TDT4136 Introduction to Artificial Intelligence Summary

On the exam: You should have the knowledge to explain how the functions work and how they would be used, but you don't need detailed knowledge of all of them.

Curriculum

Lecture notes/slides

Textbook: Artificial Intelligence - A Modern Approach, Russell & Norvig, 3rd edition. There exists two versions of this book which are both labelled "3rd edition" which differ subtly: the 2014 version (green cover) have a few chapters (6&7, 21&22) in opposite order of the 2010 version (blue cover).

Chapters 1-12 and chapter 22 (blue)/21 (green) will be the curriculum. Some details:

chapter 4: only section 4.1

2010 version (blue): chapter 7: sections 7.1 - 7.6
2014 version (green): chapter 6: sections 6.1 - 6.6

chapter 11: section 11.4 is not included.

2010 version (blue): chapter 22 (section 22.4, "Information Extraction" is not included)
2014 version (green): chapter 21 (section 21.4, "Information Extraction" is not included)

Gjennomgang

Denne gjennomgangen er i stor grad basert på slides fra forelesninger, supplert med wikipendium og bok ved behov.

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human" – Alan Turing

Course Overview	Intelligent agents Logical systems Search Knowledge representation Planning Natural language processing

Kapittel 1	Introduction
The two dimension of AI	Scientific: The science of understanding intelligent entities, developing theories, which attemt to explain and predict the nature of such entities. Discover edeas about knowledge that helt explain various sort of intelligence. Model functions of the brain Enigneering: Solving real-world problems by employing ideas of how to represent and use knowledge. Engineering og intelligent entities. Produce intelligent behaviour by any means
Involved disciplines	Philosophy, Mathematics, Statistics, Psychology, Economics, Linguistics, NeuroScience, Control Theory, Computer Science
What is AI?:	There is no formal definition covering all aspects of intelligence
Acting humanly	Turing(1950) – Turing test: Can machines behave intelligently? A computer passes the test of a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or a computer. The machine needs to posess the following capabilities - Natural language processing - Knowledge representation - Automated reasoning - Machine learning
Acting rationally	Rational behavior – doing the right thing – that which is expected to maximize goal achievement, given the available information and computational abilities. When uncertainty → The best expected outcome.
Thinking rationally	Laws of thought. Aristotle: What are correct arguments/thought processes? Formalize correct reasoning using a mathematical model → Logic.
Thinking humanly: Cognetive science	Cognetice science (top-down) and cognetive neuroscience (bottom-up) is now distinct from AI.
Two Main AI pradigms	Good old fashioned AI (GOFAI) – A physical symbol system has the necessary and sufficient means for general intelligent action (representing situation in the real world eks. Block world) Situated embodied AI (SEAI) – focuses on having a body in a physical environent

Chapter 2	**Intelligent Agents**
Outline	Agent and environments Rationality PEAS (Performance measure, Environment, Actuators, Sensors) Environment types Agent types
Agent	An agent is a computer system that is situated in some environment and that is capable of autonomous action in the environment in order to meet its design objectives. *An agent is anything that can be viewed as perceiving its environment through sensor and acting upons that environment through actuators*. Human : ears, eyes as sensors - body parts for actuators. osv. Robotic agents: Cameras and infrared range finders for sensors osv. Various motors for actuators.
Agents and environments	Agents include human, robot, softbot etc. The *agent function* maps from percept histories to action f: P* → A The agent program runs on the physical architecture to produce f.
Vacuum cleaner	See book or slides
Rational agent	A rational agent should select an action that is expected to maximize its performance measure, given - The evidence provided by the percept sequence and - Whatever built- in knowledge the agent has
Performace measure:	- Performance measure: An objective criteria for success of an agent’s behaviour

