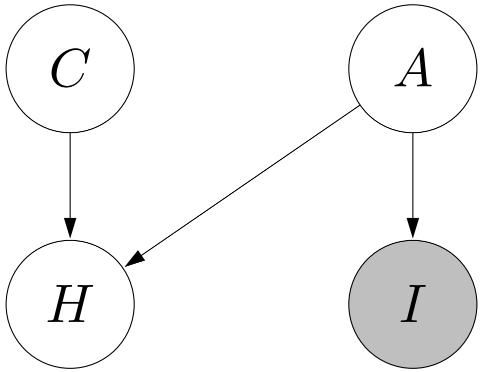


Markov network:

$$
\begin{aligned}
\mathbb{P}(C = c, A = a \mid H = 1) &= \textstyle\sum_i \mathbb{P}(C = c, A = a, I = i \mid H = 1) \\
&= \textstyle\sum_i \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a)p(i \mid a) \\
&= \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a) \textstyle\sum_i p(i \mid a) \\
&= \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a)
\end{aligned}
$$

Throw away any unobserved leaves before running inference!

Leveraging additional structure: independence

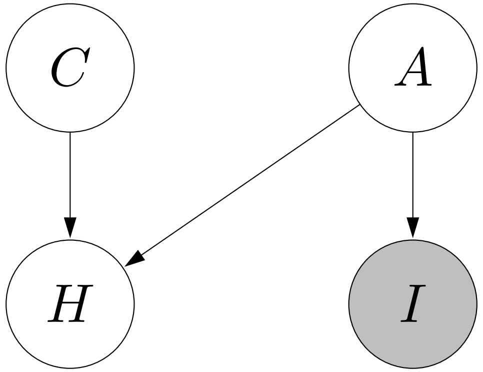


Markov network:

$$\begin{aligned}
\mathbb{P}(C = c \mid I = 1) &= \textstyle\sum_{a,h} \mathbb{P}(C = c, A = a, H = h \mid I = 1) \\
&= \textstyle\sum_{a,h} \frac{1}{Z} p(c) p(a) p(h \mid c, a) p(i = 1 \mid a) \\
&= \textstyle\sum_{a} \frac{1}{Z} p(c) p(a) p(i = 1 \mid a) \\
&= p(c) \textstyle\sum_{a} \frac{1}{Z} p(a) p(i = 1 \mid a) \\
&= p(c)
\end{aligned}$$

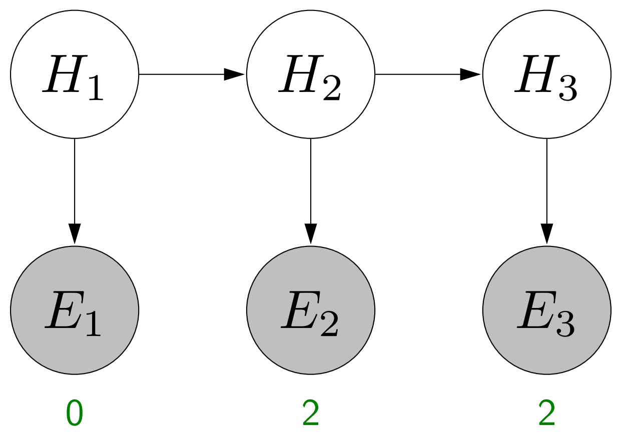
Throw away any disconnected components before running inference!

Summary



- Condition on evidence (e.g., $I = 1$)
- Throw away unobserved leaves (e.g., H)
- Throw away disconnected components (e.g., A and I)
- Define Markov network out of remaining factors
- Run your favorite inference algorithm (e.g., manual, Gibbs sampling)

Inference questions



Question (**filtering**):

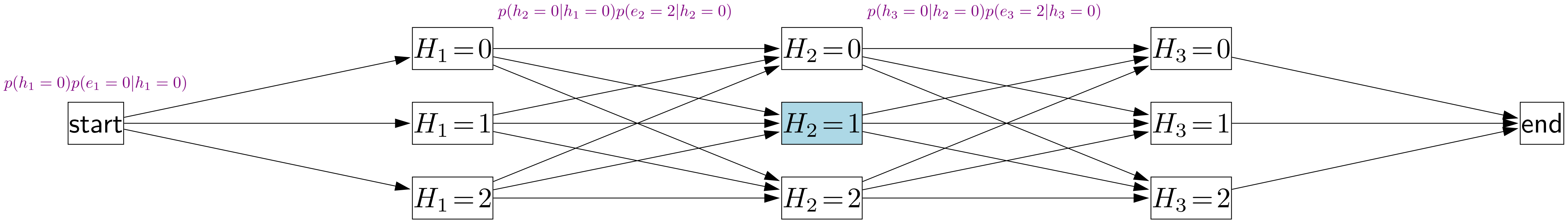
$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2)$$

Question (**smoothing**):

$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$

Note: filtering is a special case of smoothing if marginalize unobserved leaves

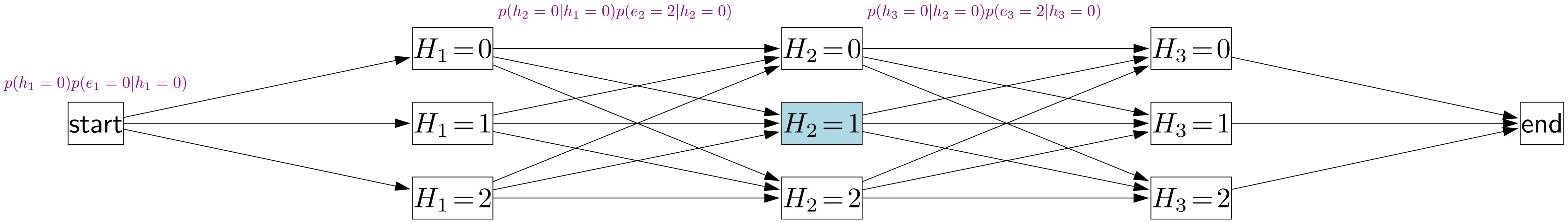
Lattice representation



- Edge $\boxed{\text{start}} \Rightarrow \boxed{H_1 = h_1}$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $\boxed{H_{i-1} = h_{i-1}} \Rightarrow \boxed{H_i = h_i}$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ is an assignment with weight equal to the product of edge weights

Key: $\mathbb{P}(H_i = h_i \mid E = e)$ is the weighted fraction of paths through $\boxed{H_i = h_i}$

Forward and backward messages



Forward: $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})\text{Weight}(\boxed{H_{i-1} = h_{i-1}}, \boxed{H_i = h_i})$

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{H_i = h_i}$

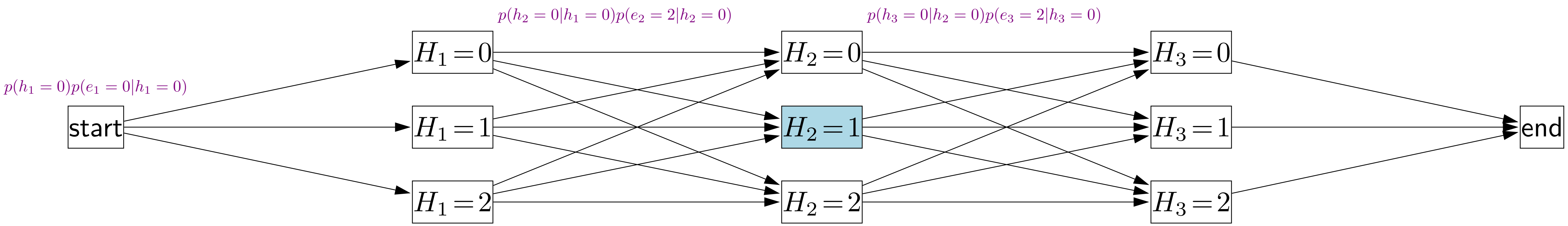
Backward: $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})\text{Weight}(\boxed{H_i = h_i}, \boxed{H_{i+1} = h_{i+1}})$

sum of weights of paths from $\boxed{H_i = h_i}$ to $\boxed{\text{end}}$

Define $S_i(h_i) = F_i(h_i)B_i(h_i)$:

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ through $\boxed{H_i = h_i}$

