Search Problem

Property	BFS	UCS	DFS	DLS	GFS	A*
`Completeness`	**YES**, b finite	**YES***	NO	NO	NO	**YES***
`Optimal`	NO	**YES**	NO	NO	NO	**YES***
`Time`	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\mathcal{O}(b^{m+1})$	$\mathcal{O}(b^{\ell+1})$	-	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$
`Space`	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\mathcal{O}(bm)$	$\mathcal{O}(b\ell)$	-	
Frontier	Queue	Priority Queue	Stack	Stack	-	

**Condition*: if b is finite, $cost \geq \epsilon > 0$

Notation: b: max num of successor (branching) of any node (maybe ∞); d: depth of (shallowest) goal node; m: max depth of a node from start node; C^*: optimal cost, $cost \geq \epsilon$

Definition: `Complete`: always find a solution; `Optimal`: find a least-cost solution; `Time C.`: number of nodes generated; `Space C.`: max number of nodes in memory

Uninformed Search

Path Checking: In every path $\langle p_k, c \rangle$, ensure that the final state c is not equal to any ancestors of c along this path: $c \notin \{s_0, s_1, ..., s_k\}$ (make sure does not go back). No increase time and space C.

Cycle Checking: Keep track of all nodes previously expanded during the search using a list (close list). When expand n_k to obtain successor c: (1) Ensure c is not equal to any previously expanded node (2) If it is, do not add c to Frontier; Expansive S.C.: $\mathcal{O}(b^{d+1})$

Bread-first Search: children at end of Frontier (queue: last in last out), extract first of the F

Depth-first Search: children at front of Frontier (stack: last in first out), extract first of the F

Depth-limited Search: DFS but only to a pre-specified depth limit D

Iterative Deepening Search: Starting at $d = 0$, loop DLS til solution or fail without cutting off

Uniform Cost Search: expand least cost node on F. (Priority Queue), same as BFS if all same cost

Informed Search

Greedy Bread-first Search: $f(n) = h(n)$; ignore cost of n; not complement or optimal

A* Search: $f(n) = g(n) + h(n)$; g: cost path; h: heuristic estimate of cost: run out of time&memory

Poof C. Implies Admissible: $(\forall n_1, n_2, a) \quad h(n_1) \leq C(n_1, a, n_2) + h(n_2) \implies (\forall n) \quad h(n) \leq h^*(n)$

(1)Base case: $k = 1$, one step away from s_g, since consistent: $h(s_i) \geq C(s_i, s_g) + h(s_g)$, since $h(s_g) = 0$, $h(s_i) \geq C(s_i, s_g) = h^*(s_i)$, therefore admissible

(2) Induction step: Suppose assumption holds for every node that is $k-1$ action away from s_g, given a node s_i, it is k action away from s_g, thus optimal path has $k > 1$ steps

(3) Since h is consistent, have: $h(s_i) \leq C(s_i, s_{i+1}) + h(s_{i+1})$

(4) Note that s_{k+1} is on a least-cost path from s_i, must have the path s_{i+1} to s_g as well, by induction hypothesis have: $h(s_{i+1}) \leq h^*(s_{i+1})$

(5) Combine inequality: $h(s_i) \leq C(s_i, s_{i+1}) + h^*(s_{i+1})$

Proof of Optimal with Consistency: $\forall n_1, n_2, \quad h(n_1) \leq h(n_2) + C(n_1, a, n_2)$

(1) WTS: $\hat{f}_{pop}(s_g) = f(s_i)$, when goal node is pooped, have found optimal

(2) Base case: $\hat{f}_{pop}(s_0) = f(s_0) - h(s_0)$

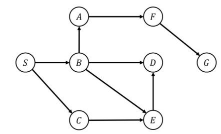
(3) Induction step: Assume $\forall s_0, s_1, ..., s_k, \hat{f}_{pop}(s_i) = f(s_i)$, given:

$$\hat{f}_{pop}(s_{k+1}) = \hat{g}_{pop}(s_{k+1}) + h(s_{k+1}) \geq g(s_{k+1}) + h(s_{k+1}) = f(s_{k+1})$$

For s_{k+1} is only explored after s_k, require $f(s_i) \leq f(s_{k+1})$, need of consistency of h, pooping s_k

$$\begin{aligned}
\hat{f}_{pop}(s_{k+1}) &= \min\{\hat{f}(s_{k+1}), \hat{g}_{pop}(s_k) + c(s_k, s_{k+1}) + h(s_{k+1})\} \\
&\leq \hat{g}_{pop}(s_k) + c(s_k, s_{k+1}) + h(s_{k+1}) \\
&= g(s_k) + c(s_k, s_{k+1}) + h(s_{k+1}) && \text{from IH} \\
&= g(s_{k+1}) + h(s_{k+1}) \\
&= f(s_{k+1})
\end{aligned}$$

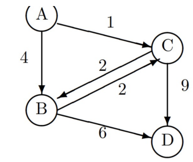
IDA* Search: reduce memory requirements of A*; cutoff is the f-value rather than the depth; at each iteration, the cutoff is the smallest f-value of any node that exceeded the cutoff on the previous iteration; avoids overhead with keeping a sorted queue of nodes, the Frontier occupies linear space.



