Intro to AI

$\neg: negation
\land: conjunction
\lor: disjunction
\to: implication
\leftrightarrow: biconditional
\forall: universal quantification
\exists: existential quantification $
α $\models$ – ENTAILMENT

Sam

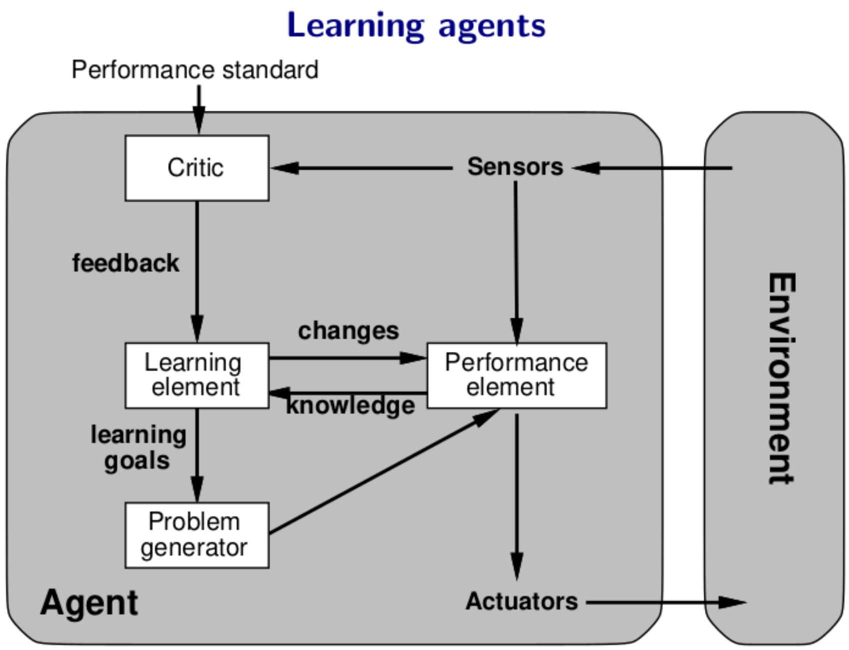
- What is AI?

Think like humans ● No theories explaining/producing human-level general intelligence	Think rationally ● Thoughts I should have vs could have ● Intelligent behavior can exist without thinking (Braitenberg vehicle)
Act like humans (Turing test) ● Not reproducible/constructive	Act rationally ● Act to maximize goal achievement ● Can be done without thinking

  - **A machine that can act rationally**
- What is learning?
  - The ability to improve performance using past experiences
- What does “acting rationally” mean?
  - Act to maximize goal achievement given the available information, though it can be done without thinking.
- What is the Turing test?
  - A human evaluator judges a conversation between a human and a machine, and must determine which is a machine. If the evaluator cannot distinguish the machine from a human, then the machine passes the Turing test.
- What is an agent?
  - An entity that perceives and acts

- What is a rational agent?
    - An agent that acts rationally relative to the performance measure because perfect rationality is unachievable.
    - An agent that **chooses whichever action maximizes the expected value of the performance measure** given the precept sequence to date
- What is a perfectly rational agent?
    - An agent that maximizes the expected performance
- How does an agent evaluate its own performance?
    - The **agent's performance is compared against a performance measur**e using outside input, either via sensors (and possibly a model) or an observer.
    - ***Critic, utility function, performance measure***
- What is an agent's environment?
    - The world, either physical or virtual, in which the agent exists and interacts using sensors and actuators.
- What is PEAS for a task environment?/What is needed to design a rational agent?
    - Performance measure, environment, actuators, sensors
- What kind of properties might an environment have?
    - **Deterministic, fully observable** =⇒ classical search problem **OFFLINE**
        - agent knows exactly which state it starts in, what each action does
        - ♦ no exogenous events (or else they're encoded into the actions' effects)
        - ♦ solution is a sequence, can predict future states exactly eg -Vacuum World with no exogenous events
    - **Non-observable -**
        - **Agent may have no idea where it is**
        - **♦ solution (if any) is a sequence that is conformant, i.e., guaranteed to work under all conditions eg-Vacuum World, no exogenous events and no sensors**
    - **Nondeterministic and/or partially observable: ONLINE**
        - **percepts provide new information about current state**
        - **♦ solution is a contingent plan or a policy**
        - **♦ often interleave search, execution**
    - Fully observable, partially observable, non-observable
        - Can see every, some, or no aspects of the world
    - Deterministic vs non Deterministic
        - **Deterministic** -fully determined by the current state and the inputs or rules governing the system
        - **non Deterministic -randomnes**s, **uncertainty,** or the presence of multiple choices or paths at each step, not fully predictable, have probabilistic models to predict
    - Episodic or sequential
        - **Episodic:** Self contained atomic episodes, the task is divided into atomic episodes; they are independent from each other, can be solved separately

            - **Sequential**: cause-and-effect relationship, an action may influence all future decisions ,sequential consequences of current and past decisions
    - Static or dynamic
        - World does or does not change while agent is thinking
    - Discrete or continuous
        - World (state of the environment) described by discrete or continuous data
    - Single-agent or multi-agent
        - One or more agents
- What is the difference between stochastic and non-deterministic?
    - **Stochastic** is always associated with probabilities (50% chance of rain tomorrow). **Non-deterministic**: diff choices lead to diff outcomes, may involve branching not fully determined by set of rules
    - can produce a range of potential outcomes, each with its own probability or likelihood
- What type of environment is the real world?
    - Partially observable, stochastic, sequential, dynamic, continuous, multi-agent
- Types of agents?
    - Simple reflex
        - Follows condition-action rules based on the perceived world
    - Reflex agents with state
        - Uses knowledge of the **current state**, how the **world evolves**, and **consequences** of previous actions to determine what the current world is like, then follows condition-action rules
    - Goal-based
        - Like a reflex agent with state, but after determining the current world, **determines the consequences of possible actions** in the current world, then chooses the action that best **fulfills the goal**
    - Utility-based
        - Like goal-based, but measures utility instead of goal completion, determining how much utility an action gives.
    - All can be turned into learning agents



Learning agents

        - Perceived information is used to make an action. The critic gives feedback based on perceived information, and the learning element changes the performance element in response. The learning element also feeds goals to the problem generator, which sends problems to the performance element.

- What type of agent can implement an “intelligent” vending machine? Lunar rover? Mario Bros player?
    - Reflex with state, utility-based, simple reflex
- What type of agent might be appropriate to implement a Braitenberg vehicle?
    - Simple reflex agent
- What are some real world applications of AI?
    - Self driving cars, smart database analysis, space exploration, game-playing/video games
- What is the “model” inside a model-based agent?
    - An internal state determined by the previously observed world and percept history .
- What is an agent function?
    - Maps precept histories to actions: f:P* ⟶ A
- What is a precept?
    - Perceived information
- What is the agent program?
    - Software that runs on the physical architecture to implement the agent function
- What is an NP-complete problem?
    - Non-deterministic polynomial time complete
    - The solution can be found via brute-force
    - The solution can be verified quickly
    - Problem that can be solved by n parameters in a polynomial solution(x^n)
    - Solution is possible by enumeration

Uninformed Search

Sam

- What is a state?
  - A representation of a physical configuration
- Is it possible to have several goal states?
  - Yes, though some may not be optimal
- How do you select the **best goal state**?
  - The best goal state has the shortest path cost
- What is the successor function?
  - A function that determines the next state given the current state and action
- Why do we use search and state space to solve problems?
  - It simplifies real problems, allowing them to be solved quickly with general algorithms.
- How does a search agent know which is the best path to take?
  - It chooses the path with the shortest path cost
- What is the definition of a search problem?
  - Agent - entity that **perceives i**ts environment
  - State **- configuration** of an agent in its environment
  - Initial state - where algo starts
  - Set of actions - func for **Choices that can be made in a state.**
  - State-transition function (successor function) - resulting state from performing any action in any state
  - Goal test - determines whether a given state is a goal state
  - Path cost (additive step cost) - num cost associ with each path
  - **Example: robotic assembly**
    - **states:** real-valued coordinates of robot joint angles parts of the object to be assembled
    - **actions:** continuous motions of robot joints
    - **goal test**: complete assembly
    - **path cost:** time to execute

- What is the solution to a search problem?
  - The sequence of actions leading from the initial state to the goal state
- What is abstraction and how does it relate to AI/search?
  - Abstraction is converting real objects into symbols/ideas. This is useful for taking real world problems and restating them in a way that they can be solved using general methods, such as standard search algorithms.
  - Real state space -> abstract state space -> abstract actions -> abstract solution -> real solution -> real actions

- What is the **difference** between a graph and a tree?
    - Graph has no hierarchical relationship of parent/child nodes. In a tree, each node has only one parent.
- What does “searching the state space” mean?
    - Exploring the possible paths between nodes to find a path from a start node to an end node.
- What does “expanding” a node mean?
    - Generating all of its children
- What is the **fringe**/frontier?
    - All nodes that have been generated but not expanded (all candidates for expansion)
- Why is search difficult?
    - Exponential time and space complexities due to branching and depth of tree
- How big can a search tree be?
    - Infinite
- What is the difference between a state and a node?
    - A state is a representation of a physical configuration (no children, depth, path cost, or parent)
    - A node is a data structure that is part of a search tree
- What does a node contain?
    - State, parent, children (if the state is expanded), depth, and path cost
- What is completeness?
    - Does the algorithm always find a solution if one exists?
- What is soundness?
    - The solution found is correct
- What is optimality?
    - Does the algorithm always find the least-cost solution?
- What is time complexity?
    - The number of nodes generated/expanded
- What is space complexity?
    - Maximum number of nodes in the memory
- What terms are time and space complexity measured in?
    - Maximum branching factor of the tree, b
    - Depth of the least cost solution, d
    - Maximum depth of the state space, m
- What are the search problem types?
    - **Deterministic**, fully observable -> classical search problem
        - Agent knows the initial state, knows the effect of every movement, no spontaneous external effect, future states can predict exactly.
    - **Non-observable ->** solution is a sequence that is **conformant**
        - Agents may have no idea of the position, or the properties of the current state. The solution, if it exists, is conformant, which grants a solution in every scenario.
    - **Nondeterministic and/or partially observable** -> solution is a contingent plan or policy. Interleave search and execution (online problem solving)

  - **Unknown state space** -> exploration problem
- What does conformant mean in regards to a solution?
  - Solution is guaranteed to work under all conditions
- What is the difference between **online and offline problem** solving?
  - Online problem solving: gather knowledge as you go
    - Necessary for exploration problems
    - Can be useful in nondeterministic and partially observable problems
  - Offline problem solving: develop the entire solution at the start, before you
  - ever start to execute it
  - e.g., the solutions for the Vacuum World examples on the last three slides

- Contents of abstraction?
  - Abstract state, abstract action, abstract solution
- How can repeated states affect a problem?
  - Make it exponential instead of linear (slide 66)
- What does NP-hard mean?
  - A decision problem *H* is NP-hard when for every problem *L* in NP, there is a polynomial-time many-one reduction from *L* to *H*.

Isti

What is a state-space?

- This is an abstraction of the real world, it is needed, because the real world is too complex, with many irrelevant details.

From what consists of a problem?

- Initial state, set of actions, state-transition function, goal test, path cost.

What does a node include in tree search algorithms?

- State, parent, children, depth, path cost.

What is a fringe/frontier?

- All nodes that have been generated but not expanded.

What is the difference between a state and a node?

- State is a representation of a physical configuration, a node is a data structure that is part of the search tree. (The nodes have parents, children, path cost, state. The stated do not have.)

What is a strategy?

- Picking the order of node expansion.

How can we evaluate a strategy?

- Completeness: does it **always** find a solution if one exists?
- Optimality: does it **always** find a least-cost solution?
- Time complexity: number of nodes generated.
- Space complexity: maximum number of nodes in memory.

How do we measure time and space complexity?

- **b: branching factor**. maximum number of child nodes that any node in the tree/possible actions
- **d: depth of the least-cost solution**. (inf if there's no sol)
- **m: maximum depth** of the state space. (may be inf)

What kind of uninformed search strategies do you know?

- Breadth-first search, depth-first search, depth-limited search, uniform-cost search, iterative deepening search.

Explain **breadth-first search!**

- Starts from root node, expands all successor nodes at current level, before moving to nodes of next level

Expand the shallowest unexpanded node.

- Fringe is a FIFO queue, it means new successors go to the end of the list of nodes to be expanded.
- Properties:
    - Complete
    - Time complexity: O(b^d)
    - Space complexity: O(b^d)
    - It finds the optimal solution, if the step cost is 1 in every case.

Explain **uniform-cost search**! / **What is Uniform Cost Search? -graph/tree**

- Fringe is a priority queue, ordered by path cost, lowest first.

- Expanded least-cost unexpanded node, priority QUEUE ordered by path cost with lowest first. This is equivalent to BFS if all step costs/path cost of all edges are equal

·

- **Properties**:
    - C* **is Cost of the optimal solution, ε** is each step to get closer to the goal node

      Complete: yes if $\exists\ \varepsilon > 0$ such that step cost $\geq \varepsilon$
    - Time complexity: $O(b^{[C^*/\varepsilon]})$
    - Space complexity: $O(b^{[C^*/\varepsilon]})$
    - Optimal.

Explain **depth-first search**!

- Starts from root node, expands to its greatest depth node, before moving to nodes of next level

Expand the deepest unexpanded node.

- Fringe is a LIFO stack ds, newest successors first.
  - deep down searching and sometimes it may go to the infinite loop.
- Properties:
    - Complete:
        - Yes, in finite spaces, where loops are eliminated.
        - No, in infinite spaces.

Where, m= maximum depth of any node

- Time complexity: $O(b^m)$
- Space complexity: $O(b*m)$
- Optimal? No, generate a large number of steps or high cost ……but in some cases, it could be helpful.

