CS 165A Artificial Intelligence Notes

Alex Mei

Fall 2021

1 Intelligent Agents

1.1 Definitions of AI

- **Artificial**: using the "type of" definition, not the "imitation/fake" definition.
- **Intelligence**: acquire and apply knowledge, faculty of thought and reason, perform symbol manipulation, achieve goals.
- **Strong AI**: computers can be made to think on a level (at least) equal to humans.
- **Physical Symbol System Hypothesis**: a version of Strong AI, noting that AI can take symbolic patterns, combine them into expression structures and manipulate them via processes to produce new expressions.
- **Weak AI**: some human-like features for computers as useful tools.
- **Ideal vs Human Reasoning**: ideal systems always think rationally, while human-like systems may think irrationally.
- **Ideal vs Human Behavior**: ideal systems always act rationally, while human-like systems may act irrationally.
- **Cognitive Modeling**: sciences related to acting humanly.
- **Ideal Intelligence**: area relating to thinking and acting rationally.
- Note: Artificial intelligence models does not necessarily want to replicate human behavior, sometimes more (i.e., emotional support) and sometimes less (i.e., driving).
- **Components of an AI Program**: knowledge database, production rule operations, choice of production rules to apply.

1.2 Goals of AI

- **Scientific:** to create models and mechanisms of intelligent action.
- **Engineering:** to understand and build intelligent systems.
- **Turing Test:** see whether an interrogator can differentiate between human and computer.
- **Chinese Room Argument:** person in a room with a Chinese translation mapping can pass a Turing test without thinking or understanding Chinese. The entire room is said to understand, but not the person inside the room itself. (Syntactical understanding is different from semantic understanding.)

1.3 Agents

- **Definition**: an entity capable of combining cognition, perception and action in behaving autonomously, purposively and flexibly in some environment.
- **Textbook View of AI**: building rational agents. Given perceptions of the environment, P, we want to build an agent function f that maps $p \in P$ to an optimal action $a \in A$.

$$f : P \to A$$

- Note: To implement f, we can use a lookup table or learning.

Knowledge, past percepts, past actions

```
function SKELETON-AGENT(percept) returns action
  static: memory, the agent's memory of the world

  memory ← UPDATE-MEMORY(memory, percept)
  action ← CHOOSE-BEST-ACTION(memory)
  memory ← UPDATE-MEMORY(memory, action)
  return action
```

e.g.,

Table-Driven-Agent

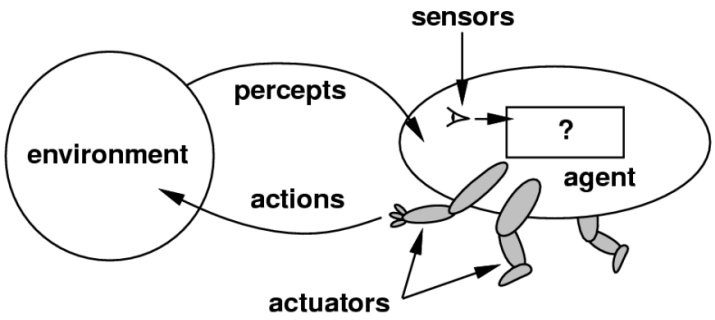
Add *percept* to percepts

LUT [percepts, table]

NOP

- **Ideal Rational Agent**: for each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

1.4 Describing the Task Environment



- **PEAS**: Performance measures, Environment, Actions, Sensed information.
- **Fully vs Partially Observable Environment**: when an agent sensor can see the complete state of an agent at each point in time, it is said to be a fully observable environment else it is partially observable.
- **Deterministic vs Stochastic Environment**: when a uniqueness in the agent's current state completely determines the next state of the agent, the environment is said to be deterministic, otherwise it's stochastic.
- **Competitive vs Collaborative Environment**: an agent is said to be in a competitive environment when it competes against another agent to optimize the output; when agents cooperate, that is said to be a collaborative environment.

– **Episodic vs Sequential Environment**: episodic is an environment where each state is independent of each other; the sequential environment is an environment where the next state is dependent on the current action.

– **Static vs Dynamic Environment**: an environment that keeps constantly changing itself when the agent is up with some action is said to be dynamic, otherwise it is static.

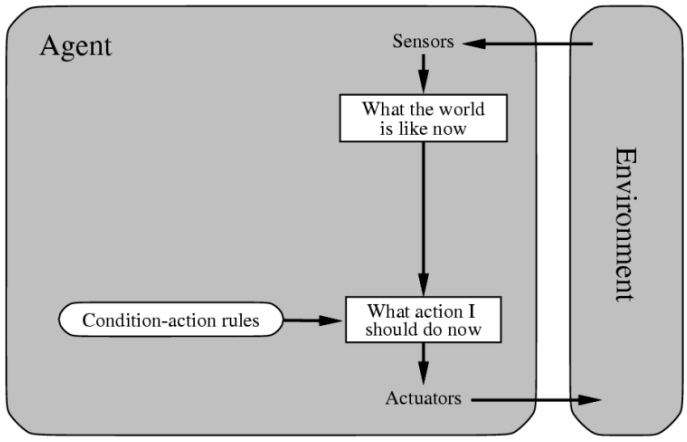
– **Discrete vs Continuous Environment**: if an environment consists of a finite number of actions that can be deliberated in the environment to obtain the output, it is said to be a discrete environment.

– **Single-agent vs Multi-agent Environment**: an environment involving one agent is a single-agent environment and involving more than one agent is a multi-agent environment.

1.5 Basic Agent Programs

1.5.1 Simple Reflex Agent

- **Definition**: Heuristic-based agent without memory that uses if-then production rules that associates inputs to outputs.

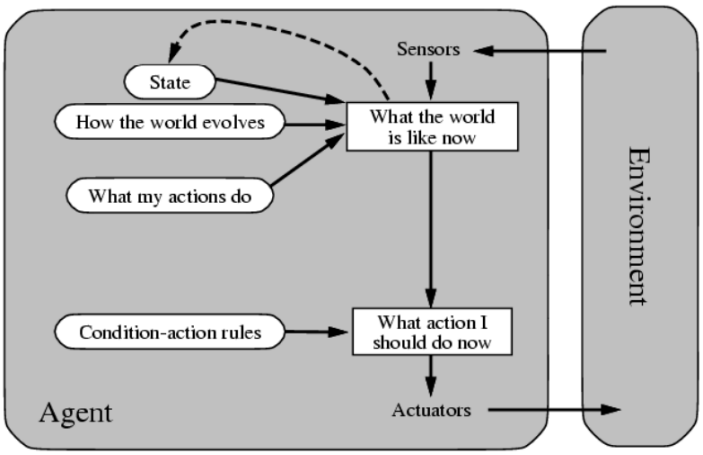


function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
static: *rules*, a set of condition-action rules

state ← INTERPRET-INPUT(*percept*)
rule ← RULE-MATCH(*state*, *rules*)
action ← RULE-ACTION[*rule*]
return *action*

1.5.2 Model-based Reflex Agent

- **Definition**: Uses an internal state to keep track of the world; tries to maximize reward of states.
- Given current state, takes percepts and tries to classify into a known state and update to that state.

