CS 161(Fundamentals of Artificial Intelligence) Notes

Lecture Notes

10/2 - Week 1

- AI is concerned with creating intelligent agents that can:
    - Perceive
    - Understand
    - Predict
    - Manipulate
    - Learn
- Knowledge representation
    - How do we represent knowledge?
        - Difference between facts and beliefs
    - What knowledge is relevant?
    - How do we acquire knowledge?
- Reasoning
    - How can we formalize the reasoning process?
        - Deduction - What is implied given knowledge base
        - Belief revision - What prior beliefs to give up based on new info
        - Causality - What is cause of the event
- Trouble with Natural Language Understanding
    - Syntactic ambiguity
    - Semantic ambiguity
    - Pragmatic ambiguity
- Classic approach to AI - Provide system with knowledge of the world and create rules
- Modern approach to AI - Rely on corpus and use ML

10/4 - Week 1

- In LISP, things are either atoms (3, x, etc) or lists.
- We manipulate lists through accessors like car and cdr.
- We create lists through constructors like cons.
- NULL is a boolean function that determines whether a list is empty, and NIL is the empty list
- Equal is a boolean function that determines whether two lists are the same.

10/6 - Week 1

- You can either have atoms or lists in LISP
- The only thing that gets evaluated to false is NIL
- OR returns the first non-nill

- List(r t) makes one big list composed of two elements, the two lists r and t, while append (r t) will actually combine the elements of r and t into a single list.

10/9 - Week 2

- Problem solving and search
    - Problem formulation (Creating the tree that describes states)
        - Need to consider the initial state and the goal state. An action is something you apply to a state to transition to another state.
        - Important to create a successor function that takes in the current state, and determines a list of available successor states.
        - Also need a function that checks whether or not the current state is actually the goal state and the problem is basically solved.
        - You can think of problem formulation having 5 components: Choosing how to represent states (primarily initial/goal states), choosings actions, creating your successor function (transition model), creating a goal state test, and choose some metric for your cost.
    - Search strategy (Finding the right path through that tree or choosing which state to expand first)
        - Goal is to operate on a search tree to find a particular path
        - Systematic
        - Local
    - Search engine will take in the above 2, and produce the solution.
- In the 8 tile problem, the hard parts are the large state space (9!), the large branching factor, and length of the solution.
- In the Missionaries and Cannibals problem, the initial state is 3M, 3C on the left side and then the goal state is 3M, 3C on the right. There are 5 possible actions, but some are not applicable in certain situations. The results of some actions would result in illegal states (where there are more cannibals than missionaries on each side).
- In 8 queen problem, the goal is to put the 8 queens on the board such that none of them are attacking each other (not in same row, col, or diagonal).
    - Initial state is the empty board, the actions can be putting a queen on the board (64 options in the beginning), and the goal state is tough to find.
    - We instead say that we create a function that takes in the current state and just determines whether or not the conditions of the goal state is satisfied.
        - This type of test is a goal state predicate.
    - Interesting note is that in this problem, the answer will always be at depth 8. This is important for us to know whenever we're choosing the proper search strategy.

10/11 - Week 2

- Problem formulation
    - IS
    - GS test

    - Successor function
- Search strategies
    - Complete/Systematic - Guaranteed to find a solution.
        - Informed - Use more information than just the problem formulation components.
        - Uninformed
    - Incomplete/Local - Not guaranteed
- Leaves of the search tree are called the fringe of the frontier. These are the collection of nodes waiting to be expanded.
- Expansion is the process of selecting a node from the fringe, and then expand that node.
    - 1) Check if it is the goal
    - 2) If not, generate children.
- The search strategy is really about choosing what node to pick first, and every node choice afterwards.
    - Informed search strategies use heuristics (some sort of additional information or constraints) to help with those choices.
    - Uninformed doesn't have any information about the path cost or number of steps.
- Evaluating an algorithm
    - Completeness = If the solution exists, is the algorithm guaranteed to find it
    - Optimality = Find the shortest solution
    - Time complexity, Space complexity
- Completeness and optimality are concerned with the quality of the path while the other 2 are more concerned with actual performance.
- In Breadth First search, the search strategy will involve looking at the depth of the nodes in the tree.
- **You only detect the GS during expansion, and not generation**.
- BFS
    - BFS is complete because it examines all nodes at each layer before moving on to the next level. Therefore, we know that no node will be missed.
        - It is also optimal (only if cost is 1) because of pretty much the same reason.
    - For BFS, when thinking about complexity, we need to know branching factor (b), the height of the tree (m), and the height of the solution (d).
    - Space complexity (basically how many nodes you'd have to count): 1 (first level) + b (2nd level) + b^2 …. b^d = O(b^d)
        - Thus, good if you have shallow but not deep solutions.
- DFS
    - DFS is not complete because there's a possibility it could miss the correct node because it keeps going down some rabbit hole path. However, if we can assume that the tree has some finite depth m, then it is complete.
    - DFS is not optimal (b/c it's not complete), but even if m is finite, still not optimal because it can miss on the shortest path.
    - Time Complexity: O(b^m)

    - Space Complexity: O(bm) because you only have to keep track of one “thread” at a time.
    - Limited depth DFS is useful if you want to make sure your algorithm completes, but might not get to your solution.
- Iterative deepening is a good combo and is doing DFS with different cap levels. Ex) During the first run, cap at some value L (Start L at 0). If you find a solution, great. But if you don’t find a solution, increase L by 1.
    - It is complete, optimal, and the same space complexity as DFS
        - Time complexity is doubled though. O(b^d * (b/b-1)^2)
        - The squared term is the new addition.
        - It won’t be that bad for large values of B, but could be problematic for smaller values.
    - Make sure you keep in mind that iterative deepening will check the first node twice
        - Draw out the separate trees that we’re going to run DFS on.
    - Problem is that it could be doing repeated work.

10/13 - Week 2

- Search problem consists of
    - A state space
    - A successor function
    - A start state and a goal state
- A solution is a sequence of actions that transforms start state to the goal state.

