State space: all possible configurations of the domain of interest
A start state: $s_0 \in S$
Goal States: The set of end states
Operators A: Actions available , defined in terms of a mapping from a state to its successor
Path: a sequence of states and operators
Path cost: number associated with each path
Solution: a path from s_0 to a goal state
Optimal solutions: a solution with minimum cost
Search node: a state, the parent state and the operator used to generate it, the cost of the path, the depth of the node
Candidate nodes: a set of nodes to be expanded
Expanding a node: Applying all legal operators to the state in the node and generating all successor states
Uninformed (Blind) search: when we don't know how far a state is to the goal. Has exponential worst case complexity
Informed (heuristic) search: a heuristic is used to guess how far the state is to the goal
Breadth-first search: all nodes at level i get expanded before all nodes at level i+1, complete, $O(b^d)$ time and space complexity
Branching factor: how many operators at most can be applied to a state
Solution depth: how long is the path to the shallowest solution
Uniform cost search: When you enqueue the nodes in a priority queue, ordered by increasing cost of the path. Guaranteed to find an optimal solution
Depth-first search: Nodes at the deepest level get searched first, $O(bd)$ space comp, $O(b^d)$ time comp, not optimal, not complete. **Depth-limited search**: DFS, but cut off at a max depth. It always terminates, but still may not complete.
Iterative Deepening: ~~What happens to your mom every night~~ It's like depth-limited search but you increase the depth successively. It is complete, and has linear space requirements.
Heuristic: Intuition about the distance from a state to the goal.
Best-First Search: Nodes are enqueued in the order of most promising to least promising. $O(b^d)$ time complexity, not complete or optimal, greedy. **Heuristic Search Algorithm**: Enqueue nodes by the cost of the path and heuristic estimate of the node.
Admissible Heuristic: if $h*(n)$ is the cost of the shortest path from n to any goal, then $h(n)$ is admissible iff $h(n) \leq h*(n)$
A* search: Heuristic search with an admissible heuristic, it is complete and optimal
Consistent: An admissible heuristic is consistent if $h(s) \leq cost(s, s') + h(s')$ for every state s and its successors s'
Iterative Deepening A*: Basically DFS, instead of max depth, we use max f $(h + c)$, we expand all nodes up to f, then we increase f. Uses less memory than A*.
Real-Time search: instead of looking for a path to the goal, we just move in the direction of the best path.
Real-Time A*: Do A* but with the g function equal to cost from current state rather than from the start
α-pruning: Maintain a value α that has the lowest f-value of any node in the current search horizon, and a node costing more than α will never be expanded.
Optimization Problems: described by a set of states and an evaluation function, we are only interested in the best solution, and not the path.
Types of search methods:

- **Constructive**: start from scratch, build a solution
- **Iterative improvement/repair**: start with a suboptimal solution, and improve it
- **Global search**: start from multiple states far apart, and go around the serch space

Hill climbing: greedy local search. Start at a configuration, go to its best successor, repeat until the best successor is worst than the current state. Can get stuck in local extrema, can get stuck on a plateau.
Simulated annealing: if a new value E_i is better than old value E, move to X_i, if it's worse, move to X_i with probability

$$e^{-\frac{E-E_i}{T}}$$

. T decreases with time. When T is high it's in exploratory phase, when T is low it's in exploitation phase. Simulated annealing is a randomized search or Monte Carlo search.
Genetic algorithms: A solution is called an individual, each individual has a fitness, a set of individuals is a population. Populations change over generations by selection/mutation/crossover. **Ways of selection**:

- **Fitness proportionate selection**: $Pr(i) = Fitness(i)/\sum_{j=1}^{p} Fitness(j)$
- **Tournament selection**: pick 2 random individuals, compare them, the superior one is picked.
- **Rank selection**: sort hypothesis by fitness, probability is proportional to rank
- **Boltzman selection**: $Pr(i) = \frac{exp(Fitness(i)/T)}{\sum_{j=1}^{p} exp(Fitness(j)/T)}$

Elitism: Best solution ever encountered in hill climbing/simulated annealing/genetic algorithms are saved.
Constraint satisfaction problem: a solution that satisfies a set of contraints, basically a cost function with minimum value at the solution, and max value somewhere else. It is defined by:

- A set of variables that can take values from a domain
- A set of constraints specifying what combination of values are allowed, they can be explicit ($A \neg 3$) or implicit ($A \neg B$)
- A CSP solution is an assignment of values to the variables such that all constraints are satisfied.

Binary CSP: each constraint relates at most two variables
Constraint Graph: Nodes are variables, arcs are constraints
Preferences(Soft constraints): represented using costs, lead to constrained optimization problems
Backtracking search: Basically like DFS.
Forward checking: Assign value to X, look at each unassigned Y connected to X and delete from Y's domain those values which are inconsistent with X's assignment
Complexity of CSP:

- Worst-case is $O(d^n)$, d is number of possible values and n is the number of variables
- Tree constraint graphs are $O(nd^2)$

- Nearly-tree structured graphs are $O(d^c(n-c)d^2)$ where c is the number of variables which when removed turns the graph into a tree.

