UCLouvain
École polytechnique de Louvain

LINFO1361

Artificial Intelligence

Yves Deville

Nathan Flamend

Dylan Goffinet

2021-2022

Contents

1 | Artificial Intelligence

1.1 Introduction

1.1.1 What is AI ?

	Human	Rationality
Thinking	Systems that think like humans. Cognitive modeling approach	Systems that think rationally. The "Law of Tought" approach.
Acting	Systems that act like humans. The Turing Test approach.	Systems that act rationally. The rational agent approach.

Systems that think like humans

The Cognitive Modeling approach : Try to construct theories of how the human think. Synergy between computer models from Ai and experimental techniques from psychology, introspection and brain imaging.

Systems that act like humans

The Turing Test approach : Testing if a program is able to achieve human-like performance in cognitive tasks.

Systems that think rationally

The "law of Tought" approach : Require formal representation of problems and knowledge. Use formal reasoning systems to derive the solution.

Systems that act rationally

The rational agent approach : *standard AI model* : Agents that do the "right thing" :

- Achieve its goals according to what it knows
- Perceive information from the environment
- May use knowledge and reasoning to select actions
- Perform actions that may change the environment

Value alignment problem

Agreement between our true preference and objectives put in the machine.

1.2 Intelligent Agents

1.2.1 What is an agent ?

An agent is an entity that interacts with its environment.

- Perception through sensors.
- Actions through actuators or effector.

Rational Agents

A rational agent does "the right thing", i.e. the action that leads to the best outcome

1.2.2 Performance of an Agent

A performance measure embodies the criterion for success of an agent's behavior.

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by its percept sequence and whatever built-in knowledge the agent has (rationality $\neq$ perfection).

Environment properties

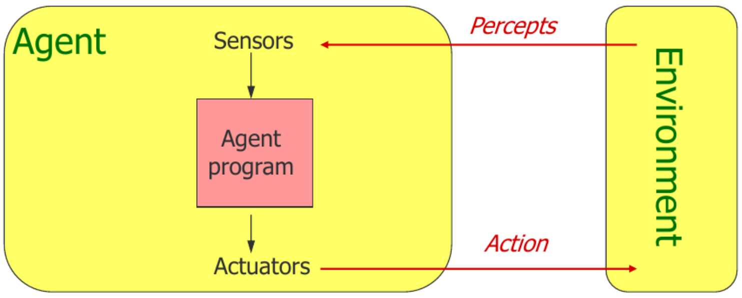
1. Fully vs. partially observable
2. Single vs. multiagent
3. Deterministic vs. stochastic
4. Episodic vs. sequential
5. Static vs. dynamic
6. Discrete vs. continuous
7. Known vs. unknown environment

1.2.3 Structure of Agents

Agent = Architecture + Program

- Architecture : operating system of the agent (computer system, specific hardware, possibly OS functions)
- Program : functions that implements the mapping from percepts to actions

Basic structure of an Agent



2 | Problem Solving

2.1 Solving Problems by Searching

2.1.1 Search

Process of looking for a (or the best) sequence of actions, that leads to a goal (specific state of the environment), starting from an initial state.

Hypothesis on the environment :

- Static
- Discrete
- Deterministic
- Fully observable

2.1.2 States and Actions

States describes distinguishable stages during problem-solving process (depend on the task and domain).

An action transports the agent from one state to another by applying an operator to a state.

2.1.3 Formulating problems

A problem is defined by the items :

- States and their representation
- Initial state
- Goal stated or goal test **Is-Goal(s)**, can be explicit (i.e."at Bucharest") or implicit (i.e. NoDirt$(x) =$ true if $x \in \{[A, C, C], [B, C, C]\}$)
- Actions available to the agent on a given state s : **Actions(s)**
- Transition model **Result(s, a)** state x action $\rightarrow$ state
- Action cost function **Action-Cost(s, a, s')** or **c(s, a, s')**

A solution is a sequence of actions from the initial state to a goal state. An optima solution has the lowest path cost among all solutions.

2.1.4 Searching for Solutions

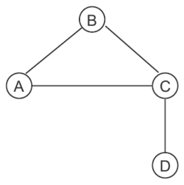
Traversal of some search space :

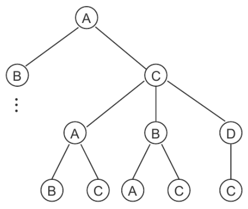
- Each node corresponds to a state
- Root correspond to the initial state
- Each edge corresponds to an action
- Expand a node n by considering the possible **Actions**(s), where s is the state in the node n.
- Use **Results**(s, a) to get the resulting states.
- Generate new nodes with these states (called child nodes or successor nodes).
- Attach each of these nodes to the current node as its parent.

State space vs Search tree

State space
Possibility to infinite set of states in the world, and the associated transitions.

Search tree
Path between states, reaching towards the goal.
Possibly multiple paths from the initial state to any state.
Unique path back from a node to the root.





State vs. Nodes

States : (Representation of) a physical configuration.

Nodes : Data structure constituting part of a search tree.

Frontier

Frontier : set of generated nodes which have not been goal-tested (visited) and which ancestors have been goal-tested (visited).

- Set on unexpanded nodes
- Separation between the visited nodes/states and the unreached nodes/states
- Represented by a queue with operations
- Different forms of queue will be used (priority queue, FIFO queue, LIFO queue or stack)

Avoiding Repeated States

Introduction of a look-up table containing all the visited nodes.

2.1.5 Measuring performance

Criteria

- Completeness : it finds a solution if one exists
- Time complexity : usually in terms of the number of nodes generated/expanded
- Space complexity : maximum number of nodes in memory
- Optimality : it finds a least cost solution ?

Problem variables
Time and space complexity are measured in terms of :

- b : maximum branching factor of the search tree
- d : depth of the least-cost solution
- m : maximum number of action in any path (may be infinite)

2.1.6 Search Strategies

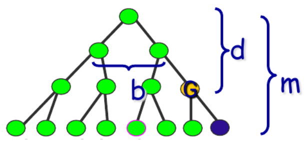
Uninformed search (blind search) : number of steps and path cost unknown. The agent only knows when it reaches a goal.

Informed search (heuristic search) : agent has background information about the problem (map, costs of actions, approximation of solutions, etc.)

Breadth-First Search

Properties

- Complete if b is finite
- Time complexity : $\mathcal{O}(b^d)$
- Space complexity : $\mathcal{O}(b^d)$
- Optimal if cost = 1 per step (not optimal in general)



Uniform-Cost Search

- The node with the lowest cost is explored first
- Frontier is implemented as a priority queue, with f(n) = n.Path-cost
- Equivalent to bfs if cost = 1
- Uniform-cost search = Dijkstra algorithm

Properties

- Complete if step cost strictly positive ($\geq \varepsilon$)
- Time and space complexity : $\mathcal{O}(b^{1+\lceil C^*/\varepsilon \rceil})$, where C^* is the cost of the optimal solution
- Optimal : nodes are expanded in increasing order of g(n)

Depth-first Search

- Expand deepest unexpanded node
- Frontier is implemented as a LIFO queue (stack)

Properties

- Not complete as it can fall in infinite depth spaces
- Time complexity : $\mathcal{O}(b^m)$
- Space complexity : $\mathcal{O}(m * b)$
- Not optimal

Depth-Limited Search

DFS (tree-search version) with a step limit.