DFS with no C.C: S–C–E–D–B–E–D–D–A–F–G

BFS with no C.C: S–B–C–A–D–E–E–F–D–D–G

BFS with C.C: S–B–C–A–D–E–F–G



A* with CC

h(A) = 8
h(B) = 3
h(C) = 7
h(D) = 0

START = A
GOAL = D

(Node, Path)	Frontier
	{(A: 0+8=8)}
(A, A)	{[illegible]=8), (AB: 4+3=7)}
(B, AB)	{(AC: 1+7=8), (ABC: 6+7=13), (ABD: 10+0=10)}
(C, AC)	{(ACB: 3+3=6), (ACD: 10+0=10), (ABC: 6+7=13), (ABD: 10+0=10)}
(B, ACB)	{(ACBC: 5+7=12), (ACBD: 9+0=9), (ACD: 10+0=10), (ABC: 6+7=13), (ABD: 10+0=10)}

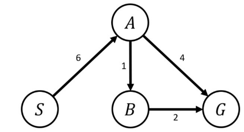


Figure 3: Graph for question 3.

	S	A	B	G	Admissible	Consistent
h_1	0	0	0	0	True	True
h_2	8	1	1	0	True	False
h_3	9	3	2	0	True	True
h_4	6	3	1	0	True	False
h_5	8	4	2	0	False	False

CSPs

(1) A set of variables $V_1, ..., V_n$;

(2) A (finite) domain of possible values $Dom[V_i]$ for each variable V_i

(3) A set of constraints $C_1, ..., C_m$,

Unary: over one variable: $C(X) : X = 2$

Binary: over two variable: $C(X, Y) : X + Y \geq 2$

Higher-order: over ¿3 variable: $All-Diff(V_1, ..., V_n) : V_1 \neq V_2, ..., V_2 \neq V_1, ...V_n \neq V_{n-1}$

(4) Each variable V_i can be assigned any value from its domain: $Vi = d$ where $d \in Dom[Vi]$

(5) Each constraint C Has a set of variables it operates over, called its scope.

(6) Solution to a CSP: An assignment of a value to all of the variables such that every constraint is satisfied; unsatisfiable if no solution exists.

Back Tracking Search: searching through the space of partial assignments, rather than paths. Decide on a suitable value for one variable at a time. Order in which we assign the variables does not matter. If a constraint is falsified during the process of partial assignment, immediately reject the current partial assignment.

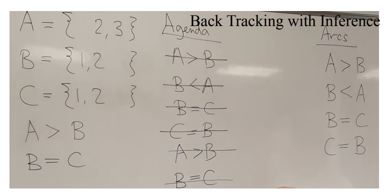
Back Tracking Search with Inference: every time assign a value to variable V, check all constrains over V and prune values from the current domain of the unassigned variables of the constrains

(1) *Value Assignment*: define current domain (`CurDom`) of a value; first step to infer other values

(2) **Degree Heuristic**: select the variable that is involved in the largest number of constrains on other unassigned variables

(3) **Minimum Remaining Values Heuristics**: always branch on a variable with the smallest remaining values (smallest `CurDom`)

(4) **Least Construing Value Heuristic**: always pick a value in `CurDom` that rules out the least domain values of other neighboring values in the constraint



Consider the following constraint satisfaction problem: there are four variables x_1, x_2, x_3, x_4 whose domain is $D = \{1, 2, 3\}$. They have the following constraints:

$$x_1 + x_2 \leq 3 \qquad x_1 + x_3 \geq 4$$
$$x_2 + x_3 \leq 3 \qquad x_3 + x_4 \leq 2$$

Variable popped/inferred	Domain update	Queue
x_2	$x_1 = \{1,2\}$ $x_2 = \{1,2,3\}$ $x_3 = \{1,2\}$ $x_4 = \{1,2,3\}$	$\{x_1, x_3\}$
x_1	$x_1 = \{1,2\}$ $x_2 = \{1,2\}$ $x_3 = \{2\}$ $x_4 = \{1,2,3\}$	$\{x_3, x_2\}$
x_3	$x_1 = \{2\}$ $x_2 = \{1\}$ $x_3 = \{2\}$ $x_4 = \{\varnothing\}$	$\{x_2, x_1\}$

Variables: X, Y, Z Domains: DX = DY = DZ = {0, 1, 2, 3, 4}
Constraints: C1 = (X = Y + 1); C2 = (Y = 2Z)

Assuming that initially no variable has been assigned and var = Y ,
(a) Pop Y : • From C1, X/ = 0 • From C2, Z/ = {3, 4} Since both X's and Z's domains are updated, we add them to the queue (i.e. queue now contains {X, Z}).

