Cheat Sheet

AI

Turing test

Created to answer the question "Is it a human or a computer I am communicating with?". Does not test intelligence.

IQ test

An IQ test measures what humans think intelligent humans should be good at - not intelligence.

Weak and strong AI

- A weak AI is a machine that can act as if it is intelligent (simulated thinking). An example is a self-driving car.
- A strong AI is a machine that is actually thinking (real thinking). Difficult to prove. We're not there yet.

Simple Reflex Agent

Select actions based on the current perception and does not care about the percept history. The simplest of the agents. Does not care for future effects of it's actions. Implemented using condition-action. For example

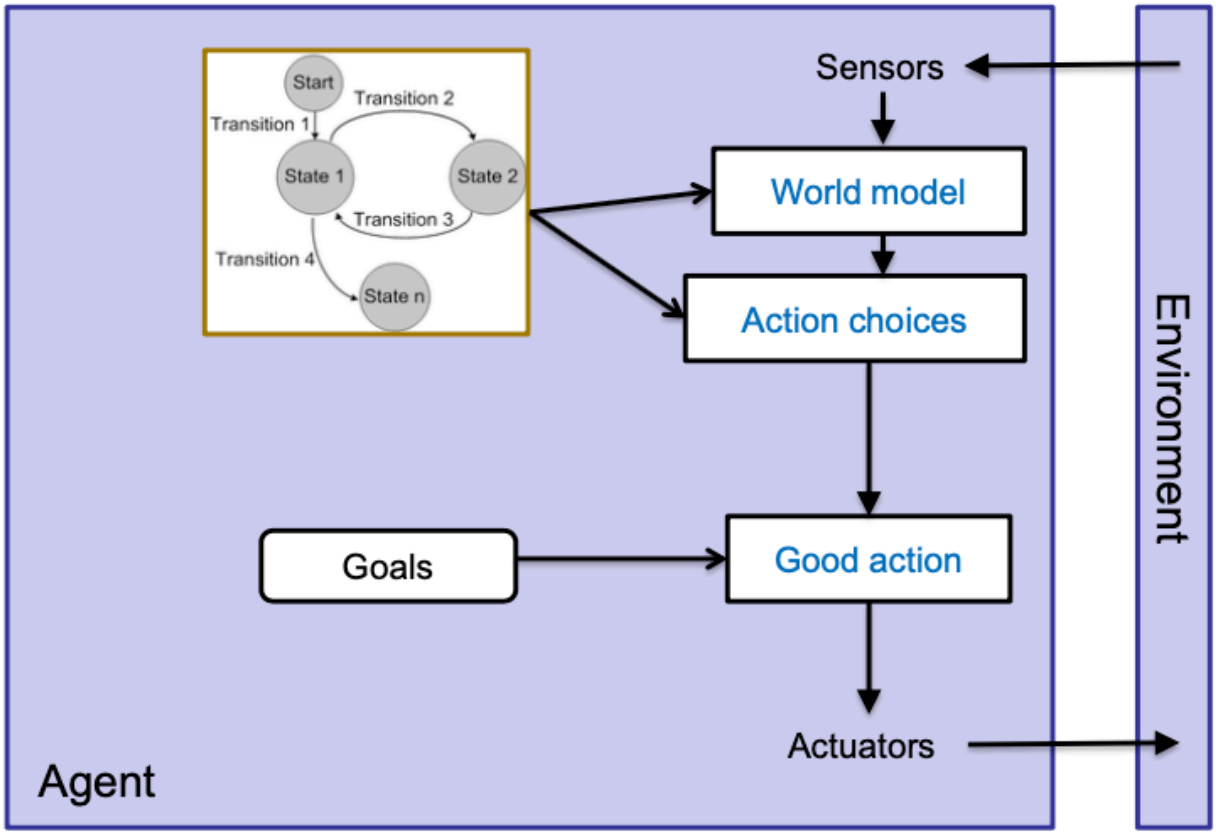
- if dirty then clean
- if clean then move

Model-based reflex agent

Has and maintains an internal model about the environment. The model is based on the percepts history. The agent also keeps track about how the world evolves and the effect the agents action might have on the environment. An example on how it might think is:

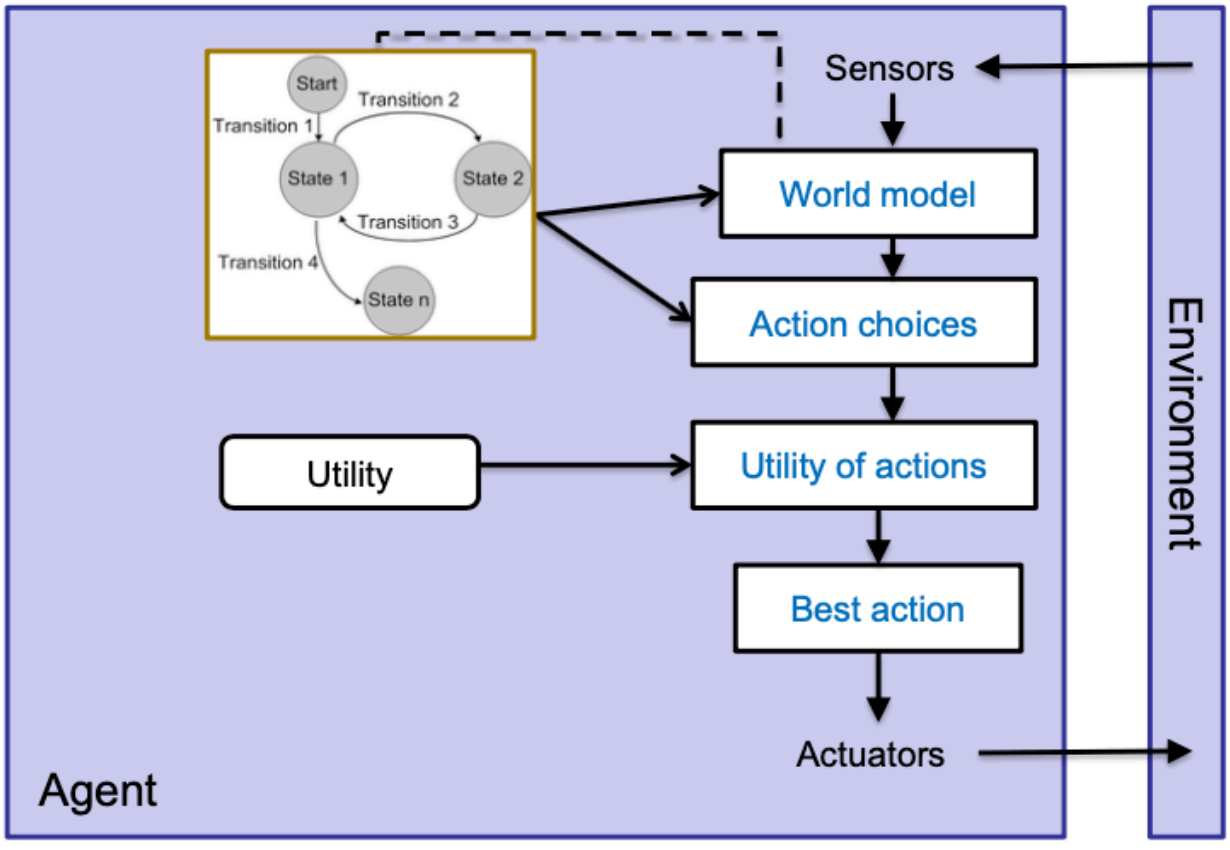
- Where would i be in the world if i turn right in this crossing and drive for 5km?

Goal-based agent



- Executes actions that leads it closer to specific goals
- Uses planning techniques

Utility-based agent



- Uses an utility-function which states how desirable a state is
- The agent chooses an action that leads to the state with the highest utility
- More flexible than goal-based agents

Learning agent

Is able to learn from it's percepts. For parts:

- Learning element: Responsible for making improvements.
- Performance: Responsible for selecting actions.
- Critic: Gives feedback on the agents action and how it's doing.
- Problem generator: Responsible for generating actions which will lead to future new experiences and new information.

The agents problem lies in having to balance exploitation(Doing the things which will teach me new things) and exploitation(Doing what is best).