Iterative Deepening

Apply Depth-Limited Search with increasing depth limit. It combines the advantages of BFS and DFS methods.
It is the preferred uninformed search method when search space is large and depth of solution is not known.

Properties

- Complete
- Time complexity : $\mathcal{O}(b^d)$
- Space complexity : $\mathcal{O}(b * d)$
- Optimal if step cost = 1. Can be modified to explore uniform-cost tree.

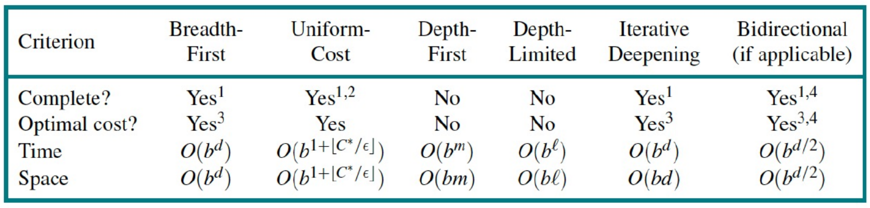
Bidirectional Search

Search simultaneously (using BFS) from goal to start and from start to goal. Stop when the two search trees intersects.

Properties

- Complete
- Time complexity : $\mathcal{O}(b^{d/2})$
- Space complexity : $\mathcal{O}(b^{d/2})$
- Optimal if cost = 1

Summary



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

2.1.7 Informed Search

Using problem specific knowledge, find and/or deduce information about future states and future paths. Use this information to make better decisions about which paths to pursue.

A heuristic function estimates the cost of the cheapest path from node to a goal (denoted h(n)) (problem-specific).

Greedy Best-First Search

- Evaluate function h(n) = f(n).
- Evaluate node that minimize h(n).
- Expand the node that seems to be the closest to a goal.

Properties

- Not complete as it can fall in infinite depth spaces.
- Time complexity : $\mathcal{O}(b^m)$
- Space complexity : $\mathcal{O}(b^m)$
- Not optimal

A* Search

- Combines greedy and uniform-cost search
- Choose the (estimated) cheapest path through the current node
  - f(n) = g(n) + h(n) = path cost + estimated cost to the goal.
  - g(n) : exact cost from initial state to node n
  - h(n) : estimated cost from node n to a goal
- A* can use the tree or graph search version of the generic BFS algorithm
- h(n) is admissible if it never overestimated the cost to reach a goal. Consequences :
  - h(n) is optimistic
  - f(n) never overestimates the cost of a solution through node n
  - h(n) = 0 if n is a goal

Properties

- Complete
- Time complexity : $\mathcal{O}(b^d)$
- Space complexity : $\mathcal{O}(b^d)$ (keeps all generated nodes in memory
- Optimal and optimally efficient (if $h(n)$ consistent (Graph search) / if $h(n)$ is admissible (Tree search))

A* is Optimal : Proof

A* (using Tree Search) is optimal if h(n) is admissible.

Proof :

- A solution is a path from the initial state to a goal state
- Let C^* be the lowest path cost among all solutions
- We must show that A* will not return a suboptimal path to a goal

Part 1 :

- Let G be a goal node in the frontier, but in a suboptimal path
- Its path cost $g(G) = C$ is not the lowest one ($C > C^*$)
- $f(G) = g(G) + h(G)$
  - $h(G) = 0$ because G is a goal node and h is admissible
  - $f(G) = g(G) = C > C^*$

  $\rightarrow$ $f(G) > C^*$

Part 2 :

- Let n be a node in the frontier, with n the path to the optimal solution (cost C^*)
- $f(n) = g(n) + h(n)$

$\rightarrow$ $f(n) \leq C^*$ because h is admissible

Part 3 :

- $f(n) \leq C^* < f(G)$

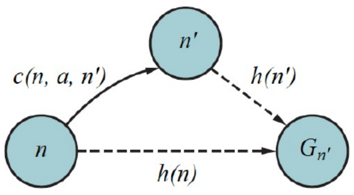
$\rightarrow$ Node G will never be selected

Consequence : A* expands no node with $f(n) > C^*$

Consistent Heuristics

A heuristic function is consistent if for every node n and successor n' obtained with action a :

- Estimated cost of reaching goal from n is no greater than cost of getting to n' plus estimated cost of reaching goal from n'
- $h(n) \leq c(n, a, n') + h(n')$



If a heuristic if consistent, then it is also admissible.
If $h(n)$ is consistent, then $f(n)$ along any path is non decreasing.

Satisficing search

Reduce the number of visited nodes by accepting suboptimal solutions, but "good enough".

Memory-Bounded Heuristic Search

Try to reduce memory needs by taking advantage of heuristic to improve performance (Iterative-deepening A* (IDA*), Recursive BFS (RBFS), Simple Memory-Bounded A* (SMA*))

Iterative Deepening A*

DFS but expand only nodes with f-cost less than or equal to smallest f-cost of non expanded nodes at last iteration. Non efficient when the number of different f-cost is high.

Properties

- Complete
- Time complexity : still exponential
- Space complexity : linear
- Optimal, optimally efficient ($h(n)$ consistent) and optimal in the absence of monotonicity

Recursive Best-First Search

DFS combined with best alternative :

- Keeps track of options along fringe
- As soon as current DF exploration becomes more expensive of best fringe option, back up to fringe, but update node costs along the way.

Properties

- Time complexity : hard to describe : efficiency is heavily dependent on quality of $h(n)$, same states may be explored many times
- Space complexity : $\mathcal{O}(bd)$ space complexity if $h(n)$ is admissible

Simple Memory-Bounded A*

Use all available memory with new expanded nodes. If new node does not fit : remove stored node with worse f-value, then propagate f-value of removed node to parent. It will regenerate a subtree only when it is needed (the path through the subtree is unknown, but cost is known)

Properties

- Complete if there is enough memory for the shortest solution path
- Time complexity : same as A* if enough memory to store the three
- Space complexity : use available memory
- Optimal if enough memory to store the best solution path

Comparing two heuristics

- Compare the total number of generated nodes N
- Compare the search trees : effective branching factor : $b*$

Choose the most dominant heuristic (most informed)

- Creating $h(n)$ by simplifying the problem by reducing restrictions on actions (relaxed problem).
- If multiple heuristics available : $h(n) = max h_1(n), h_2(n), ..., h_n(n)$
- Performance of informed search depends on the quality of the heuristics.