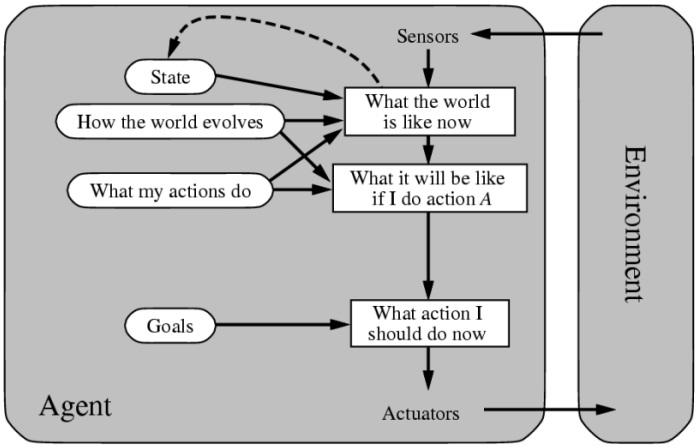


```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

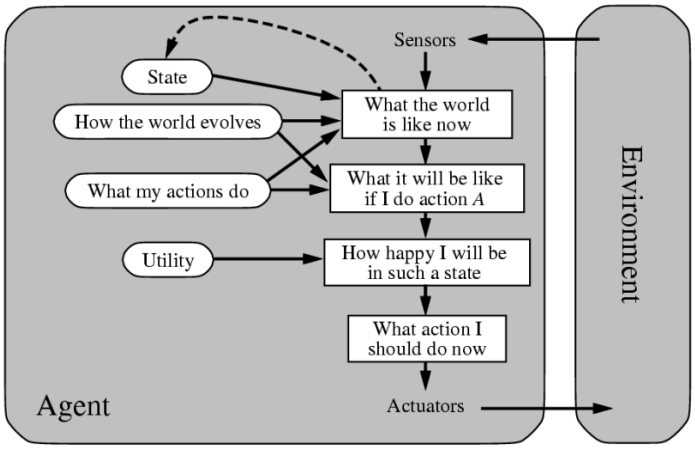
1.5.3 Goal-based Agent

- **Definition**: Finds optimal sequence of action to achieve goals; involves searching and planning actions.



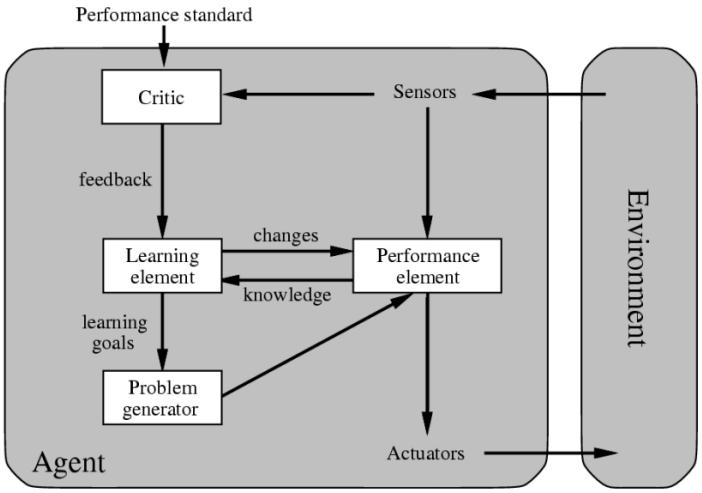
1.5.4 Utility-based Agent

- **Definition**: In addition to the Goal-based agent, specifies a utility function to map state into a utility value.



1.5.5 Learning Agent

- **Definition**: Most advanced agent that learns new methods as well by exploring and retrieving feedback.
- Problems are generated, measured by performance, creating a feedback loop for continuous improvement.



2 Probabilistic Reasoning

2.1 Definitions

- **Deterministic Process**: outcome can be predicted exactly in advance.
- **Random Process**: outcome is not known exactly, but described by a probability distribution of possible outcomes.
- **Event**: set of outcomes of a random process.

$$0 \leq P(A) \leq 1$$

- **Sample Space**: set of all possible outcomes.

$$P(S) = 1$$

- **Complement**: given event A, the complement is the event that A does not occur.

$$P(A') = 1 - P(A)$$

- **Union**: the event that either events A or B or both occurs.

$$P(A \cup B) = P(A) + P(B) – P(A \cap B)$$

- **Intersection**: the event that both events A and B occur.
- **Disjoint**: two events that cannot happen at the same time.

$$P(A \cup B) = P(A) + P(B)$$

- **Independent**: two events that do not affect the probability of one another.

$$P(A \cap B) = P(A) * P(B)$$

- **Random Variable**: numerical outcome of an event.
- **Complete Probability Model**: a single joint probability distribution over all variable in the domain.
- **Prior (Unconditional) Probability**: probabilities associated with a proposition or variable, prior to any evidence.
- **Posterior (Conditional) Probability**: probabilities of query variable(s) after evidence variable(s) is/are gathered:

$$P(X|Y) * P(Y) = P(X, Y)$$

- **Expectation**:

$$E(X) = \int_X x * P(x)$$

- **Moment**:

$$E(X^n) = \int_X x^n * P(x)$$

- **Variance**:

$$V(X) = E(X^2) - [E(X)]^2$$

- **Standard Deviation**:

$$\sigma(x) = \sqrt{V(X)}$$

- **Covariance**:

$$Cov(X, Y) = E(XY) - E(X) * E(Y)$$

- **Correlation**:

$$Corr(X, Y) = \frac{Cov(X, Y)}{\sigma(x)\sigma(y)}$$

- **Marginalization**:

$$P(X) = \int_Y P(X, Y) = \int_Y P(X|Y)P(Y)$$

- **Joint Probabilities**:

$$P(X_1) = \int_{X_2} ... \int_{X_n} P(X_1, ...X_n)$$

- **Conditional Independence**: two events A, B are conditionally independent on event C iff

$$P(A|B, C) = P(A|C)$$

- Note: independence does not imply conditional independence.

2.2 Probability Distributions

- **Binomial PDF**:

$$f(x) = \frac{n!}{x!(n-x)!} * p^x * (1-p)^x$$

- **Poisson PDF**:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

- **Gaussian PDF**:

$$f(x) = \frac{\exp(-\frac{(x-\mu)^2}{2})}{\sigma\sqrt{2\pi}}$$

- **Exponential PDF**:

$$f(x) = \lambda \exp(-\lambda x), x \geq 0$$

2.3 Linear Algebra

- **Matrix Multiplication:** given $A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{p \times n}, C = AB = B \in \mathbb{R}^{m \times n}$ where

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

- L_1 **Norm**:

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

- L_2 **Norm**:

$$||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i^2|}$$

- L_∞ **Norm**:

$$||x||_\infty = \max_{1 \leq i \leq n} |x_i|$$

- **Frobenius Norm (for Matrices)**:

$$||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}^2|}$$

- **Trace**: given $B \in \mathbb{R}^{n \times n}$:

$$trace(B) = \sum_{i=1}^{n} b_{ii}$$

- **Eigenvalues λ and Eigenvectors** v:

$$Av = \lambda v$$

- To find λ, v, solve:

$$det(A - \lambda I) = 0$$

2.4 Independent Values

- **Definition**: number of data points needed to describe a probability distribution.
- A single variable with n outcomes needs $n - 1$ independent probabilities to describe the distribution.
- An intersection of variables requires the product of the number of outcomes for each variable, minus one.
- The product of variables needs the sum of the number of independent probabilities to describe each distribution.
- Conditional probabilities need the product of the number of independent probabilities needed to describe the inference variable(s) times the number of outcomes on the conditioned variable(s).

2.5 Bayesian Networks (Belief Nets)

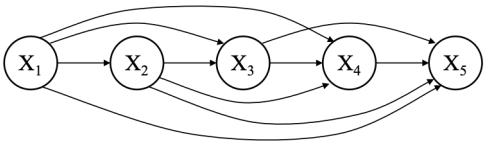
- **Definition**: a directed acyclic graph with nodes representing random variables and edges representing influence/causality.
- An edge from node $X \rightarrow Y$ means that X "directly influences" Y.
- **Conditional Probability Table (CPT)**: associated with each node, defined by $P(node|parents)$.
- **Joint Probability of a Belief Net**:

$$P(var_1, ..., var_n) = \Pi_i P(var_i | Parents(var_i))$$

- Note: we can define a utility function as a function of the probability of a belief net (or generic model) to transform the probability into a decision rule based on some other threshold unit.

2.5.1 Drawing Belief Nets

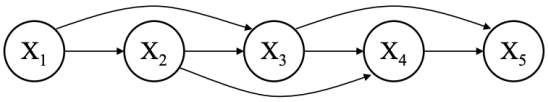
- Fully Connected Belief Nets have $n!$ ways to draw, given n nodes. One such way is:



$P(X_1,X_2,X_3,X_4,X_5) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)P(X_4|X_1,X_2,X_3)P(X_5|X_1,X_2,X_3,X_4)$

- Conditionally Independent Variables have links from parent to node:

$P(X_1, X_2, X_3, X_4, X_5) = P(X_1)\ P(X_2|X_1)\ P(X_3|X_1,X_2)\ P(X_4|X_2,X_3)\ P(X_5|X_3, X_4)$



- Independent Variables do not have common ancestors:

$$P(X_1, X_2, X_3, X_4, X_5) = P(X_1)\,P(X_2)\,P(X_3)\,P(X_4)\,P(X_5)$$

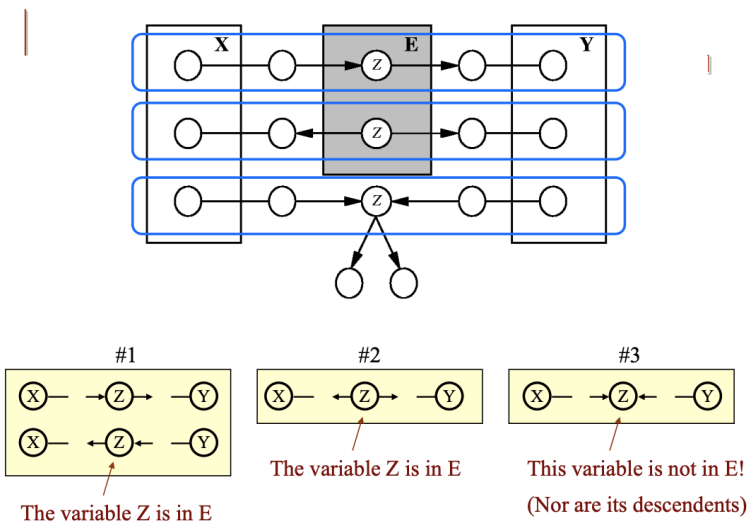