Explain depth-limited search!

- Same as depth-first search, but backtracks at each node of some depth limit, unless it is a solution.

- Depth-limited search can be terminated with two Conditions of failure
    - **Standard failure value**: It indicates no solution
    - **Cutoff failure value**: It defines no solution within a given depth limit.

·

· Properties are the same?

Explain iterative deepening search! -combining DFS and BFS

· Depth limited search to depth 0, 1, 2, and so on,

- Performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

· Properties:

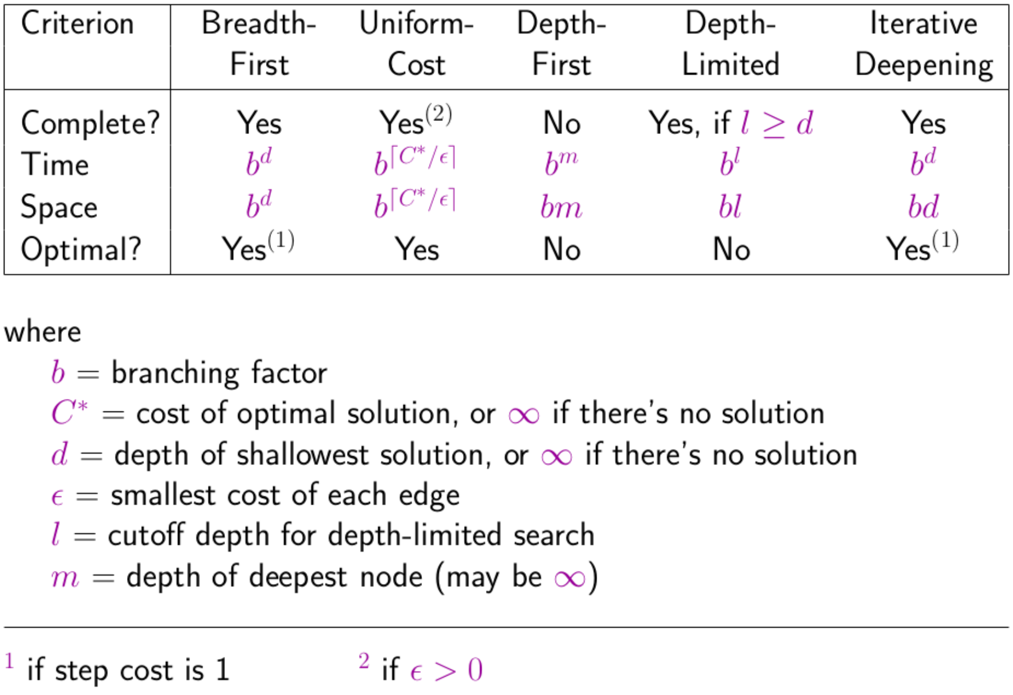
o Complete. If the branching factor is finite.

o Time complexity: O(b^d)

o Space complexity: O(b*d)

o Optimal, if path cost is 1 in every case., memory efficient

- **Describe the TREE search algorithm.**
    - From the **current nod**e, select a l**eaf node.** If the leaf node doesn't pass the goal test, expand it, and add its leaf nodes to the search tree. If it does pass, return that node. If all nodes in the search tree have been expanded (fringe is empty) and no solution is found, return failure.
    - Goal test not done until a node is selected for expansion
- **Graph search**
    - takes exponentially less time than tree search (no of paths to node is exponential in depth)
    - takes exponentially more space than tree search (when the search space is treelike)



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes[2]	No	Yes, if $l \geq d$	Yes
Time	b^d	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^d	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes[1]	Yes	No	No	Yes[1]

where

b = branching factor

C^* = cost of optimal solution, or ∞ if there's no solution

d = depth of shallowest solution, or ∞ if there's no solution

ϵ = smallest cost of each edge

l = cutoff depth for depth-limited search

m = depth of deepest node (may be ∞)

[1] if step cost is 1

[2] if $\epsilon > 0$

Informed Search (Heuristic Search)

Sam

- What is a heuristic?
  - A **rule of thumb** that ..**reduces the search** for solutions in DIFF DOMAINS, but one may not find an optimal solution or a solution at all
  - **estimate costs of shortest** paths
  - Good heuristics can **dramatically reduce search cost**

- **Heuristic choice- what node to expand next**
  - Use an **evaluation function f (n)** for each node n – estimate of **"desirability"**
  - ⇒ Expand most **desirable unexpanded node**
- What is a heuristic function?
  - A function that estimates the cost from the given node to the goal
- What is heuristic tree search?
  - basic tree search, starting from the initial state and expanding nodes until a solution is found or the search tree is exhausted
    - A* (A-star) and IDA* (Iterative Deepening A*)
- What is Heuristic graph search?
  - function performs a graph search algorithm by using a fringe for node expansion
    - **closed** set – keeps track of visited states to avoid revisiting them.
    - ensures that the algorithm does not revisit states and avoids cycles in the graph.
  - Greedy Best-First Search and Recursive Best-First Search.

- Which nodes are expanded first in heuristic tree search?
    - Nodes that APPEAR to be closest to the goal, according to the heuristic
- Does the same change to heuristic tree search apply to heuristic graph search?
    - Yes
- What is an admissible heuristic?
    - A heuristic where the **0 <= estimated cost <= true cost**
    - Never overestimates the true cost in order to not ignore potential solutions
    - can be derived from exact solution of relaxed problems
- What is a consistent heuristic? Also called monotonicity
    - A heuristic that follows the triangle inequality, i.e. if you have a route n->g, with another route n->n'->g, the length of the second route has to be >= the first route, but not shorter.
    - h(n1) <= c(n1 -> n2) + h(n2)
    - f(n) is nondecreasing along any path.
- Explain greedy search!
    - Greedy search uses f (n) = h(n),
        - keeps fringe ordered in increasing value of h
        - Always expands whatever node appears to be closest to the goal.
        - incomplete and not always optimal
- Explain Greedy Best-first search.
    - Greedy best-first search is tree/graph search using a heuristic.
    - prioritizing the nodes that appear to be the closest or most promising to the goal based on a heuristic evaluation.
- What is the time and space complexity of greedy search?
    - Not complete because it can get stuck in loops, can be avoided by **repeated-state checking.**
    - Time: O(b^m) but can be dramatically improved with a good heuristic
    - Space: O(b^m) keeps all nodes in memory
    - Not optimal
- Is greedy search the same as a greedy algorithm
    - No, a greedy algorithm doesn't remember all of the fringe, so it only remembers the current path and doesn't backtrack. Therefore,
    - ♦ Repeated-state checking cannot make it complete
    - ♦ It runs in time O(l) if it finds a solution of length l

- Explain the A* search algorithm.
    - A* is a heuristic tree search with a special heuristic.
    - A* avoids expanding paths that are already expensive by using the heuristic **f(n) = g(n)+h(n)**
    - Evaluation function: f(n) = g(n) + h(n)
    - g(n): the cost so far to reach node n
    - h(n): the estimated cost to reach the goal

        - f(n): the total cost from the initial state to the goal state

Explain A* tree search!

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

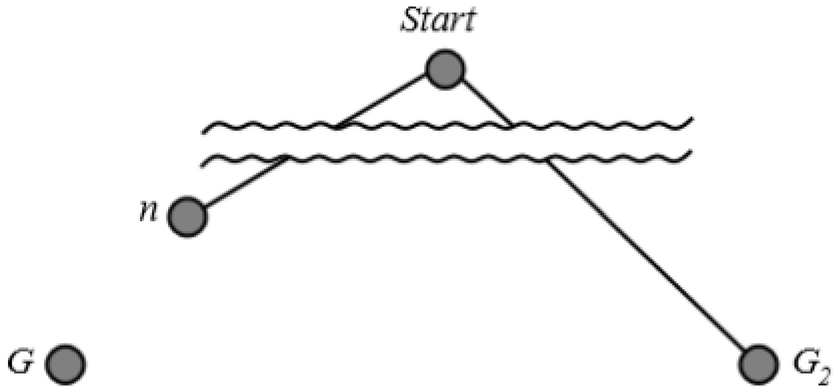
- **Admissible heuristic**: tree -it never overestimates the cost from a node to the goal,provide a lower-bound estimate of the actual cost
- **Consistency**: A* expands all nodes with f(n) < C* current best solution cost, and some nodes with f(n) = C* (based on tiebreaker)
- If f is consistent, A* expands no nodes with f(n) > C*
- not needed for A* graph-search. – it can re expand if a new path to a closed node is found to be shorter than the previously expanded path
    - ensures that A* explores alternative paths and avoids being trapped in suboptimal solutions.

If both the optimality and completeness requirements are satisfied, the A* returns the optimal solution.

- When does A* end the search process?
    - When a goal is found
- What is the time and space complexity of A*?
    - Complete as long as no infinite path as a finite cost (no step cost of 0).
    - Time: O(entire state space) in worst case, O(d) in best case
    - Space: All kept in memory
    - Optimal if heuristic is admissible
- What is a consistent heuristic?
    - The total cost should not decrease along the path. Two branch consistency.
- How does A* change with graph search? How A* Graph search is different?

    - No infinite path has finite cost: ensures that A* search terminates and guarantees completeness, means that there are **no loops or cycles i**n the search space with a finite total cost.
        - **in case of inconsistent heuristic, it is modified to re-expand nodes if** a new path to a closed node is found to be shorter than the previously expanded path
        - Finds optimal solutions even if the heuristic is inconsistent

- What is the behavior of A* with a consistent heuristic?
    - Nodes are expanded in order of increasing f value
- What is the behavior of A* with an inconsistent heuristic?
    - As one goes along a path, f may sometimes decrease as the goal is approached, instead of increasing.
    - A* doesn't always expand nodes in order of increasing f value because it may find lower cost paths to already expanded nodes, which will have to be re-expanded.
    - **may fail to find the optimal solution, and find suboptimal**
    - But – GRAPH SEARCH will not re-expand already expanded nodes
- How does the behavior of A* change with different heuristics?
    - With better heuristics, the execution time is improved.
- Explain how the optimality of A* works. Why will suboptimal goal G2 never be reached?



    - We know that G is optimal, and G2 is not. Therefore, f(G) < f(G2). f(n) has to be less than f(G), because it is higher up on the same path. Therefore, f(n) < f(G2). Because f(n) < f(G2), f(G2) will never be chosen in favor of n, so f(G2) will never be reached.
- What does it mean when a heuristic dominates another heuristic?
    - When one heuristic always gives a larger distance than another while still being <= true distance, then it dominates. This makes it more useful because it is closer to the true distance.
- What is **dominance?**
    - If the cost from the current node to the goal state (h2(n)) is greater than another cost (h1(n)), we say it dominates the other cost (h2(n) dominates h1(n)).

- What is a way to get dominance?
    - Select the highest remaining cost from the available possibilities.
    - If h_a and h_b are admissible heuristics, then h(n) = max(h_a (n), h_b (n)) is admissible and dominates h_a , h_b
- How can admissible heuristic functions be created?
    - Let **P** be a prob, **h*(s)** = minimum cost of solution path
    - Relax the problem P to P' by removing some constraints.

  - Every solution path in P is a solution path in P’, but not vice versa (P’ may have additional paths that aren’t sol paths in P)
  - h(s) is the minimum cost of a solution path in P’.
  - Then h(s)<= h’(s), then **h is an admissible heuristic for P.**
- Example of making an admissible heuristic?
  - TSP: Let solutions include paths that don’t return to the origin city and paths that revisit cities
  - Minimum spanning tree can be computed in $O(n^2)$
    - ⇒ lower bound on the least-cost path that visits all cities
    - ⇒ lower bound on the least-cost tour
- Describe the Traveling Salesman problem.
  - Find the shortest path that visits each city once and returns to the starting city (NP-hard)
- Describe the 8-Queens problem.
  - Move 8 queens on a chess board from an initial state so that no queens are in the same row or column in the smallest number of moves.
- What is the Manhattan distance?
  - Distance in grid coordinates (Ex. 4 up, 2 left = 6)
- Explain **iterative-deepening A*.**
  - Iterating a cost-limited search with an f_max that increases with each iteration.
  - Cost-limited search does DFS backtracking at nodes where f(n) > f_max
  - If no solution is found, the min(f(n) that the search backtracked to) becomes the new f_max

- IDA* is like a combination of A* and IDS
  - – complete, returns optimal solutions
  - – much lower space requirement than A*
  - – same big-O time if number of nodes grows exponentially with cost

- What is the time and space complexity of IDA*?
  - Complete unless there are infinitely many nodes with f(n) <= f(G)
  - Time: Like A* if f(n) is an integer and the number of nodes with f(n) <=k grows exponentially with k
  - O(bd)
  - Yes, given a consistent heuristic, IDA* expands all nodes with f(n) < C* and no nodes with f(n) >= C*
- Are heuristics domain specific?
  - Yes
- True/False
  - The f-values of nodes along a path must be non-decreasing. (True)
  - If n2 is expanded after n1, then f(n1) ≤ f(n2) (the f-value increases monotonically). True
  - When n is expanded every path with lower f-value has already been expanded. (True)
  - With a monotone heuristic, the first time A* expands a state, it has found the minimum cost path to that state. (True)

- What does P=NP mean?
    - In the worst case, most searches take exponential time, P=NP conjecture suggests that the class of problems that can be solved efficiently (P) is equivalent to the class of problems whose solutions can be verified efficiently (NP). In simpler terms, if a problem has a solution that can be verified quickly, then there also exists an efficient algorithm to find that solution.

Local search

Sam

- What is local search?
- · The path to the goal is irrelevant. Only the goal state itself interests us.
-
-

How different are iterative improvement algorithms (like hill climbing and simulated annealing) from classical search algorithms?