2.2 Search in complex environments (Local Search)

Local search does not keep track of paths : it keeps track of the current solution (current state).

Advantages :

- Use a small amount of memory
- They can find reasonable solutions in infinite search queries
- Reasonable $\neq$ optimal

It's an incomplete method based on iteratively improving the current solution. The next solution is found in the neighborhood of the current solution. Typically modify the value of a variable in an assignment at each step. The new solution is close to the previous one in the space of assignment.

An objective function is a function with vector inputs and scalar inputs. Search through candidate input vectors in order to *minimize* or *maximize* objective function

Search space : The set of feasible input vectors

Connected neighborhood : from each solution, there is a path to an optimal solution.
Advantages :

- No need of a restarting strategy to reach optimal solutions
- Required for convergence property of metaheuristics

Feasibility with optimality :

- Maintain feasibility at all times, explore only feasible solutions.
  Or
- Do not maintain feasibility at all time; relax a subset of the constraints. Explore a larger search space, drive the search to high quality and feasible solutions.

2.2.1 Heuristics and metaheuristics

Heuristics

- Choose the next solution in the neighborhood
- Based on local information : the current solution and its neighborhood
- Drive the search towards local optimum
- Memoryless

Metaheuristics

- Collect information on the execution sequence(s)
- Aim at escaping from local optima
- Drive the search toward global optimality
- Typically include memory or learning

Systematic heuristics

Exploration (possibly partial) of the neighborhood to determine the next solution

2.2.2 Hill Climbing

Shape of state greatly influences hill climbing. Local maxima are the Achilles heel. Hill climbing never makes downhill moves (cannot escape local maxima)

Random walks

Randomized heuristics. Select an element of the neighborhood randomly. Decide whether to accept it as the next solution.

2.2.3 Simulated Annealing[1]

- Always move uphill if possible
- Sometimes go downhill
- Optimality guaranteed with slow annealing schedule
- No need for smooth search space
- Applicable to discrete search space

2.2.4 Metropolis Step

$\mathit{current} := \mathit{next}$ only with probability $e^{\Delta E/T}$

- $\Delta E < 0$
- $|\Delta E|$ high $\rightarrow$ probability small
- T high $\rightarrow$ probability high

The rate at which T is decreased and the amount it is decreased is prescribed by an schedule.
Bltzmann distribution : if the schedule lowers T to 0 slowly enough, then :

- All the probability concentrated on the global maxima
- Global maxima will be found with probability approaching 1

2.2.5 Local Beam Search

Keep k states in memory : at each step, generate all the successors of all k states. Stop if one is a goal. Otherwise select the k best solutions from the complete list.

[1] Annealing is the process of heating metal and letting it cool slowly to lock in the stable locations of the molecules.

2.2.6 Genetic Algorithms (GAs)

Randomized search algorithms based on the theory of evolution.

- Start with k initial guesses :
  - They form a population
  - Each individual from the population is a fixed-length string (gene)
  - Each individual's fitness (score) is evaluated
- Produce a next generation by reproduction between individuals from current population

GA work best if the representation stores related pieces of the puzzle in neighboring cells of string. Crossover s not applicable to all problems.

2.2.7 Tabu search Metaheuristics

Select the best neighbor that has not yet been visited. Difficult to keep track of all the visited nodes. Short-term memory : only maintain a suffix of the sequence of visited nodes.

- Transition abstraction : represent the suffix by an abstraction (problem dependent)
- Aspiration : override the tabu status if the move improves the best solution found so far

2.2.8 Intensification vs. Diversification

Intensification
- Goal : increase search around promising areas
- Risk : premature convergence (local minima)
- Mean : favor good solutions

Diversification
- Goal : explore new areas
- Risk : convergence to optimality may be too long
- Mean : probabilistic choice of solutions

2.2.9 Other Local Search Approaches

Variable Neighborhood Search : Sequence of (increasing size) neighborhood
Guided Local Search : Use a sequence of objective functions to drive away from local optima
Adaptive Local Search : The heuristics/metaheuristics are dynamically adapted during the search
Ant Colony Optimization : The selection function is updated
Statistic Local Search : Another name for Local Search, stressing the stochastic aspect of the search

2.3 Constraint Satisfaction Problems (CSP)

2.3.1 Constraint Satisfaction Problem

What is CSP ?

- A set of variables defined over domains
- A set of constraints over the variables

What is a solution ?

- A solution is a consistent assignment of values to the variables
- Consistent assignment : does not violate any constraint

Types of CSP

- Discrete and finite domains (combinatorial problems, boolean CSP)
- Discrete and infinite domains (scheduling, linear/non-linear constraints)
- Continuous (and infinite) domains (linear programming, continuous CSP methods)

Types of constraint

- Unary constraints (SA $\neq$ green)
- Binary constraints (SA $\neq$ WA, $X + Y \leq 12$, ...)
- Higher-order constraints ($X + 5Y - 3Z \leq 8$, Alldiff(V, W, X, Y, Z), ...) can be transformed in binary constraints (with additional variables

2.3.2 Constraint Propagation

CSP as a search problems

- Initial state : empty assignment
- Successor function : assign a value to an unassigned variable
- Goal test : is the current assignment complete ?
- Path cost : constant value per step (the path is irrelevant)

Forward checking

When X is assigned a value V :

- Look at each unassigned variable Y connected to X (through a binary constraint $c(X, Y)$)
- Remove from Y's domain any value inconsistent with the value chosen for X

Objectives of constraint propagation

- Reduce the search space
- Find an equivalent CSP to the original one with *smaller domains* of variables

Techniques

- Consider the constraints locally
- Arc consistency : consider constraints between 2 variables
- Path consistency : consider constraints between n variables

Arc consistency

A constraint $c(X, Y)$ is arc consistent if for every value a in the domain X, there is a value b in the domain of Y such that $c(a, b)$ is satisfied.
Objective : make all constraints arc consistent.

Bound consistency

Weaker form of arc consistency. Only consider the bounds of the domains, never remove values in the middle of the domain; only move the bounds. Efficient propagation, less pruning.

2.3.3 Backtracking Search for CSPs

Incremental formulation

- Search tree : depth n (number of variables)
- BS is applicable
- Number of states = $\mathcal{O}(d^n)$ with n = number of variables, d = size of the domain (discrete finite domains)
- NP-complete problems (3SAT)

CSP search algorithms should only consider a single variable at each node.