- Note: the direction is not interpretable and only shows association.
- Note: can have association even when independent as based off the probability.
- Note: we can redraw a belief net to remove cycles.

2.5.2 Constructing Belief Nets

- Choose nodes as random variables.
- Choose direction of influence to satisfy DAG.
- Draw directed edges to indicate direct influence.
- Define conditional probability table for each node.
- Note: wrong conditional independences may lead to wrong answers.

2.5.3 Independence in Belief Nets

- Two nodes are independent when they do not share a common ancestor.
- Two nodes are conditionally independent on a set of evidence nodes if every path is blocked (d-separated) by the evidence nodes.



2.5.4 Applications of Belief Nets

- Inference with Belief nets is too computationally expensive since Bayesian inference is NP hard, the structure is too large, and not all the CPTs are known.
- Practically, we can calculate probability of causes given symptoms, and vice versa as an explainable model. We can determine variable sensitivity and which evidence variables are useful.

3 Learning

3.1 Classification / Supervised Learning

- **Definition**: Given set of instances $x \in X$ where we want to classify each input $x \in \mathbb{R}^n$ into a fixed set of classes $c \in C$ where $c \in \mathbb{Z}^+$.
- **Task**: find function $f : X \to C$ that optimizes performance.
- **Model Construction**: determine fixed set of classes and which algorithm to represent model.
- **Model Usage**: estimate accuracy with training data and improve as needed; then, run validation set to prevent overfitting; if acceptable, deploy to production.
- Note: training and testing data need to have similar distributions.
- Note: model could be rule based, but this is not scalable like learning algorithms, which also has the added benefit of noise and error reduction.
- **Lazy Learning**: store training data and wait until test time to predict; less time in training, but more time in predicting, which allows locally linear trends relative to the global approximation.
- **Eager Learning**: construct model before test time, forcing a commitment to one hypothesis space.

3.2 Naive Bayes Model

- **Bayesian Methods**: learning methods based on probability theory that builds a generative model to approximate how data is produced using a prior probability of each category to produce a posterior probability distribution over the possible categories.
- **Baye's Rule**:

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$

- **Note**: here, $P(Y|X)$ is causal (likelihood) knowledge.
- **Definition**: model that tries to predict a single cause given its many influence variables, which are conditionally independent given that cause.

$$c(x) = \max_{c_j \in C} P(x_1, ..., x_n|c_j) * P(c_j) = \max_{c_j \in C} P(c_j) * \Pi_{1 \leq i \leq n} P(x_i|c_j)$$

- **Maximum Likelihood Estimate**: simply use frequencies in the data.

$$\hat{P}(c_j) = \frac{count(C = c_j)}{count(*)}$$

$$\hat{P}(x_i|c_j) = \frac{count(X_i = x_i, C = c_j)}{count(C = c_j)}$$

- **Problem with MLE**: small sample sizes may result in some values not observed, rendering those probabilities zero.
- **Smoothing (for Overfitting)**: add constant m to mitigate the effect of the frequency probability.

$$\hat{P}(x_i|c_j) = \frac{count(X_i = x_i, C = c_j) + 1}{count(C = c_j) + m}$$

$$\hat{P}(x_i|c_j) = \frac{count(X_i = x_i, C = c_j) + m * P(X = x_i)}{count(C = c_j) + m}$$

- **Log Transform (for Underflow/Overflow)**: prevent multiplying a bunch of small probabilities to cause a floating point underflow by applying the monotonically increasing log transformation:

$$c(x) = \max_{c_j \in C} [log(P(c_j)) + \sum_{1 \leq i \leq n} log(P(x_i|c_j))]$$

3.2.1 Text-based Naive Bayes

- **Stochastic Language Models**: models probability of generating strings in the language.
- If we assume features are positions of text positions, and values are words, this creates too many possibilities.
- Extract vocabulary V from training corpus.
- Calculate: given $c_j \in C$ is a classification class, x_k is a query word

$$P(c_j) = \frac{count(docs \in c_j)}{count(docs)}$$

$$P(x_k|c_j) = \frac{count(x_k \in (docs \in c_j)) + \alpha}{count(words) + \alpha * count(V)}$$

- We classify using the modified function:

$$c(x) = \max_{c_j \in C}[P(c_j) * \Pi_{i \in positions} P(x_i|c_j)]$$

$$c(x) = \max_{c_j \in C}[log(P(c_j)) + \sum_{i \in positions} log(P(x_i|c_j))]$$

- **Training Time**: $O(|D|L_d + |C||V|) = O(|D|L_d)$ since $|C||V| << |D|L_d$. (L_d is average length of a document $d \in D$.)
- **Testing Time**: $O(|C|L_t)$. (L_t is the length of a test document.)
- Overall, this algorithm is linearly proportional to the data being read, which is very efficient.

3.2.2 Numerical Data for Naive Bayes

- Can model the probability as the normal pdf given the sample mean and sample variance.
- **Sample Mean**:

$$\mu = \frac{1}{n}\sum_{i}^{n} x_i$$

- **Sample Variance**:

$$\sigma^2 = \frac{1}{n-1}\sum_{i}^{n}(x_i - \mu)^2$$

3.2.3 Naive Bayes Assumptions

- $P(c_j)$ can be estimated from the frequency of classes in the training samples.
- Assume the probability of observing $P(x_1, ..., x_n|c_j)$ is equal the product of $P(x_i|c_j)$. (Otherwise, need $O(|X|^n|C|)$ parameters to build model).

3.2.4 Naive Bayes Takeaways

- Classification results are usually accurate, but the actual probabilities are not.
- Assumptions of positional and conditional independence can be badly violated and still work due to the relative probability differences in classes.
- Good in domains with equally important features and robust to irrelevant features that cancel each other out.