10/16 - Week 3

- Uninformed search strategies are ones that don’t use any info besides what is given in the problem formulation.
- A* is an informed search strategy and it involves an external heuristic.
- **Uniform cost search** is a generalization of BFS.
    - It attaches weights to edges
    - After expanding the start node, choose to expand the node with the lowest cost.
    - Complete and optimal because it visits the nodes with the lowest g(n) first.
        - g(n) is pretty much the cost it takes to get to n
    - When compared to BFS, we have the same time and space complexity since uniform is basically the same thing as BFS except with non-uniform costs.
    - Test for goal state during expansion
- When humans look at uniform cost search, we are using certain heuristics to make our decision.
    - Specifically, we’re using the shortest distance heuristic or best first or greedy.
        - “We’re going to the node that is closest to the goal”
        - “Among different choices, we’re asking ourselves what is closest to the goal”.

- With that greedy algorithm (and with the straight line distance between nodes. Label is h(n)), we create a search tree that is a little backwards. We label each node with the straight line distance to the goal and choose the node with the smallest distance.
    - This type of straight line distance is extra information that doesn't come with the problem formulation.
    - Space and time complexity is b^m, but also depends on what h is.
    - Not optimal or complete
    - This basically tells us that heuristics that we use can be good or bad and can affect the overall optimality and completeness.
- Plus side of UCS is that we get to the solution but the negative is that it does too much work
- Plus side of Greedy is that we zoom in on the solution very quickly, but we're not 100% that the answer will be correct.
- When formulating search strategies, it's important to just use local information (not have to look back at the parent nodes info)
- g(n) simply tells you have much you've paid so far. It's not looking in the future, it's looking in the past. h(n) is the opposite, it's looking at where you're going with no real regard for how much you have paid.
- f(n) = g(n) + h(n)
    - g(n) is how much I've paid to get to n. It is an **actual cost**
    - h(n) is how much I expect to pay to get from n to the goal state. It is an **estimate** of how much it will cost to get to the goal.
        - They are lower bounded estimates. You might pay more but you won't pay less.
    - f(n) is therefore a lower bound on the price you will pay with going from initial to goal state where you pass through n.
- It is important for h(n) to not overestimate. It's okay to be lower than the actual cost, but not greater.
- As long as h(n) doesn't overestimate (this property is called being admissible), A* will always be optimal.
    - h(n) is an admissible heuristic.
- A heuristic is consistent if h(n) <= C(n,p) + h(p) for all nodes p that are successors to n
- The performance of A* is sensitive to how you choose h.
- For the 8 puzzle problem, the heuristic might be the number of tiles that are not in their final position or it could be the summation of the number of moves that each number has to go through to go from initial to goal.
    - The 2nd heuristic is better since it's almost always higher but yet it still doesn't overestimate the number of moves actually required.
- For a heuristic to work from a coding POV, you need a function that takes in the current state and the goal state, and returns an estimate depending on h.

<u>**10/18 - Week 3**</u>

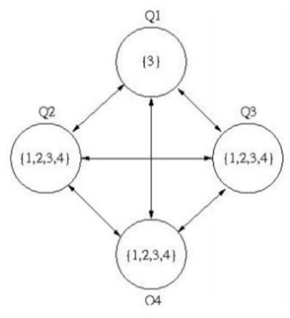
- The effective branching factors of search trees with different heuristics will be different.
- Performance of A* depends heavily on the strength of the heuristic.
- Constraint Satisfaction Problem
    - Examples are SAT and 8 Queens
- Important to determine the variables, values, and constraints in your CSP problems.
    - Similar to search problems where you have your problem formulation.
    - Here you really have to identify what variables you have, what values they can take, and what constraints they can't violate.
- The domain of a variable is the set of values that they can take on.
- You can also create a constraint graph to visualize how the constraints interact with the variables.
    - If the graph is a tree, then you can solve the problem in polynomial time.
- In CSP problems, you also have the typical initial states and goal states, and successor functions.
- DFS is a good choice for CSP problems because we know that the search tree is finite, we know that the solution is at the fringe, and we don't have a notion of optimality since we just want the first solution.
- DFS is called backtrack search with CSP problems.
    - The idea is that you start with DFS but then you backtrack if you know that one of the constraints has been violated.
- The 4 things that you can change/alter about backtrack is
    - Variable order (order in which you choose variables to assign values to in the tree)
        - One heuristic is the most constrained variable (the one that has the least number of legal values remaining)
    - Value order (order in which you assign values in the search tree)
        - Least constraining value (Value that rules out the fewest variables out of the remaining variables)
    - Forward checking
        - Assign variables and then eliminate possible values for neighbors based on the constraints.
    - Arc consistency

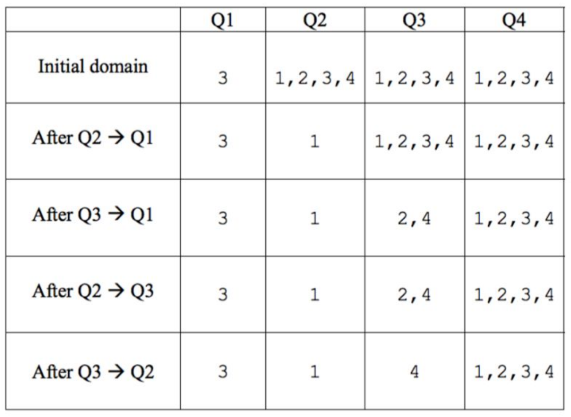
<u>10/23 - Week 4</u>

- CSP is assigning values to variables and then making sure constraints are accounted for.
- Full variable assignments are at the very last level, and so DFS would be the best way to get to the possible solutions. With DFS, you can also get to a point where you know that one of the constraints has been violated, and then you can do backtrack search.
    - Backtrack search is a variation of DFS.