Putting everything together



$$\mathbb{P}(H_i = h_i \mid E = e) = \frac{S_i(h_i)}{\sum_v S_i(v)}$$

Algorithm: forward-backward algorithm

Compute $F_1, F_2, \dots, F_n$

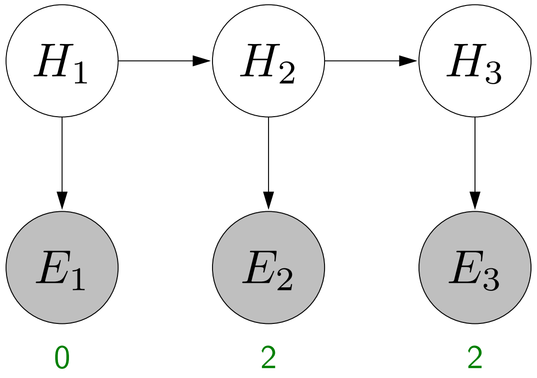
Compute $B_n, B_{n-1}, \dots, B_1$

Compute S_i for each i and normalize

Running time: $O(n|\text{Domain}|^2)$

[demo]

Review: inference in Hidden Markov models



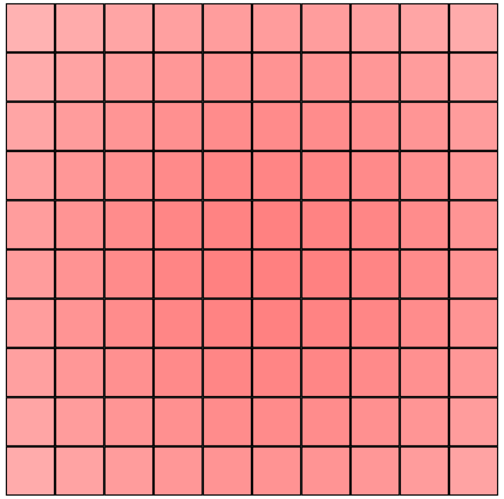
Filtering questions:

$\mathbb{P}(H_1 \mid E_1 = 0)$

$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2)$

$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

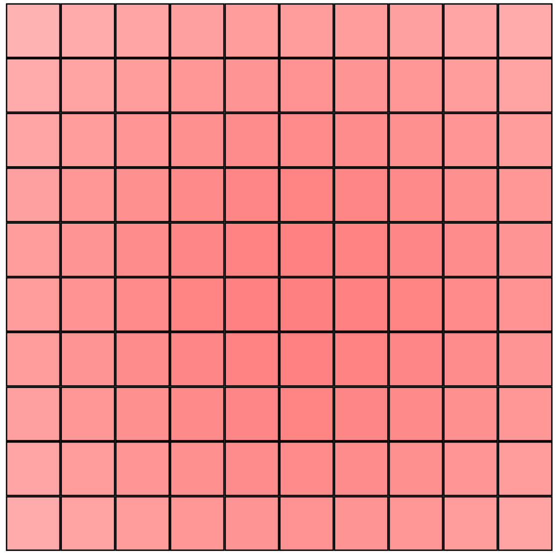
Problem: many possible location values for H_i



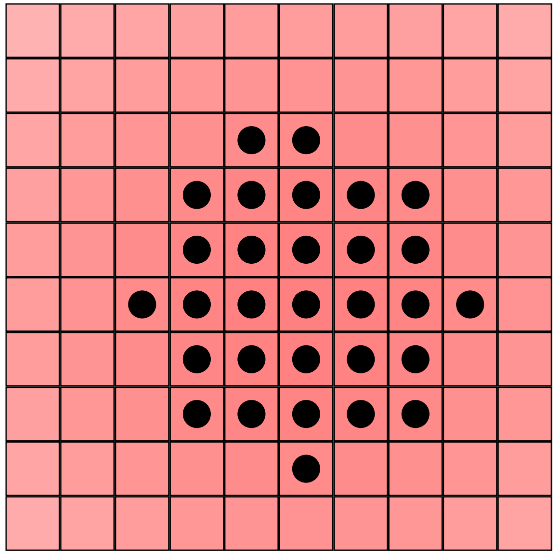
Forward-backward is too slow ($O(n|\text{Domain}|^2)$)...

Why sampling?

distribution

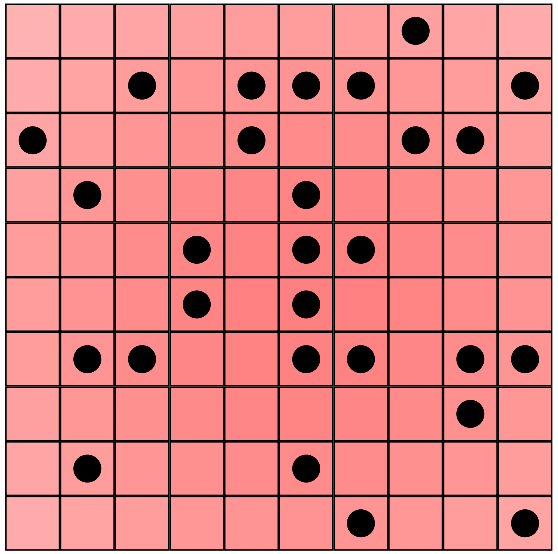


K with highest weight



not representative

K sampled from distribution



more representative

Sampling is especially important when there is high uncertainty!

Particle filtering

Algorithm: particle filtering

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

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

Propose:

$$C' \leftarrow \{h \cup \{H_i : h_i\} : h \in C, h_i \sim p(h_i \mid h_{i-1})\}$$

Weight:

Compute weights $w(h) = p(e_i \mid h_i)$ for $h \in C'$

Resample:

$C \leftarrow K$ particles drawn independently from $\frac{w(h)}{\sum_{h' \in C} w(h')}$

[demo: `particleFiltering({K:100})`]

Step 1: propose

Old particles: $\approx \mathbb{P}(H_1, H_2 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1\}$
$\{H_1 : 1, H_2 : 2\}$

Key idea: proposal distribution

For each old particle (h_1, h_2), sample $H_3 \sim p(h_3 \mid h_2)$.

h_i	$p(h_i \mid h_{i-1})$
$h_{i-1} - 1$	1/4
h_{i-1}	1/2
$h_{i-1} + 1$	1/4

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1, H_3 : 1\}$
$\{H_1 : 1, H_2 : 2, H_3 : 2\}$

Step 2: weight

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$

$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

Key idea: weighting based on evidence

For each old particle (h_1, h_2, h_3), weight it by $p(e_3 = 2 \mid h_3)$.

h_3	$p(e_3 = 2 \mid h_3)$
0	0
1	1/4
2	1/2

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ (1/4)

$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ (1/2)

Step 3: resample

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ (1/4) $\Rightarrow$ 1/3

$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ (1/2) $\Rightarrow$ 2/3

Key idea: resampling

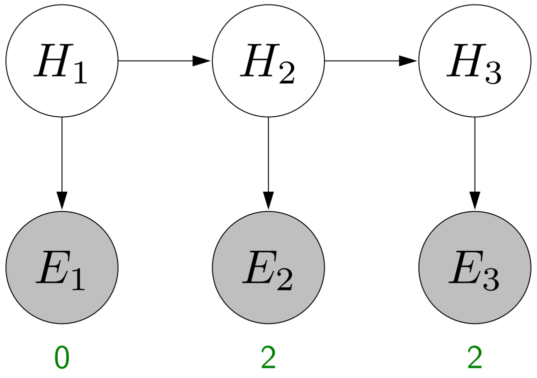
Normalize weights and draw K samples to redistribute particles to more promising areas.

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

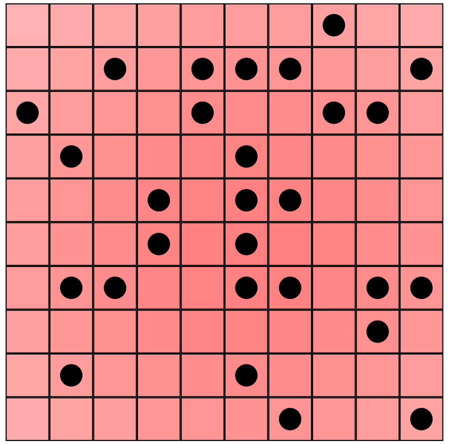
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

Summary



$$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$

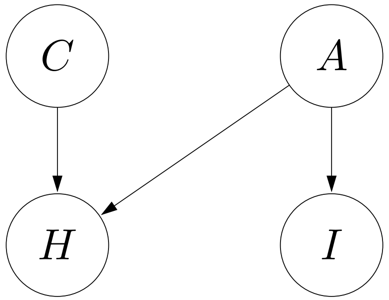


- Use particles to represent an approximate distribution

Propose (transitions) **Weight** (emissions) **Resample**

- Can scale to large number of locations (unlike forward-backward)
- Maintains better particle diversity (compared to beam search)

Where do parameters come from?



c	$p(c)$
1	?
0	?

a	$p(a)$
1	?
0	?

c	a	h	$p(h \mid c, a)$
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

a	i	$p(i \mid a)$
0	0	?
0	1	?
1	0	?
1	1	?