- Works with small samples, is very fast, and takes minimal storage.

3.3 Information Retrieval

- **Bag of Words Model**: assume classification is independent of text position.
- **Vocabulary**: $V = w_1, ..., w_n$ where w_i is a word.
- **Collection**: $D = d_1, ..., d_k$ where d_i is a document.

- **Document**: $d = d_1...d_p$ where $d_i \in V$.
- **Query**: $q = q_1...q_m$ where $q_i \in V$.
- Want to find the set of relevant documents $R(q) \subseteq D$.
- Task is to optimize $R'(q)$ to be as optimal as $R(q)$ as possible.
- **Document Selection Strategy**: given classification function f.

$$R'(q) = \{d \in D | f(d, q) = 1\}$$

- **Document Ranking Strategy**: given threshold $\theta \in \mathbb{R}$ and f maps to $\mathbb{R}$.

$$R'(q) = \{d \in D | f(d, q) > \theta\}$$

- Define Relevancy = Similarity

$$f(d, q) = \Delta(Representation(q), Representation(d))$$

- **Vector Space Model**: represent text as a vector, with each term defining one dimension, each vector element corresponding to term weight, and vector similarity to determine relevance.
- **Stop Words**: irrelevant words that can be removed to save computational complexity.
- **Stemming**: unify syntactic variants of the same word.

3.3.1 Term Frequency, Inverse Document Frequency (TFIDF)

- **Premise**: need to weight query terms as some terms more important than others and need to weight document terms as some provide more information.
- Raw Term Frequency (TF):

$$TF(term) = \frac{count(term, doc)}{count(words \in doc)}$$

- Log Term Frequency:

$$TF(term) = log(\frac{count(term, doc)}{count(words \in doc)} + 1)$$

- Note: need to normalize term frequency to penalize (not over-penalize) long documents. There is a relationship between normalization and smoothing.
- **Pivoted Normalization**: given normalization factor b.

$$1 - b + \frac{b * len(d)}{L_d}$$

- **Maximum Frequency Normalization**:

$$TF(term) = 0.5 + \frac{0.5 * (count(term, doc)}{max_{term}(doc)}$$

- **Popularized TF Normalization**:

$$TF(term) = 0.5 + \frac{(1 + k) * count(term, doc)}{count(term, doc) + k * (1 - b + \frac{b * len(d)}{L_d})}$$

- **Inverse Document Frequency (IDF)**: given n docs and k docs with given term.

$$IDF(term) = 1 + log(n/k)$$

- **TDIDF Weighting**:

$$weight(term, doc) = TF(term, doc) * IDF(term)$$

- Define the relevance as the TFIDF vectors for doc and query and take the dot product.
- **Advantages**: effective, intuitive, easy to implement, well-studied and evaluated.
- **Disadvantages**: assumes term independence, document and query equivalence, many hyperparameters.
- **Summarization Application**: choose sentences that rank highest in comparison to other sentences or whole text.

3.4 Measurements

- **Precision**:

$$\frac{TruePositive}{TruePositive + FalsePositive}$$

- **Recall**:

$$\frac{TruePositive}{TruePositive + FalseNegative}$$

- **F1 Score**:

$$\frac{2 * Precision * Recall}{Precision + Recall}$$

- Can measure quality by computing the area under a Precision-Recall tradeoff curve.
- **Dot Product**:

$$q^T d$$

- **Cosine**: normalized dot product.

$$\frac{q^T d}{||q||_2 ||d||_2}$$

- **Euclidean**: affected by scale.

$$\sqrt{\sum_i (q_i - d_i)^2}$$

3.5 Decision Trees

- **Definition**: a tree that divides data into partitions based on features, and then makes predictions from those features.
- **Break Point**: we can stop partitioning the tree when there are no labels left or the partition represents a single class.
- **Entropy**: probability that an arbitrary tuple in $x \in X$ belongs to class $c_i \in C$:

$$H(Y) = \sum_{c_i} P(x \in c_i) * \log_2(P(x \in c_i))$$

$$H(Y|X_i) = \sum_{x_i \in X_i} P(x_i) * H(Y|x_i))$$

- **Information Gain**: entropy needed to classify X after splitting X by some feature X_i, where $H(Y)$ is the information before splitting X:

$$Gain(X_i) = H(Y) - H(Y|X_i)$$

- **Tree Splits**: split by the attribute that maximizes information gain and recursively apply on resulting subtrees.

3.5.1 Continuous Valued Attributes

- **Discretizing Data**: we convert the attribute into discrete buckets by sorting the data and then consider whether to split among split point candidates.
- **Split Point Candidate**: the midpoint of two continuous values of a feature that has different labels.
- Note: to prevent overfitting by splitting into buckets of size 1, we add a normalization term on the number of splits, split info.
- **Split Info**: given v is the number of discrete buckets, attribute A, and X_j is the jth bucket:

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{|X_j|}{|X|} * \log_2(\frac{|X_j|}{|X|})$$

- **Gain Ratio**: we use gain ratio as the metric for continuous valued attributes for the decision tree splitting choice:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

3.5.2 Overfitting in Decision Trees

- **Definition**: cannot generalize outside the training data (since trees can easily overfit with too many branches).
- **Prepruning**: decide threshold to stop tree construction early; brute-force non-scalable approach.
- **Postpruning**: remove branches from complete tree and test on validation set for result difference.
- **Training Set**: data which the model trains on.
- **Validation Set**: data which the model tests on.
- **Testing Set**: data which the model runs on.
- Note: data must be seeded and shuffled for correct results.

3.6 Linear Classifier

- Given input vector x and want to classify to label y.
- Task: find weight vector w that optimizes $y = w^T x$.

3.7 k-Nearest Neighbors

- **Definition**: algorithm that classifies based on plurality vote based on its nearest neighbors' labels (or the mean if returning a real value).