(b) Pop X: • From C1, Y/ = 4 Since Y 's domain is updated, we add Y to the queue (i.e. queue now contains {Z, Y }).

(c) Pop Z: • From C2, Y/ = {1, 3} Since the queue contains Y , we do not add it back (i.e. queue now contains {Y }).

Pop Y : • From C1, X/ = {2, 4} • From C2, Z/ = 2 Since both X's and Z's domains are lated. we add them to the queue (i.e. queue now contains {X, Z}).

(e) Pop X • No inference (i.e. queue now contains {Z}).

(f) Pop Z: • No inference (i.e. queue is now empty).

From the inference above, we derive the following:
• X/ = {0, 2, 4}, DX = {1, 3}
• Y/ = {1, 3, 4}, DY = {0, 2}
• Z/ = {2, 3, 4}, DZ = {0, 1}

Games

Properties: two player; finite number of states and moves (large- heuristic cutoffs); deterministic (perfect info/observable); **zero-sum**: fully competitive, total payoff to all players is constant

- 2 players Max and Min
- A set of positions P (states of the game)
- A starting positions $P \in P$ (game begins)
- A set of Terminal positions $T \subseteq P$ (game end)
- A set of directed edges E_{Max} between some positions, representing Max's move
- A set of directed edges E_{Min} between some positions, representing Min's move
- A utility/payoff function $U : T \to \mathbb{R}$, representing quality each terminal state is for player **Max**

Minmax Search

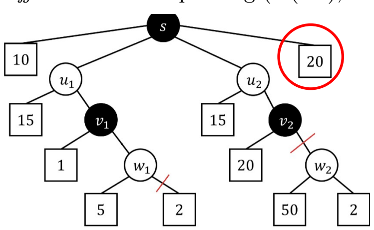
Max plays a move to change the state to the highest valued child $U(S_0) = max\{U(S_i), ..., U(S_n)\}$
Min plays a move to change the state to the lowest valued child $U(S_0) = min\{U(S_i), ..., U(S_n)\}$
Use **DFS** to save space (finite depth); T.C.: $\mathcal{O}(b^d)$; S.C: $\mathcal{O}(bd)$

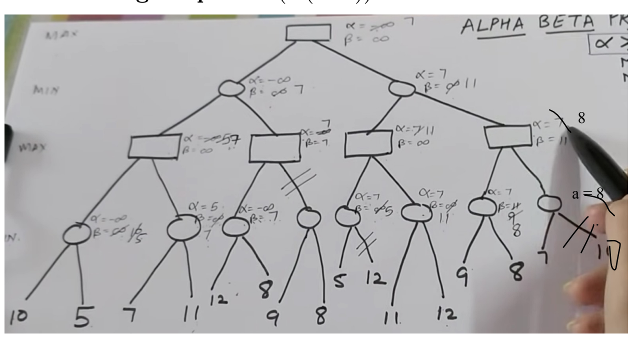
Alpha-Beta Pruning

At a Max node s
(1.1) α_s: the highest value of s **children** examine so far (changes as examine more children)
(1.2) β: the lowest value of s **parent** examine so far (fixed)
(2) If α_s becomes $\geq \beta$, stop expanding children of s; Min never choose to move from s parent, would choose one of s lower valued siblings
At a Min node s
(1.1) α: the highest value found so far by s **parent** by previous explored siblings (fixed)
(1.2) β_s: the lowest value of value of s **children** examine so far (changes as explore more children)
(2) If α_s becomes $\geq \beta_s$, stop expanding children of s; Max never choose to move from s parent, would choose one of s higher valued siblings

- Set initial values: $\alpha = -\infty$ and $\beta = \infty$
- While backing the utility values up the tree, identity α, β for each node (α/β: best already explored along the path to the root of MAX/MIN)
- At every node s, if $\alpha \geq \beta$ **prune** (remaining) children of s (α/β-cuts: pruning of MAX/MIN nodes)

Ordering Moves: Max prune best if best move for Max explored first; Min prune best if best move for Min explored first; can use heuristics to estimate and choose
Effectiveness: no pruning ($\mathcal{O}(b^d)$); if move **ordering is optimal** ($\mathcal{O}(b^{d/2})$)