Learning task

Training data

$\mathcal{D}_{\text{train}}$ (an example is an assignment to X)

$\Downarrow$

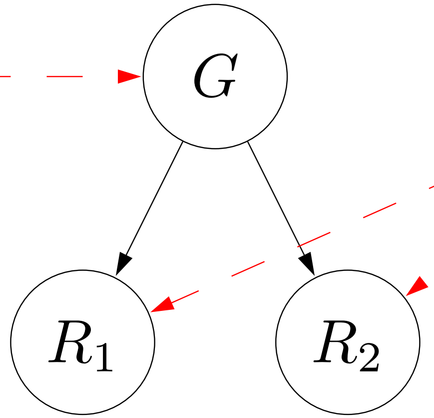
Parameters

θ (local conditional probabilities)

Parameter sharing

Key idea: parameter sharing

The local conditional distributions of different variables can share the same parameters.



g	$p_G(g)$
c	2/5
d	3/5

g	r	$p_R(r \mid g)$
d	3	1/6
d	4	3/6
d	5	2/6
c	1	1/4
c	2	1/4
c	4	1/4
c	5	1/4

Impact: more reliable estimates, less expressive model

General case

Bayesian network: variables $X_1, \dots, X_n$

Parameters: collection of distributions $\theta = \{p_d : d \in D\}$ (e.g., $D = \{\text{start}, \text{trans}, \text{emit}\}$)

Each variable X_i is generated from distribution p_{d_i}:

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^{n} p_{d_i}(x_i \mid x_{\text{Parents}(i)})$$

Parameter sharing: d_i could be same for multiple i

General case: learning algorithm

Input: training examples $\mathcal{D}_{\text{train}}$ of full assignments

Output: parameters $\theta = \{p_d : d \in D\}$

Algorithm: count and normalize

Count:

For each $x \in \mathcal{D}_{\text{train}}$:

For each variable x_i:

Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

Normalize:

For each d and local assignment $x_{\text{Parents}(i)}$:

Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$

Maximum likelihood

Maximum likelihood objective:

$$\max_{\theta} \prod_{x \in \mathcal{D}_{\text{train}}} \mathbb{P}(X = x; \theta)$$

Algorithm: maximum likelihood

Count:

For each $x \in \mathcal{D}_{\text{train}}$:

For each variable x_i:

Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

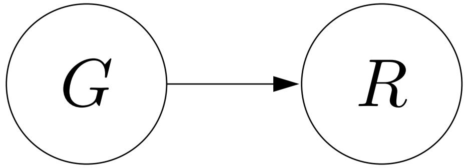
Normalize:

For each d and local assignment $x_{\text{Parents}(i)}$:

Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$

Closed form — no iterative optimization!

Review: maximum likelihood



$$\mathbb{P}(G = g, R = r) = p_G(g)p_R(r \mid g)$$

$$\mathcal{D}_{\text{train}} = \{(\text{d}, 4), (\text{d}, 4), (\text{d}, 5), (\text{c}, 1), (\text{c}, 5)\}$$

θ:

g	$\text{count}_G(g)$	$p_G(g)$
d	3	3/5
c	2	2/5

g	r	$\text{count}_R(g, r)$	$p_R(r \mid g)$
d	4	2	2/3
d	5	1	1/3
c	1	1	1/2
c	5	1	1/2

Do we really believe that $p_R(r = 2 \mid g = \text{c}) = 0$?

Overfitting!

Laplace smoothing

Key idea: maximum likelihood with Laplace smoothing

For each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$:

Add λ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$.

Further increment counts $\{\text{count}_d\}$ based on $\mathcal{D}_{\text{train}}$.

Hallucinate λ occurrences of each local assignment

Interplay between smoothing and data

Larger $\lambda \Rightarrow$ more smoothing $\Rightarrow$ probabilities closer to uniform

g	$\text{count}_G(g)$	$p_G(g)$
d	$1/2+1$	$3/4$
c	$1/2$	$1/4$

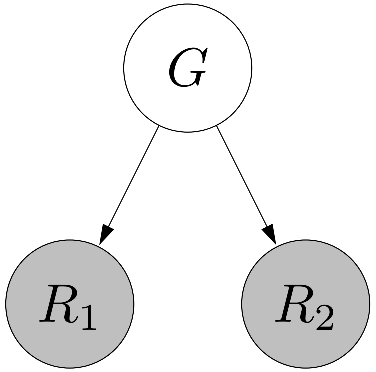
g	$\text{count}_G(g)$	$p_G(g)$
d	$1+1$	$2/3$
c	1	$1/3$

Data wins out in the end (suppose only see $g =$ d):

g	$\text{count}_G(g)$	$p_G(g)$
d	$1+1$	$2/3$
c	1	$1/3$

g	$\text{count}_G(g)$	$p_G(g)$
d	$1+998$	0.999
c	1	0.001

Motivation



Genre $G \in \{\text{drama}, \text{comedy}\}$

Jim's rating $R_1 \in \{1, 2, 3, 4, 5\}$

Martha's rating $R_2 \in \{1, 2, 3, 4, 5\}$

If observe all the variables: maximum likelihood $=$ count and normalize

$$\mathcal{D}_{\text{train}} = \{(\text{d}, 4, 5), (\text{d}, 4, 4), (\text{d}, 5, 3), (\text{c}, 1, 2), (\text{c}, 5, 4)\}$$

What if we **don't observe** some of the variables?

$$\mathcal{D}_{\text{train}} = \{(?, 4, 5), (?, 4, 4), (?, 5, 3), (?, 1, 2), (?, 5, 4)\}$$

Expectation Maximization (EM)

Intuition: generalization of the K-means algorithm

cluster centroids = parameters θ cluster assignments = hidden variables H

Variables: H is hidden, $E = e$ is observed

Algorithm: Expectation Maximization (EM)

Initialize θ randomly

Repeat until convergence:

E-step:

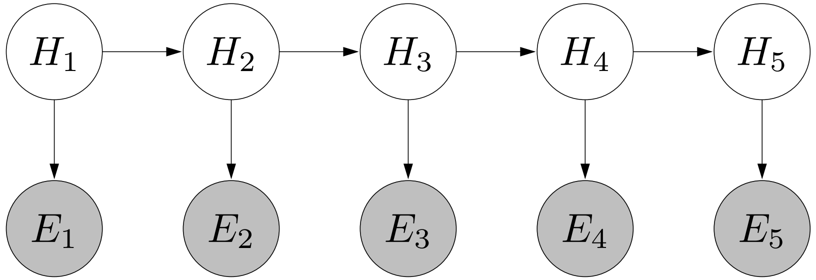
Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (probabilistic inference)

Create fully-observed weighted examples: (h, e) with weight $q(h)$

M-step:

Maximum likelihood (count and normalize) on weighted examples to get θ

Summary



Maximum marginal likelihood:

$$\max_\theta \prod_{e \in \mathcal{D}_{\text{train}}} \mathbb{P}(E = e; \theta)$$

EM algorithm:

$\Leftarrow$ probabilistic inference (E-step)

hidden variables $q(h)$

parameters θ

count and normalize (M-step) $\Rightarrow$

Applications: decipherment, phylogenetic reconstruction, crowdsourcing

Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Modeling paradigms

State-based models: search problems, MDPs, games

Applications: route finding, game playing, etc.

Think in terms of **states, actions, and costs**

Variable-based models: CSPs, Bayesian networks

Applications: scheduling, tracking, medical diagnosis, etc.

Think in terms of **variables and factors**

Logic-based models: propositional logic, first-order logic

Applications: theorem proving, verification, reasoning

Think in terms of **logical formulas and inference rules**

Natural language

Example:

- A **dime** is better than a **nickel**.
- A **nickel** is better than a **penny**.
- Therefore, a **dime** is better than a **penny**.

Example:

- A **penny** is better than **nothing**.
- **Nothing** is better than **world peace**.
- Therefore, a **penny** is better than **world peace**???

Natural language is slippery...

Two goals of a logic language

- **Represent** knowledge about the world

- **Reason** with that knowledge

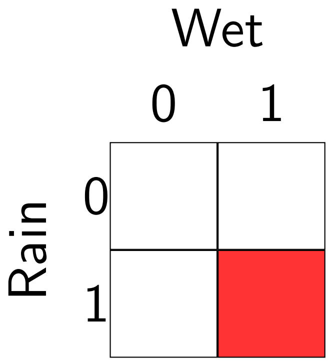
Ingredients of a logic

Syntax: defines a set of valid **formulas** (Formulas)

Example: Rain $\wedge$ Wet

Semantics: for each formula, specify a set of **models** (assignments / configurations of the world)

Example:



Inference rules: given f, what new formulas g can be added that are guaranteed to follow ($\frac{f}{g}$)?