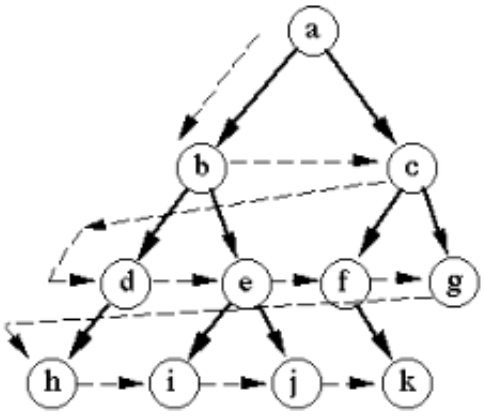
Environments

Fully observable environments are easier to deal with, but often not possible. An example is chess. On the other hand, partially observable environments are more common. An example is self-driving cars - there are sensors but the car cannot know what another driver is thinking (poker is another example).

If the next state of an environment only depends on the current state and the agent's action - the environment is deterministic. Easier for agents - but often not possible in the real world. Driving a car is stochastic - a lot of things can happen in traffic.

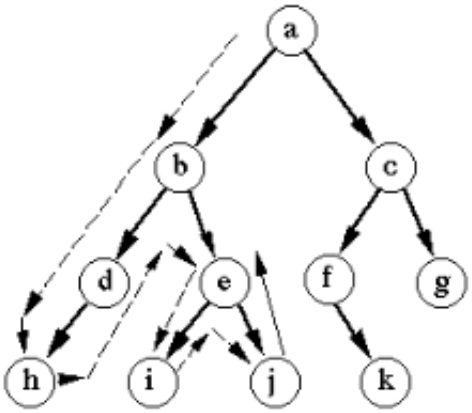
Algorithms

Breadth-first search (BFS)



- Search level by level in a tree
- Is optimal if the costs between nodes are identical
- Is complete if the branching factor is finite
- Slow and memory intensive

Depth-first search (DFS)

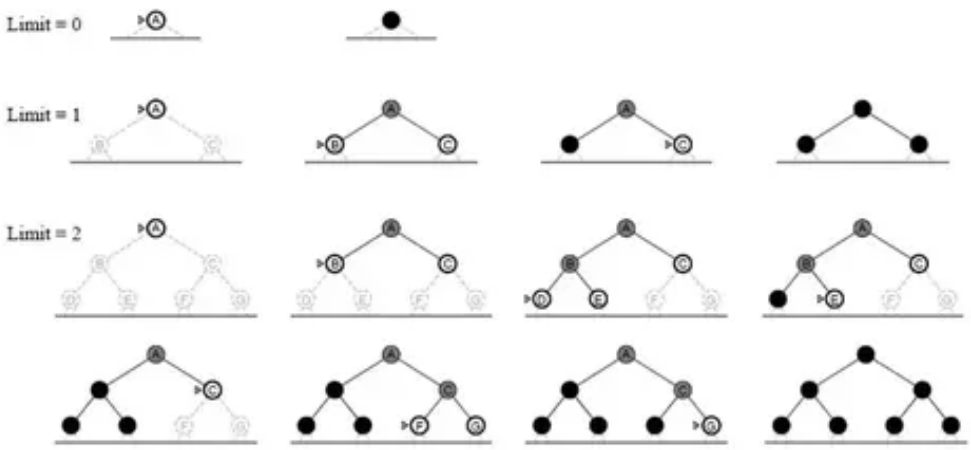


- Search branch by branch
- Is complete if the depth of the tree is finite
- Optimality cannot be guaranteed if there are two or more goal nodes
- Very fast for well-balanced trees
- Very low memory consumption