- Backtrack search is when you insert a test before the successor generation step where you check whether any constraint has been violated based on the variable assignments you have made up to this point.
    - The problem with backtrack search is that it only finds the problem when it gets to that particular node. If the first 6 variables cause the 8th variable assignment to cause a violation no matter what, the 7th variable will still be assigned.
- Forward checking looks ahead to detect unsolvability. Each time a variable is assigned, forward checking deletes from the domains of the non-assigned variables all the values that conflict with the assignments so far.
- As you expand each node, you can do a check to make sure that the set of constraints has not been violated for the variables that you have assigned so far.
    - A bit of a trade-off because of the computation required to make those checks at every single node choice might not be worth it.
- CSP
    - Variable/Value Ordering
    - Detecting failure early
        - Backtrack Search
        - Arc consistency
            - Forward checking
    - Structure based CSP
- A state is arc consistent if every variable has a value in its domain that is consistent with each of the constraints in the problem.
- Arc consistency example

Q. Fill in the table below with the candidate values of each queen after each of the following steps of applying the arc consistency algorithm to the figure. An arc "x -> y" is consistent if for each value of x there is some value of y that is consistent with it.





	Q1	Q2	Q3	Q4
Initial domain	3	1,2,3,4	1,2,3,4	1,2,3,4
After Q2 → Q1	3	1	1,2,3,4	1,2,3,4
After Q3 → Q1	3	1	2,4	1,2,3,4
After Q2 → Q3	3	1	2,4	1,2,3,4
After Q3 → Q2	3	1	4	1,2,3,4

- Structure based CSP is where you can represent constraints as edges between variables. This graph structure can give us clues and intuitions as to how to solve it.
    - When the graph is a tree, we can solve the problem in linear time.

    - If it’s not, the extent of its difficulty is tied to how close it is to a tree.
        - We need a measure to describe how close something is to a tree. Tree width is an example.
    - Cut set conditioning: You can simply the tree graph by set a variable to a value, and then remove that variable until you get a tree, at which point you input that to your linear solver, and then check whether or not a solution is found. If yes, then you’re done. If no, then you try the other value assignments for that variable.
- Normally, CSPs are O(n * d^w) where n is the # of variables, d is the # of values for those variables, and w is the tree width.
- Local search in CSPs is concerned with looking at a single state (full variable assignment) at a time. We’re also looking at neighbors (subset of all the next states possible)
    - Contrast this with CSPs where we were building a tree that had partial variable assignments.
- When choosing which neighbor to explore next, you want to pick the neighbor with the smallest number of violations.
- With local search, there is a possibility that you could just oscillate around and never actually find the answer. The initialization of that first state is extremely important
    - In these cases, you can do a restart initialization.

  Another approach is to look at how bad your next position is + the time you’ve already spent searching.
- Local search is incomplete, while building up the search tree is complete.
- Minimax games can be formulated by thinking about the states (specifically initial state), the operators/actions, a terminal test for when the game is over, and a utility function to determine who wins.

<u>**10/25 - Week 4**</u>

- Knowledge Representation and Reasoning
    - Logical Reasoning
        - Propositional/Boolean
        - First order logic
    - Reasoning under uncertainty
- Deduction is when you have a knowledge base (set of facts about the game), make new observations, and then drawing further conclusions based on those two sets of information.
- Logic
    - Syntax
    - Semantics
    - Inference algorithms
- Propositional Logic
    - Syntax means defining the variables.
        - P1, p2, ….pn

    - Semantics refers to the ways we use these variables and the logical equivalence of two statements.
        - Pi and alpha ^ beta are all examples of sentences
        - Or, And, Equivalent, Negation are all valid operations
- Alpha implies Beta = not Alpha OR Beta
- Alpha implies Beta = not Beta implies Alpha
- Variables to truth values is a truth assignment or a world
    - You can say a truth statement is true/false in a particular world
    - You can say this particular world has this truth assignment
- Lets say B1 = LV or something else
    - M(B1) = 1...9, 11, 13, 15
- Alpha is equivalent to Beta is equal to M(alpha) = M(beta)
- Alpha is inconsistent M(Alpha) = {}
- Alpha is consistent M(Alpha) != {}
- Alpha is a tautology is where M(Alpha) = W
    - True at every possible world
- Alpha and Beta are mutually exclusive if M(Alpha) Intersect M(Beta) = {}
    - You can also say M(Alpha Intersect Beta) = {} or inconsistent
    - No world that makes both true at the same time
- Alpha implies Beta means that M(Alpha) is a subset of M(Beta)
    - If a world makes Alpha true, that it makes Beta true
    - You can think of Alpha as having more information, as being the constraining factor, as being the smaller set.
- An inference procedure is complete if it can find a proof for any sentence that is entailed.
- An inference rule is sound if the conclusion is true in all cases where the premise is true.
- M is a model of a sentence A if A is true in M
- 2 statements are consistent if there is some variable assignment that causes both to be true.

- Symbols of propositional logic
    - Logical constants
        - True and False
    - Propositional Variables (Symbols)
        - Atom
        - E.g., P, Q, R
        - Each variable can have binary value
            - P = {true | false}
    - Logical Connectives
        - $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
    - Sentences
        - Made by putting these symbols together

- $A \Rightarrow B = \neg A \vee B$
- $\neg(A \wedge B) = \neg A \vee \neg B$
- $\neg(A \vee B) = \neg A \wedge \neg B$
- $\neg\neg A = A$
- $A \Rightarrow B = \neg B \Rightarrow \neg A$

10/30 - Week 5

- If you're given a set of sentences and logic, how can you determine inference methods
- Propositional variables: A, B, C…
- Literal: A, ~A
- Clause is a disjunction of literals
- Term is a conjunction of literals
- Syntactic Form
    - CNF - (A V ~B) ^ (B V ~C) ….
        - If you have OR inside, it is a clause
        - CNF is a conjunction of clauses
            - Each is called a conjunct
    - DNF - (A ^ B) V (A ^ ~C) ….
        - If you have AND inside, it is a term
        - DNF is a disjunction of terms
    - Horn
        - Subset of CNF where every clause has <= 1 positive literal
        - People talk about whether a clause is Horn or not
- Can every sentence be represented in CNF form?
    - Is the CNF language complete? Can everything be represented in CNF?
    - If you take every sentence in arbitrary logic, you can convert it to CNF
- DNF is complete but Horn is not complete
- Some tasks that are hard in arbitrary logic are easier in other forms.
    - Doing SAT on DNF is very easy while SAT on CNF is NP Complete