Example: from Rain $\wedge$ Wet, derive Rain

Syntax versus semantics

Syntax: what are valid expressions in the language?

Semantics: what do these expressions mean?

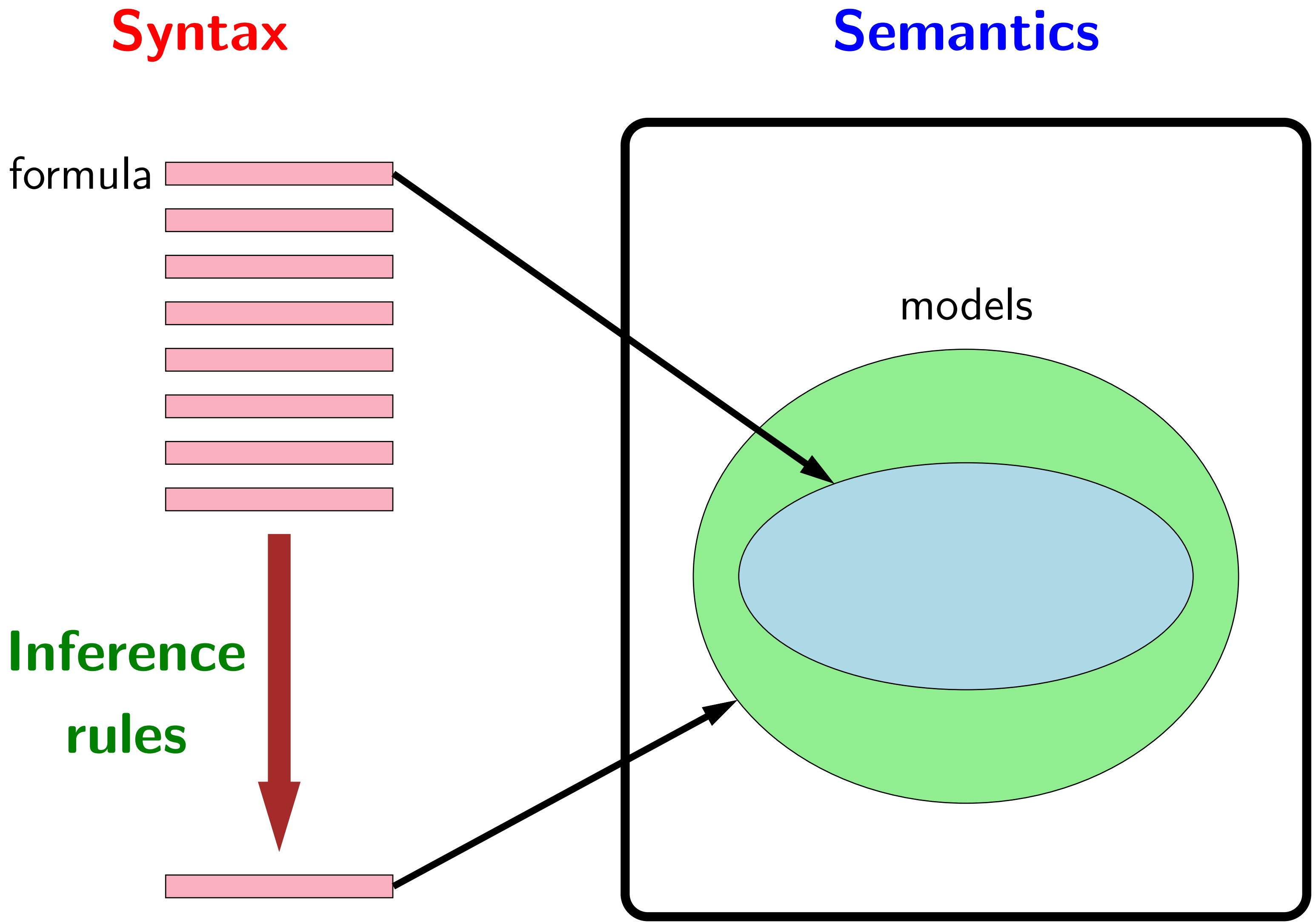
Different syntax, same semantics (5):

$$2 + 3 \Leftrightarrow 3 + 2$$

Same syntax, different semantics (1 versus 1.5):

$$3 \;/\; 2 \text{ (Python 2.7)} \not\Leftrightarrow 3 \;/\; 2 \text{ (Python 3)}$$

Propositional logic



Logics

- **Propositional logic with only Horn clauses**
- **Propositional logic**
- Modal logic
- **First-order logic with only Horn clauses**
- **First-order logic**
- Second-order logic
- ...

Key idea: tradeoff

Balance **expressivity** and **computational efficiency**.

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: $f \leftrightarrow g$

Syntax of propositional logic

Key idea: syntax provides symbols

Formulas by themselves are just symbols (syntax).

No meaning yet (semantics)!

Model

Definition: model

A **model** w in propositional logic is an **assignment** of truth values to propositional symbols.

Example:

- 3 propositional symbols: A, B, C
- $2^3 = 8$ possible models w:

$$\begin{gathered}\{A:0, B:0, C:0\}\\ \{A:0, B:0, C:1\}\\ \{A:0, B:1, C:0\}\\ \{A:0, B:1, C:1\}\\ \{A:1, B:0, C:0\}\\ \{A:1, B:0, C:1\}\\ \{A:1, B:1, C:0\}\\ \{A:1, B:1, C:1\}\end{gathered}$$

Interpretation function

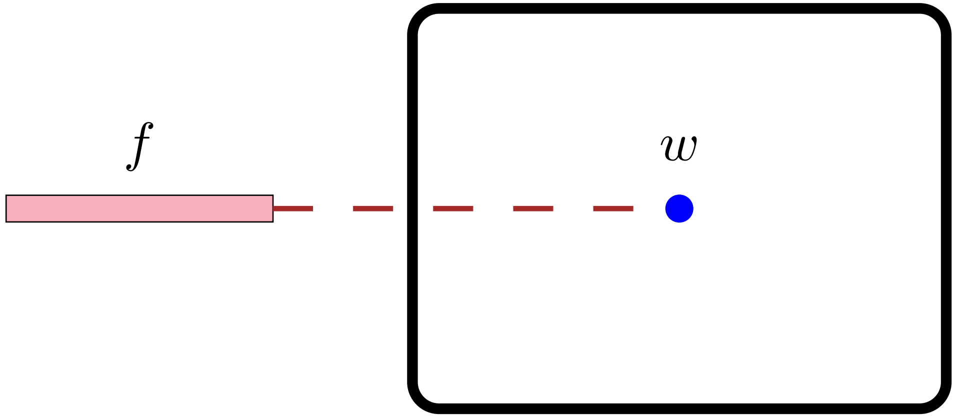
Definition: interpretation function

Let f be a formula.

Let w be a model.

An **interpretation function** $\mathcal{I}(f, w)$ returns:

- true (1) (say that w satisfies f)
- false (0) (say that w does not satisfy f)



Interpretation function: example

Example: interpretation function

Formula: $f = (\neg A \wedge B) \leftrightarrow C$

Model: $w = \{A : 1, B : 1, C : 0\}$

Interpretation:

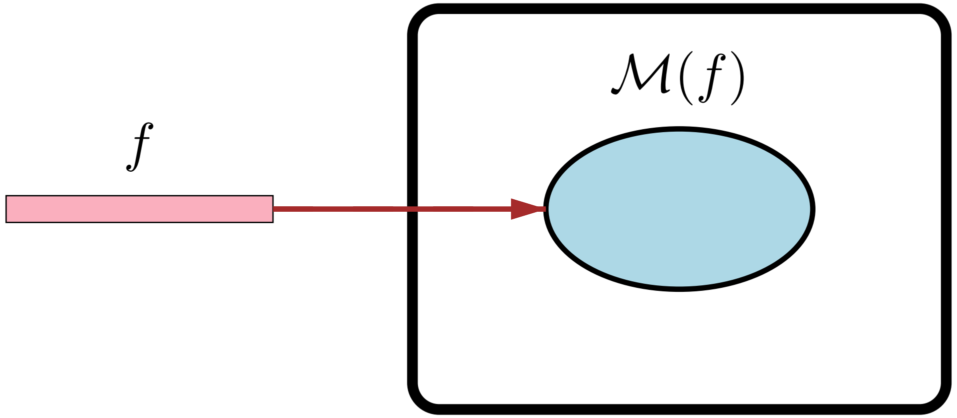
- $\mathcal{I}((\neg A \wedge B) \leftrightarrow C, w) = 1$
  - $\mathcal{I}(\neg A \wedge B, w) = 0$
    - $\mathcal{I}(\neg A, w) = 0$
      - $\mathcal{I}(A, w) = 1$
    - $\mathcal{I}(B, w) = 1$
  - $\mathcal{I}(C, w) = 0$

Formula represents a set of models

So far: each formula f and model w has an interpretation $\mathcal{I}(f, w) \in \{0, 1\}$

Definition: models

Let $\mathcal{M}(f)$ be the set of **models** w for which $\mathcal{I}(f, w) = 1$.