General algorithm for any CSP

- No initial state, successor function, goal test.
- The same for each CSP

Minimum remaining value (MRV) heuristics

- Choose the variable with the fewest legal values
- Most constrained variable
- First-fail heuristics

Degree heuristic

- Variable involved in the largest number of constraints
- Reducing the branching factor of substree
- Usually used as tie-breaker of MRV

2.3.4 Local search for CSPs

Complete state formulation

- Initial state : a value to every variable
- Successor function : change the value of one variable
- Goal test : is the current assignment consistent ?
- Path cost : constant value per step

Min-Conflict Algorithm

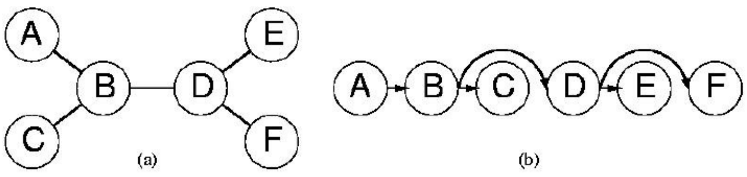
- Specialized version of Hill Climbing
- Choosing a neighbor with a variable and new value with the min number of conflicts

2.3.5 Structure of the problems

- Independent subproblems (partition of variables and constraints, forming k different CSPs)
- Can be solved separately
- Global solution is the union of the different solutions
- Complexity $\mathcal{O}(d^{n/k}k)$

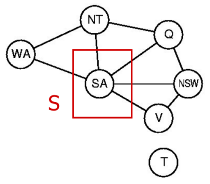
Tree-structure Subproblems

- Any 2 variables are connected by at most one path.
- Can be solved by in $\mathcal{O}(n * d^2)$
- Few CSP are tree structured
- How to reduce a CSP to a tree-structured CSP ? Remove nodes / collapse nodes



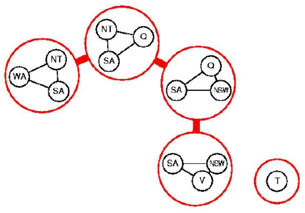
Removing nodes

- Determine the cycle cutset S (size c)
- Solve the reduced problem for each consistent assignment of the cutset
- Complexity $\mathcal{O}(d^c(n-c)d^2)$



Grouping nodes

- Tree decomposition of the constraint graph into a set of connected subproblems
- Each variable appears in at least one subproblem
- Two variables connected by a constraint must appear together in at least one subproblem
- If a variable appears in 2 subproblems, it must appear in all subproblems in the path connected these subproblems.



2.4 Adversarial Search & Games

2.4.1 Types of games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, Poker, Scrabble, Nuclear war

Problems involving :

- Multiple agents
- Competitive environments
- Agents have conflicting goals

2.4.2 Defining a Game

A game can be defined by the following properties :

- The initial state S_0 (the board position)
- To-Move(s) : the player to play in state s
- Actions(s) : the set of legal moves in state s
- Result(s, a) : the transition model; state resulting from taking action a in state s
- Is-Terminal(s) : a terminal test on states
- Utility(s, a) : a utility function on terminal states

2.4.3 Game Tree

Initial states and Action(s) and Results(s, a) define the state space search and a game tree.
Two players : *MAX* and *MIN*.

Minimax Strategy

Minimax(s) =

- Utility(s, MAX) if s is a terminal state
- $\max_{a \in Actions(s)}$ Minimax(Result(s, a) if To-Move(s) = MAX
- $\min_{a \in Actions(s)}$ Minimax(Result(s, a) if To-Move(s) = MIN

Optimal strategy

- Perfect play for deterministic games
- Leads to outcomes at least as good as any other strategy when playing an infallible opponent
- if MIN does not play optimally, MAX will do even better

Algorithm

1. Generate the game tree to the terminal states
2. Apply the utility function to all terminal states
3. Determine the utility of the successor nodes of the terminal states
4. Treat one layer at a time, applying Min or Max
5. The value at the top of the tree determines the best move

Properties

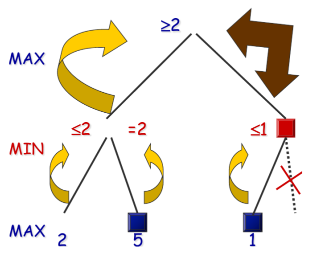
- Complete (if tree is finite)
- Time complexity : $\mathcal{O}(b^m)$
- Space complexity : $\mathcal{O}(bm)$ where b is the branching factor and m the maximum depth of the search tree

Pruning

- Discards parts of the search tree (guaranteed not to contain good moves)
- Same best result than Minimax
- Substantial time and space savings (cuts the exponent in half but still exponential)

Alpha-Beta

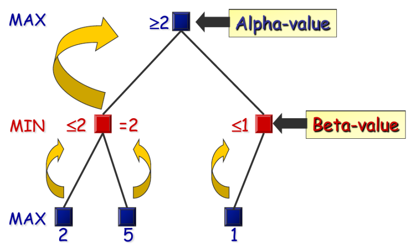
- Generate the tree depth-first, left-to-right
- Propagate final values of nodes as initial estimates for their parent nodes



1. The MIN-value (1) is already smaller than the MAX-value of the parent (2)
2. The MIN-value can only decrease further
3. The MAX-value is only allowed to increase
   → No point in computing further below this node

Terminology

The (temporary) values at $\begin{matrix}\text{MAX}\\ \text{MIN}\end{matrix}$-nodes are $\begin{matrix}\text{ALPHA}\\ \text{BETA}\end{matrix}$-values



Pruning

- $\begin{matrix}\alpha\\ \beta\end{matrix}$: the value of the best $\begin{matrix}\text{highest}\\ \text{lowest}\end{matrix}$ choice found so far at any choice point along the path for $\begin{matrix}\text{MAX}\\ \text{MIN}\end{matrix}$
- Order of considering successors matters : if possible, consider best successors first

Properties

- Pruning does not affect final result
- If successors are ideally visited, time complexity is $\mathcal{O}(b^{m/2})$ where b is the branching factor and m the maximum depth of the search tree (can look twice as far as minimax)
- If successors are visited randomly, time complexity is $\mathcal{O}(b^{3m/4})$
- Use heuristic ordering function to optimize the choice

Game Tree versus Graph

- Possible to memorized visited states
- Store the evaluation of these states
- Called transposition tables (reached list)
- Impractical to store all the states

Complete search is *impractical for most games.* Alternative :

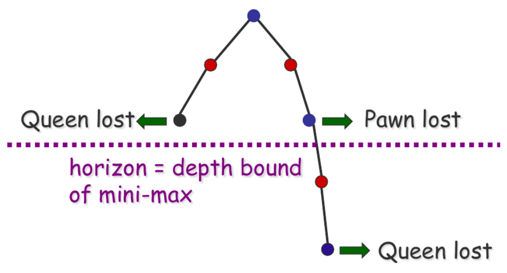
- Search only part of the tree
- Use a heuristics-based evaluation function to estimate the expected utility of the game from a given position

Evaluation Function

- H-Minimax(s, d) =
  - Eval(s) if Is-cutoff(s, d)
  - $\max_{a \in Actions(s)}$H-Minimax(Result(s, a), $d + 1$) if To-Move(s) = MAX
  - $\min_{a \in Actions(s)}$H-Minimax(Result(s, a), $d + 1$) if To-Move(s) = MIN
- Must be consistent with the utility function
- Tradeoff between accuracy and time cost
- Should reflect the actual chances of winning
- Weighted linear functions are frequently used

The horizon effect

Because of the depth-bound, we prefer to delay disasters although we don't prevent-them.