11/1 - Week 5

- First order logic allows you to have greater expression ability. Can let you write expressions more succinctly.
- In the concept of a world in first order logic, we have the following components
  - Objects - People, houses, colors, etc
  - Relations - Bigger than, inside, part of, etc
  - Functions - Father of, best friend of, etc
- Ex) One plus one equals two
  - Objects - One, Two
  - Relations - Equals
  - Functions - Plus
- Ex) Squares neighboring the wumpus are smelly
  - Objects - Squares, wumpus
  - Relations - Smelly, neighbors
  - Functions - ???
- Ex) Evil King John ruled England in 1200
  - Objects - John, England, 1200
  - Relations - Evil, King
  - Functions - ???
- Primitives from which you build your sentences
  - Constants - King John, 2, UCLA
  - Predicates (Name of a relation) - Brother, neighbor, >
  - Functions - Sqrt, +, father of
  - Variables - x, y
    - Ways to refer to objects in general ways
  - Connections - ^ V ~ implies
  - Equality - =
  - Quantities - For all x
- The first 3 above refer to our ontology.
- Atomic sentence - Just a single predicate
  - Predicate (term1 … termN)
  - A term can be a constant or a variable (when you don't want to be specific) or the result of a function (term1 … termN).
- In propositional logic, the sentences are always true or false. But here we have terms that just represent objects, they're not inherently true or false.
- Anytime you have a variable that can't be quantified, you have a free variable.
- A sentence is not well formed if you have free variables.
- At(x, Berkley) implies Smart(x) is not well formed
- For all x At(x, Berkley) implies Smart(x) is well formed because we've quantified x to all possible values.
- There exist x At(x, Stanford) ^ Smart(x)
- If you want to represent above without quantifiers and you have a finite number of objects, you can say
  - [At(x1, Stanford) ^ Smart(x1)] V [At(x2, Stanford) ^ Smart(x2)]

- If you have an infinite number of objects, then you have to use the there exists quantifier.
- For all is a universal quantifier

11/3 - Week 5

- Sentence is valid if it is true in all models
- Satisfiable and consistent if it is true in some models.
- Unsatisfiable and inconsistent if it is false in all models.

11/8 - Week 6

- Knowledge base contains parts that are domain specific and knowledge specific.

12/8 - Week 10

- Cannot use conjunctions inside predicates.
- A polytree is where there is at most one undirected path between two nodes in the network = singly connected
- Multiply-connected is where there is more than one undirected path.

Textbook Notes

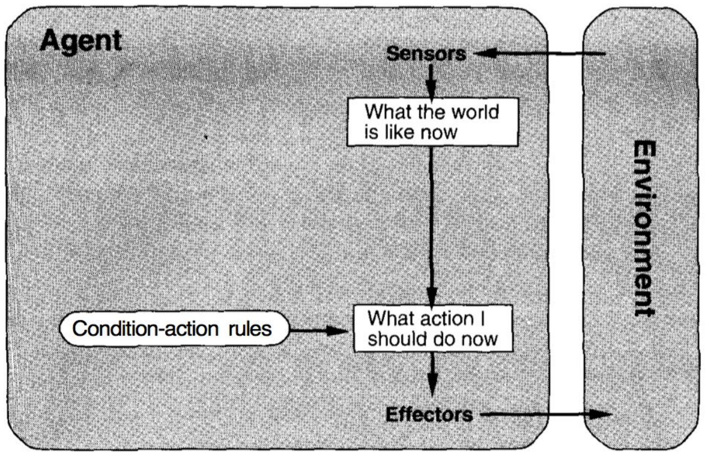
Chapter 1 - Introduction

- Field of AI hopes to understand and build intelligent entities.
- Multiple definitions of AI
  - Systems that think like humans
  - Systems that act like humans
  - Systems that think rationally
  - Systems that act rationally
- When thinking about AI, ask yourself two questions.
  - Are you concerned with thinking or behavior?
  - Do you want to model humans or work from an ideal standard?
- The Turing Test was designed to provide an operational definition of intelligence.
  - Intelligent behavior is the ability to achieve human level performance in all cognitive tasks, sufficient to fool an interrogator.
- An agent is something that perceives and acts.
  - AI is also viewed as a field that studies and constructs rational agents.
  - An intelligent agent takes the best possible action in a situation.

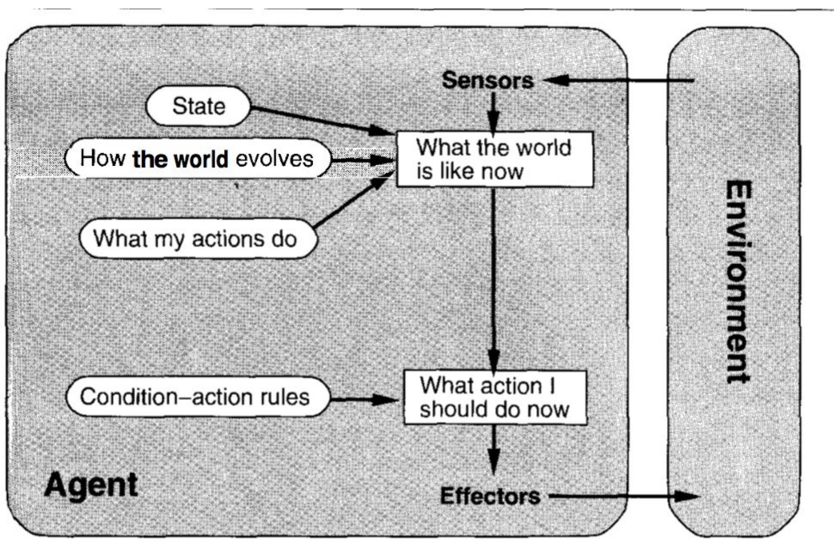
Chapter 2 - Intelligent Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.
  - A rational agent is one that does the right thing, basically the thing that will cause the agent to be the most successful.