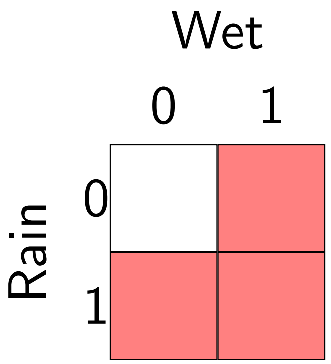
Models: example

Formula:

$$f = \text{Rain} \vee \text{Wet}$$

Models:

$$\mathcal{M}(f) =$$



Key idea: compact representation

A **formula** *compactly* represents a set of **models**.

Knowledge base

Definition: Knowledge base

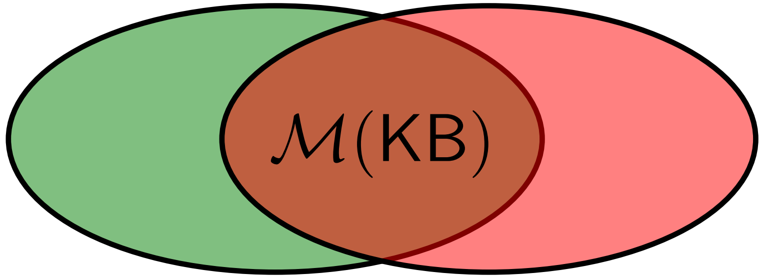
A **knowledge base** KB is a set of formulas representing their conjunction / intersection:

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

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

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

$\mathcal{M}(\text{Rain} \vee \text{Snow})$



$\mathcal{M}(\text{Traffic})$

Adding to the knowledge base

Adding more formulas to the knowledge base:

$$\text{KB} \longrightarrow \text{KB} \cup \{f\}$$

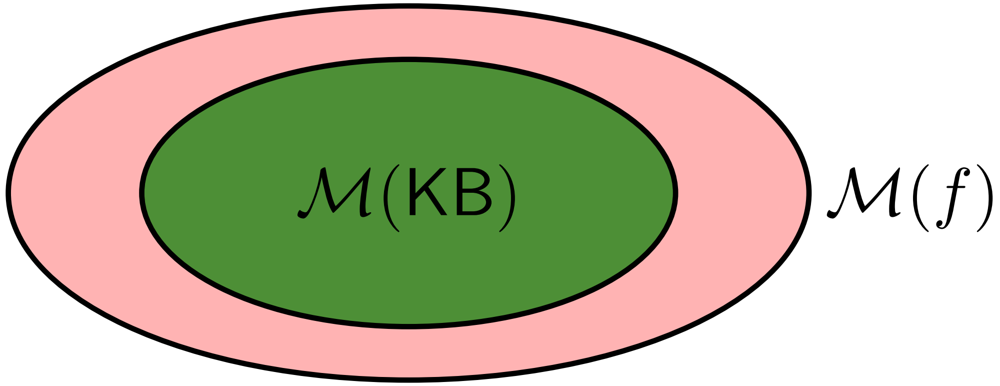
Shrinks the set of models:

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

How much does $\mathcal{M}(\textbf{KB})$ shrink?

[whiteboard]

Entailment



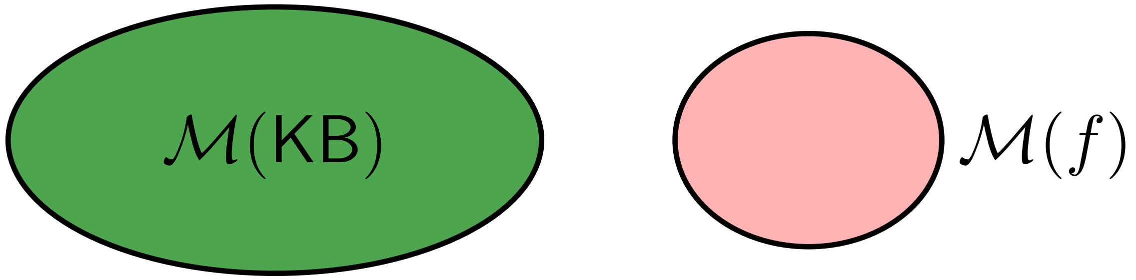
Intuition: f added no information/constraints (it was already known).

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).

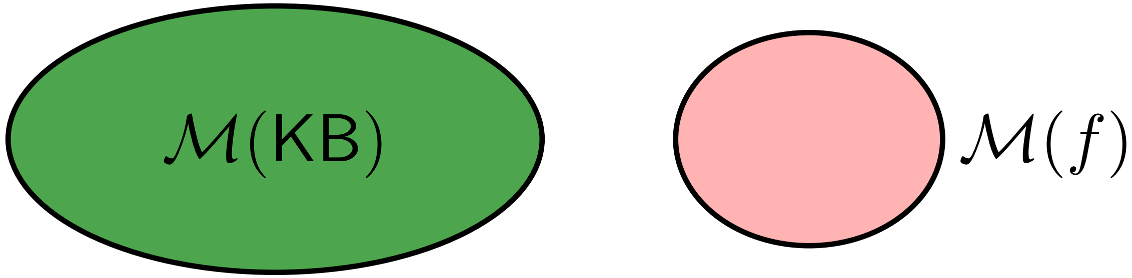
Definition: contradiction

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

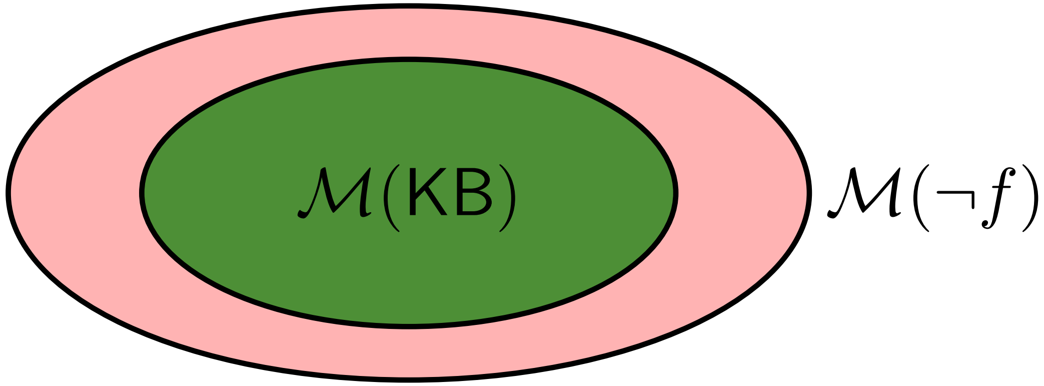
Example: Rain $\wedge$ Snow contradicts $\neg$Snow

Contradiction and entailment

Contradiction:



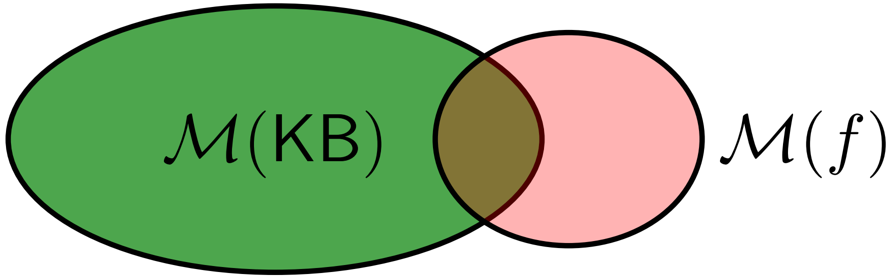
Entailment:



Proposition: contradiction and entailment

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

Contingency



Intuition: f adds non-trivial information to KB

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

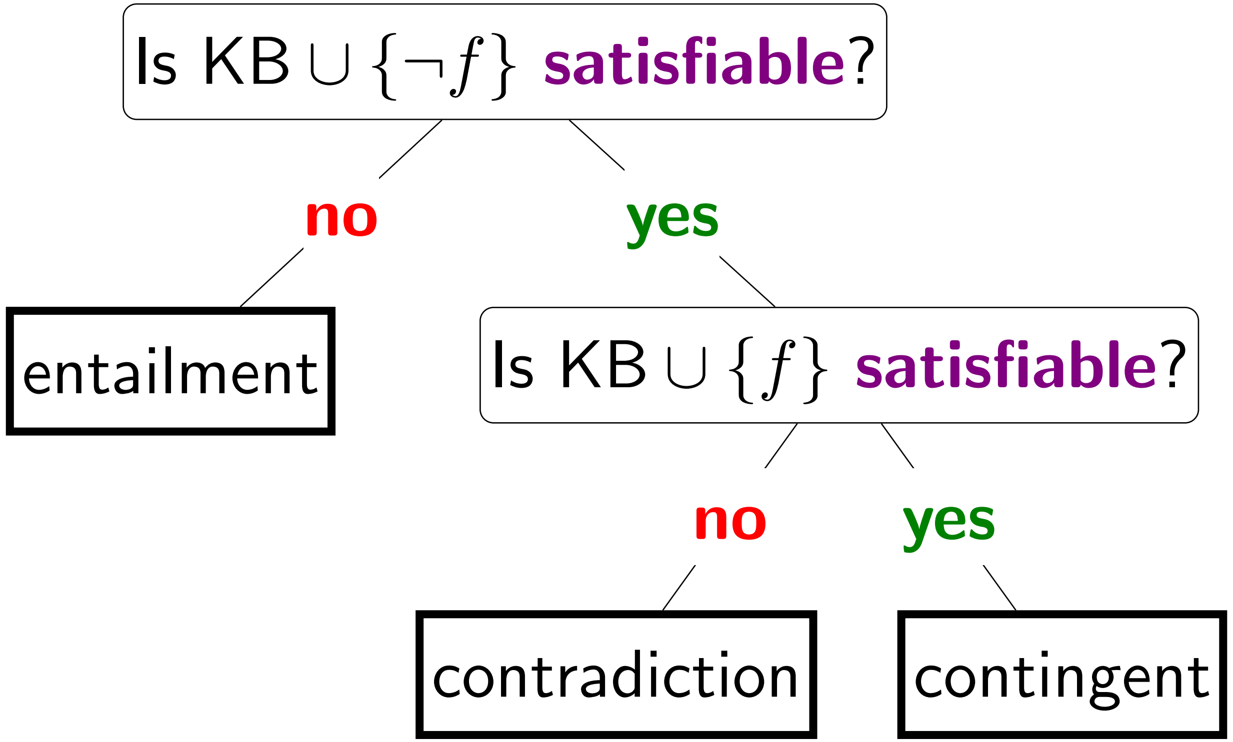
Example: Rain and Snow

Satisfiability

Definition: satisfiability

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

Reduce $\text{Ask}[f]$ and $\text{Tell}[f]$ to satisfiability:



Model checking

Checking satisfiability (SAT) in propositional logic is special case of solving CSPs!

Mapping:

propositional symbol	$\Rightarrow$	variable
formula	$\Rightarrow$	constraint
model	$\Leftarrow$	assignment

Model checking

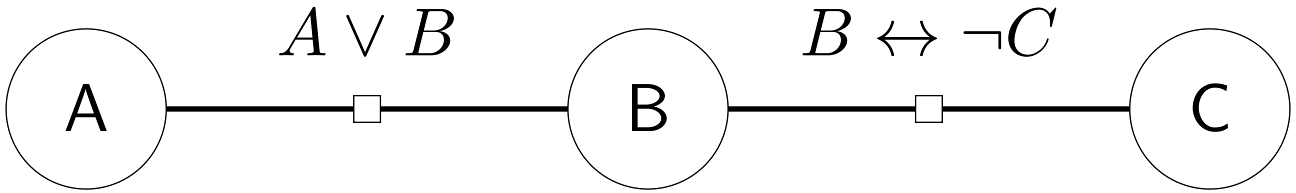
Example: model checking

KB $= \{A \vee B, B \leftrightarrow \neg C\}$

Propositional symbols (CSP variables):

$$\{A, B, C\}$$

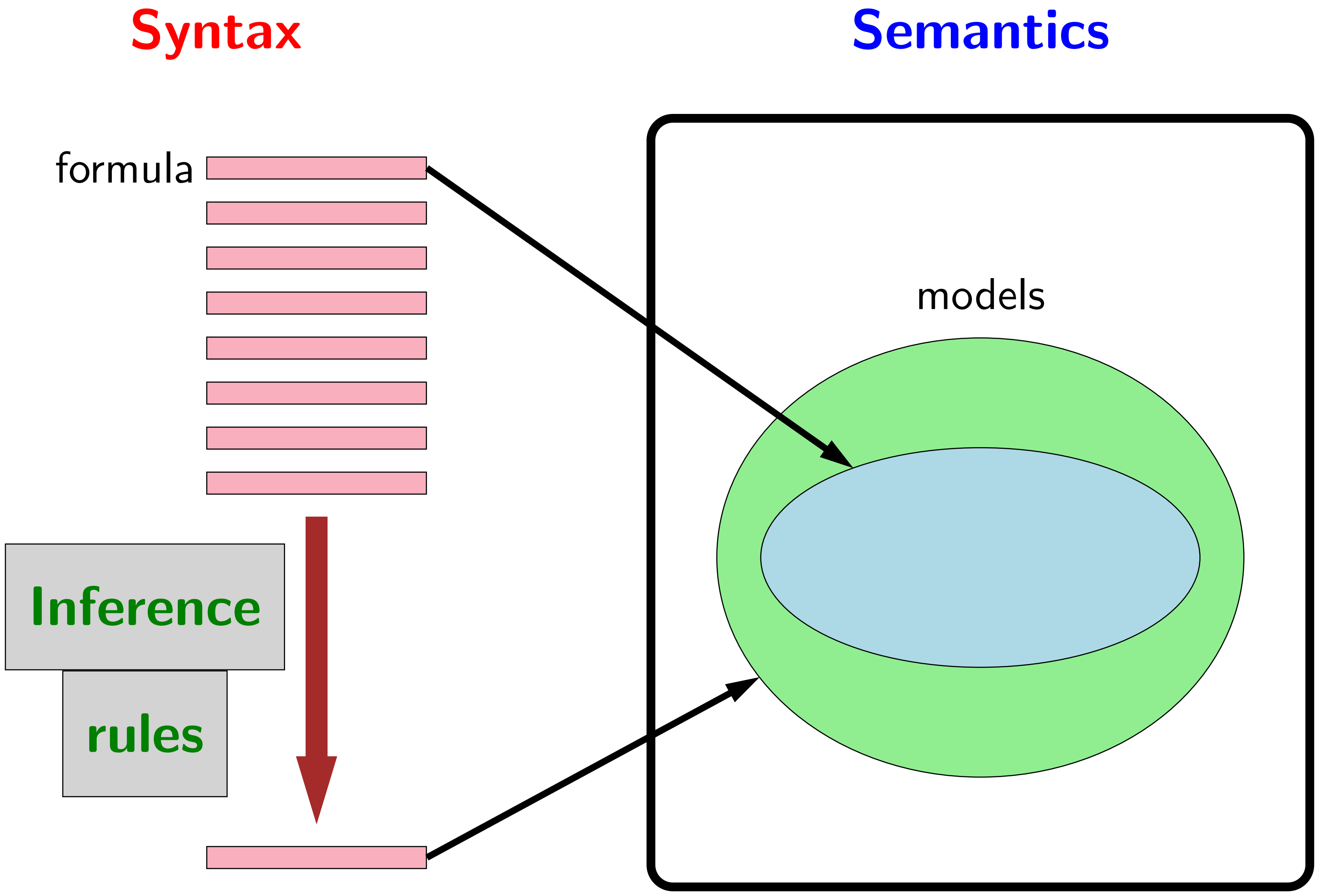
CSP:



Consistent assignment (satisfying model):

$$\{A : 1, B : 0, C : 1\}$$

Propositional logic



Inference rules

Example of making an inference:

It is raining. (Rain)

If it is raining, then it is wet. (Rain $\rightarrow$ Wet)

Therefore, it is wet. (Wet)

$$\frac{\text{Rain}, \quad \text{Rain} \rightarrow \text{Wet}}{\text{Wet}} \qquad \frac{\text{(premises)}}{\text{(conclusion)}}$$

Definition: Modus ponens inference rule

For any propositional symbols p and q:

$$\frac{p, \quad p \rightarrow q}{q}$$

Inference framework

Definition: inference rule

If $f_1, \dots, f_k, g$ are formulas, then the following is an **inference rule**:

$$\frac{f_1, \quad \dots \quad, f_k}{g}$$

Key idea: inference rules

Rules operate directly on **syntax**, not on **semantics**.

Inference algorithm

Algorithm: forward inference

Input: set of inference rules Rules.

Repeat until no changes to KB:

  Choose set of formulas $f_1, \dots, f_k \in$ KB.

  If matching rule $\frac{f_1, \quad \dots \quad, f_k}{g}$ exists:

    Add g to KB.