- How is local search different from other search techniques?
    - It searches for the optimal goal state itself, not a path to the optimal goal state
- How is the state space different in local search?
    - The state space is a set of goal states
- What are iterative improvement algorithms?
    - Algorithms that keep a single current state and try to improve on them to reach an optimal goal state
- Can local search algorithms be used for online search?
    - Yes, both online and offline
- Mention some example problems that can be solved through local search.
    - N-queens problem. Start with an initial state and move a single queen at a time to reduce the number of conflicts until no conflicts remain.
    - TSP. Perform pairwise exchanges of connected cities until a least cost tour is found.
- Explain **hill-climbing**.
    - When a node is expanded, move on to the highest/lowest value child/neighbor node. If there is no higher/lower value node, return the current node as the solution.
    - At each step, the algorithm examines the neighbors. If a neighbor is more desirable than the current state, it moves to that neighbor. It repeats this procedure until there is no neighbor which has more favorable properties than the current one.
- Describe the typical “landscape” of the state space.
    - 2D graph of objective function as a function of state space. Trying to find the highest/lowest point in the graph (optimal goal state in the state space) to maximize/minimize the objective function.

- What is the problem with algorithms like **hill-climbing**? How can this problem be fixed?
    - Prone to getting stuck on local extrema, shoulders, and plateaus.
    - **Random-restart**: repeat with randomly chosen starting points (trivially complete)
    - **Local-beam search**: Randomly generate k states, chooses k highest-valued neighbors
    - **Stochastic beam search** chooses k successors randomly, biased towards good ones
- Describe the **local beam search algorithm.**
    - Randomly generate k states, then generate their successors. If any is a solution, return it, otherwise select the k best successors to repeat with.
    - Often all k states end up on the same local extrema
        - Stochastic beam search chooses k successors randomly, biased towards good ones
- What is the idea of **simulated annealing**?
    - Allow some bad moves to escape local extrema, but gradually decrease their size and frequency.
- Explain the role of the temperature function in simulated annealing.
    - It reduces the size and frequency of bad moves over time by decreasing the probability of a bad mov (which is a function of temperature).
- Explain the role of **randomness in simulated annealin**g.
    - The probability of allowing a bad move decreases exponentially over time due to the temperature function.

- Do algorithms like HC or SA always find the best solution?
    - No, but SA comes close
- How do we use HC/SA in continuous state spaces?
    - The objective function (continuous) measures desirability. Use the gradient of the objective function. It is 0 at the goal.
    - Discretize the continuous state space

Constraint satisfaction

Constraint Satisfaction Problem (CSP) has consistent and valid values in its domain based on the constraints imposed by other variables

Sam

- Describe the structure of a CSP.
    - A CSP consists of an initial state, a goal test, a successor function, and a domain
- What is a state in a CSP?
    - A set of assignments of values to variables with domains
- What is a domain in a CSP?
    - The set of possible values that can be assigned to a variable
- What is the goal in a CSP?
    - A state with all variables assigned a value without violating any constraints

- What is a goal test in a CSP?
    - A set of constraints specifying allowing combinations of values for various sets of variables
- What is a formal representation language?
    - a language or notation used to express domain, constraints, variables, and their relationships in a structured & formalized manner
- How are constraints expressed?
    - In a constraint graph, constraints are represented as edges between nodes, which represent variables.
- What is a binary CSP?
    - Each constraint relates at most two variables
- What types of CSP exist?
    - **Discrete variables**; finite domains or infinite domains (linear or nonlinear constraints, requires a constraint language)
    - **Continuous variables:** linear constraints solvable using linear programming methods in polynomial time (very high overhead)
- What kind of constraints do you know?
    - • **Unary**: constraints involve only one variable.
    - • **Binary**: constraints involve pairs of variables.
    - • **Higher orde**r: constraints involve 3 or more variables.
    - • **Preferences**: soft constraints, determine which value is more desirable. Often represented by a cost.

Node consistency— when all values in var domain satisfy the variable's unary constraints

- Give some examples of real-world CSPs.
    - Assignment problems
    - Timetabling problems
    - Floor planning
- Describe how a CSP can be solved by using search.
    - Do DFS, doing 1 assignment per step. Continue until a goal state is found.
    - If n variables, b = (n-i)d at depth i, and n!d^n leaves
    - **Called backtracking search** for CSPs with single-variable assignments
- Does the order of variable assignments matter?
    - No, they are commutative
- What is the size of the search space?
    - n!d^n
- Describe the backtracking search algorithm.
    - **Depth-first search for CSPs with single-variable assignments**
    - **Backtracking search is the basic uninformed algorithm for CSPs**
    - **Can solve n-queens**
    - ***ALGO-***
    - Given CSP and an assignment of values to variables (initially **NONE**). If all variables have a value assigned, it returns the assignment as the solution.

    - Otherwise, it selects an unassigned variable for expansion. The children of the current node are all possible value assignments to the selected variable. For each child, it checks if the assigned value violates any constraints. If it does, it moves onto the next child. If not, then it adds the value to the assignments list, and continues recursively.

How can you improve the backtracking search?

- Select the most beneficial variable to assign a value to it. (By using a heuristic).
- Figure out in which order the values should be tried.
- Detecting inevitable failure early.
- Take advantage of problem structure.

- What heuristics can be used to improve the search process?
    - Minimum value remaining (MRV)
    - Degree
    - Least constraining value (LCV)
    - Forward checking
    - Arc consistency
- How do we select which variable to assign next?
    - **MRV**, the variable with the **fewest legal values remaining**, because it is the one most likely to have no options left if other variables are filled first.
    - **Degree** is used as a **tie-breaker**, where the variable with the most constraints on remaining variables (constrains more neighbors) is chosen
- How do we decide on which order to try values?
    - LCV, the value that limits the other variables the least
    - After the variable is selected, choose the least constraining value. the value that rules out the fewest values in the remaining variables
    - This value minimizes the limitations in the future.
- How can we detect inevitable failure early?
    - Forward checking. Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values. propagates information from assigned to unassigned variables, but **doesn't provide early detection for ALL failures:**
    - **detect inconsistencies that guarantee later failure**
    - **Constraint propagation r**epeatedly enforces constraints locally– Arc consistency
- Explain **arc consistency.**
    - An **arc** is a directed connection between two variables, indicating that a constraint exists between them.
    - **AC-3 (Algorithm for Consistency 3)**, iteratively checks & enforces consistency between arcs. It works by removing inconsistent values from domains of variables until all arcs are consistent
    - For each constraint on X and Y, there are two arcs, X -> Y and Y -> X. X -> Y is consistent if and only if for every value x of X, there is some allowed y for Y. Make X -> Y consistent by removing conflict values of X. Ex, if X = b,r and Y = b, then remove b from X.

        - **reduces the search space by pruning inconsistent values, detect failures earlier, improve pruning, domain reduction**
- How can we exploit the problem structure?
    - Handle independent subproblems separately to reduce complexity
        - N variables with d values, worst case is d^n leaf nodes, exponential in n
        - Divide into n/c subproblems with c variables
        - Worst case (n/c)d^c leaf nodes, linear in n
- What do we do if the CSP has a tree structure?
    - Choose a variable as the root, and then turn the CSP graph into a tree
    - Apply arc consistency with one way arcs
    - Each node must be consistent with its parent
- What do we do if the CSP is nearly tree structured?
    - Conditioning -> instantiate the most constrained variable and prune its neighbors domains
    - Cutset conditioning -> instantiate a set of variables such that the remaining constraint graph is a tree
- What other kinds of algorithms can be used to solve a CSP?
    - Iterative algorithms
        - Hill climbing, simulated annealing can be used by allowing complete states to have unsatisfied constraints (normally they only work with complete states, i.e. all variables assigned). Then reassign variable values until all constraints are satisfied, like moving pieces in n-queens.
        - An operator reassigns variable values, so the parts of the problem are:
            - States, operators, goal test, and evaluation

Game playing

Sam

- What is a game?
    - A competition between players to maximize their own utilities.
- What is a zero-sum game?
    - The sum of the agents' utilities does not change between the start and the end of the game (also called constant sum).
- What is a finite game?
    - Finitely many agents, actions, and states
- What is perfect information?
    - Every agent knows the current state, all of the actions, and what the actions do
    - No simultaneous actions
- What is a strategy?
    - Specifies what an agent will do in every possible situation

    - May be pure (deterministic) or mixed (probabilistic)
- What is utility in the context of a game?
    - Score
- How can minimax be applied to play games?
    - Min and Max are two players, competing to maximize and minimize the utility by taking turns.
- **Explain the minimax algorithm / What is the minimax theorem?**
    - Let G be a two-person finite zero-sum game with players Max and Min
    - Then there are strategies s* and t* , and a number V_G called G's minimax value
    - Minimax takes a starting state and returns a utility value. If the state is a terminal state, then it returns the utility for Max. Otherwise, if it is Max's move, it recursively does Minimax with each of Max's possible actions in that state, returning the maximum utility value it gets back. If it is Min's move, it does Minimax with each of Min's possible actions, returning the minimum utility it gets back.
    - If Min and Max use their optimal strategies, the end utility for both will be the minimax value of the game.
        - U(s*,t*)=V_G
        - If G is a perfect-information game, there are pure strategies s* and t* that satisfy the theorem.

- What are the properties of the minimax algorithm?
    - Number of pure strategies for max and min are <= b^(h/2) with h = tree height
    - Below ques
- What is the time and space complexity of the minimax algorithm?
    - Soundness: yes
    - Complete on finite trees (games like chess require specific rules)
    - Time: O(b^h)
    - Space: O(bh)
- Explain alpha beta pruning.
    - Let Alpha is the biggest lower bound of any ancestor. Minimax will never explore a path with a lower maximum value than alpha because max can get a better payoff from alpha **(alpha cutoff)**
    - Beta is the smallest upper bound of any ancestor. Minimax will never explore a path with a higher minimum value than beta because min can get a better payoff from beta.
    - Alpha is the best value for Max, so if V is less than alpha, Max will avoid it. Beta is the best value for Min, so if V is greater than beta, Min will avoid it.
- What is the time complexity of minimax with alpha beta pruning?
    - O(b^(m/2)), doubles solvable depth —maximum depth of the game tree
- What other approaches can be used to use minimax in real-world applications?
    - Use a cut-off test instead of a terminal test (ex. Depth limit)
    - Use evaluation function instead of utility function (estimate desirability of a position)
- What is an evaluation function?

  - ○ A function that gives an approximation of a nodes minimax value (often a weighted sum of features)
  - ○ Exact values don't matter in deterministic games because behavior is preserved under any monotonic transformation(preserves the relative order of values)
- How can we deal with non-deterministic games?
  - ○ Monte Carlo roll-outs: do a minimax search of a randomly selected subtree, randomly select some children of each node visited, call self recursively on these nodes
  - ○ Expectiminimax algorithm
- Explain the **expectiminimax algorithm**.
  - ○ If s is a terminal state, return Max's payoff. Otherwise, if s is a chance node, return the sum of its expectiminimax values of each successor. Otherwise, if it is Max's move, return the maximum value for the expect-minimax of each possible action of Max. Otherwise, do the same for Min.
  - ○ Behavior is preserved only by positive linear transformations, so we need to compute weighted averages at chance nodes.
  - ○ Hence Eval should be proportional to the expected payoff

Isti

What are legal values?

- value that can be assigned to a variable within the constraints of the problem

What is the difference between states in case of standard search problems and constraint satisfaction problems?

- In standard search problems the state is a data structure, which represents a physical configuration.
- In CSP, a state is a set of assignments of values to variables with domains.

What is a goal test?

- A goal test is **a set of constraints specifying allowable combination**s of values for variables.

What are the solutions of CSP?

- Combinations of values to variables that satisfy all the constraints.

Why does CSP use graphs to solve problems and what does a CSP graph consist of?

- It is a general solution; we can apply it for many problems. Graphs provide a fast solution.
- **<u>Nodes represent variables, edges represent constraints</u>.**

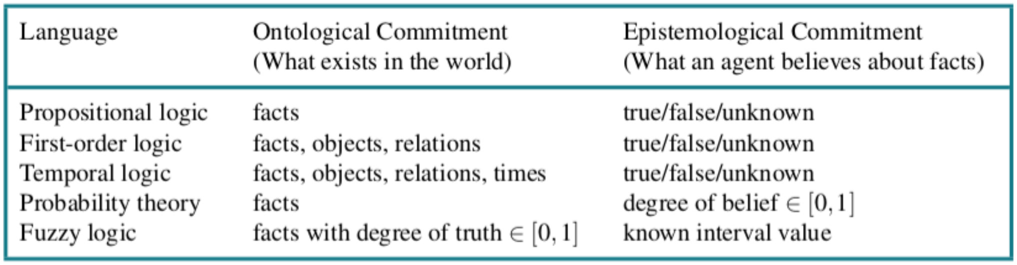
How can we take advantage of the problem structure?

- Usually, the problems can be represented as graphs. We can apply many transformations on these graphs to make the problem more feasible. For example, divide the whole problem into sub-problems.