Bayesian Networks

Probability

∩: OR; ∪: AND
Basic Rules: $P(\mathcal{U}) = 1$, $P(A) \in [0, 1]$, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
Summing out Rule: $P(A) = \sum_{C_i} P(A \cap C_i)$, $P(A \mid B) = \sum_{C_i} P(A \mid B \cap C_i)P(C_i \mid B)$
Normalizing: dividing each number by the sum of the numbers:
(1) normalize $[x_1, x_2, ...x_k] = [\frac{x_1}{\alpha}, \frac{x_2}{\alpha}, ..., x_k/\alpha]$, where α is the sum of all x_k
(2) normalize $[x_1, x_2, ...x_k] = [x_1 \cdot \beta, x_2 \cdot \beta, ..., x_k \cdot \beta]$, where β is any constant
Conditional Probability:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Bayes Rule:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Chain Rule:

$$P(A_1 \cap A_2 ... \cap A_n) = P(A_n \mid A_1 ... \cap A_{n-1}) \cdot P(A_{n-1} \mid A_1 ... \cap A_{n-1}) \cdot ... \cdot P(A_2 \mid A_1) \cdot P(A_1)$$

Independence: $P(V_1 \mid V_2) = P(V_1)$ (V_1, V_2 are independent)

$$P(A \mid B) = P(A) \quad P(A \cap B) = P(A) \cdot P(B)$$

Conditional Independence: A is conditionally independent of B given C

$$P(A \mid B \cap C) = P(A \mid C) \quad P(A \cap B \mid C) = P(A \mid C) \cdot P(B \mid C)$$

Joint Distribution: $Dom[V_1] = Dom[V_2] = \{1, 2, 3\}$, $P(V_1, V_2)$ are vector of 9

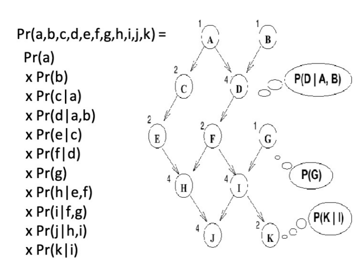
$$P(v_1 = 1, V_2 = 1), P(V_1 = 1, V_2 = 2), ..., P(V_1 = 2, V_2 = 1), ..., P(V_1 = 3, V_2 = 3)$$

Conditional Probabilty Table (CPT): $Dom[V_1] = Dom[V_2] = Dom[V_3]\,\{1, 2, 3\}$, $P(V_1 \mid V_2, V_3)$ are 27 values; $P(V_1 = 1 \mid V_2 = 1, V_3 = 1), ..., P(V_1 = 3 \mid V_2 = 3, V_3 = 3$

Bayesian Network

Conditional Independence:
`E → C → A → B → H:`
(1) B is independent of E, and C, given A, A is independent of E given C
(2) Computation: $P(H \mid B, \{A, C, E\}) = P(H \mid B)$
(3) Chain rule: $P(H, B, A, C, E) = P(H \mid B, A, C, E)P(B \mid A, C, E)P(A \mid C, E)P(C \mid E)P(E)$
(4) Independenece assumption: $P(H, B, A, C, E) = P(H \mid B)P(B \mid A)P(A \mid C)P(C \mid E)P(E)$
(5) Joint distributions:
$P(\mathcal{A} = \sum_{x_i \in Dom(X)} P(\mathcal{A} \mid x_i)P(x_i) = \sum_{x_i \in Dom(X)} P(\mathcal{A} \mid x_1) \sum_{y_i \in Dom(Y)} P(x_i \mid y_i)P(y_i)$
Ex. $P(c) = P(c \mid e)P(e) + P(c \mid \neg e)P(\neg e)$

Network and Chain Rule



$$P(R) = P(L_1)P(R|L_1) + P(L_2)P(R|L_2) + P(L_3)P(R|L_3)$$
$$= (0.5)(0.05) + (0.3)(0.08) + (0.2)(0.1)$$
$$= 0.069$$

Therefore,

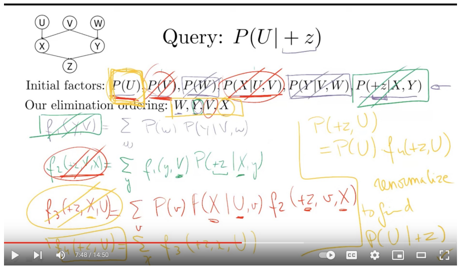
(a) $P(L_1|R) = (0.5)(0.05)/0.069 = 0.3623$

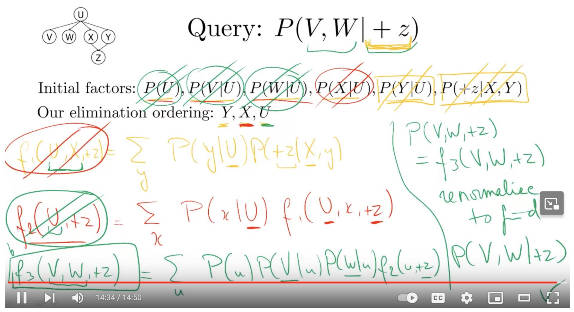
(b) $P(L_2|R) = (0.3)(0.08)/0.069 = 0.3478$

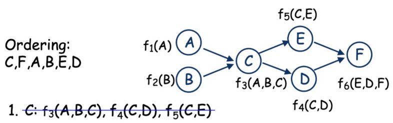
(c) $P(L_3|R) = (0.2)(0.1)/0.069 = 0.2899$

Variable Elimination & Factoring

VE sum out the innermost variable, computing a new function over variables in that sum.







