Propositional Logic

We say m is a model of a sentence α if α is true in m: $m \models \alpha$.
$\mathrm{M}(\alpha) = \{\, m \mid m \models \alpha \,\}$
$\mathrm{KB} \models \alpha \Leftrightarrow \mathrm{M}(\mathrm{KB}) \subseteq \mathrm{M}(\alpha)$
$\alpha \equiv \beta$ if $\mathrm{M}(\alpha) = \mathrm{M}(\beta)$
α is valid $\Leftrightarrow$ α is true in all models (tautology).
α is satisfiable $\Leftrightarrow \mathrm{M}(alpha) \neq \emptyset$
Resolution rule.

$$\frac{\begin{array}{l} l_1 \vee \cdots \vee l_k \\ m_1 \vee \cdots \vee m_n \end{array}}{\begin{array}{l} l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee \\ m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n \end{array}}$$

where $l_i = \neg m_j$ or $\neg l_i = m_j$ on a syntactic level.
A sentence is in CNF iff it is a conjunction of disjunctions of literals.

Approach 1 (Writing propositional logic in CNF)

1. Replace $S_1 \Leftrightarrow S_2$ by $(S_1 \Rightarrow S_2) \wedge (S_2 \Rightarrow S_1)$.
2. Replace $S_1 \Rightarrow S_2$ by $\neg S_1 \vee S_2$.
3. Push $\neg$ inwards until it hits a literal. Use:
$$\neg(S_1 \wedge S_2) = \neg S_1 \vee \neg S_2$$
$$\neg(S_1 \vee S_2) = \neg S_1 \wedge \neg S_2$$
$$\neg\neg S_1 = S_1$$
4. Distribute $\vee$ over $\wedge$ whenever possible:
$$(S_1 \wedge S_2) \vee S_3 = (S_1 \vee S_3) \wedge (S_2 \vee S_3)$$

Approach 2 (Proof by Resolution for propositional logic)

1. Add α to KB.
2. Write $\mathrm{KB} \wedge \neg\alpha$ in CNF.
3. Show that $\mathrm{KB} \wedge \alpha \vdash \bot$ by applying the resolution rule until no new clause can be added anymore, or `false` is derived.

Predicate Logic

Substitution: $\{\, \mathrm{Old}_1/\mathrm{New}_1, \ldots, \mathrm{Old}_n/\mathrm{New}_n \,\}$
A unifier is a substitution which makes two literals equal. We unify as follows:

1. If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same constant.
2. If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2 (and vice versa).
3. If T_1 and T_2 are complex terms then they unify if:
   (a) They have the same functor and arity, and,
   (b) All their corresponding arguments unify, and,
   (c) The variable instantiations are compatible.

The Most General Unifier (MGU) is a unifier with the least amount of variables replaced by concrete values.

$$\forall_x \neg P \equiv \neg\exists_x P \qquad \neg\forall_x P \equiv \exists_x \neg P$$
$$\forall_x P \equiv \neg\exists_x \neg P \qquad \exists_x P \equiv \neg\exists_x \neg P$$

Approach 3 (Removing quantifiers)
First remove existential quantifiers by replacing quantified variables by Skolemisation constants or functions.

$$\forall_{a,b,\ldots}\exists_x P(x,y) \equiv \forall_{a,b,\ldots} P(S(a,b,\ldots),y)$$

Then, drop all universal quantifiers.

Approach 4 (Proof by Resolution for predicate logic)
Same as for propositional logic.

Search

Tree search and graph search can both be used on trees and graphs. The difference is that graph search keeps track of completely visited nodes. Tree search is favorable in acyclic action paths.
A* search is a *best-first* search which uses an evaluation function $f(n) = g(n) + h(n)$ to find the next candidate node n, with $g(n)$ the cost to reach the node n from the start state (known) and $h(n)$ the estimated cost to get from n to the goal.
The tree version of A* is optimal if $h(n)$ is admissible: $h(n)$ never overestimates the cost.
The graph version of A* is optimal if $h(n)$ is consistent: the triangle inequality holds: $h(n) \leq \mathrm{cost}(n, an\prime) + h(n\prime)$.