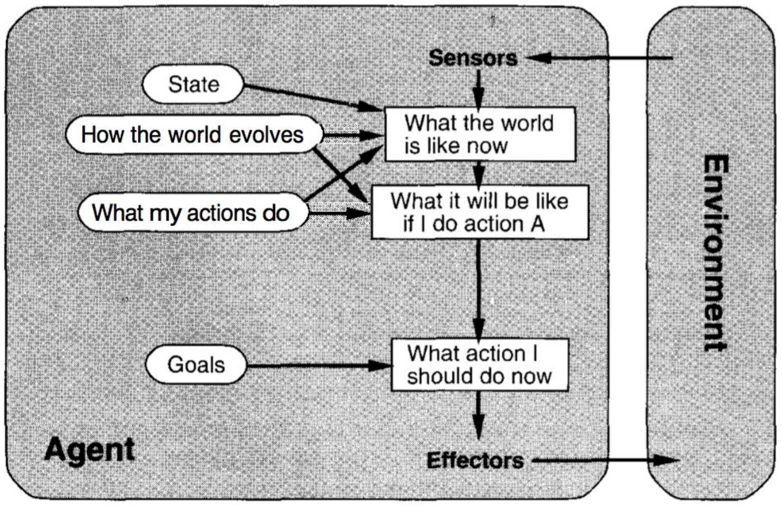
    - This means that there has to be an objective performance measure for how well an agent is doing.
- The rational action at a given time is based on
    - The performance measure that defines success
    - Everything the agent has perceived so far, the percept sequence
    - What the agent knows about the environment
    - The actions that the agent can perform
- For each 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 sequence and whatever knowledge the agent has.
- Agents will likely have some mapping from percept sequences to actions.
- AN agent is autonomous to the extent that its action choices depend on its own experience, rather than on knowledge of the environment that has been built in by the designer.
- There are 4 types of agent programs
    - Simple reflex agents - These agents have some set of condition-action rules and based on the percepts it has at a given moment, it will find a rule that matches with the current situation, and then do the action associated with that rule.



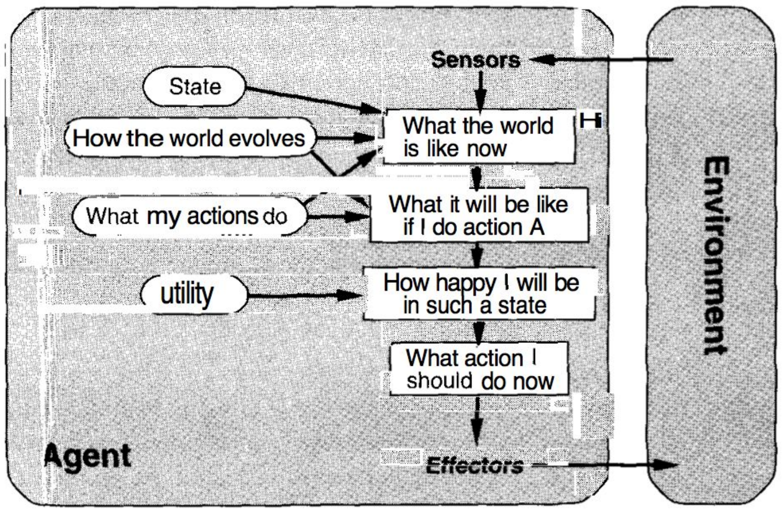
    - Agents that keep track of the world - These agents have some internal state about how the world evolves independently of the agent and how an agent's actions can impact the state of the world.



- Goal-based agents - These agents have some information about the goals they are trying to achieve. This goal information describes situations that are desirable. These agents are different from reflex agents, because goal based agents involve consideration of the future, not just making actions based on current precepts.


Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

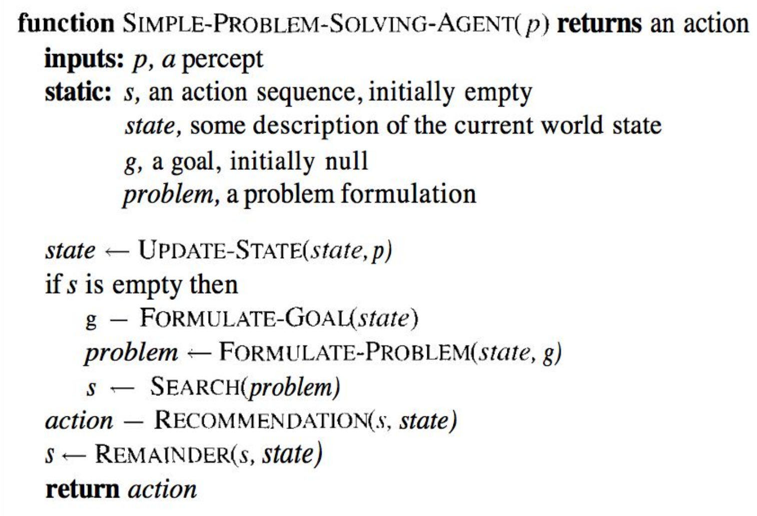
- Utility-based agents - These agents will keep in mind the measure of “goodness” of being in a certain state. If one world state is preferred to another, then it has higher utility for the agent. Utility is a function that maps a state onto a real number, which describes the happiness or goodness.


Sensors
State
How the world evolves
What the world is like now
Hi
What my actions do
What it will be like if I do action A
utility
How happy I will be in such a state
What action I should do now
Agent
Effectors
Environment

- Agents can be put into different environment types.
    - Accessible v.s inaccessible - Depending on whether an agent's sensory apparatus can give it access to the complete state of an environment.
    - Deterministic v.s nondeterministic - Depending on whether the next state of the environment is determined strictly by the current state and the actions chosen by the agents (deterministic) or if there are other factors.
    - Episodic vs nonepisodic - Depending on whether the agent's experience is divided into episodes where the agent perceives and then acts and the quality of the action depends only on the current episode.
    - Static vs dynamic - Depending on if the environment can change.
    - Discrete vs continuous - Depending on if there is a limited number of percepts and actions.