- **Nearest Neighbor**: the data point that minimizes Euclidean distance in $\mathbb{R}^d$ space. (Discrete case: most common training example similar to test example).
- **Vonoroi Diagram**: partition of space to show which nearest neighbor that area corresponds to.
- **Weighting Neighbors**: assign a weight w to each neighbor x_i based on their distance d to the test data point x_q:

$$w = \frac{1}{d(x_q, x_i)^2}$$

- **Advantages**: robust to noisy data as number of neighbors k increases.
- **Disadvantages**: distance of neighbors can be dominated by irrelevant attributes, which need axes normalization; additionally, in high dimensional space, neighbors can all be far apart / close to each other.

3.8 Clustering / Unsupervised Learning

- Given n observations from a joint probability distribution and want to infer properties about this distribution.
- **Task**: group collection of data into partitions where the data is close to each other.
- Note: no measure of success; heuristic arguments for judgement.

3.9 Distance

- **Metric Functions**: satisfy the three properties:

$$d(x, y) \geq 0, d(x, y) = 0 \iff x = y$$

$$d(x, y) + d(y, z) \geq d(x, z)$$

$$d(x, y) = d(y, x)$$

- **Nonmetric Functions**: violate above properties (i.e., graphs).
- **Euclidian Distance**:

$$\sqrt{(x - y)^2}$$

- **Manhattan Distance**:

$$|x - y|$$

- **Infinity Distance**:

$$\max_i |x_i - y_i|$$

- **Intracluster Distance**: distance measure after K clusters are formed:

$$W(K) = 0.5 \sum_{k=1}^{K} \sum_{i=1}^{|k|} \sum_{j=i+1}^{|k|} d(x_i, x_j)$$

- **Sum of Squared Error Distance**:

$$J = \sum_{k=1}^{K} \sum j = 1^{|k|} ||x_i - \overline{x_k}||^2$$

$$\overline{x_k} = \frac{1}{|k|} \sum_{i=1}^{|k|} x_i$$

- **Intercluster Distance**: distance between clusters, defined as $B(C)$.
- **Single Link**: smallest distance between elements in two clusters.
- **Complete Link**: longest distance between elements in two clusters.
- **Average Link**: average distance between elements in two clusters.
- **Centroid Link**: distance between centroids two clusters.
- **Medoid Link**: distance between medoids two clusters.
- **Total Distance**:

$$T(K) = W(K) + B(K)$$

3.10 K-Means Clustering

- Randomly select k centers and partition into k clusters.
- Compute the mean for each cluster.
- Assign each point to the closest partition.
- If any point changed clusters, repeat mean computation and assignment.
- **Strengths**: relatively efficient with $O(tkn)$ runtime where t is number of iterations.
- **Weaknesses**: only works for data with defined mean and number of clusters; not robust to outlier and noisy data; cannot handle high dimensional and non-convex shape data; no defined global minimum; order dependent for real time data.
- **Number of Clusters**: loop through increasing number of clusters and continue while sharp drops in $W(K)$ is observable (data is intended to split); we can also randomly generate data on a uniform distribution and apply this process.

3.11 K-Medoids Clustering

- **Medoid**: point in cluster that minimizes distance with rest of data in the cluster.
- **Definition**: k-means algorithm, except choose medoids as centers instead of the mean to be more robust.
- **Agglomerative Hierarchical Clustering**: generate large number of groups and merge closest two groups recursively to build a merge tree and see which subtree performs best.
- **Divisive Hierarchical Clustering**: top down approach to generating a merge tree, splitting with increasing clusters for increased efficiency.

4 Search

- **Problem Solving Goal**: find a sequence of actions that leads to goal states while minimizing cost.
- **Offline Problem Solving**: formulate problem with known states, apply search, execute results.
- **Environment Assumptions**: observable, static, discrete, deterministic.
- **State Diagram**: represents configurations of the agent.

- **State Tree**: data structure with nodes including the parent, children, depth, and cost. Root is the initial state. Leaves are unexpanded states.
- **Repeating States**: removing repeated states can significantly prune search space; don't return to parent, don't create cyclical paths, don't generate a past state. Tradeoff is memory tracking and potential unoptimality.
- **Tree Search**: don't remember states and may repeat states.
- **Graph Search**: remember all states that have been explored and don't repeat some states.

4.1 Formulating Problems

- 5-tuple $(\{S\}, S_0, \{S_G\}, \{O\}, g(.)$.
- $(\{S\}$ denotes set of possible states.
- S_0 denotes initial state.
- $\{S_G\}$ denotes set of goal states.
- $G(S)$ denotes the goal test function.
- $O : \{S_i\} \subseteq \{S\} \rightarrow \{S_j\} \subseteq \{S\}$ denotes the possible operators.
- $g(.)$ denotes the the cost function.
- **Search Problem**: determine optimal $o \in O$ at any given time.

4.2 Search Criteria

- **Completeness**: guaranteed to find a solution (if one exists)?
- **Optimality**: Does it find a "best" solution?
- **Time complexity**: number of nodes generated/expanded.
- **Space complexity**: memory usage.
- **Performance Measures**: best case, worst case, average case, real-world case.
- **Branch Factor**: number of possible choices at each state.
- **Effective Branching Factor**: $N = 1 + b^* + ... + (b^*)^d$ where b^* is the EBF for N nodes expanded at depth d.
- **Brute Force Runtime**: given, branching factor b and steps m, runtime is $O(b^m)$.

4.3 Blind Search

- **Information Available**: current state, complete path from initial state, current operators, goal test, current path cost.
- **Tree Search Algorithm**: without queue.