Logical agents

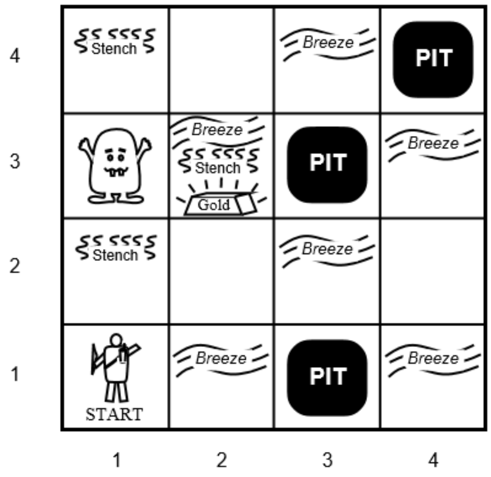
Sam

- What is logic?
    a) Formal languages for representing information such that conclusions can be drawn
    b) the science of the formal principles of reasoning
- Explain a simple knowledge-based agent?
    - Given a percept, the agent will add that percept to its KB and deduce appropriate actions. The agent then tells the KB that the action has been taken.
- What is a sentence?
    - In logic, a sentence is a proposition that can hold a logical entity true or false.
- What is a knowledge base?
    - A set of sentences in a formal language
    - It is a declarative approach to build an agent.
    - We TELL the agent what it needs to know, and then
    - We ASK the agent what to do? The answers/actions the agent returns are derived from the KB
- *How does one logic differ from another? (additional)*
    a) Logics differ in their ontological commitments and epistemological commitments. For example propositional logic commits only to the existence of facts, first-order logic commits to the existence of objects and relations and thereby gains expressive power.



Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0,1]$
Fuzzy logic	facts with degree of truth $\in [0,1]$	known interval value

- What does declarative mean?
    - Telling the knowledge base (declaring) information (tell, ask, tell, ask)
    - Pieces of syntax correspond to facts
- What are procedural KBs?
    - Tell everything at once
- What is the difference between the knowledge and implementation level?
    - Knowledge level: What the agent knows, regardless of how it is implements
    - Implementation level: Data structures in KB and algorithms that manipulate them
- What is the Wumpus world?
    - *There is a wumpus, gold bars, and pits. Pits make a breeze in adjacent squares, the wumpus makes a stench in adjacent squares, gold bars glitter in their own square.*

  - 

  - Wumpus world is a cave consisting of rooms connected by passageways.
  - One of the rooms consists of a Wumpus which will eat anyone who enters.
  - The rooms adjacent to Wumpus will have a stench, and the Wumpus can be killed by an arrow, but the agent has only one.
  - There are also pits in some rooms which will trap the agent, and the rooms adjacent to pits are breezy.
  - The objective is to navigate the rooms and find the gold, which is in one of the rooms without getting eaten or trapped.
  - PEAS:
  - P: gold +1000, death -1000, -1 per step, -10 for using arrow
  - E:
  - A: Left turn, Right turn, Backward, Forward, Grab, Release, Shoot
  - S: Breeze, Glitter, Smell
- What are the properties of the Wumpus world?
  - Fully observable? No - only local perception (room where agent is there)
  - Deterministic? Yes - outcomes are exactly specified.
  - Episodic? No - next actions follow previous actions
  - Static? Yes
  - Discrete? Yes
  - Single-agent? Yes - wumpus is a natural feature
- What are the aspects of a language?
  Logics - formal languages for representing information, such that conclusions can be drawn.
  Syntax - defines the rules for sentences in the language
  Semantics - defines the meaning of sentences (truth of a sentence in a world)
  Pragmatics = context (used in NLP)
- What is syntax?
  - Defines the rules for sentences in a language
- What are semantics?
  - rules to define meaning of sentences **(truth of a sentence wrt model in a world**)

- What is entailment?
    - KB entails A if and only if A is true in all worlds where KB is true
        - Can be checked with model checking
    - Relationship between sentences (syntax) that is based on semantics
    - Consequences of KB are a haystack; α is a needle.
    - Entailment = needle in haystack; inference = finding it

- What are well-formed sentences/formulas ?
    - PL does not allow variables, only symbols.
    - Any sentence that follows the rules of propositional logic is a well-formed formula.
    - Rules:
        - Any capital letter by itself is a well-formed formula.
        - Any WFF can be prefixed with $\neg$.
        - Any two WFF's can be joined together with the logical connectivities such as $\lor, \land, \Longrightarrow$ and enclosed in parentheses.

- What is a **clause**?
    - A Disjunction of literals, and conjunction of literals is **term**

- What is inference?
    - Deriving new sentences from KB based on what it has been told previously.
    - A can be derived from KB by inference procedure like Model checking
    - Consequences of KB are **a haystack; A is a needle.**
    - Entailment = needle in haystack; inference = finding it
- **What is the difference between entailment and implication?**
    - It is a matter of scope
    - An implication is something that may be true or false, depending on which truth assignment you're considering at the moment, whereas an entailment is a statement about all truth assignments.
    - Implication depends on an individual model (assignment of truth values) while entailment is for all models (all possible assignments of truth values)
    - KB |= A iff KB => A is valid (valid means true in every world)
    - If every assignment of values to KB => A is true, then KB |= A

- **Logical equivalence and bi-implication?**

    Logical equivalence is bi-entailment. Implication and entailment have a different scope,
- What is a **model**?
    - A formally structured world in which truth can be evaluated
    - An assignment of truth values to symbols such that the sentence is true
- What is model checking?

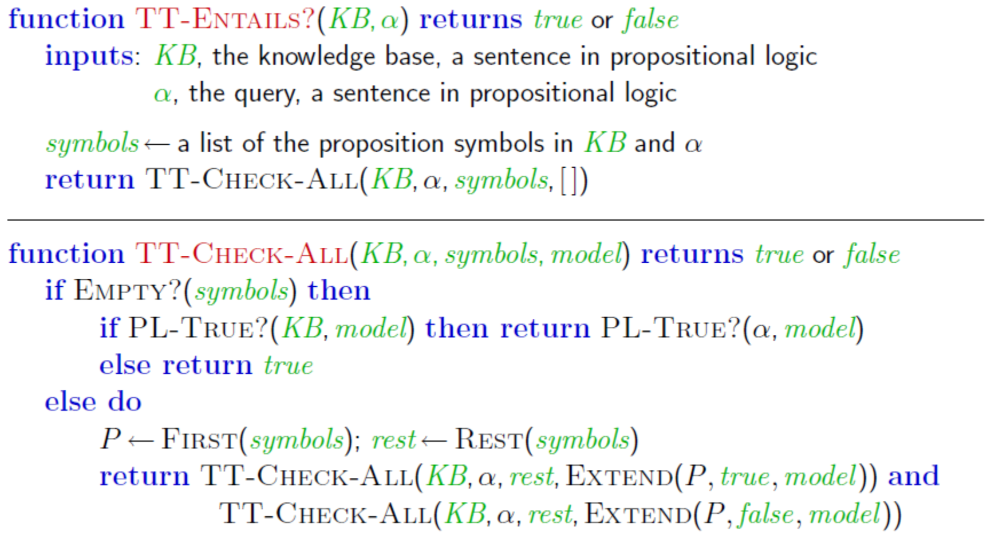
- Inference using truth table
- If α is true wherever KB is true in TT, then KB entails α
- Each row in TT is a potential model.
- sound and complete if there are finite models

- What is soundness?
  - i is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \vDash \alpha$, guarantees that only valid conclusions/entailed sentences can be derived from true premises

- What is completeness?
  - i is complete if whenever $KB \vDash \alpha$, it is also true that $KB \vdash_i \alpha$, ensures that all valid conclusions/entailed sentences can be derived from true premises.
- What are the logical connectives of PL?
  - ¬, ⋀, v, →,↔
  - Negation, conjunction, disjunction , implication, bidirectional implication

- Explain propositional logic syntactic symbols!
  - · Negation: if S is true, neg(S) is false
  - · Conjunction: con(S1, S2) is true, if S1 and S2 are both true
  - · Disjunction: dis(S1, S2) is true, if S1 or S2 true
  - · Implication: S1 => S2 is false, if S1 is true and S2 is false. Other cases, it is true.
  - · Biconditional: is true, if S1 => S2 is true and S2 => S1 is true.
- What does "propositional logic is declarative" mean?Do you know any other approaches? (procedural approach, learning based approach)
  a. You declare an object to be true or false, but not both or in between.
- What is the simplified form of A => B?
  - -A or B — Implication Elimination
- What is De Morgan's law?
  - -(A or B) = (-A and -B)
  - -(A and B) = (-A or -B)
- What is contraposition?
  - A=> B = -B => -A
- Explain Enumeration?
  - Enumeration is simply exploring all the possible models in the knowledge base.
- **Explain inference by enumeration.**
  - Depth-first enumeration of all models is sound and complete
  - O(2 n) for n symbols; problem is co-NP-complete
  - Given a KB and a query, alpha**, TT-entails** makes a list of all the symbols. It then calls **TT-check-all,** which uses KB, alpha, symbols, and an empty model. If the **list of symbol**s is empty, it checks if **the model is true** for the KB. If so, it returns whether the model is true for alpha. Otherwise, it returns true. If there are symbols in the list of symbols, it takes the first

symbol from the list of symbols, and extends the model. The new model, KB, alpha, and remaining symbols are then used in a recursive TT-check-all call.

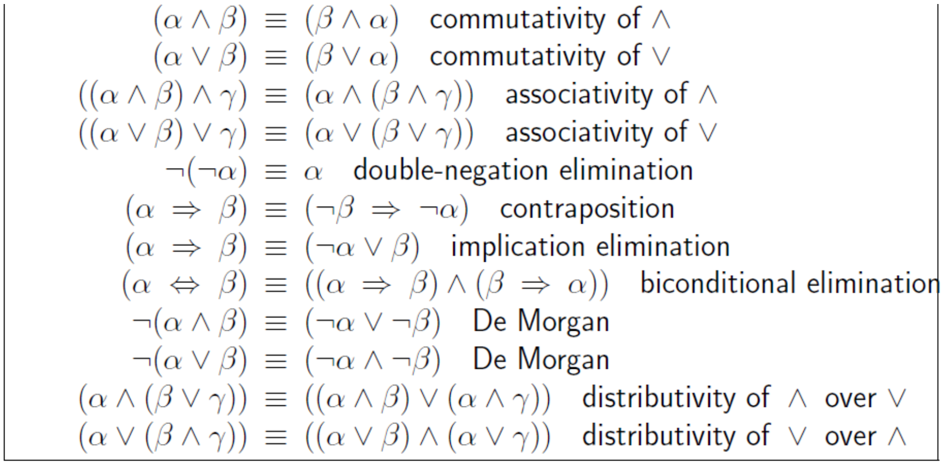


```
function TT-ENTAILS?($KB, \alpha$) returns $true$ or $false$
   inputs: $KB$, the knowledge base, a sentence in propositional logic
           $\alpha$, the query, a sentence in propositional logic

   $symbols \leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
   return TT-CHECK-ALL($KB, \alpha, symbols, [\,]$)
```

```
function TT-CHECK-ALL($KB, \alpha, symbols, model$) returns $true$ or $false$
   if EMPTY?($symbols$) then
      if PL-TRUE?($KB, model$) then return PL-TRUE?($\alpha, model$)
      else return $true$
   else do
      $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
      return TT-CHECK-ALL($KB, \alpha, rest$, EXTEND($P, true, model$)) and
             TT-CHECK-ALL($KB, \alpha, rest$, EXTEND($P, false, model$))
```

- **What does logical equivalence mean?**
    - Two sentences are logically equivalent if they have the same truth values for every possible interpretation of truth value assignments.
    - α ≡ β if and only if α |= β and β |= α



$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

- What is a valid sentence?
    - A sentence that is true
        - KB |= A IF AND only if KB => A is valid in all models
- What is the deduction theorem?
    - Known as implication introduction rule or the conditional proof
    - if you can prove a sentence A => a sentence B, then you can conclude that if A is true, then B is also true
    - Validity is connected to inference via the Deduction Theorem
    - KB |= alpha if and only if KB => alpha is valid
    - For any sentences α and β, α ⊨ β if and only if the sentence (α ⇒ β) is valid

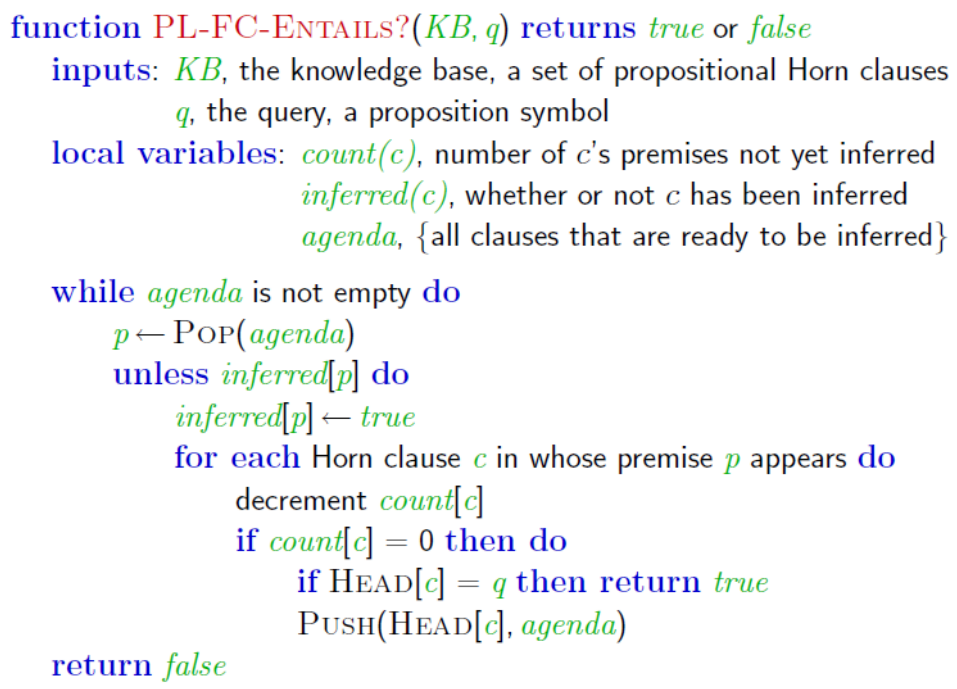
- What is a satisfiable sentence?
  - A sentence that is true in at least one model — eg A ⋁ B, C
- What is an unsatisfiable sentence?
  - Known as contradiction, A sentence that is true in no models — eg A ⋀ -A
- How is satisfiability connected to inference?
  - KB |= alpha if and only if (KB ⋀ -alpha) is unsatisfiable
- What is “reductio ad absurdum”?
  - method of proof by contradiction where one assumes the negation of what is to be proven and demonstrates that it leads to a contradiction, concluding that original statement must be true
  - Prove that the opposite of what you want to prove is impossible. To prove A, show that -A is impossible/contradictory.
- Explain **Horn Form?**
  - Formula is said to be in Horn form if it is a ***conjunction (AND) of literals***, where at most one literal is positive
  - Where ***KB = Conjunction of Horn Clauses***
  - Horn Clause = follows the Horn form.
    - Horn clauses can be written in the form:
    - H :- L1 ⋀ L2 ⋀ ... ⋀ Ln
    - Horn clause consists of a head and a body, represented as an implication
    - The head of a Horn clause is a single positive literal, and the body is a **conjunction of literals.**
    - A sentence consisting of a single positive literal, such as $L_{1,1}$, is called a Fact.
    - $\frac{\alpha_1, \ldots, \alpha_n \quad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$
  - Inference with Horn clauses can be done through Forward chaining and Backward chaining algorithms, which are very natural and run in linear time.
  - Eg $\neg C \vee \neg B \vee A$ becomes $C \vee B \rightarrow A$ is a combination of two Horn clauses
- What is the Definite clause?
  - Disjunction of literals of which exactly one is positive.
  - All definite clauses are Horn clauses
  - Horn clauses with **no positive literals**; these are called goal clauses
- What is Modus Ponens?
  - $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$
  - Whenever any sentence of the form α ⇒ β and α are given, then the sentence β can be inferred.
  - **Rule of inference**, If in our knowledge base there are α1, α2, …, αn, and we can conjunct them and implicate a sentence β, then sentence β is true.
- *When is a theorem proving more efficient than model checking? (additional)*
  a) If the number of models are large but the proof is short.