Chapter 3 - Solving Problems by Searching

- There is another type of agent called a problem-solving agent which decides what to do by finding sequences of actions that lead to desirable states.
    - The agent has to formulate an appropriate view of the problem, and that will depend on the knowledge available to the agent (whether it knows the current state and the outcomes of actions).
- Goal formulation is the first step in problem solving.
    - Agent has to figure out what actions it needs to take to get to that goal state.
- Problem formulation is the process of deciding what actions and states to consider.
    - This formulation follows goal formulation.
- The process of looking for a sequence of actions that lead to states of known value is called search.
    - The search algorithm takes a problem as input and returns a solution in the form of an action sequence. It is a path from the initial state to a final state that satisfies the goal test.
- Performing the actions returned by the search is called the execution phase.
- "Formulate, search, execute"

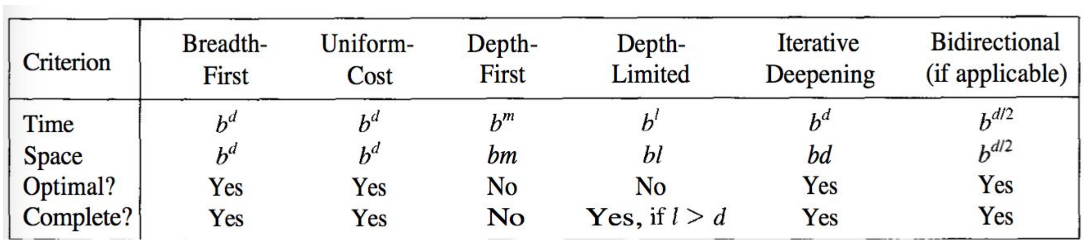


```
function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
          state, some description of the current world state
          g, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, p)
  if s is empty then
     g — FORMULATE-GOAL(state)
     problem ← FORMULATE-PROBLEM(state, g)
     s ← SEARCH(problem)
  action — RECOMMENDATION(s, state)
  s ← REMAINDER(s, state)
  return action
```

- Above shows a simple problem solving agent
- There are 4 types of problems
  - Single state problems: Agent's sensors give it enough information to tell exactly what state it is in and it knows exactly what each of its actions does, and it can calculate which state it will be in after a sequence of actions.
  - Multiple state problems: When the world is not fully accessible and the agent must reason about sets of states it might get to, rather than considering the single state that it will get to if it takes a certain action (this is the situation with single state problems).
  - Contingency problems: Calculating a tree of actions, given that there are certain contingencies that may arise, causing exact prediction to be impossible.
  - Exploration problems: Agent has no information about the effects of its actions, and thus the agent must experiment to discover new actions and see what sorts of states exist.
- A problem is really a collection of information that the agent will use to decide what to do.
- The initial state of an agent and the set of possible actions available to the agent define the state space of the problem.
  - The state space is the set of all states reachable from the initial state by any sequence of actions.
- The essence of search is that we need to choose one option and put the others aside for latter, assuming that the first choice does not lead to a solution.
  - The choice of which state to expand first is determined by the search strategy.
- Search strategies are determined by 4 different criteria.
  - Completeness: Guaranteed to find a solution when there is one?
  - Time complexity: How long it takes?
  - Space complexity: How much memory it takes?
  - Optimality: Does it find highest quality solution when there are several?
- Uninformed search is where the search algo has no information about the number of steps from the current state to the goal.

    - Informed search is when you do have that information that helps you decide the best choice.
- 6 different search strategies that are all uninformed
    - Breadth first search: Root node is expanded, than all the nodes generated by the root nodes, and then their successors, etc.
        - Nodes at depth d are expanded before the nodes at d+1
        - Good because it will find the solution if there is one, but the amount of time and memory needed is not great. It also finds the shallowest goal state, but not always the least cost solution.
    - Uniform cost search: Modifies BFS by expanding the lowest cost node on the fringe, rather than the lowest depth node.
    - Depth first search: Expands one of the nodes at the deepest level of the tree.
        - Should be avoided though for trees with large or infinite maximum depths.
    - Depth limited search: DFS with a cutoff on the maximum depth of a path.
    - Iterative deepening search: Trying DLS with all possible depth limits.
    - Bidirectional search: Simultaneously search both forward from the initial state and backward from the goal, and stop when the two searches meet in the middle.



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l > d$	Yes	Yes

- Above shows the pros and cons with the 6 search strategies.
- In order to avoid going to repeated states, do not return to the state you just came from, do not create paths with cycles in them, and do not generate any state that was ever generated ever before.
- Constraint satisfaction problem is a kind of problem that satisfies some additional structural properties beyond the basic requirements.
    - States are defined by the values of a set of variables.
    - Goal test specifies the constraints the the values must obey.
    - Every variable has a domain which specifies the set of possible values that the variable can take on.
    - A constraint specifies the allowable subset of the domain.
- Backtracking search is a good way to search for solutions in CSP problems. This is because the tree will know when to stop and backtrack if a particular constraint has been violated.
- Forward checking is where each time a variable is instantiated, forward checking deletes from the domains of the yet to be instantiated variables all of the values that would cause a violation of a constraint.
    - If any domain becomes empty, then the search backtracks immediately.

- Forward checking is a special case of arc consistency checking. A state is arc consistent if every variable has a value in its domain that is consistent with each of the constraints on that variable.
    - Exhibits a form of constraint propagation.