Rationality	Fixed performance measure evaluates the environment sequence. A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date - E.g. One point per square cleaned in time t Percept may not supply all relevant information Rational != successful Rational → Exploration, learning, autonomy
Autonomy	Lacking autonomy means relying on prior knowledge (being fed) rather than on its own percepts
Rationality depend on	Task environment: P: The performance measure E: The agents prior knowledge about the environment A: The actions that the agent can perform S: The percept sequence
To design a rational agent, we must specify the task environment	Consider the task of designing an: Automated taxi: Performance measure? Safety, destination, profits, lagality, comfort. Environments? Streets, traffic, pedestrians, weather. Actuators? Steering, accelerator, brake, horn. Sensor? Video, accelometers, gauges, engine sensor, GPS. Internet shopping agent Performance measure? Price, quality, apporpriateness, efficience. Environment? Current and future www sites, vendors, shippers. Actuators? Display to user, follow URL, fill in form. Sensors? HTML pages Medical diagnosis system Performance measure: Healthy patient, minimize costs, lawsuits Environment: Pation, hospital, staff Actuators: Screen, display

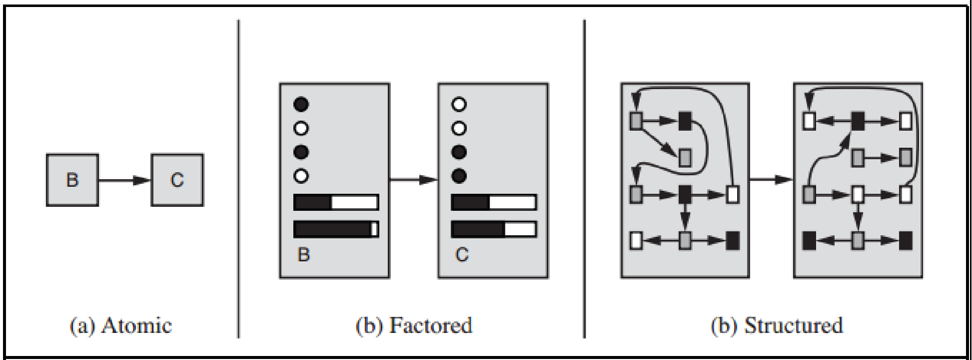
	Sensors: keyboard
Properties of environments	**Fully observable:** All relevant information acquired through percepts The agents sensors give it access to the complete state of the environment at each point in time. The agent can obtain complete, accurate, up-to-date information about the state of the environment. Convenient because the agent need not maintain any internal state to keep track of the world. The more accessible the environment is, the simpler it is to build agent to operate in it. **Partially observable:** Some relevant information

	Because of noisy and inaccurate sensors, or because parts of the state are simply missing from the sensor data. Agent should make informed guesses about world. **Single agent vs Multiagent:** Crossword vs chess. - **Competitive vs Cooperative** **Deterministic environment:** The next state depend only on current state and agent's action Any action has a single guaranteed effect. There is no uncertainty about the state that will result from performing an action If the environment is deterministic except for the action of other agents, we say the environment is strategic. **Stochastic** There is some uncertainty about the outcome of an action. Non-deterministic environments present greater problems for agent design **Episodic environments** The agent's experience is dividied into atomic episodes. Each episode consist of the agent perceiving and then performing a single action. The episodes are independent. The choice of action in each episode depends only of the episode itself. **Sequential** The current decision could affect all future decision. Episodic environments are much simpler than sequential because the agent does not need to think ahead (planning) How state is defined, how time is handled, and the percepts and actions of the agent **Discrete:** Finite number of distinct states and percepts/actions e.g. chess **Continuous** Continuous time/state/action: taxi driver **Static** environment Can be assumed to remain unchanged except by the performance of actions by the agent

	Static environment er eay to deal with because the agent need not keep looking at the world while its deciding on an action, nor need it worry about the passage of time. **Dynamic** Can change while agent is deliberating. Has other processes operating on it, and which change in ways beyond the agent's control. Demands quick decisions from the agent - real-time decisionmaking (Semidynamic) **Known environment** The outcomes for all actions are given. Note that the known environment (the agent knows all rules that apply) may only be partially observable if the sensors are not properly working. Depending on property of deterministic of course. **Unknown:** The agent will have to learn how it works The environment type largely determines the agent, design. The real world (of course) is partially observable, stochastic, sequential, dynamic, continuous, multi agent, unknown. Figure 2.6 page 45.