Probabilistic Reasoning

$$P(A \wedge B) = P(A \mid B)P(B)$$

$$P(A \vee B) = P(A) + P(B) - P(A,B)$$

$$P(X_1, \ldots, X_n) = \Pi_{i=1} P(X_i \mid X_1, \ldots, X_{i-1})$$

$$\mathbf{P}(X) = \Big\langle\, P(X = x_1), \ldots, P(X = x_n) \Big\rangle$$

$$\mathbf{P}(Y) = \sum_z \mathbf{P}(Y, Z = z)$$

$$\mathbf{P}(Y) = \sum_z \mathbf{P}(Y \mid z) P(z)$$

Independence (equivalent statements):

$$\begin{array}{ll} P(a \mid b) = P(a) & \mathbf{P}(X \mid Y) = \mathbf{P}(X) \\ P(b \mid a) = P(b) & \mathbf{P}(Y \mid X) = \mathbf{P}(Y) \\ P(a \wedge b) = P(A)P(b) & \mathbf{P}(X,Y) = \mathbf{P}(X)\mathbf{P}(Y) \end{array}$$

Bayes rule: $P(b \mid a) = \frac{P(a|b)P(b)}{P(a)}$

$$P(A|B) = \begin{bmatrix} P(a,a), P(\neg a,b)/\alpha_b \\ P(a,\neg b), P(\neg a,\neg b)/\alpha_{\neg b} \end{bmatrix}$$
$$\alpha_b = P(a,a) + P(\neg a,b)$$
$$\alpha_{\neg b} = P(a,\neg b) + P(\neg a,\neg b)$$

Bayesian Networks

A Bayesian network is a directed acyclic graph $G(V,E)$ with random variables as nodes and direct influence between random variables as edges. We define $\mathrm{Parents}(X_i) = \{p \mid (p,X_i) \in E\}$ the parents of node X_i.

$$P\left(X_1, X_2, \ldots, X_n\right) = \Pi_{i=1}^n P\left(X_i \mid \mathrm{Parents}(X_i)\right)$$

For the set of nodes A, B and C, $A \perp B \mid C$ if all paths from A to B are blocked.

$$A \xrightarrow{+} W \xrightarrow{+} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

$$A \xleftarrow{+} W \xrightarrow{+} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

$$A \xrightarrow{+} W \xleftarrow{+} B \Rightarrow \begin{cases} A \text{ does not affect } B & \text{if } W \text{ not observed} \\ A \text{ affects } B & \text{otherwise} \end{cases}$$

Machine Learning

- **Unsupervised** learning: we only supply data and give no labels.
- **Semi-supervised** learning: the data is only partly labeled.
- **Supervised** learning: the data is fully labeled.

$$D = \Big\{\, \{\xi^\mu, S^\mu\} \,\Big\}_\mu = 1^P$$

$$\begin{array}{rr} D = \cup_{i=1}^n D^{(i)} & \text{all data} \\ D^{(i)}_{\text{train}} = D \setminus D^{(i)} & \text{training data } (i) \\ D^{(i)}_{\text{test}} = D^{(i)} & \text{test data } (i) \end{array}$$

We wish to aim two competing goals:

- Low **bias**: the system only slightly deviates from the "true solution".
- Low **variance**: the system weakly depends on the training set.

		Actual P	**Actual** N
Pred.	P	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$= \text{Sensitivity} = \text{True Positive Rate}$$
$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$
$$= \text{True Negative Rate}$$
$$\text{F-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{E} = \frac{1}{2} \sum_{n=1}^{N} \left(y(X_n) - t_n \right)^2$$

Two internal measures for clustering: cohesion, separation.

$$\text{WSS} = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

$$\text{BSS} = \sum_i |C_i| \, (m - m_i)^2$$

Decision Trees

$$\text{Entropy}(S) = \sum_{i=1}^{c} -P_i \log_2 (P_i)$$

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{Entropy}\left(S_v\right)$$

Approach 5 (Iterative Dichotomiser)
Input. A data set with attributes.
Output. A decision tree.