1. ~~C: $f_3(A,B,C)$, $f_4(C,D)$, $f_5(C,E)$~~
2. ~~F: $f_6(E,D,F)$~~
3. ~~A: $f_1(A)$, $f_7(A,B,D,E)$~~
4. ~~B: $f_2(B)$, $f_9(B,D,E)$~~
5. ~~E: $f_8(E,D)$, $f_{10}(D,E)$~~
6. D: $f_{11}(D)$

5. $\Sigma_E f_8(E,D) f_{10}(D,E) = f_{11}(D)$

f_{11} is he final answer, once we normalize it.

D-Separation (Independence)

Independence: every X_i is conditionally independent of all of its nondescendants given it parents

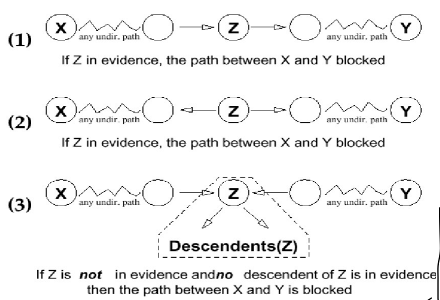
(1) A set of variables E d-separates X, Y if it blocks every undirected path in the BN between X, Y

Let P be an undirected path from X, Y in a BN; let E (evidence) be a set of variables

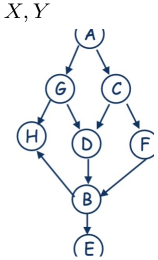
E blocks path P iff there is some node Z on path P such that:

- $Z \in E$ and one arc on P enters (goes into) Z and one leaves (goes out of) Z
- $Z \in E$ and both arcs on P leave Z
- Both arcs on P enter Z and neither Z, nor any of its descendants are in E

(2) X, Y are conditionally independent given evidence E if E d-separates X, Y



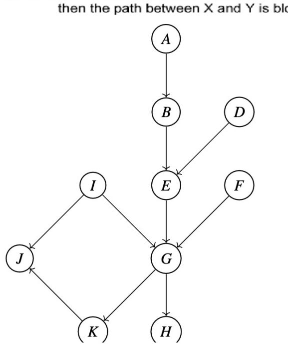
1. A and E given no evidence? No
2. A and E given {C}? No
3. A and E given {G,C}? Yes
4. A and E given {G,C,H}? Yes
5. A and E given {G,F}? No
6. A and E given {F,D}? Yes
7. A and E given {F,D,H}? No
8. A and E given {B}? Yes
9. A and E given {H,B}? Yes
10. A and E given {G,C,D,H,D,F,B}? Yes



(a) Which variables are independent of I? A, B, D, E, F

(b) Which variables are conditionally independent of A given K? *None*

(c) Which variables are conditionally independent of B given E? F, G, H, I, J, K



$$P(F=\text{yes}|N,M) = \frac{P(F=\text{yes},N,M)}{P(N,M)} = \frac{P(F=\text{yes},N,M)}{\sum_{\forall F} P(F,N,M)}.$$

Using the Bayesian network, we can factorize the above as follows:

$$P(F.N.M) = \sum_{\forall T,S} P(F,N,M,T,S)$$

$$= \sum_{\forall T,S} P(N)P(M)P(T|M,N)P(S|N)P(F|T,S)$$

$$= P(N)P(M)\underbrace{\sum_{\forall T} P(T|M,N) \underbrace{\sum_{\forall S} P(S|N)P(F|T,S)}_{f_1(F,T,N)}}_{f_2(F,M,N)}$$

Using the conditional probability tables, we have

F	T	N	$f_1(F,T,N)$
yes	early	0	$0.9 \times 0.95 + 0.1 \times 0.70 = 0.925$

P(E)	e	-e
	1/10	9/10

P(B)	b	-b
	1/10	9/10

P(S|E,B)	s	-s
e ∧ b	9/10	1/10
e ∧ -b	2/10	8/10
-e ∧ b	8/10	2/10
-e ∧ -b	0	1

P(W|S)	w	-w
s	8/10	2/10
-s	2/10	8/10

P(G|S)	g	-g
s	1/2	1/2
-s	0	1

What are the eight probability values of $P(G|S \wedge W)$?