  - b) A proof can be more efficient because the proof can ignore irrelevant propositions no matter how many are there.
- *Is modus ponens sound and complete ?(additional)*
  - a) By considering the general definition of modus ponens and the possible truth values of α and β , one can easily show that modus ponens is sound. Using the same statement we can not say that modus ponens is complete (i.e for every truth value).
- *Explain the monotonicity of logical systems? (additional)*
  - a) Monotonicity says that the set of entailed sentences can only increase as information is added to the knowledge base.

Premise refers to a sentence or clause in the KB that is used as a basis for inference. It is a statement or condition that serves as the starting point for reasoning or deduction.

- **What is forward chaining?**
  - ○ Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until the query is found
- **Explain the forward chaining algorithm in PL**
  - ○ While the **agenda (all clauses ready to be inferred)** is not empty,
    - ■ **remove** the first clause, p. If the agenda is empty and the **while loop** ends, return false.
    - ■ If p is not inferred then infer it. For each Horn clause in which p appears, decrement the amount of Horn clauses. If there are no Horn clauses left, then if the **head** of the Horn clauses is the **query**, **return true**. **Push the head** of the Horn clauses to the agenda.
  - ○ FC is an example of a general form of **data-driven reasoning and is a bottom-up approach**.
  - ○ It begins from known facts (positive literals) in KB
  - ○ DETAIL-

    It determines if a single proposition symbol q - the query is entailed by a KB of definitive clauses.
  - ○
  - ○ If all the premises of an implication are known, then its conclusion is added to the set of known facts.
  - ○ For example, if L1,1 and Breeze are known and (L1,1 Λ Breeze) ⇒B1,1 is in the knowledge base, then B1,1 can be added.
  - ○ This process continues until the query q is added or until no further inferences can be made
  - ○ The count table keeps track of how many premises of each implication are not yet proven, and is reduced by one when a premise is appeared.
  - ○ If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda.
  - ○ Loops are avoided by keeping track of processed symbols.

- 

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional Horn clauses
          q, the query, a proposition symbol
  local variables: count(c), number of c's premises not yet inferred
                   inferred(c), whether or not c has been inferred
                   agenda, {all clauses that are ready to be inferred}