```
function GENERAL-SEARCH(problem, strategy) returns a solution or failure

 initialize the search tree using the initial state of problem
 loop do
   if there are no candidates for expansion then return failure
   choose a leaf node for expansion according to strategy
   if the node contains a goal state then return the corresponding solution
   else expand the node and add the resulting nodes to the search tree
 end
```

- **Queue Search Algorithm**: with queue.

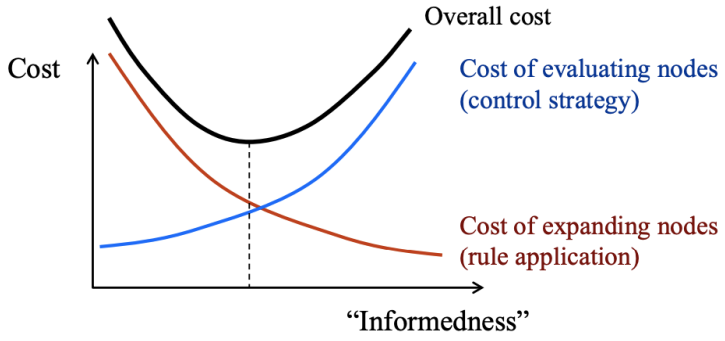
```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution or failure

 nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
 loop do
   if nodes is empty then return failure
   node ← REMOVE-FRONT(nodes)
   if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
   nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
 end
```

- **Breadth First Search**: expands nodes in increasing depth d of tree; considers shortest path, not lowest cost.
- **Breadth First Search Properties**: complete, optimal for uniform cost, exponential time $O(b^d)$, exponential space $O(b^d)$.
- **Depth First Search**: expands nodes at deepest tree level. Uses queue search with a stack. Denote m as max tree depth.
- **Depth First Search Properties**: not complete, not optimal, exponential time $O(b^m)$, polynomial space $O(b * m)$.
- **Depth Limited Search**: DFS with depth limit l to prevent long and unfruitful paths.
- **Depth Limited Search Properties**: complete if $d \leq l$, not optimal, exponential time $O(b^l)$, polynomial space $O(b * l)$.
- **Iterative Deepening Search**: DLS with iteratively increasing l to combine BFS properties of optimality and completeness with DFS memory savings. Preferred blind search when space is large and solution depth is unknown.
- **Iterative Deepening Search Properties**: complete, optimal uniformly, exponential time $O(b^d)$, polynomial space $O(b*d)$.
- **Bidirectional Search**: expands from both start and goal state. Needs action inverse production rules to be defined, and only works with a single goal state. Also need to test for intersection of two searches in constant time. Not practical.
- **Bidirectional Search Properties**: complete, optimal for uniform cost, exponential time and space $O(b^{d/2})$.

4.4 Informed Search

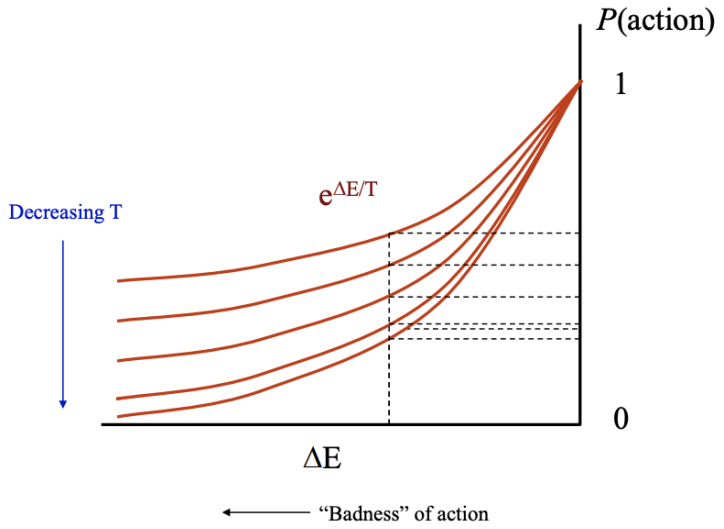
- **Tradeoff**: assumes reasonable heuristics exist evaluated in negligible time; can be hard to evaluate desirability of heuristic.



- **Best First Search**: class of algorithms that expand the most promising node first.
- **Uniform Cost Search**: breadth-first search, but expands lowest historical cost node. Optimal when path cost is monotonically increasing. Denote C as optimal cost and ϵ as minimum step cost.
- **Uniform Cost Search Properties**: complete, optimal if $\epsilon > 0$, exponential time $O(b^{C/\epsilon})$, exponential space $O(b^{C/\epsilon})$
- Note: C/ϵ estimates the depth of search.
- **Greedy Best-First Search**: breadth-first search, but expands lowest future cost node.
- **Greedy Best-First Search Properties**: not complete, not optimal. Time and Space complexity dependent on heuristic.
- **A Search**: minimizes $f(n) = g(n) + h(n)$ where $g(n), h(n)$ is past, future cost. Estimates cheapest complete path.
- **A* Search**: A Search with admissible heuristic.
- **Admissible**: a heuristic that never overestimates the cost. (Ensures we don't miss goal state.)
- **A* Search Properties**: optimal, complete, exponential time $O(b^d)$, exponential space $O(b^d)$.
- Note: A* is optimal because the search continues until all other nodes have a higher cost than the potential optimal cost. It is also efficient because no other optimal algorithm is guaranteed to expand fewer nodes. (Otherwise, may skip solution.)
- **Consistency**: a consistent heuristic satisfies $h(n \to goal) \leq cost(n \to n') + h(n' \to goal)$. Consistency guarantees admissibility (can prove definition by induction).
- Note: a consistent heuristic guarantees the first path to a node is the best path and avoids having to repeat states.
- **Iterative Deepening A***: optimal variant of A* that does not save a queue of nodes to visit and instead explores as long as $path < d$ where d is the bound of the f-cost. Space complexity $O(bd)$ where d is longest path explored. Exponential time $O(b^d)$.
- **Simplified Memory-Bounded A***: A* search with memory cap. Once memory cap is hit, drop the high f-cost nodes from consideration. Optimality and completeness depend on whether it is achieveable within given memory limit. Time is exponential like A*, but memory is fixed.

4.5 Iterative Improvement Algorithms

- **Premise**: start with proposed state and modify. Used for problems where non-optimal solutions are easily generated.
- **Gradient Descent**: move in direction of decreasing cost. If stuck at local minimum, restart and see global optimum overtime. Does not maintain search tree.
- **Simulated Annealing**: add vibration to gradient descent denoted by T based on the goodness of solution E. Can lower vibration as the goodness increases. At every current state, take action if $\Delta E > 0$ or with probability $\exp(\Delta E/T)$ otherwise.



- **Local Beam Search**: Simulated Annealing that keeps track of k states instead of 1. Then, choose k best descendants, which is better than k independent solutions from SA.
- **Genetic Algorithm**: Start with k randomly generated states. Use fitness function to select best candidates based on probability of fitness. Combine best solutions into one and mutate. Replace least fit individuals with this new solution.
- **Genetic Algorithm Challenges**: need to determine fitness solution, how to merge parents.