1. Find the attribute that provides the highest information gain within this data set.
2. Create a node using this attribute.
3. Sort out the data set to the corresponding split.
4. For each split data set: if all data points correspond to the same class, create a leaf node.
5. Otherwise, apply this algorithm to the split data set and insert the resulting tree here.

- If all the records in the current data set have the same class, then stop recursing.
- If all records have the same set of input attributes, then stop recursing.

Approach 6 (Pruning decision trees)

1. Consider a node with p positive and n negative examples. Let p_1, n_1 and p_2, n_2 be the positive and negative examples for the two alternatives of the node.
2. Define $\hat{p_1} = p \cdot \frac{p_1 + n_1}{p + n}$ and $\hat{n_1}$, $\hat{p_2}$, and $\hat{n_2}$.
3. Calculate $\Delta = \frac{(p_1 - \hat{p_1})^2}{\hat{p_1}} + \frac{(n_1 - \hat{n_1})^2}{\hat{n_1}} + \frac{(p_2 - \hat{p_2})^2}{\hat{p_2}} + \frac{(n_2 - \hat{n_2})^2}{\hat{n_2}}$
   $\Delta \sim \chi^2_{d-1}$, with d the number of splits. Small values for Δ imply no rejection of the null hypothesis, and hence justify the pruning of the tree.

Reinforcement Learning

$$U_h\left[s_0, s_1, s_2, \ldots\right] = \text{R}\left(s_0\right), \gamma \text{R}\left(s_1\right), \gamma^2 \text{R}\left(s_2\right), \ldots$$

$$\text{U}^\pi(s) = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t \text{R}\left(S_t\right)$$

$$\pi^*(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s'} P\left(s' \mid s, a\right) \text{U}(s')$$

$$\text{U}(s) = \text{R}(s) + \gamma \max_{a \in A(s)} \sum_{s'} P\left(s' \mid s, a\right) \text{U}\left(s'\right)$$

Value Iteration

$$\text{U}_{i+1}(s) \leftarrow \text{R}(s) + \gamma \max_{a \in A(s)} \sum_{s'} P\left(s' \mid s, a\right) \text{U}_i\left(s'\right)$$

Policy Iteration
Start with random $\pi = \pi\prime$. Until there is no change anymore, for each $s \in S$, consider a such that

$$\left[\text{R}(s) + \gamma \sum_{s' \in S} P\left(s' \mid s, a\right) \text{U}\left(\pi, s'\right) \right] > \text{U}(\pi, s)$$

Direct Utility Estimation

$$\text{U}^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\left(S_t, \pi(S_t), S_{T+1}\right) \right]$$

Adaptive Dynamic Programming

$$\text{U}^\pi(s) = \text{R}(s) + \gamma \sum_{s'} P\left(s' \mid s, \pi(s)\right) \text{U}^\pi\left(s'\right)$$

$$P\left(s' \mid s, a\right) = \frac{\text{times you visited } s' \text{ from } s \text{ with action a}}{\text{times you performed action } a \text{ at state } s}$$

Active learning
Estimate $P(s' \mid s, a)$ for more than one action. The action the agent should choose is

$$\pi(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s' \in S} P(s' \mid s, a) U(s')$$

To balance exploitation and exploration, use an exploration function $f(u, n)$:

$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s' \mid s, a) \Big[R(s, a, s') + \gamma U^+(s') \Big],\ N(s, a) \right)$$
$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

with $N(s, a)$ is the number of times the action a has been executed in state s.

Machine Learning for Security

$$\overline{\phi_i}(d) = \frac{\phi_i(d) - \min_i}{\max_i - \min_i} \in [0, 1]$$

$$\mu_+ = \underset{\mu \in \mathbb{R}^N}{\text{argmin}} \max_{1 \le i \le n} ||x_i - \mu||^2$$

$$\min_{R, \mu, \xi} R^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i$$

subject to $||x_i - \mu||^2 \le R^2$ for all $i = 1, \ldots, n$.