  while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
      inferred[p] ← true
      for each Horn clause c in whose premise p appears do
        decrement count[c]
        if count[c] = 0 then do
          if HEAD[c] = q then return true
          PUSH(HEAD[c], agenda)
  return false
```

- **What is backward chaining?**
    - It is a form of goal-driven reasoning and is a top-down approach.
    - Work backwards from the query q by chaining through conclusions to find out facts that supports the query
    - to prove q by BC,
        - check if q is known already, or
        - recursively call BC to prove all premises of some rule concluding q
    - Avoid loops: check if new subgoal is already on the recursion stack
    - Avoid repeated work: check if new subgoal
        - has already been proved true, or
        - has already failed
- What is **CNF? What is DNF?**
    - If clauses in a sentence are expressed as a conjunction where each clause is a disjunction of literals, then it is called **Conjunctive Normal Form**.
        - Clause1 ⋀ Clause2 ⋀ Clause3
        - $(A \lor \neg B) \land (B \lor \neg C \lor D)$
        - Literals are an atomic formula or its negation (A, -A)
    - If clauses in a sentence are expressed as a disjunction where each clause is a conjunction of literals, then it is called **Disjunctive Normal Form.**
        - Clause1 ⋁ Clause2 ⋁ Clause3
        - $(A \land \neg B) \lor (B \land \neg C \land D)$
- What time do forward and backward chaining run in?
    - Linear
- When is it useful to use backward chaining and when is it useful to use forward chaining?
    - **FC is data-driven**, good for automatic, unconscious processing (object recognition)

  - **BC is goal driven**, good for problem solving (finding an answer to a question)
  - BC complexity can be much less than linear in size of KB
- Are FC and BC complete?
  - Yes, for Horn clauses
- What is **Modus Ponens**?
  - $\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$
  - Whenever any sentence of the form α ⇒ β and α are given, then the sentence β can be inferred.
  - Rule of inference, If in our knowledge base there are α1, α2, …, αn, and we can conjunct them and implicate a sentence β, then sentence β is true.
- How to convert to **CNF**?
  - Replace α ⇔ β with(α ⇒ β) ∧ (β ⇒ α).
  - Replace α ⇒ β with ¬α ∨ β
  - Move negation inwards using De Morgan's rules and double negation
  - Apply distributivity laws (∨ over ∧) and flatten:
- What is **Resolution**?
  - It is used for a KB, which is not in Horn Form
  - Inference procedure based on resolution uses proof by contradiction (reductio ad absurdum)
  - To show that $KB \vDash \alpha$, we show that $(KB \wedge \neg\ \alpha)$ is unsatisfiable.
  - First $(KB \wedge \neg\ \alpha)$ is converted into CNF
  - Then, the resolution rule is applied to the resulting clauses. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
  - The process continues until one of two things happens:
    - there are no new clauses that can be added, in which case KB does not entail α or,
    - two clauses resolve to yield the empty clause, in which case KB entails α
  - The empty clause—a disjunction of no disjuncts—is equivalent to False because a disjunction is true only if at least one of its disjuncts is true.
  - Moreover, the empty clause arises only from resolving two contradictory unit clauses such as P and ¬P.
  - Resolution is sound and complete for propositional logic
  - Resolution is equal to Modus Ponens:

clauses from CNF: $A \vee \neg B \qquad B \vee \neg C \vee \neg D$

rewrite as implications: $\neg A \Rightarrow \neg B \qquad \neg B \wedge C \Rightarrow \neg D$

apply modus ponens: $\neg A \wedge C \Rightarrow \neg D$

rewrite as clauses: $A \vee \neg C \vee \neg D$

  -
  - P v Q … -P, then Q is True

- Explain the **resolution algorithm**.
  - For each pair of clauses in the set of all clauses, resolve them. If the resolvent is then empty, return true. Add all resolvents to new. Once all the clause pairs have been tested, if new is already in the clauses, return false. Add the new clauses to the set of all clauses, and repeat.
- *Is resolution sound and complete for PL?*
  - *Yes*
  - When coupled with a complete search algorithm, the resolution rule yields a sound and complete algorithm for deciding the *satisfiability* of a propositional formula, and, by extension, the validity of a sentence under a set of axioms."

- *When does the resolution algorithm stop?*
  - The process continues until one of two things happens:
    - there are no new clauses that can be added, in which case KB does not entail α or,
    - two clauses resolve to yield the empty clause, in which case KB entails α

Isti

How can an agent be viewed from the knowledge base perspective?

· It can be viewed from the knowledge level. The way how was it implemented is not important, only the knowledge.

· It can be viewed from the implementation level. In this case, the way how was it implemented matters: data structure and algorithms are important.

What are the two main branches of proof methods?

· Application of inference rules and model checking.

What is a proof?

· It is the application of a sequence of inference rules.

FOL

Sam

Query refers to the statement or proposition that you want to determine the truth value of based on the given knowledge base.

- **What is FOL or Predicate Logic?**
  - First-order logic is a formal language whose syntax and semantics are built around objects and relations.
  - First-order logic:
  - – syntax: constants, functions, predicates, equality, quantifiers

- What are the pros and cons of PL?
  - PL is declarative: pieces of syntax correspond to facts
  - PL allows partial/disjunctive/negated information
  - PL is compositional: meaning of sentences is derived from meaning of symbols
  - Meaning in PL is context independent
- **What is a con of propositional logic?**
  - Propositional logic has very limited expressive power E.g., cannot say “pits cause breezes in adjacent squares” except by writing one sentence for each square.

- What is the main difference between propositional logic and first-order logic?
  - Propositional logic represents the world by facts.
  - First order logic represents the world by:
    - **Objects:** people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
    - **Relations**: can be **unary relations** can represent **properties such as** *red, round, bogus, prime*, multistoried, **binary represent father(x,y)** or more general n-ary **relations** involve more than 2 obj and represent complex rlp
      - They return truth values
    - **Functions**: relations in which **there is only one “value”** for a given “input.” *father of, best friend, third inning of*, one more than, beginning of . . .
      - They return value types, and act on 1+ items

- What are the basic elements of FOL?
  - Constant symbols : KingJohn, 2, UniversityofMaryland, .. (relates to objects)
  - Predicate symbols: Brother, >, . . . (relates to relations - return truth values)
  - Function symbols: Sqrt, LeftLegOf , . . . (returns value types)
  - Variable symbols: x, y, a, b, . . .
  - Connectives: $\wedge$ $\vee$ $\neg$ $\Rightarrow$ $\Leftrightarrow$
  - Equality: =
  - Quantifiers: $\forall$ (for all) , $\exists$ (for some)
  - Punctuation: ( )
- What is an **atomic sentence** in FOL?
  - Predicate of terms
  - Predicate (term_1, ... , term_n) or $\text{term}_1 = \text{term}_2$
  - Term = function(term_1, ... , term_n) or constant or variable

    - An atomic sentence (or atom for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as
      Eg: Brother(Richard, John).
- What are complex sentences in FOL?
    - made from atomic sentences using connectives
- What is the difference between relations and functions in FOL?
    - A relation act on 2+ values, a function modifies 1+ variables
    - Functions return value types rather than truth values.
- How is FOL different from PL?
    - More expressive power bc it has quantifiers
- What is the difference between a sentence in FOL and PL?
    - In PL, an atomic sentence is a symbol, whereas in FOL it is a predicate of terms.
- What is a term in FOL?
    - function(term_1,...,term_n) or constant or variable
- What is the difference between a term and an atomic sentence in FOL ?
    a) A term is a logical expression that refers to an object. Constant symbols are terms. An Atomic sentence in FOPL is Predicate of terms.
- What is a model in FOL?
    - It is a mapping of real world objects to symbols
    - A model is a pair, $M = (D, I)$
        - D - a domain - a world of objects
        - I - an interpretation - meaning of symbols (relations, function, objects)
    - An atomic sentence, Predicate (term$_1$, …. , term$_n$), is true in M iff the objects referred to by term$_1$, …. , term$_n$ are in relation referred to by predicate.
- What is the issue with enumeration while computing entailment in FOL?
    - The functions in the FOL lead to infinite recursions making it unable to compute.
- *Can we move from PL to FOL and FOL to PL easily?*
    - Moving from PL to FOL involves adding additional expressive capabilities to account for quantification, variables, and relations. This expansion typically requires introducing new symbols, such as constants, functions, and predicates, along with quantifiers (universal and existential)
    - Moving from FOL to PL is more challenging because FOL is a more expressive logic that allows for quantification and relations, which are not directly representable in PL.
- What is a domain?
    - A domain consists of at least one object and relations among the objects if there are more than one object.
- What is an interpretation?

  interpretation refers to the assignment of meaning or semantics to the various components of the logical language

  o Constant symbols -> objects in the domain.

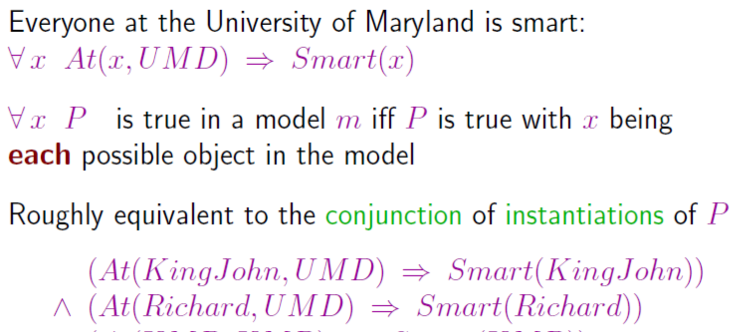
  o Predicate symbols -> relations over objects in the domain.

- Function symbols -> functional relations over objects in the domain.

When is a sentence true in a model?

· An atomic sentence is true in a model, if and only if the objects referred to by (term1,…termn) are in the relation referred to by predicate.

- How does one evaluate truth in FOL?
    - Model checking doesn’t work because many worlds are possible in FOL
- What is universal quantification?
    - ∀ <variables> <sentence>
    - Is true in a model if and only if the sentence is true for every possible object in the model
    - **Equivalent to the conjunction of instantiations of the sentence**



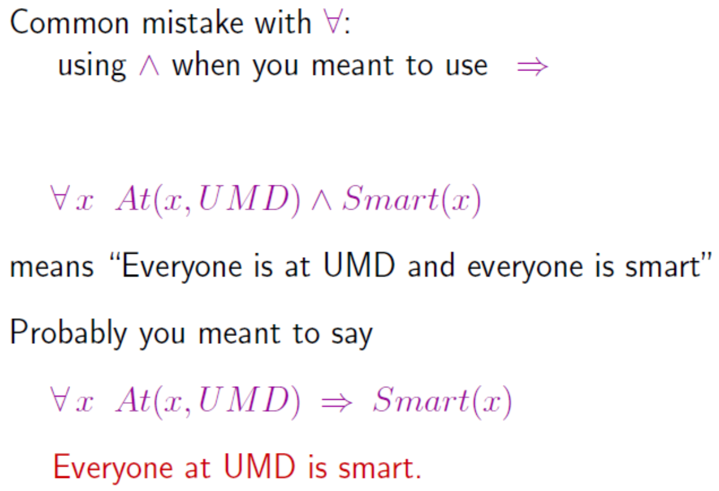
Everyone at the University of Maryland is smart:

$\forall x \;\; At(x, UMD) \;\Rightarrow\; Smart(x)$

$\forall x \;\; P$ is true in a model m iff P is true with x being **each** possible object in the model

Roughly equivalent to the conjunction of instantiations of P

$$(At(KingJohn, UMD) \;\Rightarrow\; Smart(KingJohn))$$
$$\wedge\; (At(Richard, UMD) \;\Rightarrow\; Smart(Richard))$$



Common mistake with $\forall$:

using $\wedge$ when you meant to use $\Rightarrow$

$\forall x \;\; At(x, UMD) \wedge Smart(x)$

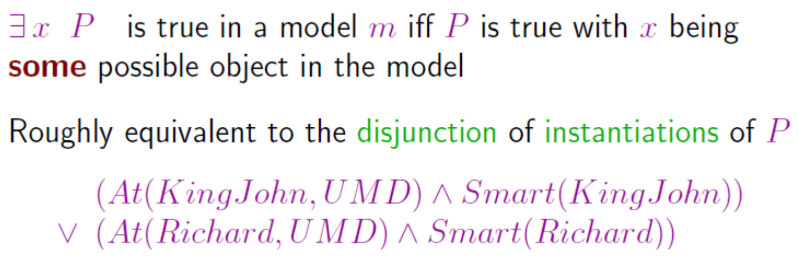
means “Everyone is at UMD and everyone is smart”

Probably you meant to say

$\forall x \;\; At(x, UMD) \;\Rightarrow\; Smart(x)$

Everyone at UMD is smart.

- What is existential quantification?
    - ∃ <variables> <sentence>
    - A sentence is true in a model if and Only if the sentence is true with some object in the model.
    - **Equivalent to the disjunction of instantiations of the sentence**

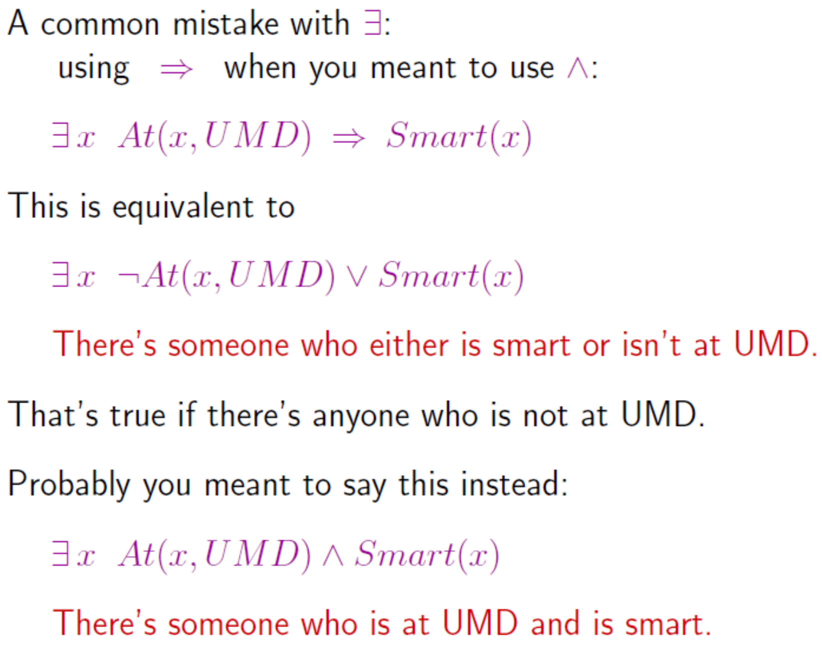


$\exists x\ \ P$ is true in a model m iff P is true with x being **some** possible object in the model

Roughly equivalent to the disjunction of instantiations of P

$$\begin{aligned}&(At(KingJohn, UMD) \land Smart(KingJohn))\\ \lor\ &(At(Richard, UMD) \land Smart(Richard))\end{aligned}$$

○



A common mistake with $\exists$:
using $\Rightarrow$ when you meant to use $\land$:

$$\exists x\ \ At(x, UMD) \Rightarrow Smart(x)$$

This is equivalent to

$$\exists x\ \ \neg At(x, UMD) \lor Smart(x)$$

There's someone who either is smart or isn't at UMD.

That's true if there's anyone who is not at UMD.

Probably you meant to say this instead:

$$\exists x\ \ At(x, UMD) \land Smart(x)$$

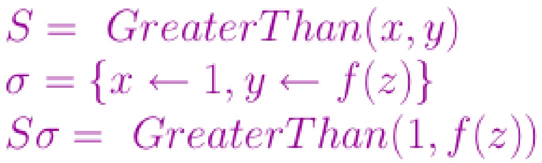
There's someone who is at UMD and is smart.

○

What are the properties of the quantifiers?

- The order of universal quantifiers doesn't matter if there are only universal quantifiers.
- The order of existential quantifiers doesn't matter if there are only existential quantifiers.
- The order matters if the universal and existential quantifiers are mixed.
- Duality: they can be expressed by using the other (negation).

● Explain substitutions?
a)substitution is a set of variable binding (replacing a variable by a term)
Eg: Given a sentence S and a substitution σ, Sσ (postfix notation) is the result of applying σ to S



$$\begin{aligned}S &= \ GreaterThan(x, y)\\ \sigma &= \{x \leftarrow 1, y \leftarrow f(z)\}\\ S\sigma &= \ GreaterThan(1, f(z))\end{aligned}$$

What is diagnostic rule?

- Infer cause from the effect.

What is causal rule?

· Infer effect from the cause.

What is a result function?

· Situations are connected by a result function.

- What is quantifier duality?
  - *For some* and *for all* can be expressed using the other
  - Each quantifier can be expressed using the other.

    $$\forall x \;\; Likes(x, IceCream) \qquad \neg\exists x \;\; \neg Likes(x, IceCream)$$

    $$\exists x \;\; Likes(x, Broccoli) \qquad \neg\forall x \;\; \neg Likes(x, Broccoli)$$
  -
- Explain Equality?
  1. a) It states that both terms refer to the same object in the domain.
  2. b) term 1 = term 2 is true under a given interpretation if and only if term 1 and term 2 refer to the same object.

     Eg:

     x = 2 and ∀x x(Sqrt(x), Sqrt(x)) = x are satisfiable (true under at least one interpretation)

     2 = 2 is valid (true in every interpretation)
- What is a quantifier and equality in FOL? How are they useful?
  - Quantifiers allow variables to be abstracted (for some, for all)
  - Quantifiers allow us to make generalizations and reason about groups of objects rather than specific individuals.
  - Equality says that if term1 = term2 if and only if both terms refer to the same object
  - Equality is important for making assertions and drawing conclusions based on the identity of objects
- What is a predicate?
  - A predicate is a relation between objects
- What are the properties of quantifiers?
  - Scope: Each quantifier has a scope, defining the part of the sentence it applies to.
  - Binding: Quantifiers bind variables, giving them meaning and restricting their scope.