- **Tabu Search**: guides local search algorithm to explore beyond a potentially stuck zone for a local minimum.

4.6 Adversary Search

- **Environment Properties**: stochastic? perfect information? turn-taking? zero-sum?
- **Perfect Information**: all players know everything, no random events, players do not need to have same moveset.
- **Payoff**: game outcome has payoffs represented as numbers.
- **Game Trees**: search tree with alternating players at each depth. Trees are too large to be complete, need winning strategy.
- **Evaluation Functions**: impossible to solve for games completely, need to cut off search at certain depth. Loses completeness; need to estimate cost of internal nodes using heuristic.
- **Minimax Search**: alternates between max and min optimal players. Uses DFS implementation.
- **Multiplayer Minimax**: circulate between players and maximize an element in corresponding tuples.
- **Minimax Search Properties**: complete if tree is finite, optimal against optimal opponent if tree is finite, exponential time $O(b^m)$, polynomial space $O(b * m)$.
- **Quiescent Search**: extend Minimax search to deeper depths in volatile and risky states. Mitigates the horizon effect.
- **Horizon Effect**: negative event that is inevitable but postponable, which hinders depth limited search performance.
- **Pruning**: eliminate branch of search tree from consideration.
- **Alpha-beta Pruning**: prune unnecessary nodes. α is best running choice for max. β is best running choice for min.
- **Alpha Pruning**: due to maximizing parent choice.
- **Beta Pruning**: due to minimizing parent choice.
- **Search Cutoff**: given constant time, we can stop at a fixed number of nodes or depth, or iterative deepening until time.
- **Expectimax Search**: instead of optimal minimizing player, use expected value at those depths.
- **Payoff Matrix**: representation of game theory choices.

Prisoner B

Prisoner A	Confess	Not Confess
Confess	-3,-3	0,-9
Not Confess	-9,0	-1,-1

- **Nash Equilibrium**: a state where neither player can benefit by changing strategies.
- **Mixed Strategy Nash Equilibrium**: an optimal strategy involving multiple choices.
- **Nash's Theorem**: Any finite game will have at least one Nash equilibrium possibly involving mixed strategies.

5 Knowledge and Reasoning

- **Knowledge**: methods for representing facts and how to act in world.
- **Reasoning**: methods for deducing additional information and a course of action to achieve goals.
- **Logic**: formal languages for representing information to support reasoning.
- **Syntax**: defines the sentences in the language; allowable symbols and rules for constructing grammatically correct sentences.
- **Semantics**: defines the meaning of sentences and their correspondence with facts. Sentences are booleans.
- **Proof Theory**: specifies the inference procedures that are sound.
- **Knowledge-Based Agent**: uses reasoning based on prior and acquired knowledge to achieve its goals.
- **Knowledge Base (KB)**: represents facts about the world (the agent's environment). Set of sentences in the KRL.

function KB-AGENT(*percept*) **returns** an *action*
  static: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t ← *t* + 1
  return *action*

- **Fact**: sentence in a particular knowledge representation language (KRL).
- **Inference Engine**: program that applies inference rules to knowledge to infer new knowledge.
- **Entailment**: A knowledge base (KB) entails (semantic dependent) sentences α is denoted by

$$KB \models \alpha$$

- **Inference Procedure**: i can derive α from KB (semantic independent):

$$KB \vdash \alpha$$

$$\frac{KB}{\alpha}$$

- **Soundness**: i is sound if whenever $KB \vdash \alpha$, it is also true that $KB \models \alpha$.
- **Completeness**: i is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash \alpha$.
- **Search Equivalence**: initial state = KB, operators = inference rules, goal test = KB contains S, node = state of KB.

5.1 Propositional Logic

- Symbols represent propositions. Sentences are generated by combining proposition symbols with Boolean connectives.
- **Syntax and Order of Precedence**: true, false, propositional symbols, $(), \neg, \wedge, \vee, \rightarrow, \leftrightarrow$.
- **Equivalence**:

$$\alpha \equiv \beta \iff \alpha \models \beta \ \& \ \beta \models \alpha$$

- **Satisfiable**: some interpretations of sentences are true and others false.
- **Unsatistifiable, Contradictory**: interpretations of sentences are always false. (Negation of tautology.)
- **Tautology, Valid**: interpretations of sentences are always true.
- **Complexity**: validity and satisfiability are NP-Complete. Checks for satisfiably are efficient iff same for validity.
- **Semantics**: interpreted by constructing a truth table with all 2^n combinations where n is the number of propositions.
- **Inference Rules**:

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \quad \neg\beta}{\alpha}$$

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

5.2 First Order Logic

- **Definition**: extension to propositional logic in which quantifiers can bind variables in sentences.
- **Syntax**: objects, predicates, functions, variables, logical connectives, quantifiers, and equivalence.
- **Objects**: constants.
- **Predicates**: true or false statements about objects.
- **Functions**: return objects.
- **Variables**: unspecified objects.
- **Atomic Sentence**: a predicate applied to some terms.
- **Logical Sentence**: atomic sentences connected by connectives.
- **Quantified Sentence**: sentences with quantified variables.
- **Universal Quantifier**: for all, denoted $\forall$.