Search strategy

- Basic strategy : depth-limited
- Beam search : only consider the n best moves
- Iterative deepening :
  - Answer is refined progressively
  - Ensure an answer within a time limit (real-time decision)
  - Doesn't take advantage of knowledge about the problem

Search vs. lookup

Many game-playing programs use table lookup rather than search for the opening and ending of games. End game can be completely solved by the computer.

Monte Carlo tree search

- Utility function estimated as the average utility (e.g. win percentage) over simulations (playout) of complete games
- Playout with random legal moves by both players

Maintain a search tree, growing it at each iteration :

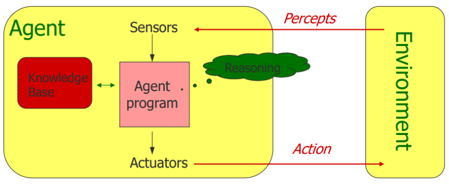
- Selection : starting from the root, go to a leaf using a selection strategy (focus on relevant parts)
- Expansion : add one new child to the node
- Simulation : playout (without recording the moves)
- Back-propagation : update all the parents in the search tree

3 | Knowledge, Reasoning & Planning

3.1 Logical Agents and Propositional Logic

3.1.1 Knowledge Based Agents

- Knowledge representation
- Reasoning with knowledge : determining which action to take
- Logic deals with both of these :
  - Classic logic : definite knowledge (knowledge that is true or false in the world)
  - Other logic : uncertain knowledge



- The knowledge base (KB) is the central component (a set of sentences representing assertions about the world, sentences are represented with a knowledge representation language)
- Two operations on KBs : Tell and Ask. Both may involve inferencing, deriving new sentences from old.
- Agent may have initial information in KB (background knowledge)

Declarative vs. Procedural

Knowledge representation

Declarative approach
- Design representation language making it easy to express knowledge
- Simplify construction of solutions

Procedural approach
- Knowledge is embodied in the algorithms and program code itself
- Potentially more efficient, but more difficult to develop solutions

3.1.2 Logic

A logic consists of syntax and semantics

- Syntax defines well formed sentences
- Semantics defines "meaning" of sentences. in logic, defines the truth of each sentence with respect to each possible word (interpretation)

Interpretation and Models

An interpretation is a mathematical abstraction of a possible world

- Possible worlds represents real environments
- Given an interpretation, a sentence is either true or false in this interpretation

A model of a sentence α is an interpretation where the sentence α is true

- The phrase "m is a model of α" ($\alpha \models \beta$) means that the sentence α is true in interpretation m

Logical Reasoning

- Logical entailment between sentences α and β
- β is a logical consequence of α
- $\alpha \models \beta$ if and only if
  - $\mathrm{M}(\alpha) \subseteq \mathrm{M}(\beta)$ where $\mathrm{M}(\alpha)$ is the set of all models of α
  - β is true in every model of α
  - in every interpretation in which α is true, β is also true

How to verify entailment ?

$\alpha \models \beta$

Logical inference, transformation to derive conclusions.
Example : Model checking :

- Enumeration of the interpretations
- Find the interpretations which are models of α
- Check that β is true in these models $\alpha \vdash \beta$

$$\begin{tikzcd} \alpha \models \beta \arrow[r, shift right, "\text{Completeness}"'] & \alpha \vdash \beta \arrow[l, shift right, "\text{Soundness}"'] \end{tikzcd}$$

Monotonicity

- If KB $\models \alpha$ then KB $\wedge \beta \models \alpha$
- The set of entailed formulas can only increase as information is added to the KB
- If a formula is entailed by a subset of KB, it is also entailed by KB

3.1.3 Propositional Logic

A very simple logic (syntax, semantics, inference algorithm) centered around propositions (statements about the world that may be true or false).

Syntax

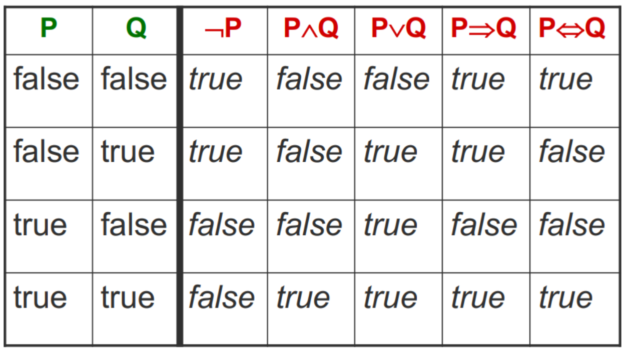
- Symbols
  - logical constants : True, False
  - propositional symbols : P, Q, ...
  - logical connectives : negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, equivalence $\Leftrightarrow$
  - parentheses (,)
- Sentences
  - Constructed from simple sentences

$$\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & (\,\textit{Sentence}\,) \\
& \mid & \neg\,\textit{Sentence} \\
& \mid & \textit{Sentence} \wedge \textit{Sentence} \\
& \mid & \textit{Sentence} \vee \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Rightarrow\; \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Leftrightarrow\; \textit{Sentence} \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{array}$$

Model

- An interpretation of a proposition formula is an assignment of a truth value to each of its proposition symbol
- If n proposition symbols : 2^n different interpretations

Semantics



P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
false	false	*true*	*false*	*false*	*true*	*true*
false	true	*true*	*false*	*true*	*true*	*false*
true	false	*false*	*false*	*true*	*false*	*false*
true	true	*false*	*true*	*true*	*true*	*true*

Model checking

Depth-first enumeration of all interpretations. Is is sound and complete.
For n symbols :

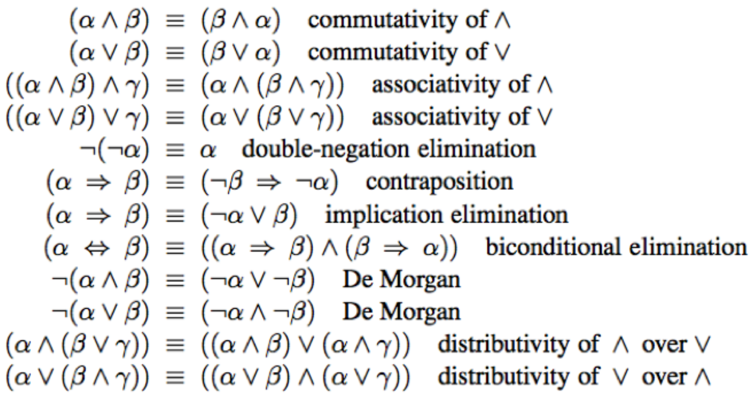
- Time complexity is $\mathcal{O}(2^n)$
- Space complexity is $\mathcal{O}(n)$