Solution:
$P(g|s,\neg w) = P(g|s,w) = P(g|s) = \frac{1}{2}$
$P(\neg g|s,\neg w) = P(\neg g|s,w) = P(\neg g|s) = \frac{1}{2}$
$P(g|\neg s,\neg w) = P(g|\neg s,w) = P(g|\neg s) = 0$
$P(\neg g|\neg s,\neg w) = P(\neg g|\neg s,w) = P(\neg g|\neg s) = 1$

What are the eight probability values of $P(G|S \wedge W)$?

Solution:
$P(g|s,\neg w) = P(g|s,w) = P(g|s) = \frac{1}{2}$
$P(\neg g|s,\neg w) = P(\neg g|s,w) = P(\neg g|s) = \frac{1}{2}$
$P(g|\neg s,\neg w) = P(g|\neg s,w) = P(g|\neg s) = 0$
$P(\neg g|\neg s,\neg w) = P(\neg g|\neg s,w) = P(\neg g|\neg s) = 1$

Knowledge Representation

Representation: Symbolic encoding of propositional believed

Reasoning: Manipulation of symbolic encoding of propositions to produce propositions that believed by the agent but are not explicitly stated

First-order Logic

Syntax: A grammar specifying what are legal syntactic constructs of the representation.

- Propositional variable: True or False variables
- $A \wedge B$ (conjection); $A \vee B$ (disjection); $A \implies B$ (implication); $A \iff B$ (bi-implication)
- A st V of variables; A set of F of function symbols; A set of P of predicate/relation symbols

Let $\mathcal{L}$ be a vocabulary, the set of first-order $\mathcal{L}$-formulas as defined:

- Atomic formula: $P(t_1, t_2, ..., t_n)$ where P is an n-ary predicate symbol in $\mathcal{L}$, and t_n are $\mathcal{L}$ terms
- Negation: $\neg f$, where f is a $\mathcal{L}$-formula
- Conjection: $f_1 \wedge f_2 \wedge ... \wedge f_n$, where $f_1, ..., f_n$ are $\mathcal{L}$-formula
- Disjunction $f_1 \vee f_2 \vee ... \vee f_n$, where $f_1, ..., f_n$ are $\mathcal{L}$-formula
- Implication: $f_1 \implies f_2$, where f_1, f_2 are $\mathcal{L}$-formula
- Existential: $\exists x f$, where x is a variable and f is a $\mathcal{L}$-formula
- Universal $\forall x f$, where x is a variable and f is a $\mathcal{L}$-formula

Ex. $AC(x)$: x belongs to Alpine Club; $L(x,y)$: x likes y

Semantics: A formal mapping from syntactic constructs to set theoretic assertions

Truth Assignment: a function τ from the propositional variables into the set of $\{T, F\}$

Let τ be a t.a. extension $\bar{\tau}$ of τ assigns either T, F to every formula and is defined as:

- If $A = x$, where x is a variable, then $\bar{\tau} = \tau(x)$
- $\bar{\tau}(\neg A) = T$, IFF $\bar{\tau}(A) = F$
- $\bar{\tau}(A \wedge B) = T$ IFF $\bar{\tau}(A) = T$ AND $\bar{\tau}(B) = T$
- $\bar{\tau}(A \vee B)$ IFF $\bar{\tau}(A) = T$ OR $\bar{\tau}(B) = T$
- $\bar{\tau}(A \implies B) = F$ IFF $\bar{\tau}(A) = T$ AND $\bar{\tau}(B) = F$

τ satisfies a set Φ of formulas IFF τ satisfies all formula in Φ

a formula A is a logical consequence of $\Phi \vDash A$ IFF for every t.a. τ satisfies Φ, then τ satisfies A

Structure: let $\mathcal{L}$ be a first order vocabulary, an $\mathcal{L}$-structure $\mathcal{M}$ consists:

- Nonempty set M called the universe (domain) of discourse
- For each n-ary function symbol $f \in \mathcal{L}$, and associated function $f^{\mathcal{M}} : M^n \to M$ (if $n = 0$, then f is a constant symbol and $f^{\mathcal{M}}$ is simply an element of M. $f^{\mathcal{M}}$ is called the extension of the function symbol f in $\mathcal{M}$)
- For each n-ary predicate symbol $P \in \mathcal{M}$, an assorted relation $P^{\mathcal{M}} \subseteq M^n$. $P^{\mathcal{M}}$ is called the extension of the predicate symbol P in $\mathcal{M}$

What are the four probability values of $P(G|W)$?

Solution: We perform variable elimination. Query variable is **G**. Ordering: E, B, S, G.