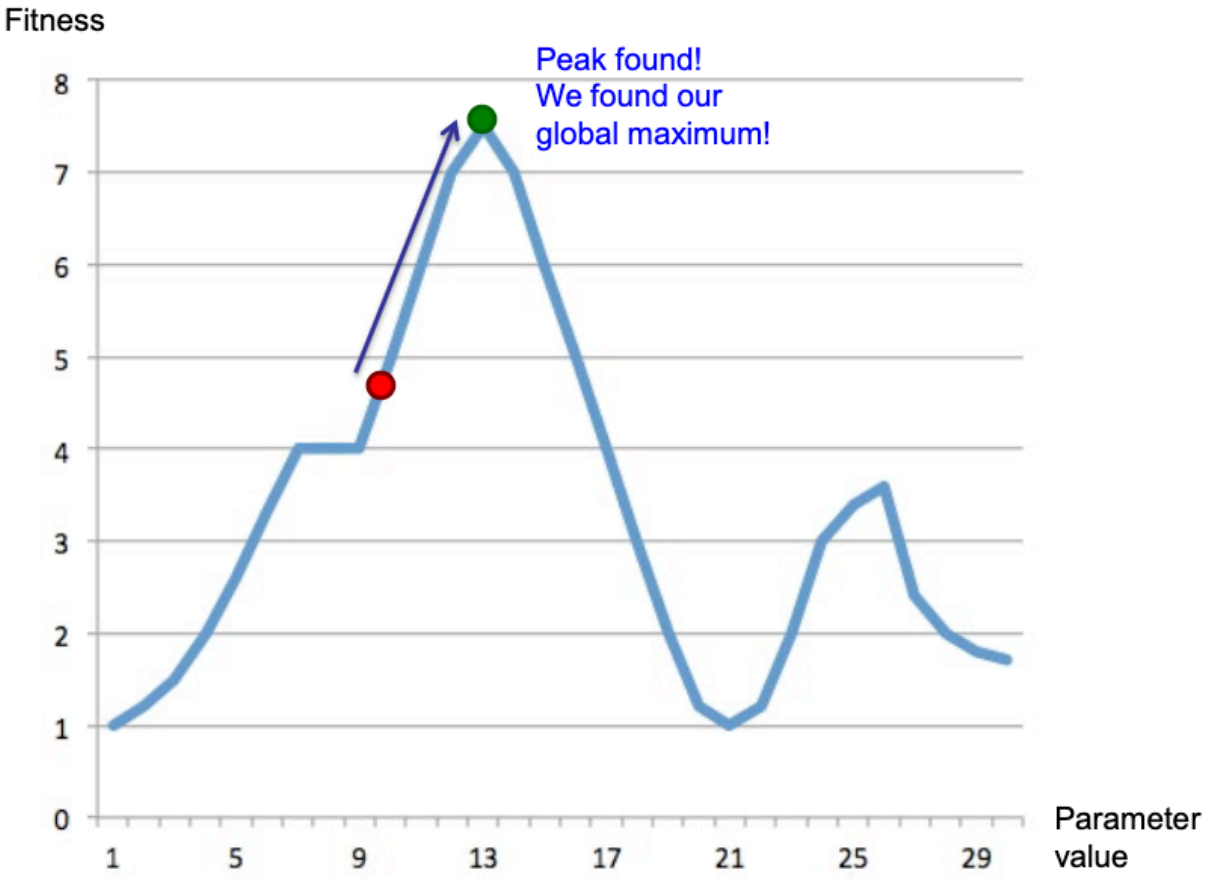
Iterative deepening

Combines the strengths of BFS and DFS. Relatively fast with a low memory requirement, optimal if the branching factor is finite, optimal if the costs between nodes are identical.

1. Perform DFS to a pre-defined level l
2. If no solution was found, increase l and perform a new serach
3. Repeat step 2