3.1.4 Propositional Theorem Proving

Validity and satisfiability

- Logical equivalence : α and β are logically equivalent ($\alpha \equiv \beta$) if they are true in the same interpretations ($\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$)
- A sentence is valid (tautology) if it is true in all interpretations
- A sentence is satisfiable if it is true is some interpretations
- A sentence is unsatisfiable if it is true in no interpretations

Standard Equivalences



$$\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge\\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee\\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge\\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee\\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination}\\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}\\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}\\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}\\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}\\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}\\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee\\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}$$

Proof methods

Two kinds :

1. Application of inference rules
   - Legitimate (sound) generation of new sentences from old
   - Proof = a sequence of inference rule applications
   - Can use inference rules as operators in a standard search algorithm
   - Typically require transformation of sentences into a normal form
2. Model checking
   - Truth table enumeration (always exponential in n)
   - Improved backtracking, e.g. : David-Putnam-Logemann-Loveland (DPLL)
   - Heuristic search in model space (sound but incomplete), e.g. : mon-conflicts-like hill-climbing algorithms

Proof As Search

- Initial state : Initial KB
- Successor function : Each KB that results from applying one inference rule to selected rules in the KB
- Goal test : Does the KB contain the goal sentence ?
- Path cost : The number of inference rules applied

Conjunctive Normal form (CNF)

Every sentence in propositional logic is logically equivalent to a conjunction of clauses (i.e. disjunction of literals)

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$

- Eliminate $\Leftrightarrow$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$

$$(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$

- Eliminate $\Rightarrow$ with $(\neg\alpha \lor \beta)$

$$(\neg B_{1,1} \lor (P_{1,2} \lor P_{2,1})) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$$

- "Move $\neg$ inwards", double negation, de Morgan

$$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$$

- Distribute $\lor$ over $\land$

$$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$$

Horn Clauses

Linear time algorithms exist when knowledge bases are restricted to Horn clauses. A Horn clause is a disjunction of literals of which at most one is positive.

Inference with Horn clauses

Forward Chaining

- Start with the knowledge base and through repeated applications of Modus Ponens, derive all logically entailed atomic sentences.
- Fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found
- Data driven
- Sound and complete

Backward Chaining

- Apply resolution until empty clause if found
- Start with a query q
- Find all implications that conclude q
- If all premises are true, then q is true, otherwise use backward chaining on premises with unknown values.
- Goal-directed
- Sound and refutation complete

3.1.5 Effective Propositional Model Checking

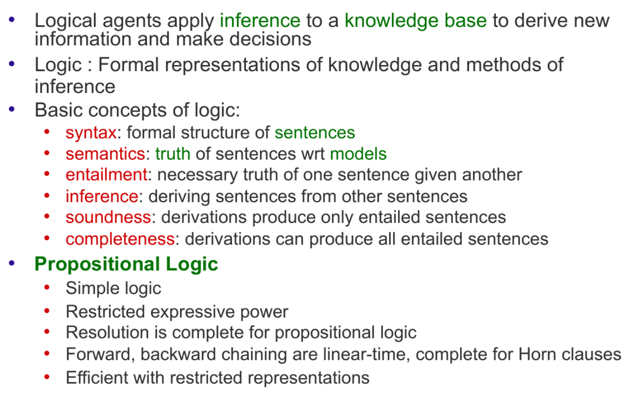
DPLL : Davis Putman algorithm

- Similar to TT-entail
- Use a CNF input sentence
- Recursive depth-first enumeration of interpretations
- Various improvements (consistency, early detection of true clauses, pure symbol heuristics, unit clause heuristics)
- One of the fastest satisfiability algorithm

WalkSAT : Local Search

- Model-based approach
- Based on MinConflict CSP local search
- Balance between greediness and randomness
- Introduce some perturbations to avaoid local minima
- Sound but not complete

3.1.6 Summary



- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Logic : Formal representations of knowledge and methods of inference
- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences wrt models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences
- **Propositional Logic**
  - Simple logic
  - Restricted expressive power
  - Resolution is complete for propositional logic
  - Forward, backward chaining are linear-time, complete for Horn clauses
  - Efficient with restricted representations

3.2 First-Order Logic

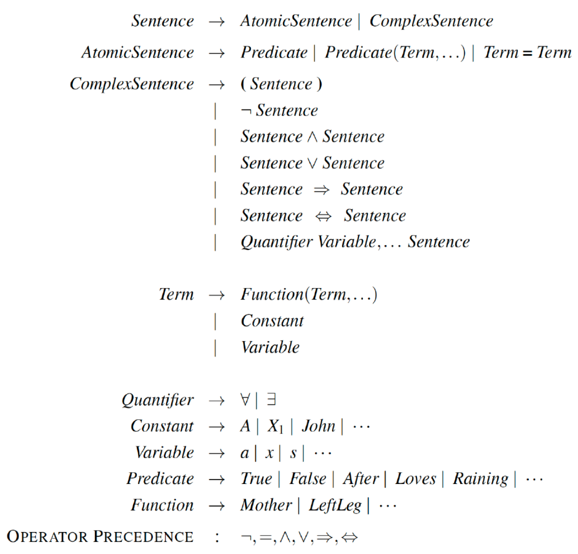
3.2.1 Representation revisited

- Procedural representation : representing knowledge with data structure has limitations. Need a declarative structure.
- Propositional logic :
  - Declarative : relationships between variables are described through sentences. A method for propagating relationships
  - Expressive : can represent partial information using disjunction
  - Compositional : if A means "it is raining" and B means "i like beer", A $\wedge$ B means "it is raining and I like beer"
  - But lacks expressive power to describe the environment concisely
- FOL basic blocks
  - PL assumes the world contains facts
  - FOL (like natural language) assumes the world contains objects, relations, functions.