First run of VE. Evidence is $W = w$.

i. E: $P(E), P(S|E,B)$
ii. B: $P(B)$
iii. S: $P(w|S), P(S|G)$
iv. G:

$F_1(S,B) = \sum_E P(E) \cdot P(S|E,B) = P(e) \cdot P(S|e,B) + P(\neg e) \cdot P(S|\neg e,B)$

$F_1(\neg s,\neg b) = P(e) \cdot P(\neg s,e,\neg b) + P(\neg e) \cdot P(\neg s,\neg e,\neg b) = 0.1 \cdot 0.8 + 0.9 \cdot 1 = 0.98$

$F_1(\neg s,b) = P(e) \cdot P(\neg s,e,b) + P(\neg e) \cdot P(\neg s,\neg e,b) = 0.1 \cdot 0.1 + 0.9 \cdot 0.2 = 0.19$

$F_1(s,\neg b) = P(e) \cdot P(s,e,\neg b) + P(\neg e) \cdot P(s,\neg e,\neg b) = 0.1 \cdot 0.2 + 0.9 \cdot 0 = 0.02$

$F_1(s,b) = P(e) \cdot P(s,e,b) + P(\neg e) \cdot P(s,\neg e,b) = 0.1 \cdot 0.9 + 0.9 \cdot 0.8 = 0.81$

Variable Assignments: let $\mathcal{M}$ be a structure and X be a set of variables. An object assignment σ for $\mathcal{M}$ is **mapping** from variables in X to the universe of M
Recursive definition: let $\mathcal{L}$ be a set of function and predicate symbols
(1) Every variable x is a term;
(2) if f is an n-ary function symbol in $\mathcal{L}$ and $t_1, t_2, ..., t_n$ are $\mathcal{L}$-terms, then $f(t_1, t_2, ..., t_n)$ is a $\mathcal{L}$-term
Let $\mathcal{L}$ be a vocabulary and $\mathcal{M}$ be an $\mathcal{L}$-structure, the extension $\bar{\sigma}$ of σ is defined recursively:
(1) For every variable $x, \bar{\sigma}(x) = \sigma(x)$;
(2) For every function symbol $f \in \mathcal{L}, \bar{\sigma}(f(t_1, ...t_n)) = f^{\mathcal{M}}(\bar{\sigma}(t_1), ..., \bar{\sigma}(t_n))$

Model Interpretation
For an $\mathcal{L}$-formula C, $\mathcal{M} \vDash C\,[\sigma]$ ($\mathcal{M}$ satisfies C under σ, or $\mathcal{M}$ is a model of C under σ) is defined recursively on the structure of C as:

- $\mathcal{M} \vDash P(t_1, ..., t_n)\,[\sigma] \iff \langle \bar{\sigma}(t_1), ..., \bar{\sigma}(t_n) \rangle \in P^{\mathcal{M}}$
- $\mathcal{M} \vDash (s = t)\,[\sigma] \iff \bar{\sigma}(s) = \bar{\sigma}(t)$
- $\mathcal{M} \vDash \neg A\,[\sigma] \iff \mathcal{M} \nvDash A\,[\sigma]$
- $\mathcal{M} \vDash (A \vee B)\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma] \vee \mathcal{M} \vDash B\,[\sigma]$
- $\mathcal{M} \vDash (A \wedge B)\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma] \wedge \mathcal{M} \vDash B\,[\sigma]$
- $\mathcal{M} \vDash (\forall x A)\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma(m/x)]$ for all $m \in M$
- $\mathcal{M} \vDash (\exists x A)\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma(m/x)]$ for some $m \in M$

$\sigma(m/x)$ is an o.a. function exactly like σ, but maps the variable x to the individual $m \in M$:
(1) For $y \neq x : \sigma(m/x)(y) = \sigma(y)$ (2) For $x : \sigma(m/x)(x) = m$

Bounded: an occurrence of $x \in A$ is bounded iff it is an sub-formula of A of the form $\forall x B$ or $\exists x B$; otherwise the occurrence is free
In a structure $\mathcal{M}$, formula with free variables might be true for some object assignments to the free variable and false to others
Sentence: a formula A is closed if it contains no free occurrence of a variable
If σ and σ' agree on the free variables of A, then $\mathcal{M} \vDash A\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma']$
Corollary: if A is a sentence, then for any object assignments σ, and σ':

$$\mathcal{M} \vDash A\,[\sigma] \iff \mathcal{M} \vDash A\,[\sigma']$$

so if A is a sentence (no free variables), σ is irrelevant and omit mention of σ, $\mathcal{M} \vDash A$
Satisfiability: let Φ be a set of sentences