Iterative improvement algorithm: Start with a broken assignment, reassign conflicted variables until less conflicts occur
Min-conflicts heuristic: choose value that violates the fewest constraints. It solves CSP in almost linear time except for a very small subset of problems
Minimax search: Expand a complete search tree, then go back up picking the worst value at min levels and best value at max levels. Complete if game tree is finite, optimal against optimal opponent. Time complexity $O(b^m)$, space complexity $O(bm)$. Cope with resource limitation by cutting off, and use heuristic to estimate values at cutoff.
$\alpha - \beta$ **pruning**: We keep the best possible value for max as α and the best possible value for min as β and if any node is lower we don't expand it. It does not affect the final result.
Monte Carlo tree search: Play the game randomly according to some random policy for each player, then the value of each node is the average of the evaluations after the simulation. usually you have a minimax proportion and a monte carlo portion.
Rapid Action-Value Estimate: Assume the value of the move is the same no matter when it is played.
Plan: A collection of actions for performing some task with forms of knowledge representation to describe sets of states. **Declarative approach**: Build agents with two parts. A knowledge base which contains a set of facts expressed in formal/standard lang. An inference engine with general rules for deducing new facts
Logic: Formal language for representing information. Syntax defines sentences in the language, semantics define the "meaning" of sentences
Ontological Commitment: What exists in the language: facts/objects/relations/time
Epistemological Commitment: What states of knowledge are in the language: true/false/etc
Interpretation: A way of matching objects in the world with symbols in the sentence: a truth assignment: A sentence is valid if it's true in 1 interpretation, satisfiable if in all, unsatisfiable if in none.
Entailment(KB $\vDash \alpha$): KB entails α iff α is true in all worlds where KB is true. **Inference(KB $\vdash_i \alpha$)**: α can be derived from KB by inference procedure i. i is sound if when KB $\vdash_i \alpha$, KB $\vDash \alpha$. i is complete if when KB $\vDash \alpha$, KB $\vdash_i \alpha$. **Model checking**: an inference proof method by enumerating a truth table.
Conjunctive normal form: conjunction of disjunction of literals: OR clauses connected by ANDs
Disjunctive normal form: disjunction of conjunction of literals: And clauses connected by ORs
Horn form: Clauses with ≤ 1 positive literal, implications connected by ANDs
Resolution (for CNF): $\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$
Modus Ponens (Horn form): $\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \vee \ldots \vee \alpha_n \rightarrow \beta}{\beta}$
Forward chaining: when a new sentence is added to KB, resolution happens, new sentences are added to KB. Data driven, eager.
Backward chaining: when query q is asked, if q is in KB, return true, else resolve q with other sentences in KB and continue. Goal driven, lazy
Implication elimination: $\frac{\alpha \rightarrow \beta}{\neg\alpha \vee \beta}$
Planning graph: Proposition and Action nodes arranged in levels in which they alternate. Lines between levels indicate pre/post conditions, lines within levels indicate mutual exclusions. **Predicates**: used to describe objects, properties, and relationships between objects.
Quantifier: $\forall$ or $\exists$
Atomic sentences: predicate($term_1, \ldots, term_n$) or $term_1 = term_2$, Term = function($term_1, \ldots, term_n$) or constant or variable
Complex sentences: made from atomic sentences using connectives
Universal quantification: $\forall x Taking(x, AI) \rightarrow Smart(x)$, **Existential quantification**: $\exists x Taking(x, AI) \hat{S}mart(x)$ **Quantifier properties**: $\forall x \forall y \leftrightarrow \forall y \forall x$, $\exists x \exists y \leftrightarrow \exists y \exists x$, $\exists x \exists y \not\leftrightarrow \exists y \exists x$ $\forall x f(x) \leftrightarrow \neg\exists x \neg f(x)$, same with exists.
Proofs: Modus Ponens $\frac{\alpha, \alpha \rightarrow \beta}{\beta}$, And Introduction $\frac{\alpha\ \beta}{\alpha\hat{\beta}}$, Universal elimination $\frac{\forall x \alpha}{\alpha\{x/\tau\}}$, Resolution $\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee gamma}$
Skolemization: $\exists f(x) = Rich(G1)$, if $\exists$ is inside $\forall$: $\forall x f(x) \leftarrow \exists y g(y) = \forall x f(x) \leftarrow g(H(x))$. $H(x)$ is a skolem function.
STRIPS: Domain: a set of typed objects, states: first-order predicates over objects in conjunction, operators/actions defined in terms of preconditions and effects, goals: conjunction of literals.
STRIPS Operator: preconditions are conjunctions of positive literals, postconditions are in terms of an Add-list and a Delete-list.
State-space planning: finding a plan by looking through state space looking for a path from start to goal. Progression planners start from start, regression planners start from goal.
Progression (Forward) Planning: Determine all applicable operators in the start state, apply operator, determine new content of knowledge base, repeat until goal is reached.
Goal Regression: Pick action that satisfy (some of) the goal's propositions, make a new goal by removing conditions satisfied by this condition, adding the preconditions of this action, repeat until goal state is satisfied by the start state.
Variations of Goal Regression: linear planning is a stack of goals, not complete. non-linear (set of goals) is complete but expensive.
Total vs Partial Order: total: plan is always a strict sequence of actions, partial: plan steps may be unordered
Bayes rule: P(H|e) = P(e|H)P(H)/P(e), P(e) = P (e|H)P (H) + P (e|¬H)P(¬H). P(H|e) is posterior probability, P(H) is prior probability, P(e|H) is likelihood, P(e) is normalizing constant.
Conditional Independence: $P(x|y,z) = P(x|z), \forall x, y, z$, Knowing the value of y does not change the probability of x if z is known. If C and F are conditionally independe, $P(C, F, B)$ decomposes to $P(C|B)P(F|B)P(B)$