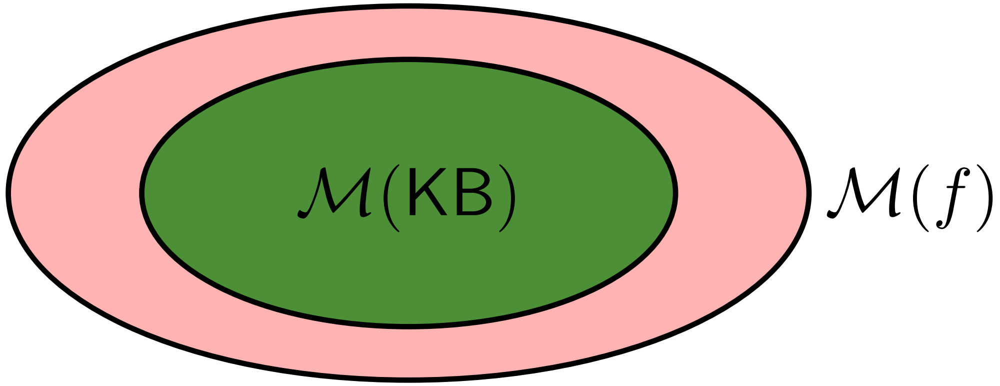
Definition: derivation

KB **derives/proves** f (KB $\vdash f$) iff f eventually gets added to KB.

Desiderata for inference rules

Semantics

Interpretation defines **entailed/true** formulas: KB $\models f$:



Syntax:

Inference rules **derive** formulas: KB $\vdash f$

How does $\{f : \text{KB} \models f\}$ relate to $\{f : \text{KB} \vdash f\}$?

Truth

$$\{f : \mathsf{KB} \models f\}$$

Soundness

Definition: soundness

A set of inference rules Rules is sound if:

$$\{f : \mathsf{KB} \vdash f\} \subseteq \{f : \mathsf{KB} \models f\}$$

Completeness

Definition: completeness

A set of inference rules Rules is complete if:

$$\{f : \mathsf{KB} \vdash f\} \supseteq \{f : \mathsf{KB} \models f\}$$

Soundness and completeness

The truth, the whole truth, and nothing but the truth.

- **Soundness**: nothing but the truth
- **Completeness**: whole truth

Completeness: example

Recall completeness: inference rules derive all entailed formulas (f such that $\text{KB} \models f$)

Example: Modus ponens is incomplete

Setup:

$$\text{KB} = \{\text{Rain}, \text{Rain} \vee \text{Snow} \rightarrow \text{Wet}\}$$

$$f = \text{Wet}$$

$$\text{Rules} = \left\{\frac{f, \quad f \rightarrow g}{g}\right\} \text{ (Modus ponens)}$$

Semantically: $\text{KB} \models f$ (f is entailed).

Syntactically: $\text{KB} \not\vdash f$ (can't derive f).

Incomplete!

Fixing completeness

Option 1: Restrict the allowed set of formulas

propositional logic

↓

propositional logic with only Horn clauses

Option 2: Use more powerful inference rules

Modus ponens

↓

resolution

Definite clauses

Definition: Definite clause

A **definite clause** has the following form:

$$(p_1 \wedge \cdots \wedge p_k) \to q$$

where $p_1, \dots, p_k, q$ are propositional symbols.

Intuition: if $p_1, \dots, p_k$ hold, then q holds.

Example: (Rain $\wedge$ Snow) $\to$ Traffic

Example: Traffic

Non-example: $\neg$Traffic

Non-example: (Rain $\wedge$ Snow) $\to$ (Traffic $\vee$ Peaceful)

Horn clauses

Definition: Horn clause

A **Horn clause** is either:

- a definite clause ($p_1 \wedge \cdots \wedge p_k \to q$)
- a goal clause ($p_1 \wedge \cdots \wedge p_k \to \text{false}$)

Example (definite): $(\text{Rain} \wedge \text{Snow}) \to \text{Traffic}$

Example (goal): $\text{Traffic} \wedge \text{Accident} \to \text{false}$

equivalent: $\neg(\text{Traffic} \wedge \text{Accident})$

Modus ponens

Inference rule:

Definition: Modus ponens

$$\frac{p_1, \quad \cdots \quad , p_k, \quad (p_1 \wedge \cdots \wedge p_k) \to q}{q}$$

Example:

Example: Modus ponens

$$\frac{\text{Wet}, \quad \text{Weekday}, \quad \text{Wet} \wedge \text{Weekday} \to \text{Traffic}}{\text{Traffic}}$$

Completeness of modus ponens

Theorem: Modus ponens on Horn clauses

Modus ponens is **complete** with respect to Horn clauses:

- Suppose KB contains only Horn clauses and p is an entailed propositional symbol.
- Then applying modus ponens will derive p.

Upshot:

$$\text{KB} \models p \text{ (entailment) is the same as KB} \vdash p \text{ (derivation)!}$$

Resolution [Robinson, 1965]

General clauses have any number of literals:

$$\neg A \vee B \vee \neg C \vee D \vee \neg E \vee F$$

Example: resolution inference rule

$$\frac{\text{Rain} \vee \text{Snow}, \quad \neg\text{Snow} \vee \text{Traffic}}{\text{Rain} \vee \text{Traffic}}$$

Definition: resolution inference rule

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \quad \neg p \vee g_1 \vee \cdots \vee g_m}{f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m}$$

Conjunctive normal form

So far: resolution only works on clauses...but that's enough!

Definition: conjunctive normal form (CNF)

A **CNF formula** is a conjunction of clauses.

Example: $(A \vee B \vee \neg C) \wedge (\neg B \vee D)$

Equivalent: knowledge base where each formula is a clause

Proposition: conversion to CNF

Every formula f in propositional logic can be converted into an equivalent CNF formula f':

$$\mathcal{M}(f) = \mathcal{M}(f')$$

Conversion to CNF: general

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg (f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg (f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg \neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

Resolution algorithm

Recall: relationship between entailment and contradiction (basically "proof by contradiction")

$$\text{KB} \models f \quad \Longleftrightarrow \quad \text{KB} \cup \{\neg f\} \text{ is unsatisfiable}$$

Algorithm: resolution-based inference

- Add $\neg f$ into KB.
- Convert all formulas into **CNF**.
- Repeatedly apply **resolution** rule.
- Return entailment iff derive false.

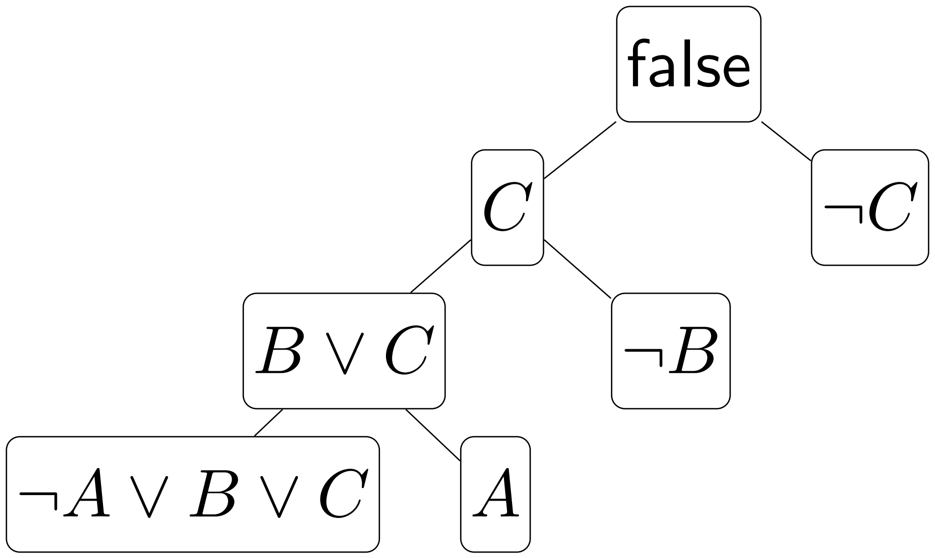
Resolution: example

$$\mathsf{KB}' = \{A \to (B \vee C), A, \neg B, \neg C\}$$

Convert to CNF:

$$\mathsf{KB}' = \{\neg A \vee B \vee C, A, \neg B, \neg C\}$$

Repeatedly apply **resolution** rule:



Conclusion: KB **entails** f

Limitations of propositional logic

All students know arithmetic.

$$\text{AliceIsStudent} \rightarrow \text{AliceKnowsArithmetic}$$

$$\text{BobIsStudent} \rightarrow \text{BobKnowsArithmetic}$$

$$\ldots$$

Propositional logic is very clunky. What's missing?