  - Variable Substitution: Quantifiers allow for the substitution of variables with specific objects or terms.
  - Quantifier Order: The order of quantifiers can impact the interpretation of a sentence.
  - Duality: Quantifiers have a duality property, allowing them to be expressed using the other quantifier and negation.
  - Quantifier Restrictions: There may be restrictions on the domain of quantification or the types of objects that can be quantified over.
- What is the difference between a term and an atomic sentence?
  - A term is a constant, a variable, or a function of other terms, and an atomic sentence is a **predicate of terms** in FOL

- What are objects and relations in FOL?
    - Objects are entities or individuals in the world, represented using constant symbols, such as "a", "b", "c", or specific names like "John" or "Mary".
    - Relations can be unary, binary, or n-ary, depending on the number of objects involved, indicate how objects are related or interact with each other. Unary relations apply to a single object, binary relations involve two objects, and n-ary relations involve more than two objects.
- What is substitution in FOL?
    - substitution is a set of variable binding (replacing a variable by a term)
    - Ex. A = {x <- 1, y <- f(z)}
    - Substitutions are performed simultaneously
    - Given a sentence S and a substitution sigma, S_sigma is the result of applying sigma to S

$$S = \ GreaterThan(x, y)$$
$$\sigma = \{x \leftarrow 1, y \leftarrow f(z)\}$$
$$S\sigma = \ GreaterThan(1, f(z))$$

- Explain Diagnostic and Causal properties of FOL?
    1. Diagnostic rule—infer cause from effect
       Eg: effect/ observation/ symptom - - -> cause

$$\forall y \ \ Breezy(y) \ \Rightarrow \ \exists x \ \ Pit(x) \wedge Adjacent(x, y)$$

    2. Causal rule—infer effect from cause
       Eg: cause - - -> effect/ observation/ symptom

$$\forall x, y \ \ Pit(x) \wedge Adjacent(x, y) \ \Rightarrow \ \ Breezy(y)$$

    3. With the help of diagnostic, causal rules we can attain generalization (this was not possible in PL) in FOL. Combining the ideas of these two we can get the definition of a predicate.
       Breezy =

$$\forall y \ \ Breezy(y) \ \Leftrightarrow \ [\exists x \ \ Pit(x) \wedge Adjacent(x, y)]$$

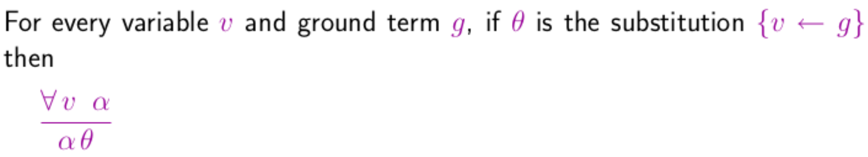
- What is the diagnostic rule?
    - Infer cause from effect
- What is the causal rule?
    - Infer effect from cause
- Are facts eternal?
    - No, they only hold for situations
- What is situation calculus?
    - Situation calculus conventions to represent actions and change in FOL.
    - Can formulate and solve planning problems

    - It adds a situation argument to each non-eternal predicate, ex. Holding(Gold,s) with s being the situation. Result(a,s) is the result of doing action a in situation s.
- What is the frame axiom?
    - Describes non-changes due to action
- What is the effect axiom?
    - Describes changes due to action
- What is the frame problem?
    - Find a way to handle non-change
- What is the qualification problem?
    - True descriptions of real actions require endless caveats
- What is the ramification problem?
    - Real actions have many secondary consequences

Inference in FOL

Sam

- What is universal instantiation?
    - Every instantiation of a universally quantified sentence is entailed by it

      For every variable v and ground term g, if θ is the substitution $\{v \leftarrow g\}$ then

      $$\frac{\forall v \;\; \alpha}{\alpha\theta}$$

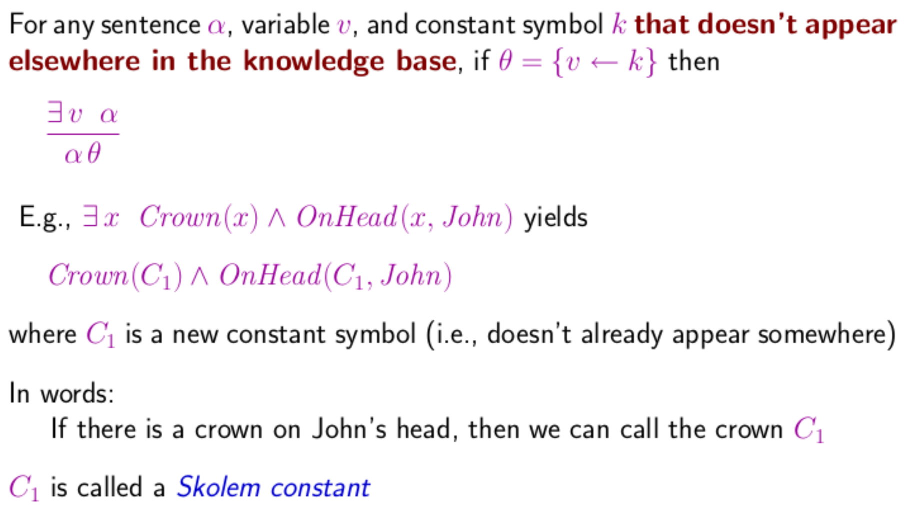
        -
        - For all x, if x is a king and x is greedy, then x is evil represented as
            - ∀x (King(x) ∧ Greedy(x) → Evil(x)).
            - UI, we can substitute a specific ground term, let's say "John," for the universally quantified variable "x." This results in the new sentence: "If John is a king and John is greedy, then John is evil" represented as King(John) ∧ Greedy(John) → Evil(John).

    - It says that we can infer any sentence obtained by substituting a ground term (a term without variables) for a universally quantified variable.

- What is existential instantiation?
    - It replaces an existentially quantified variable with a single new constant symbol.

    - For any sentence alpha,existentially quantified variable v, and constant symbol k **that doesn't appear elsewhere in KB** (Skolem constant), if theta = {v <- k}, then
        - For some v alpha infers alpha theta

        - Some sentence alpha with the variable v infers the sentence alpha with k substituted for v
        - Ex. for some x crown(x) and OnHead(x, John) yields
            - Crown(C1) and OnHead(C1, John) where C1 is a Skolem constant



For any sentence α, variable v, and constant symbol k **that doesn't appear elsewhere in the knowledge base**, if $\theta = \{v \leftarrow k\}$ then

$$\frac{\exists v \ \ \alpha}{\alpha\theta}$$

E.g., $\exists x \ \ Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

where C_1 is a new constant symbol (i.e., doesn't already appear somewhere)

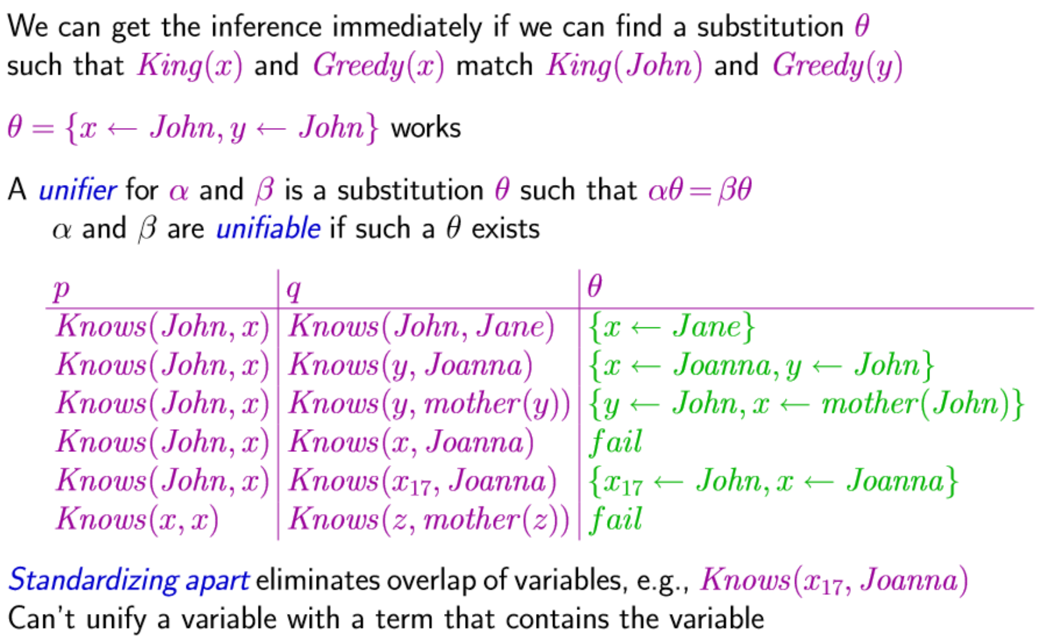
In words:
If there is a crown on John's head, then we can call the crown C_1

C_1 is called a *Skolem constant*

- What is a Skolem constant?
    - A constant that does not appear elsewhere in KB
    - The new constant C1 that replaces an existentially quantified variable during existential instantiation that does not appear elsewhere in the knowledge base
- What are UI and EI used for?
    - UI can be used several times to add new sentences where the new KB is logically equivalent to the old KB
    - EI can be applied once to to replace an existential sentence. The new KB is not equivalent to the old KB, but it is satisfiable if and only if the old KB was satisfiable.
- How can FOL be reduced to propositional inference (**propositionalized**)?
    - Instantiate universal sentences in all possible ways
    - This *involves instantiating universal quantifiers and creating ground instances* of the sentences.
    - Propositionalization involves replacing variables - constants and functions - corr. ground terms. The resulting propositionalized sentences consist of atomic propositions that can be evaluated using truth values.
- What is the problem with instantiating universal sentences in all possible ways? / What are the issues of **Propositionalization**
    - Creates many irrelevant sentences
    - Can create definitive clauses infinitely many sentences (nested function symbols)
    - It can create lots of irrelevant sentences, which are not needed for the proof.
        - With $p\ k - ary$ predicates and n constants, there are $p \cdot n^k$ instantiations.
- Explain Herbrand Theorem?
    - **provides a way to reduce (FOL) formulas to propositional logic (PL) formulas,**

    - **If a sentence alpha is entailed by an FOL KB, then it is entailed by a finite subset of the propositionalized KB.**
        - **works if α is entailed, loops if α is not entailed**
    - *To resolve the above issues with propositionalization*, this theorem is used.
    - It is a kind of depth limited search.
    - At every depth, we verify entailment and break the loop if the query is proved.
    - If α is entailed, the theorem works otherwise it will loop forever.
- **Why is entailment in FOL semidecidable?**
    - If a sentence α is entailed by an FOL KB, then it is entailed by a finite subset of the proportionalized KB
    - If a sentence α is not entailed, the inference will loop forever.

- Can every FOL KB be propositionalized to preserve entailment?
    - Yes by following
        - Eliminate biconditionals and implications
        - Move negations inwards: -For all p = for some -p, -for some p = for all -p
        - Standardize variables so each quantifier uses a different one
        - Skolemize: each existential variable is replaced by a Skolem function
        - Drop universal quantifiers
        - Distribute conjunction over disjunction
- When is a ground sentence in the new KB entailed?
    - If it was entailed by the original KB
- How is inference performed in FOL?
    - Forward Chaining: Backward Chaining, Resolution, Unification
- What is unification?
    - **finding substitutions for variables that make different logical expressions equal. Unification is used, for example, in resolution, where it helps in combining disjunctions by canceling out complementary literals**.
        - A unifier for alpha and beta is a substitution theta such that alpha theta = beta theta
    - Alpha and beta are unifiable if such a theta exists
    - Find a substitution for variables in both sentences so that the sentences become equal. This is difficult or impossible if the sentences share variables in a conflicting way. Ex. Knows(John,x) and Knows(x,Joanna), x cannot be both Joanna and John to fulfill both sentences.
    - **DETAIL**
    - Process of finding substitutions that make different logical expressions look identical.
    - The UNIFY **algorithm takes two sentences and returns a unifier for them** (a substitution) if one exists



We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\theta = \{x \leftarrow John, y \leftarrow John\}$ works

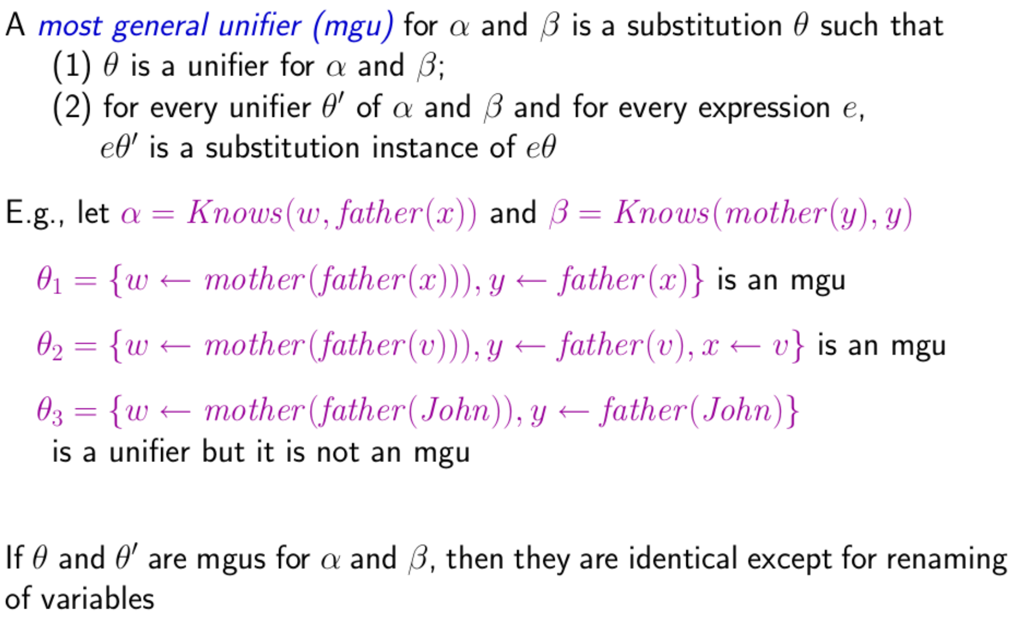
A *unifier* for α and β is a substitution θ such that $\alpha\theta = \beta\theta$
α and β are *unifiable* if such a θ exists

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x \leftarrow Jane\}$
$Knows(John, x)$	$Knows(y, Joanna)$	$\{x \leftarrow Joanna, y \leftarrow John\}$
$Knows(John, x)$	$Knows(y, mother(y))$	$\{y \leftarrow John, x \leftarrow mother(John)\}$
$Knows(John, x)$	$Knows(x, Joanna)$	$fail$
$Knows(John, x)$	$Knows(x_{17}, Joanna)$	$\{x_{17} \leftarrow John, x \leftarrow Joanna\}$
$Knows(x, x)$	$Knows(z, mother(z))$	$fail$

Standardizing apart eliminates overlap of variables, e.g., $Knows(x_{17}, Joanna)$
Can't unify a variable with a term that contains the variable

- What is standardizing apart?
  - Make a variable more specific to allow unification (turn a variable into multiple specific variables)
  - Ex. Knows(John,x) and Knows(x17,Joanna), x = Joanna, x17 = John
- Why is Standardizing apart necessary in Unification?
  - To avoid overlapping of variables.
  - It is done by Standardizing apart one of the two sentences being unified, which means renaming its variables to avoid name clashes

- Can a variable with a term that contains a variable be unified?
  - No, ex. Knows(x,x) and Knows(z,mother(z))
- What is a most general unifier (MGU)?



A *most general unifier (mgu)* for α and β is a substitution θ such that
(1) θ is a unifier for α and β;
(2) for every unifier θ' of α and β and for every expression e,
$e\theta'$ is a substitution instance of $e\theta$

E.g., let $\alpha = Knows(w, father(x))$ and $\beta = Knows(mother(y), y)$

$\theta_1 = \{w \leftarrow mother(father(x))), y \leftarrow father(x)\}$ is an mgu

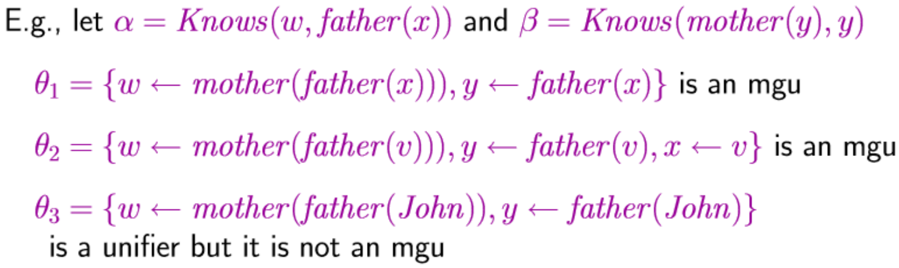
$\theta_2 = \{w \leftarrow mother(father(v))), y \leftarrow father(v), x \leftarrow v\}$ is an mgu

$\theta_3 = \{w \leftarrow mother(father(John)), y \leftarrow father(John)\}$
is a unifier but it is not an mgu

If θ and θ' are mgus for α and β, then they are identical except for renaming of variables

- An MGU is a unifier that leaves the most variable symbols remaining

- **How is most general unifier (mgu) different from unifier?**
    - Every unifiable pair of expressions has a single most general unifier (MGU) that is unique up to renaming and substitution of variables.
    - A most general unifier (mgu) for α and β is a substitution θ such that
        - (1) θ is a unifier for α and β;
        - (2) for every unifier θ’ of α and β and for every expression e, eθ’ is a substitution instance of eθ
        - If θ and θ’ are mgus for α and β, then they are identical except for renaming
        - of variables



E.g., let $\alpha = Knows(w, father(x))$ and $\beta = Knows(mother(y), y)$

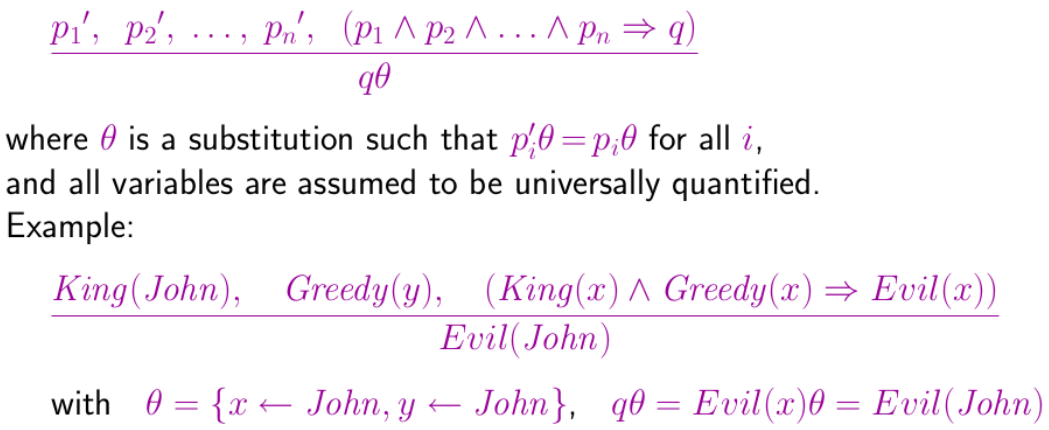
$\theta_1 = \{w \leftarrow mother(father(x))), y \leftarrow father(x)\}$ is an mgu

$\theta_2 = \{w \leftarrow mother(father(v))), y \leftarrow father(v), x \leftarrow v\}$ is an mgu

$\theta_3 = \{w \leftarrow mother(father(John)), y \leftarrow father(John)\}$
is a unifier but it is not an mgu

- How do we find an MGU?
    - Compare the expression element by element, building up a substitution along the way
    - Apply the substitution made so far. If two elements are the same after the substitution, continue. Otherwise, if one of the elements is a variable x and the other is an expression e, and if x doesn’t appear in the expression (Occur check, ex. X, mother(X)), then incorporate x= e into the substitution. Else, the process fails.
    - Runs in quadratic time (linear without the occur check)

- What is Occur check?
  - Checking if the **variable x** occurs in the **expression e** is the occur check.
- **What is Generalized Modus Ponens? Why is it needed?**
  a) **Generalized Modus Ponens** is more general than Modus Ponens in the sense that the known **facts and the premise of the implication need match only up to a substitution, rather than exactly**. On the other hand, Modus Ponens allows any sentence α as the premise, rather than just a conjunction of atomic sentences.

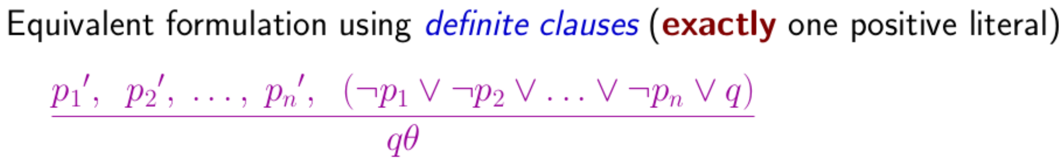


$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where θ is a substitution such that $p_i'\theta = p_i\theta$ for all i,
and all variables are assumed to be universally quantified.
Example:

$$\frac{King(John),\quad Greedy(y),\quad (King(x) \wedge Greedy(x) \Rightarrow Evil(x))}{Evil(John)}$$

with $\theta = \{x \leftarrow John, y \leftarrow John\}$, $q\theta = Evil(x)\theta = Evil(John)$



Equivalent formulation using *definite clauses* (**exactly** one positive literal)

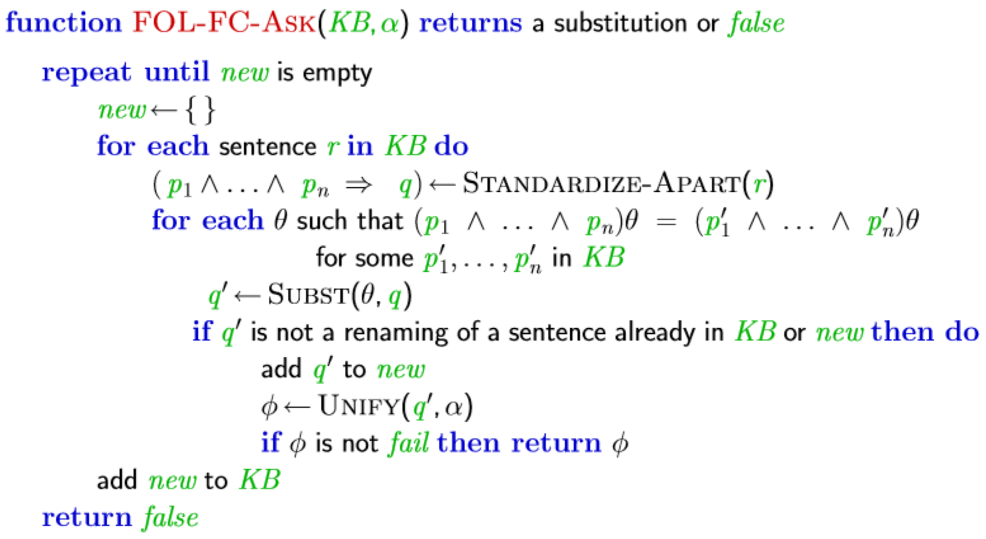
$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}',\ \ (\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee q)}{q\theta}$$

  b)
    i) p' is true
    ii) p => q
    iii) Therefore, p' => q'
    iv) ' means substitution

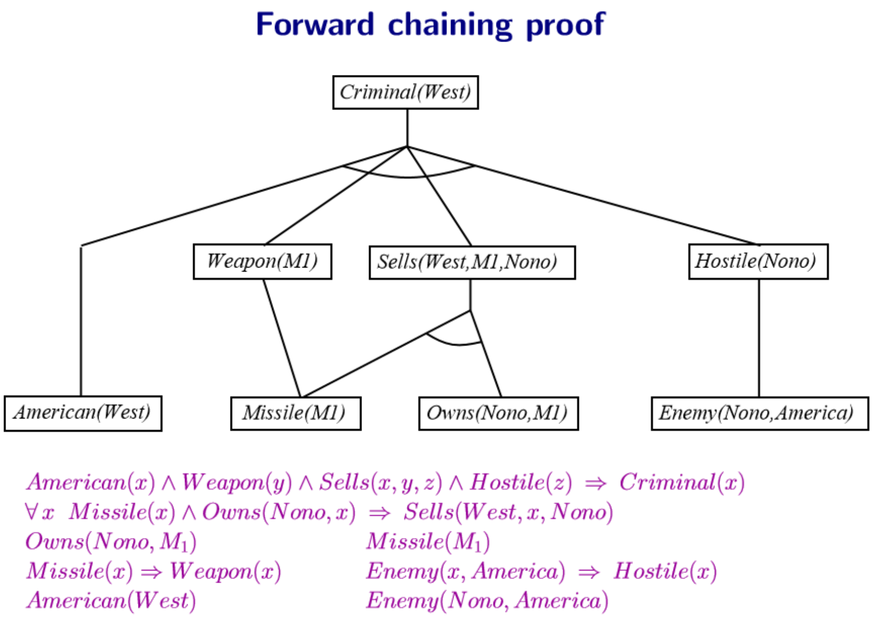
- What is a definite clause?
  - A clause with exactly one positive literal
  - For any definite clause p, universal instantiation gives p |= p theta

- Explain Forward chaining in FOL?
  - **Start with known sentences and** repeatedly apply inference rules to derive new conclusions.
  - **involves matching the antecedents (premises) of logical rules with the known facts, applying substitutions for variables, and adding the consequents (conclusions) to the knowledge base**
  - **useful for deductive databases and expert systems.**
  - **bottom-up approach**
  - **DETAIL**
  - Works only for **definitive clauses**

- Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.
- The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added.
- The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.