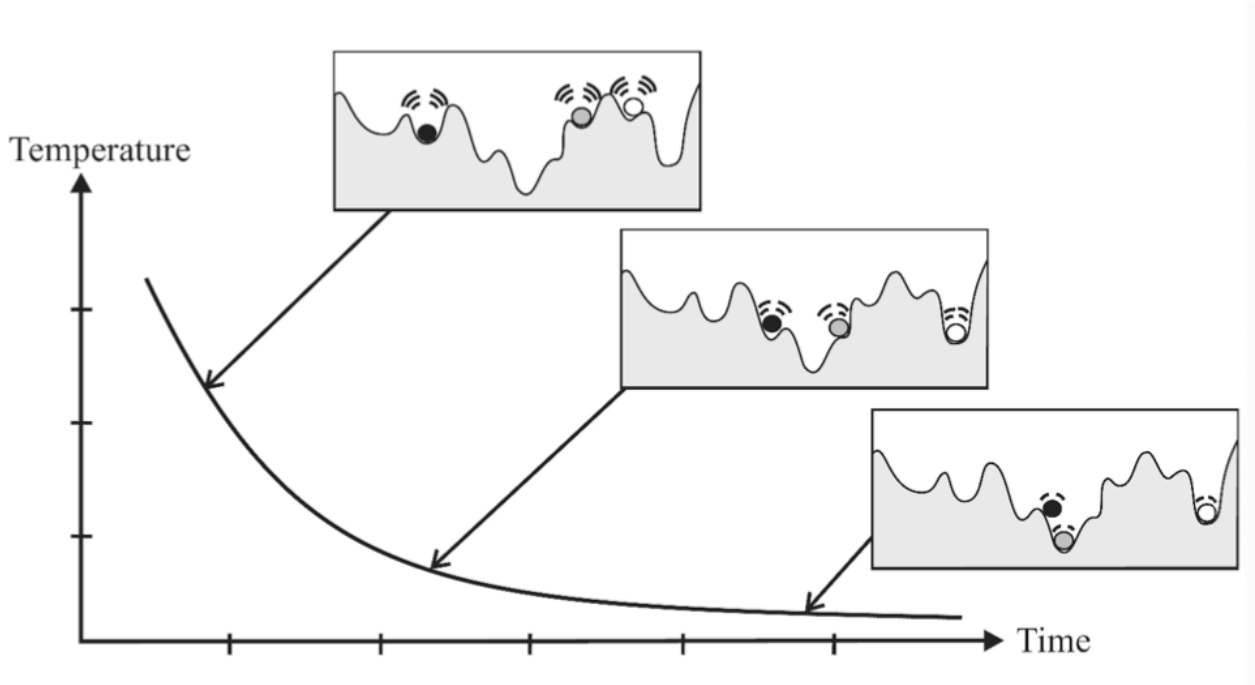
Hill Climbing



1. Randomly select a start value between min and max
2. Find the fitness for the start value and its neighbors
3. Move in the direction of increasing fitness values (uphill)
4. Stop when the peak is reached

The issue with Hill Climbing is that if we start at, say, 21 in the graph above - we will find a local maxima, not the global.

Simulated Annealing



Combines Hill Climbing with a random walk, reducing the possibility of getting stuck in local maxima.

1. Randomly select a new position
2. If the new position is better (uphill), it is always accepted
3. If it is worse (downhill), it is accepted with some probability less than 1

Genetic Algorithms

Uses techniques learned from biological evolution. If the evaluation is time consuming, it might take a long time to evaluate the entire population through several generations. There is randomness in the system which may cause the solution to never converge.

1. Represent the problem as a string of parameters - the DNA / optimization parameters
2. Generate the initial population - random DNA
3. Compute fitness - how well each individual has solved the task
4. Perform a selection from the population - such as roulette, tournament
5. Perform a crossover - "mate" individuals, sharing their DNA
6. Mutate the population - change some random genes. Important to maintain genetic diversity
7. Compute fitness
8. If the population has not converged, repeat from step 4

Min-Conflicts

Used to schedule things (such as the Hubble Telescope).

- Roughly independent of size
- Suitable for very hard problems
- A bad initial placement can increase the search time (drawback)

1. Select random value for each variable
2. Select a random variable to update

3. Update the variable to the value that causes the least number of conflicts
4. Continue from step 3 until we are done or we reach the maximum number of iterations

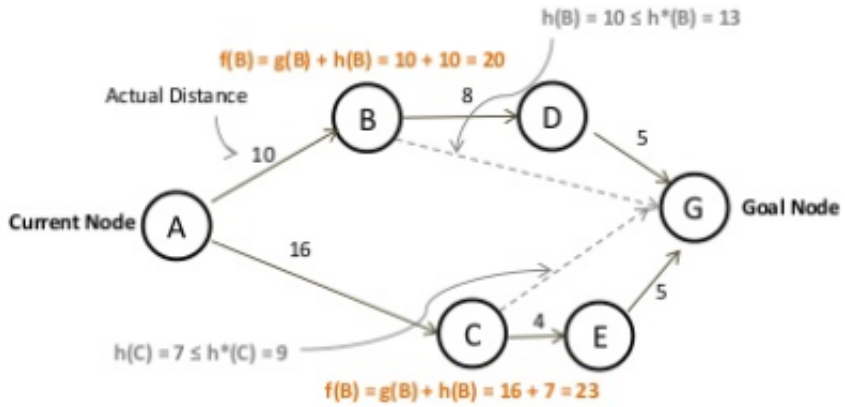
Backtracking Search

A depth-first search with backtracking for constraint checks.