3.2.2 Syntax and semantics of FOL

Syntax

- An alphabet consists of variables, constants, function symbols, predicate symbols (all user-defined) and of connectors, punctuations and quantifiers.
- Terms are either variables, constants or function symbols provided with as many terms as arguments, as the function expects
- Well-formed formulas are constructed from predicate symbols, provided with terms as arguments, and from connectors, quantifiers and punctuation - according to the rules of the connectors



Sentence	$\rightarrow$	*AtomicSentence* | *ComplexSentence*
AtomicSentence	$\rightarrow$	*Predicate* | *Predicate*(*Term*, ...) | *Term* = *Term*
ComplexSentence	$\rightarrow$	(*Sentence*)
	|	$\neg$ *Sentence*
	|	*Sentence* $\wedge$ *Sentence*
	|	*Sentence* $\vee$ *Sentence*
	|	*Sentence* $\Rightarrow$ *Sentence*
	|	*Sentence* $\Leftrightarrow$ *Sentence*
	|	*Quantifier Variable*, ... *Sentence*
Term	$\rightarrow$	*Function*(*Term*, ...)
	|	*Constant*
	|	*Variable*
Quantifier	$\rightarrow$	$\forall$ | $\exists$
Constant	$\rightarrow$	*A* | X_1 | *John* | ⋯
Variable	$\rightarrow$	*a* | *x* | *s* | ⋯
Predicate	$\rightarrow$	*True* | *False* | *After* | *Loves* | *Raining* | ⋯
Function	$\rightarrow$	*Mother* | *LeftLeg* | ⋯
OPERATOR PRECEDENCE	:	$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Semantics

- Provided byinterpretations for the basic constructs, usually suggested by meaningful names
- Domain of the interpretation : set of objects
- Constants : the interpretation identifies the object in the real world
- Predicate symbols : the interpretation specifies the particular relation in the domain. Can be defined implicitly, or explicitly through the set of tuples of objects that satisfy the relation.
- Function symbols : identifies the object referred to by a tuple of objects. Can be defined implicitly or explicitly through tables.

Interpretation

- A set D (the domain)
- A (total) function that maps constants to D
- A (total) function that maps functions symbols to (total) functions : D $\rightarrow$ D
- A (total) function that maps predicate symbols to predicates : D $\rightarrow$ Booleans

Usually, one has a specific interpretation in mind when writing sentences. Beware sentences can be interpreted according to any possible interpretation.

Terms

A term is a logical expression referring to an object in the interpretation.

- Let I be an interpretation, $t_1, ..., t_n$ be terms and f be a n-ary function symbol :
  - $f(t_1, ..., t_n)$ is a term
  - This term refers to the object $F(T_1, ..., T_n)$ in the domain I
    - $\rightarrow$ where F is the function $D^n \rightarrow D$ associated to f in I and $T_1, ..., T_n$ are the interpretation of the terms $t_1, ..., t_n$ in I

Atomic sentence

An atomic sentence state a fact (true or false). It is composed of a predicate with possible arguments (terms) and is true or false in an interpretation.

- Let I be an interpretation, $t_1, ..., t_n$ be terms and p be a n-ary function symbol :
  - $p(t_1, ..., t_n)$ is an atomic formula
  - The formula is true if in I the relation $P(T_1, ..., T_n)$ holds
    - $\rightarrow$ where P is the predicate $D^n \rightarrow Boolean$ associated to P in I and $T_1, ..., T_n$ are the interpretation of the terms $t_1, ..., t_n$ in I

Complex sentences

A complex sentence is built by combining atomic sentences using logical connectives $(neg, \wedge, \vee, \Rightarrow, \Leftrightarrow)$ and parentheses, and is true or false in a given interpretation.

Quantifiers

Can be used to express properties of collections of objects $(\forall, \exists)$

Universal quantifiers

States that a predicate P holds for all objects x in the domain. Given an interpretation I with domain D, the sentence is true if and only if all the individual sentences where the variable x is replaced by the individual objects in D are true in the given interpretation. Typically, $\Rightarrow$ is the main connective with $\forall$ (common mistake : using $\wedge$ as the main connective with $\forall$).

Existential quantifiers

States that a predicate P holds for some object in the universe. Given an interpretation I with domain D, the sentence is true as there is at least one of the individual sentences where the variable x is replaced by the individual objects in D is true in the given interpretation. Typically, $\wedge$ is the main connective with $\exists$ (common mistake : using $\Rightarrow$ as the main connective with $\exists$).

Properties of quantifiers

- $\begin{matrix}\forall x \forall y \\ \exists x \exists y\end{matrix}$ is the same as $\begin{matrix}\forall y \forall x \\ \exists y \exists x\end{matrix}$
- $\exists x \forall y$ is not the same as $\forall y \exists x$

Connection between quantifiers

- $\begin{matrix}\forall x P \\ \exists x P\end{matrix}$ is equivalent as $\begin{matrix}\neg \exists x \neg P \\ \neg \forall x \neg P\end{matrix}$
- This generalize : $\begin{matrix}\neg (P \wedge Q) \\ \neg (P \vee Q)\end{matrix}$ equivalent to $\begin{matrix}\neg P \vee \neg Q \\ \neg P \wedge \neg Q\end{matrix}$

Closed and ground formulas

A ground term (or formula) is a term (or formula) without any variable.
A closed formula is a formula where each occurrence of a variable x is in the scope of a quantifier $\forall x$ or $\exists x$

Logical Reasoning

An interpretation where α is true is called a model of α
Logical entailment between sentence α and β : $\alpha \models \beta$ if and only if, in every interpretation in which α is true, β is also true.

Validity and satisfiability

- Validity : a sentence that is true in all interpretations
- Satisfiability : a sentence that is true is some interpretations
- Inconsistency, unsatisfiability : a sentence that is false is all interpretations
- α is valid iff $\neg\alpha$ is unsatisfiable
- $\alpha \models \beta$ iff $(\alpha \neg \beta)$ is unsatisfiable

3.2.3 Using FOL

FOL can be used to model natural numbers, sets and subsets, lists.

3.2.4 Some complements on FOL

- Completeness theorem : the set of logical consequences is recursively enumerable
- Turing : the set of logical consequences is not recursive

3.3 Inference in FOL

3.3.1 Propositional vs. FOL interference

How to transform FOL sentences into a propositional sentence ? Using propositional inference system.

Universal instantiation

For any sentence α, variable v, and ground term g : $\frac{\forall v \ \alpha}{subst(v/g,\alpha)}$
A universally quantified variable can be replaced with any instance. Similar to and-elimination since $\forall v \ P(v)$ is "identical" to $P(v_1) \wedge P(v_2) \wedge ... \wedge P(v_n)$

Existential instantiation

For any sentence α, variable v, and new constant symbol k : $\frac{\exists v \ \alpha}{subst(v/k,\alpha)}$
A existentially quantified variable can be replaced with a new symbol.

Reduction to Propositional Logic (Propositionalization

- Reduce FOL sentences into a set of propositional sentences.
- Entailment is preserved
- The set of propositional sentences is finite if no function symbol, infinite otherwise
- Can be reduced to a finite set (for each proof)

Reduction (technical issues)

- Theorem : Herbrand : If a sentence α is entailed by a FOL KB, it is entailed by a finite subset of the propositionalized KB
  - Application : For $n = 0$ to inf do :
    - Create a propositional KB by instantiating with depth-n terms
    - See if α is entailed by this KB
  - Problem : Works if α is entailed, loops if α is not entailed
- Theorem : Turing / Church : Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence).

3.3.2 Unification

How to make 2 terms / atoms identical ? Find values for variables that makes the two terms identical.