- **Existential Quantifier**: for all, denoted $\exists$.
- **Variable Scope**: local until redefined more locally.
- Note: $\forall x \exists y$ is not equivalent to $\exists x \forall y$.
- Note: $\forall x(S)$ is equivalent to $\neg \exists x(\neg S)$ and $\exists x(S)$ is equivalent to $\neg \forall x(\neg S)$.
- **Assertions**: Denoted $TELL(KB, Sentence)$ adds sentence to KB.
- **Queries**: Denoted $ASK(KB, Sentence)$ returns whether KB entails the sentence, conditioned on a substitution binding.
- **Refutation**: To prove KB entails P, assume $\neg P$ and show a contradiction.

$$(KB \wedge \neg P \rightarrow False) \equiv (KB \rightarrow P)$$

- **Substitution**: Denoted $SUBST(\theta, Sentence)$ where θ is a binding list. Applies substitution and returns function.
- **Binding List**: List with elements $\frac{var}{grounding}$ where the grounding term has no variables.
- **Additional Inference Rules**:
  - Universal Instantiation

$$\frac{\forall v \quad \alpha}{SUBST(\{v/g\}, \alpha)} \qquad g - \text{ground term}$$

  - Existential Instantiation

$$\frac{\exists v \quad \alpha}{SUBST(\{v/k\}, \alpha)} \qquad k - \text{constant that does not appear elsewhere in the knowledge base}$$

  - Existential Introduction

$$\frac{\alpha}{\exists v \; SUBST(\{g/v\}, \alpha)} \qquad v - \text{variable not in } \alpha, \; g - \text{ground term in } \alpha$$

- **Generalized Modus Ponens**: for atomic sentences p_i, p_i', q where there is a substitution θ s.t. $SUBST(\theta, p_i') = SUBST(\theta, p_i)$ for all i, then,

$$\frac{(p_1', ..., p_n'), (p_1 \wedge ... \wedge p_n) \rightarrow q}{SUBST(\theta, q)}$$

- GMP is efficient with combining steps and purposeful with substitutions.
- GMP is the only inference rule necessary for horn clauses, and is only complete for horn clauses.
- **Unification**: returns a substitution θ for atomic sentences p, q that makes p, q equivalent.
- **Forward Chaining**: data driven process to derive end result.
- **Horn Clause**: subset of FOL sentences in the form

$$\forall x(P_1(x) \wedge ...P_n(x)) \rightarrow Q(x)$$

- **Backwards Chaining**: start with goal, find implications that prove goal, and prove parts needs to solve that proof.
- **Choice of Proof**: branching factor, number of goal states, justification of user.
- **Generalized Resolution**: for literals p_i, q_i where $UNIFY(p_j, \neg q_k) = \theta$,

$$\frac{(p_1 \vee ... \vee p_m), (q_1 \vee ... \vee q_n)}{SUBST(\theta, p_1 \vee ... \vee p_{j-1} \vee pj + 1 \vee ... \vee p_m \vee q_1 \vee ... \vee p_{k-1} \vee pk + 1 \vee ... \vee q_n)}$$

- **Godel's Completness Theorem**: FOL entailment is semi-decidable (by GR). (Only entailed sentences can be decided.)
- **Godel's Incompleteness Theorem**: Arithmetic cannot be axiomatized.

5.3 Conjunctive Normal Form and Implicative Normal Form

- **Conjunctive Normal Form**: Or statements joined by ands.
- **Implicative Normal Form**: Implications with a conjunction of atoms on the left and a disjunction of atoms on the right.

- **CNF Conversion Procedure**: replace equivalences with implications, eliminate implications, distribute negations, [FOL: standardize variables, move quantifiers left, skolemize, drop universal quantifiers] distribute ands over ors, flatten nesting.
- **INF Conversion Procedure**: convert to CNF, convert disjunctions to implications.
- Note: CNF and INF are logically equivalent.
- **Skolemization**: remove existential quantifiers. Substitute with constant if existential quantifier on outside. If on inside of a universal quantifier, introduce a Skolem function $H(x)$ that returns a constant based on the universal variable x.

6 Markov Decision Processes

- **Definition**: given a set of states $s \in S$, set of actions $a \in A$, transition function $T(s, a, s')$ with probability $s \to s'$ with a, reward function $R(s, a, s')$, start state, and terminal state.
- **Markov**: given the present state, the future and the past are independent.
- **Policy**: π gives an action for each state. Optimal policies maximizes expected utility. Explicit policies define a reflex agent.
- **MDP Search Trees**: each MDP state projects an expectimax search tree with start node s and actions as branches for first depth to Q-states, with each Q-state having transitions as branches to state s'.
- **Stationary Preferences**: assume stationary preferences, which defines utilities in two ways.
- **Additive Utility**: sum of all rewards.
- **Discounted Utility**: sum of all rewards subject to a decay factor γ^i over time step i.
- **State Values, Bellman Equations**: compute the (expectimax) value of a state.

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

- **Value Iteration**: start with all state values $V = 0$. Then, calculate expectimax of each state ($O(S^2A)$) and repeat until convergence. This uses policy extraction since policy is implied by max operation.
- **Value Iteration Theorem**: will converge to unique optimal values since approximation gets refined toward optimal values.
- **Policy Evaluation with Fixed Policies**: reduce complexity of each value iteration to $O(S^2)$ by fixing one action per state. Also, since there is no max operation, this is now linearly solvable.

$$V^*(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^*(s')]$$

- **Problems with Value Iteration**: slow, action rarely changes, policy converges before values.
- **Policy Iteration**: evaluate utilities for some fixed policy until convergence, then update policy action with one-step look ahead. Repeat until convergence. Optimal and can converge faster in correct conditions.
- **Policy Update**:

$$\pi_{i+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$