Decision Trees

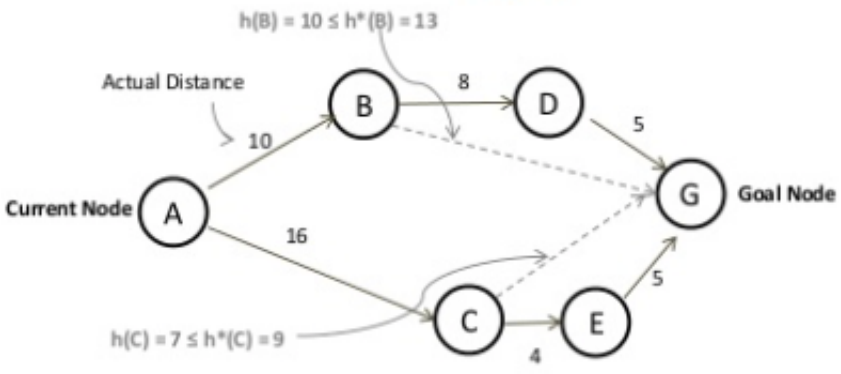
- Each node in the tree tests an attribute
- Each leaf represents a class
- Suitable for nominal attributes - temp < 15 = cold etc.
- Fast at learning and classification
- Human-readable
- Bad at handling inconsistent data
- Cannot learn all concepts

A*



A* is an implementation of greedy search which also uses the cost of each path in addition to the heuristic.

Greedy search



A type of informed search (knowledge about the heuristic such as distance to goal). Always chooses the step with the lower heuristic.

Learning-based algorithms

Supervised learning

- Learning with an instructor
- We have some known data
- Learn to handle known data and assume a general concept can be taught - to apply on unknown data

Unsupervised learning

- Learning without an instructor
- Try something to see if it works
- Use a utility function to calculate how well it worked

Overfitting

Overfitting is when a model begins to describe random errors in the data instead of an actual relationship. Usually occurs when the model is too complex. Reduces its generalizability outside the original dataset.

Logic

Propositional Logic

- One limitation is that positions are absolute
- Cannot handle dynamic environments
- Assumes the world contains facts

Example: *The sun is hot and the ocean is wet*:

- $p =$The sun is hot
- $q =$ The ocean is wet
- $p \wedge q$

First-Order Logic

- More expressive than propositional logic
- Not limited to facts - also has objects and relationships
- Also supports temporal logic (facts true only some of the time)

There are three types of symbols:

- Constant symbols (objects) - people, houses, numbers
- Predicate symbols (relations) - red, round, brother of
- Function symbols (functions) - father of, best friend, plus

Example: *Every student takes both math and chemistry*:

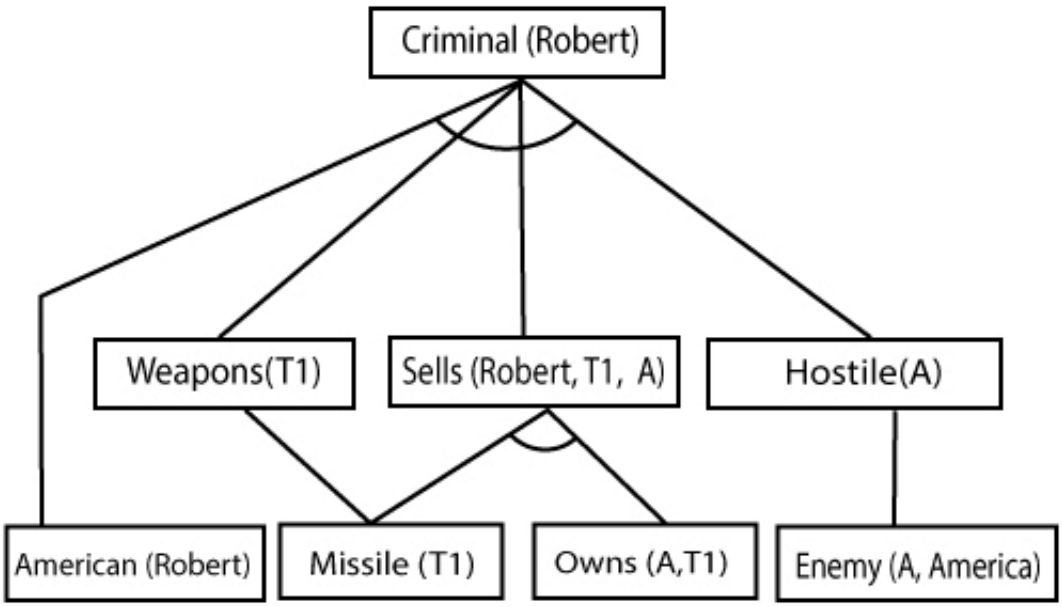
$$\forall x : \text{Student}(x) \Rightarrow \text{Takes}(x, \text{Math}) \wedge \text{Takes}(x, \text{Chemistry})$$

Knowledge base

Resolution Algorithm

The main idea is to prove by contradiction, that $\Phi \wedge \neg(\Phi)$ is unsatisfiable.