Unification process : Unify(p, q)

- Take two atomic sentences p and q
- Return a substitution would make p and q identical
- Unify$(p, q) = \theta$ where Subst(θ, p) = Subst(θ, q)
- θ is called the unifier of the two sentences, there is possibly more than one unifier.
- Unique Most General Unifier (up to variable renaming)
- Quadratic complexity (size of the sentences)

Most General Unifier (mgu)

The substitution that makes the least commitment about the binding variables. It is unique (up to variable renaming).

Standardize apart

Problem if 2 sentences share variables. Standardize apart is to renaming the variables of one or both to avoid name clashes.

Generalized Modus Ponens

Let Subst(θ, p_i) = Subst(θ, p'_i) for all I :

$$\frac{p'_1, p'_2, ...p'_n, (p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow q)}{\text{Subst}(\theta, q)}$$

3.3.3 Forward chaining

First Order definite clauses

- Extension of propositional definite clauses
- Definite clauses :
  - A disjunction of literals of which one is positive
  - Often written as : $A1 \wedge ... \wedge An \Rightarrow H; H \leftarrow A1, A2, ..., An$
  - Not always possible to transform a set of FOL sentences to definite clauses
- Horn clauses : at most one positive literal

Analysis of FC algorithm

- Soundness : Does it only derive sentences that are entailed ? Yes, because only (Generalized) Modus Ponens is used and it is sound
- Does it answer every query whose answers are entailed by the KB ? Yes if the clauses are definite clauses. May not terminate in general if query is not entailed. This is unavoidable: entailment with definite clauses is semidecidable

3.3.4 Backward chaining

Start with the premises of the goal. Each premise must be supported by KB. Start with first premise and look for support from KB (looking for clauses with a head that matches premise, the head's premise must then be supported by KB).

BC algorithm

- FOL-BC-Ask(KB, α)
- Simplistic version of BC
- A recursive, depth-first algorithm (suffers from repetition and incompleteness, space is linear)
- Application domains : logic programming : Prolog language, and constraint programming

Logic programming : Prolog

- Algorithm = Logic + Control
- basis :
  - Backward chaining with horn clauses + bell & whistles
  - Program = set of choices
- Execution :
  - DF, left-to-right backward chaining
  - Built-in predicates for arithmetic etc.
  - Built-in predicates that have side effects
  - Closed-world assumption ("negation as failure")

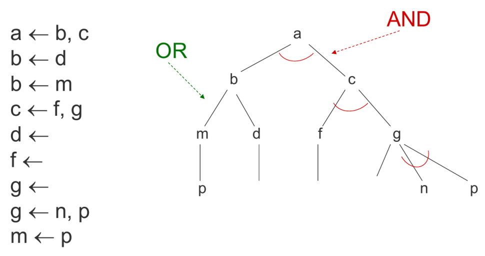
And-Or Tree vs. List of goals Tree

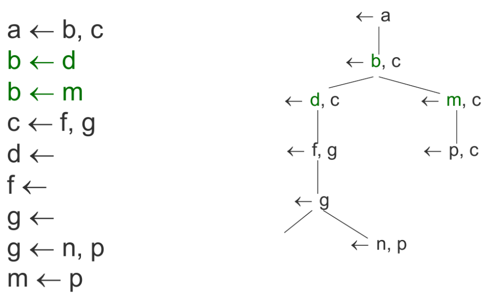
And-Or Tree

- Not really suitable in FOL, because of substitutions
- Suitable for describing forward chaining
- Not suitable for backward chaining : difficult to handle substitutions within and nodes

List of goals Tree

- Not suitable for forward chaining
- Suitable for backward chaining : substitutions applies on all goals in the list of goals (see algorithm FOL-BC-Ask)
- Resulting substitution = composition of substitutions along the successful branch





3.3.5 Resolution

First-Order CNF

Every sentence of first-order logic can be converted into an inferentially equivalent Conjunctive Normal Form (CNF) sentence. The two sentences are both unsatisfiable or both satisfiable.

Conversion to CNF

- **Eliminate implications** : Convert all implications with corresponding disjunctions

  $p \Rightarrow q$ becomes $\neg p \vee q$

- **Move $\neg$ inwards** : Use de Morgan's laws, quantifier equivalences, and double negations

  $\neg(p \vee q)$ becomes $\neg p \wedge \neg q$

  $\neg(p \wedge q)$ becomes $\neg p \vee \neg q$

  $\neg \forall x, p$ becomes $\exists x \neg p$

  $\neg \exists x, p$ becomes $\forall x \neg p$

  $\neg\neg p$ becomes p

- **Standardize variables** : Change all duplicate variable names to separate names

  $(\forall x\ P(x)) \vee (\exists x\ Q(x))$ becomes $(\forall x\ P(x)) \vee (\exists y\ Q(y))$

- **Move quantifiers left** :

  $p \vee \forall x\ q$ becomes $\forall x\ p \vee q$

- **Skolemization** : Replace variables with brand new constants (not existing elsewhere in KN) in order to remove all existential qualifiers

  $\forall x\ Q(x)$ becomes $Q(A)$ where A is unique

  - Skolem Functions are universally quantified variable sin whose scope the existential quantifier appears.
- **Distribute $\wedge$ over $\vee$** :

  $(a \wedge b) \vee c$ becomes $(a \vee c) \wedge (b \vee c)$
- **Flatten nested conjunctions and disjunctions** :

  $(a \vee b) \vee c$ becomes $(a \vee b \vee c)$

  $(a \wedge b) \wedge c$ becomes $(a \wedge b \wedge c)$

Resolution inference rule

- Simple version :

  $\frac{A \wedge B, \neg B \wedge C}{A \wedge C}$
- Modus Ponens does not allow us to derive new implications, it only derives atomic conclusions.
- Resolution : make it more powerful. Sound and complete inference system for propositional logic.
- A lifted version of propositional resolution rule :
  - Two clauses must be standardized apart (no variables are shared)
  - Can be resolved if their literals are complementary
  - For p_i and q_i where UNIFY$(p_j, \neg q_k) = \theta$:

$$\frac{\substack{p_1 \vee ... p_j ... \vee p_m, \\ q_1 \vee ... q_k ... \vee q_n}}{\text{SUBST}(\theta\ , (p_1 \vee ... p_{j-1} \vee p_{j+1} ... \vee p_m \vee q_1 \wedge ... q_{k-1} \vee q_{k+1} ... \vee q_n))}$$

- How to prove that a is a logical consequence of KB (in CNF) ?
  - Proof by contradiction : KB $\models\ \alpha$ iff (KB $\wedge \neg\alpha$ unsatisfiable
  - Transform $\neg\alpha$ into conjunctive normal form
  - Show that KB $\wedge \neg\alpha$ is unsatisfiable (apply resolution rule until derivation of the empty clause)
  - Possible to get an answer (substitution) that makes the goal to fail