- $\mathcal{M}$ satisfies Φ ($\mathcal{M} \vDash \Phi$), if for every sentence $A \in \Phi, \mathcal{M} \vDash A$
- If $\mathcal{M} \vDash \Phi$, say $\mathcal{M}$ is a model of Φ
- Say that Φ is satisfiable if there is a structure $\mathcal{M}$ such that $\mathcal{M} \vDash \Phi$

Unsatisfiable: if A is a **logical consequence** of Φ, then there is no $\mathcal{M}$ such that $\mathcal{M} \vDash \Phi \cup \{\neg A\}$

Resolution by Refutation

Knowledge Base: a collection of sentences that represent what the agent believes about the world
Sentences in KB are explicit knowledge; logical consequences of the KB are implicit
Resolution works with formulas expressed in clausal form
Literal: an atomic formula or the negation of an a.f. (ex. $dog(Fido), \neg cat(fido)$)
Clause: disjunction of literals (ex. $P(x) \vee \neg Q(x, y) \quad \neg O(fido) \vee \neg Dog(fido)$)
Clausal Theory: a set of clauses, can be considered as conjection of clauses
$(P(x) \vee \neg Q(x, y), \neg O(fido))$
Resolution Proof using inference rule

$$\frac{a_1 \vee a_2 \vee ... \vee a_n \vee c \quad b_1 \vee b_2 \vee ... \vee b_m \vee \neg c}{a_1 \vee a_2 \vee ... \vee a_n \vee b_1 \vee b_2 \vee ... \vee b_m}$$

Resolution by Refutation to Show: KMA

- Assume $\neg A$ is true to generate a contradiction (**refutation**)
- Convert $\neg A$ and all sentences in KM to a clausal theory C
- Resolve the clauses in C until an empty clause is obtained

Eliminate Implications

Implication Rule: $A \rightarrow B \iff \neg A \vee B$

- $\neg(A \wedge B) \iff \neg A \vee \neg B$
- $\neg(A \vee B) \iff \neg A \wedge \neg B$
- $\neg \forall x A \iff \exists x \neg A$
- $\neg \exists x A \iff \forall x \neg A$

Standardize Variables: rename variables so that each quantified variable is unique

Skolemizaation: remove existential quantifiers by introducing new function symbols

Convert to Prenex Form: bring all quantifiers to the front
(1) $\forall x P \wedge Q \iff Q \wedge \forall x P \iff \forall x (P \wedge Q)$; (2) $\forall x P \vee Q \iff Q \vee \forall x P \iff \forall x (P \vee Q)$;
Conjunctions over disjunctions: $A \vee (B \wedge C) \iff (A \vee B) \wedge (A \vee C)$
Flatten nested $\wedge, \vee$: $A \vee (B \vee C)$ to $(A \vee B \vee C)$
Convert to Clauses
Resolution is refutation complete: If a set of clauses is unsatisfiable (i.e., when the answer is "YES") and so some branch contains [ ], a breadth-first search guaranteed to find [ ].
First-order unsatisfiability is semi-decidable, but not decidable. Thus, calculating entailments is semi-decidable and undecidable. First-order satisfiability is undecidable.
Decidable if there is some algorithm that correctly generates a "YES-NO" answer for every possible input. Otherwise, it's undecidable.
Semi-decidable if there is some algorithm that correctly generates "YES" answers, but does not terminate on some inputs for which the answer is "NO"

Convert the following FOL sentences into clausal form:

1. $\forall x(\text{child}(x) \rightarrow \text{loves}(x, \text{santa}))$
   $\neg\text{child}(x) \vee \text{loves}(x, \text{santa})$
2. $\forall x, y(\text{loves}(x, \text{santa}) \wedge \text{raindeer}(y) \rightarrow \text{loves}(x, y))$
   $\neg\text{loves}(x, \text{santa}) \vee \neg\text{reindeer}(y) \vee \text{loves}(x, y)$
3. $\text{reindeer}(\text{rudolph}) \wedge \text{has}(\text{rudolph}, \text{red-nose})$
   $\text{reindeer}(\text{rudolph})$
   $\text{has}(\text{rudolph}, \text{red-nose})$
4. $\forall x(\text{has}(x, \text{red-nose}) \rightarrow (\text{weird}(x) \vee \text{clown}(x)))$
   $\neg\text{has}(x, \text{red-nose}) \vee \text{weird}(x) \vee \text{clown}(x)$
5. $\neg\exists x(\text{reindeer}(x) \wedge \text{clown}(x))$
   $\neg\text{reindeer}(x) \vee \neg\text{clown}(x)$
6. $\neg\exists x(\text{loves}(\text{scrooge}, x) \wedge \text{weird}(x))$
   $\neg\text{loves}(\text{scrooge}, x) \vee \neg\text{weird}(x)$
7. $\neg\text{child}(\text{scrooge})$
   $\neg\text{child}(\text{scrooge})$

.