Backward and forward chaining



The facts are stated at the bottom row with conjunctions above. For example, from this we can read that $\text{Missile}(T_1) \wedge \text{Owns}(A, T_1) \Rightarrow \text{Sells}(\text{Robert}, T_1, A)$.

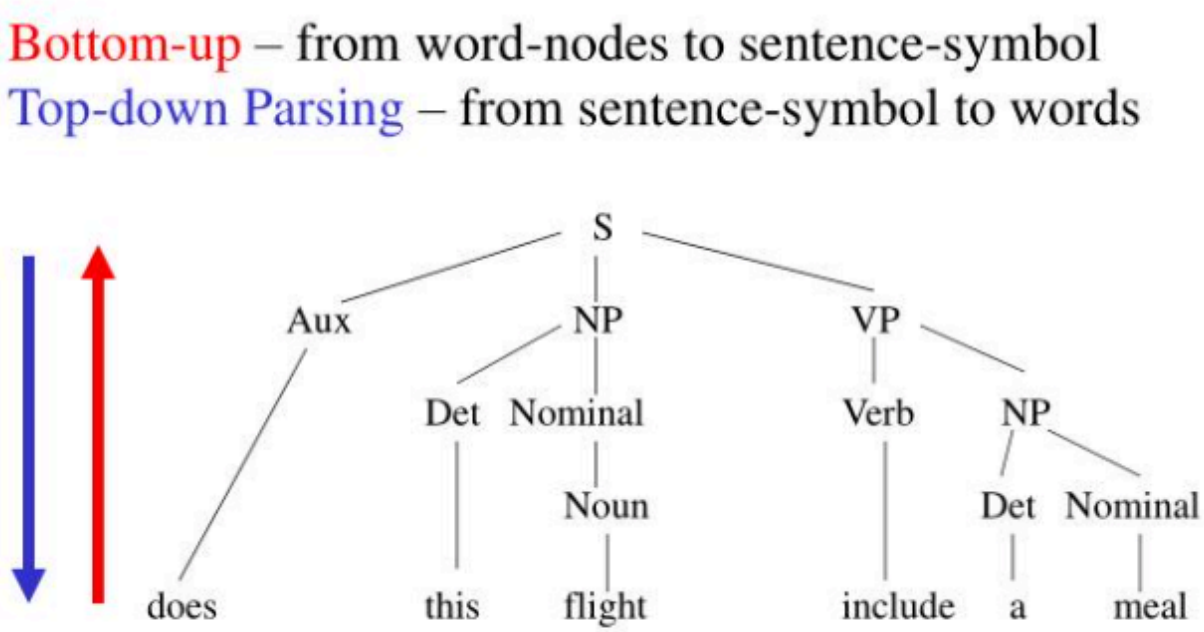
Conjunctive Normal Form

1. Eliminate implications and equivalences
    1. Replace $P \Rightarrow Q$ with $\neg P \vee Q$
    2. Replace $P \Leftrightarrow Q$ with $(P \vee \neg Q) \wedge (\neg P \vee Q)$
2. Replace all and with or: $A \wedge B \rightarrow \neg(A \vee B)$

Example: from first-order logic to conjunctive normal form:

$$\forall y : \text{King}(y) \wedge \text{Greedy}(y) \Rightarrow \text{Evil}(y)$$
$$\text{to}$$
$$\neg\text{King}(y) \vee \neg\text{Greedy}(y) \vee \text{Evil}(y)$$

NLP



Top-down parsing

Can be efficient with good heuristics. Often gets stuck in infinite loops.

- Start with the sentence symbol, find further symbols and move onto words

Bottom-up parsing

Can be efficient with good heuristics. Can generate partial parsers - searching irrelevant portions.

- Find the symbol for each word, pair the symbols and build up the sentence