Chapter 4 - Informed Search Methods

- We want to expand nodes on our search tree depending on an evaluation function. We want this function to give us the node that appears to be the best choice, dependent on some metric.
- One best first strategy is to choose the node that is judged to be the closest to the goal state. The function that calculates the approximate cost of reaching a goal from a particular state is called a heuristic function.
    - A best first search that uses h to select the next node to expand is called greedy search.
    - This greedy search however is not optimal and is not complete.
- f(n) = g(n) + h(n) where g is the cost of the path so far and h is the approximate cost of the path from the current node to the goal state.
    - The key is to choose an h that never overestimates the cost to reach the goal. This characteristic is called being admissible.
- Best first search using f as the evaluation function and an admissible h function is known as A* search.
- If the f cost never decreases, then the heuristic exhibits monotonicity.
- A* usually runs out of space before it runs out of time.
- The quality of a heuristic function can be determined by its effective branching factor. You can determine this by setting N = 1 + b + b^2 … and solving for b where N is the total number of nodes expanded.
    - Well designed heuristics have b values close to 1.
- If h2(n) > h1(n) for any node n, then h2 dominates h1 and thus h2 will expand fewer nodes on average.
- Thinking of heuristics means thinking of a problem with less restrictions on the operators. We call this a relaxed problem.
- We can also create a set of heuristic functions and then at every node, just take h(n) = max(h1(n), h2(n), ...). Because the components are all admissible, so is the overall h.
- Can use heuristics when trying to solve CSP problems. Examples are most constrained variable heuristic, least constraining variable, etc.
- We can turn A* into Iterative Deepening A*. This helps with the storage cost.
- Simulated annealing is a form of hill climbing, but instead of picking the best move, we pick a random move. If it improves the situation, it is executed. Otherwise it makes it with a probability < 1. The parameter delta E calculates how bad the move was. A second parameter T determines the probability.

Chapter 5 - Game Playing

- When dealing with games, we want to find techniques for choosing a good move when time is limited. Pruning allows us to ignore portions of the search tree, and heuristic evaluation functions allows us to approximate the true utility of a state without doing a complete search.
- In two player games, MAX must find a strategy that will lead to a winning terminal state regardless of what MIN does.
- Minimax algo is designed to determine the optimal strategy for MAX.
- 1) Generate the whole game tree
  2) Apply utility function to each terminal state to get its value.
  3) Determine the utility of the nodes one level higher up in the search tree. When doing this you have to keep in mind what the other player MIN is going to do.
  4) Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Called the minimax decision because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it.
- Minimax relies on the ability to search all the way through to the terminal states. The solution to to have a estimate of the expected utility of the game from a current position. If something is below the cutoff, then it won't be explored. You can also cut off nodes based on depth.
- The process of eliminating a branch of the search tree without examining it is called pruning the search tree. Alpha beta prunes away branches that can't possibly influence the final decision.
- Effectiveness of alpha beta depends on the ordering in which the successors are examined.
- O(b^d/2) for alpha beta instead of O(b^d) for minimax.
- A game with any sort of dice rolls or luck must have chance nodes in addition to MAX and MIN nodes. We want to calculate expected values.

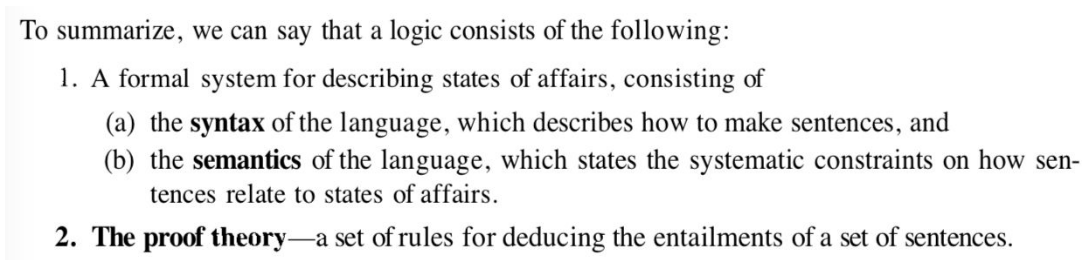
Chapter 6 - Agents that Reason Logically

- In a knowledge based agent, it has a knowledge base which are a set of representations of facts about the world, where each representation is a sentence.
- The sentences are expressed in a knowledge representation language. The language is defined by
  - Syntax: All the possible configurations that can constitute sentences.
  - Semantics: Determines the facts of the world to which the sentences refer to
- We want to generate new sentences that are necessarily true, given the old sentences that are true. The relation is called entailment.

$$KB \models \alpha.$$

- Above you can see that KB (knowledge base) entails a sentence a.

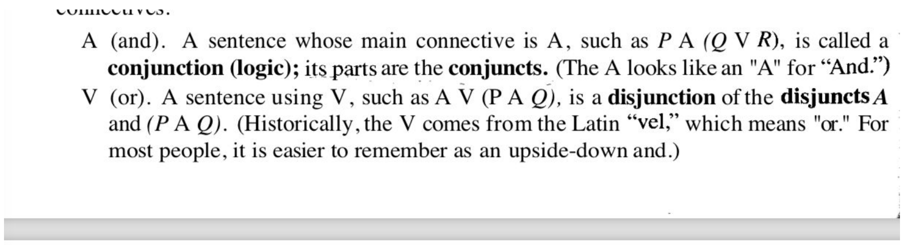
- Inference procedures that generate only entailed sentences are called sound or truth preserving.
- Logical inference is a process that implements the entailment relation between sentences.
- A sentence is valid if and only if it is true under all possible interpretations in all possible worlds.
    - There is a stench at [1,1] or there is not.
- A sentence is satisfiable if and only if there is some interpretation in some world for which it is true.
    - Self contradictory statements are unsatisfiable.
- Logic



To summarize, we can say that a logic consists of the following:

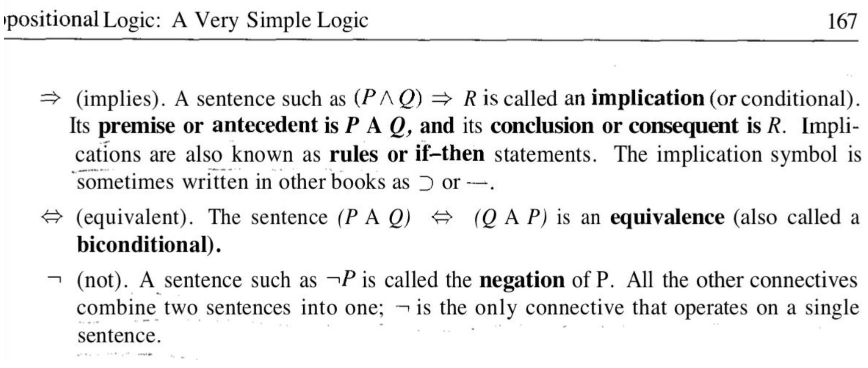
1. A formal system for describing states of affairs, consisting of
    (a) the **syntax** of the language, which describes how to make sentences, and
    (b) the **semantics** of the language, which states the systematic constraints on how sentences relate to states of affairs.
2. **The proof theory**—a set of rules for deducing the entailments of a set of sentences.