```
function FOL-FC-ASK($KB, \alpha$) returns a substitution or $false$
    repeat until $new$ is empty
        $new \leftarrow \{\}$
        for each sentence $r$ in $KB$ do
            $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
            for each $\theta$ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
                        for some $p'_1, \ldots, p'_n$ in $KB$
                $q' \leftarrow$ SUBST($\theta, q$)
                if $q'$ is not a renaming of a sentence already in $KB$ or $new$ then do
                    add $q'$ to $new$
                    $\phi \leftarrow$ UNIFY($q', \alpha$)
                    if $\phi$ is not $fail$ then return $\phi$
        add $new$ to $KB$
    return $false$
```



Forward chaining proof

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$

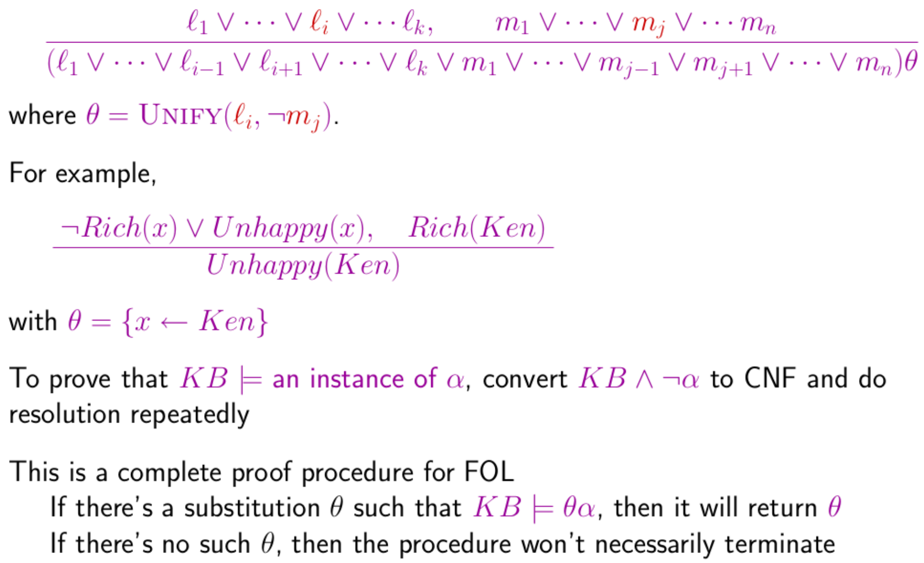
$American(West)$ $\quad$ $Enemy(Nono, America)$

- What are the properties of forward chaining?
    - Sound and complete(for first-order definite clauses)
    - May not terminate if alpha is not entailed
    - Entailment with definite clauses is semidecidable
    - Can guarantee termination if restrictions are satisfied
        - Datalog = first order definite clauses + no functions, ex. Colonel West example

        - FC terminates for Datalog in poly iterations at most *p.n^k* literals, p predicates, n no of obj, k max literals

- What is the efficiency of forward chaining?
    - Database indexing allows O(1) retrieval of known facts
    - Matching conjunctive premises against known facts is NP-hard
        - Partial fix: store partial matches in data structures such as rete networks
    - FC is used in deductive databases and expert systems

- Explain **Backward chaining in FOL?**
    - **starts with a query and recursively applies inference rules in reverse to find the premises that would lead to the goal.**
    - **The process continues until all necessary facts or conditions are found or until no more rules can be applied, which in turn prove the premises and the query.**
    - **involves matching the consequents (conclusions) of logical rules with the goal, applying substitutions for variables, and recursively working backward to find the antecedents (premises).**
    -
    - *used in logic programming and theorem proving.*
    - *top-down approach*
- What are the properties of backward chaining?
    - Incomplete due to infinite loops
        - Partial fix: check current goal against every goal on stack. Can prevent looping if there are no functions
    - Inefficient due to repeated subgoals, can be fixed by caching previous results
    - Used for logic programming
- What are Prolog systems?
    - Programming that uses BC with Horn clauses
    - Program = set of clauses of the form head:- literal_1, … literal_n.
    - Prolog programs are sets of definite clauses written in a notation somewhat different from standard first-order logic.
    - It is used primarily as a rapid-prototyping language and for symbol-manipulation task
- Explain resolution in FOL.
    - It works for any KB, not just definite clauses.
    - First, sentences are converted into CNF for FOL
        - Eliminate biconditionals and implications
        - Move negations inwards: -For all p = for some -p, -for some p = for all -p
        - Standardize variables so each quantifier uses a different one
        - Skolemize: each existential variable is replaced by a Skolem function
        - Drop universal quantifiers
        - Distribute conjunction over disjunction

  - Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.
  - first-order literals are complementary if one unifies with the negation of the other.
  - This rule is called the Binary resolution rule because it resolves exactly two literals.
  - 

    $$\frac{\ell_1 \vee \cdots \vee \ell_i \vee \cdots \ell_k, \qquad m_1 \vee \cdots \vee m_j \vee \cdots m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

    where $\theta = \textsc{Unify}(\ell_i, \neg m_j)$.

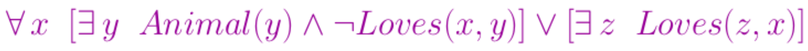
    For example,

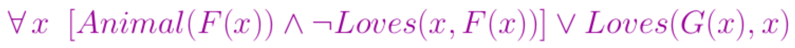
    $$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Ken)}{Unhappy(Ken)}$$

    with $\theta = \{x \leftarrow Ken\}$

    To prove that $KB \models$ an instance of α, convert $KB \wedge \neg\alpha$ to CNF and do resolution repeatedly

    This is a complete proof procedure for FOL
    If there's a substitution θ such that $KB \models \theta\alpha$, then it will return θ
    If there's no such θ, then the procedure won't necessarily terminate
  - Like PL, take two disjunctions of literals. Unify combinations of literals that cancel out given a certain substitution. Apply the substitution and the unification to combine the disjunctions.
  - The process continues until a contradiction (empty clause) is derived or no new clauses can be derived.

- Explain conversion to CNF in FOL.
  - Eliminate biconditionals and implications
  - Move negations inwards: -For all p = for some -p, -for some p = for all -p
  - Standardize variables so each quantifier uses a different one
  - Skolemize: each existential variable is replaced by a Skolem function
    - 

      $\forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ \ Loves(z, x)]$
    - 

      $\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$
  - Drop universal quantifiers
  - Distribute conjunction over disjunction
- Why FC/BC cannot handle other than definite clauses?
  - Symbols cannot be inferred if there are more than one positive literal / no positive literals.
- How is inference in FOL different from inference in PL?
  - FOL inference methods must account for quantifiers, because variables may not have objects/values assigned to them in FOL
- *Difference between inference rules in PL and FOL?*

Inference Rules	Propositional Logic (PL)	First-Order Logic (FOL)
Expressive Power	Deals with propositions and truth values	Extends PL with quantifiers and variables for more expressiveness
Quantifiers	Does not have quantifiers	Introduces quantifiers (universal and existential)
Variables and Objects	Does not have variables	Uses variables to represent objects and quantification
Model Checking	Uses truth tables for model checking	More complex due to potential infinite models in FOL
Handling Quantifiers	Not applicable as quantifiers are not present	Requires handling quantifiers and instantiating variables