- Objects and predicates: propositions (e.g., AliceKnowsArithmetic) have more internal structure (alice, Knows, arithmetic)
- Quantifiers and variables: *all* is a quantifier which applies to each person, don't want to enumerate them all...

First-order logic: examples

Alice and Bob both know arithmetic.

$$\text{Knows}(\text{alice}, \text{arithmetic}) \wedge \text{Knows}(\text{bob}, \text{arithmetic})$$

All students know arithmetic.

$$\forall x \, \text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$$

Syntax of 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., $\text{Student}(x) \to \text{Knows}(x, \text{arithmetic})$)
- Quantifiers applied to formulas (e.g., $\forall x\, \text{Student}(x) \to \text{Knows}(x, \text{arithmetic})$)

Quantifiers

Universal quantification ($\forall$):

Think conjunction: $\forall x \, P(x)$ is like $P(A) \wedge P(B) \wedge \cdots$

Existential quantification ($\exists$):

Think disjunction: $\exists x \, P(x)$ is like $P(A) \vee P(B) \vee \cdots$

Some properties:

- $\neg \forall x \, P(x)$ equivalent to $\exists x \, \neg P(x)$
- $\forall x \, \exists y \, \text{Knows}(x, y)$ different from $\exists y \, \forall x \, \text{Knows}(x, y)$

Natural language quantifiers

Universal quantification ($\forall$):

Every student knows arithmetic.

$\forall x \, \text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$

Existential quantification ($\exists$):

Some student knows arithmetic.

$\exists x \, \text{Student}(x) \wedge \text{Knows}(x, \text{arithmetic})$

Note the different connectives!

Models in first-order logic

Definition: model in first-order logic

A model w in first-order logic maps:

- constant symbols to objects

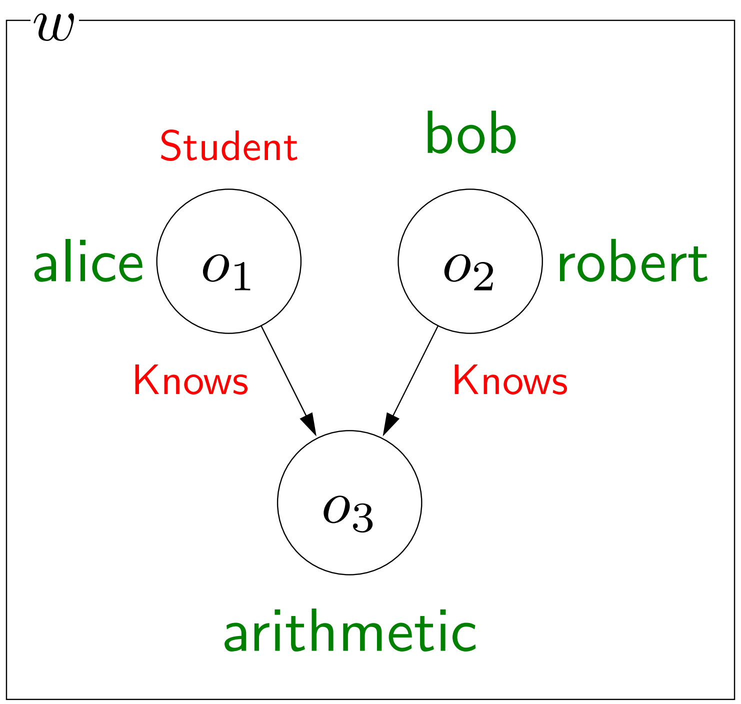
$w(\text{alice}) = o_1, w(\text{bob}) = o_2, w(\text{arithmetic}) = o_3$

- predicate symbols to tuples of objects

$w(\text{Knows}) = \{(o_1, o_3), (o_2, o_3), \dots\}$

Graph representation of a model

If only have unary and binary predicates, a model w can be represented as a directed graph:

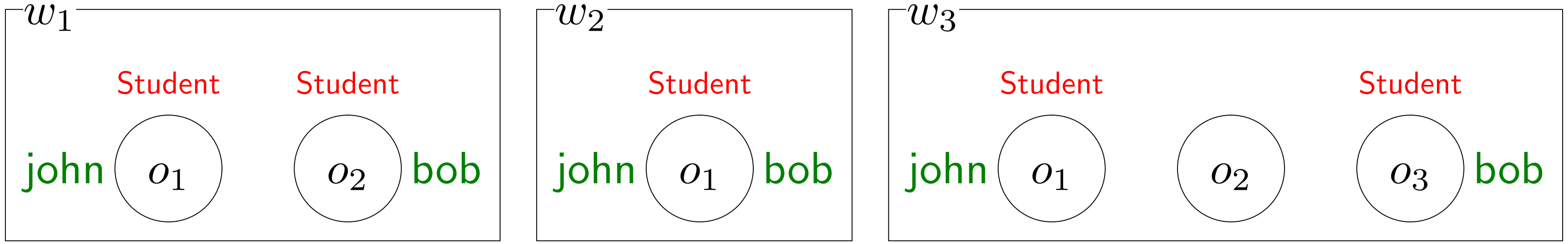


- Nodes are objects, labeled with constant symbols
- Directed edges are binary predicates, labeled with predicate symbols; unary predicates are additional node labels

A restriction on models

John and Bob are students.

$$\text{Student}(\text{john}) \wedge \text{Student}(\text{bob})$$



- Unique names assumption: Each object has **at most one** constant symbol. This rules out w_2.
- Domain closure: Each object has **at least one** constant symbol. This rules out w_3.

Point:

constant symbol ⟷ object

Propositionalization

If one-to-one mapping between constant symbols and objects (**unique names** and **domain closure**),

first-order logic is syntactic sugar for propositional logic:

Knowledge base in first-order logic

$$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$$
$$\forall x\, \text{Student}(x) \rightarrow \text{Person}(x)$$
$$\exists x\, \text{Student}(x) \wedge \text{Creative}(x)$$

Knowledge base in propositional logic

$$\text{Studentalice} \wedge \text{Studentbob}$$
$$(\text{Studentalice} \rightarrow \text{Personalice}) \wedge (\text{Studentbob} \rightarrow \text{Personbob})$$
$$(\text{Studentalice} \wedge \text{Creativealice}) \vee (\text{Studentbob} \wedge \text{Creativebob})$$

Point: use any inference algorithm for propositional logic!

Substitution

$\text{Subst}[\{x/\text{alice}\}, P(x)] = P(\text{alice})$

$\text{Subst}[\{x/\text{alice}, y/z\}, P(x) \wedge K(x, y)] = P(\text{alice}) \wedge K(\text{alice}, z)$

Definition: Substitution

A substitution θ is a mapping from variables to terms.

$\text{Subst}[\theta, f]$ returns the result of performing substitution θ on f.

Unification

$\mathsf{Unify}[\mathsf{Knows}(\mathsf{alice}, \mathsf{arithmetic}), \mathsf{Knows}(x, \mathsf{arithmetic})] = \{x/\mathsf{alice}\}$

$\mathsf{Unify}[\mathsf{Knows}(\mathsf{alice}, y), \mathsf{Knows}(x, z)] = \{x/\mathsf{alice}, y/z\}$

$\mathsf{Unify}[\mathsf{Knows}(\mathsf{alice}, y), \mathsf{Knows}(\mathsf{bob}, z)] = \mathsf{fail}$

$\mathsf{Unify}[\mathsf{Knows}(\mathsf{alice}, y), \mathsf{Knows}(x, F(x))] = \{x/\mathsf{alice}, y/F(\mathsf{alice})\}$

Definition: Unification

Unification takes two formulas f and g and returns a substitution θ which is the most general unifier:

$\mathsf{Unify}[f, g] = \theta$ such that $\mathsf{Subst}[\theta, f] = \mathsf{Subst}[\theta, g]$

or "fail" if no such θ exists.

Modus ponens

Definition: modus ponens (first-order logic)

$$\frac{a'_1, \dots, a'_k \quad \forall x_1 \cdots \forall x_n (a_1 \wedge \cdots \wedge a_k) \rightarrow b}{b'}$$

Get most general unifier θ on premises:

- $\theta = \text{Unify}[a'_1 \wedge \cdots \wedge a'_k, a_1 \wedge \cdots \wedge a_k]$

Apply θ to conclusion:

- $\text{Subst}[\theta, b] = b'$

Complexity

Theorem: completeness

Modus ponens is complete for first-order logic with only Horn clauses.

Theorem: semi-decidability

First-order logic (even restricted to only Horn clauses) is **semi-decidable**.

- If KB $\models f$, forward inference on complete inference rules will prove f in finite time.
- If KB $\not\models f$, no algorithm can show this in finite time.