- There are two kinds of logic, propositional/boolean logic and first order logic.
- In propositional logic, symbols represent propositions or facts. And we can combine these symbols using Boolean connectives to generate sentences.
- First order logic looks to represent worlds in terms of objects and predicates on objects as well as connectives and quantifiers.
- Propositional logic
    - Syntax in this language are True, False, symbols like P and Q, and connectives like AND, OR, NOT, IMPLIES, etc



A (and). A sentence whose main connective is A, such as *P* A *(Q* V *R*), is called a **conjunction (logic);** its parts are the **conjuncts.** (The A looks like an "A" for "And.")

V (or). A sentence using V, such as A V (P A *Q*), is a **disjunction** of the **disjuncts** ***A*** and *(P* A *Q)*. (Historically, the V comes from the Latin "vel," which means "or." For most people, it is easier to remember as an upside-down and.)

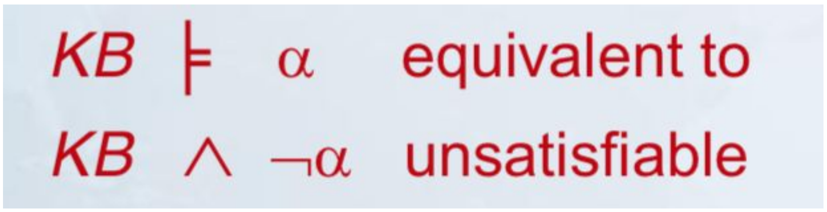


⇒ (implies). A sentence such as $(P \wedge Q) \Rightarrow R$ is called an **implication** (or conditional). Its **premise or antecedent is *P* A *Q*, and** its **conclusion or consequent is** *R*. Implications are also known as **rules or if–then** statements. The implication symbol is sometimes written in other books as ⊃ or —.

⇔ (equivalent). The sentence *(P* A *Q)* ⇔ *(Q* A *P)* is an **equivalence** (also called a **biconditional).**

¬ (not). A sentence such as $\neg P$ is called the **negation** of P. All the other connectives combine two sentences into one; ¬ is the only connective that operates on a single sentence.

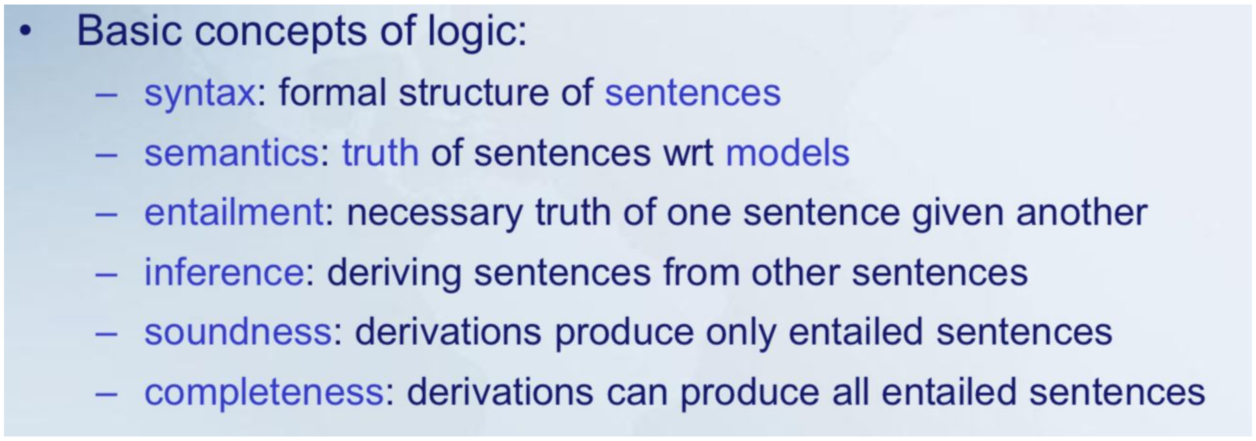
- Atomic sentences have one symbol only. Complex ones have connectives or parenthesis.
- Any world in which a sentence is true under a particular interpretation is called a model of that sentence.
- All inference rules are sound with search algorithms, but they are not guaranteed to be complete.
- Resolution is sound and complete for propositional logic for checking contradictions but not for checking (something else?).
- Any sentence in propositional logic can be transformed into an equivalent sentence in CNF form.



$KB \models \alpha$ equivalent to

$KB \wedge \neg\alpha$ unsatisfiable

- Resolution can be exponential in space and time but if we reduce all clauses to Horn it becomes linear.
- Summary



- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences wrt models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences

Chapter 7 - First Order Logic

- Propositional logic is limited because it makes the only commitment that the world is made up of facts.
- First order logic says that the world is made of objects that have identities, and properties that distinguish them from other objects.
- First order logic consists of
  - Objects
  - Relations
  - Properties
  - Functions
- This way of representing knowledge makes it easier for us to reason about it.
- First order logic has sentences that represent facts and terms that represent objects.

- Composed of
    - Constant symbols - Specifying some object in the world
    - Predicate symbols - Refers to a particular relation
    - Function symbols - Any given object is related to exactly one other object by the relation
- A term is a logical expression that refers to an object.
- An atomic sentence is formed from a predicate symbol followed by a list of terms.
- Quantifiers let us express properties of entire collections of objects.