Agent types	4 Basic types in order increasing generality **Simple Reflex Agents: Simples kind: If-then.** Select actions on the basis of the current perceptions, ignoring the percept history. Fully obersvable, else not optimal. In partially observable it can use randomization to escape loops. **Reflex agents with state ((best-guess) model based)**: Select actions on the basis of a model of observed world(model/state NB it's a best guess model), taking into account the percept history. Partial observable. **Goal-based agents:** Select action on the basis of a model (as model based relfex agents) and a set of goals its trying to achieve. Search and planning are some concepts here. **Utility based agents:** Uses a model of the world, along with a utility (degree-of-happiness) function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of outcome. All these can be turned into **Learning agents**. Learning agents learns from their action. Can be divided into four conceptual components - **Learning element:** Responsible for making improvements - **Performance element**: Responsible for selecting external actions - **Critic:** Gives feedback to the learning elements on how the agent is doing and determines how the performance element should be modified. - **Problem generator:** Responsible for suggesting actions that lead to new experiences.

Agent	Agent = architecture + program
Conditionaction rule	Break light on car in front = you have to break as well
Rational agent in short	All agents should strive to do right thing, based on percept, what it knows, and possible actions. Rational != omnicent. Should be able to explore – perform actions in order to modify the future percepts so as to obtain useful information – active perception. Should be autonomous. It is autonomous if behavior is determined by its own experience.

Representaton of state	 **Figure 2.16** Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects. **Atomic:** Each state is one unit – indivisible **Factored:** Each state has attributes or parameters **Structured:** State has objects, which in turn have parameters and relations to each other.
Summary	Agents interact with environment through actuators and sensors. The agent function describes what the agent does in all circumstances. The performance measure evaluates the environment sequence. A perfectly rational agent maximizes expected performance/outcome. Agent programs implements (some) agent functions. PEAS description define task environment. Environment are categorized along several dimensions: Observable, determinstic, episodic, static, descrete, singe agent? Several basic agent architecture exist: reflext, reflex with state, goalbased, utility based → learning agents

Chapter 3	**Solving problems by searching**
Search is	Searching for - Pre-existing entities (information, objects etc) - Strategies for creating/designing entities Examples: Web search, AI search (creates) **Uninformed –vs- Informed**: Do points in the search space give information that helps the searcher to determine the next step? Is there any information outside definition (heuristic function etc) **Partial –vs- Complete solutions**: Could the current state of search always be considered a complete solution (local search), though not necessarily good or optimal? Or is it often a partial state that must be incrementally enhanced to become a solution.
Problem Solving Agent	Goal Based Agent Atomic representation - Factored or structured → Planning agent Goal formulation – start/init state to end goal/state Unknown environment – explore Searches for sequence of actions leading to solution Problem definition: - Initial State - Actions: Actions available in current state - Transition model/Successor function: Defines result of action - Goal test: Test if state is goal - Path cost, step cost: optimal solution problem
Formulating Problem	Abstraction – Removing details from state or actions. Useful as it is a simplification of original Valid – If abstraction can be expanded to real world Toy problem – Illustrate/exercise method/world
Searches	Leaf nodes – nodes with no children Frontiers – Expandable leaf nodes Reapeted state – loopy path Redutnant paths – multiple paths to state (“overflødig”) Explored set/closed list of visited/finished nodes FIFO,LIFO(stack) or priority queue An **incremental formulation** augment the state description (incrementally adds objects), while a **complete-state formulation** starts with problem equalling goal.

Measuring problem-solving performance	Completeness: Is the algorithm guaranteed to find a solution when there is one? Optimality: Does the strategy find the optimal solution? Time complexity: How long does it take to find a solution? Space complexity: How much memory is needed to perform the search?

	In AI we use - b: Branching factor or maximum number of successor of any node - d: depth of the shallowest goal node - m: Maximum length of any path in the state space
Uninformed search	Blind search – no knowledge of state outside definition **BFS: Breadth-first search** Complete: Yes Optimal: Yes, if cost = 1 per step. Not in general Time: O(b^d) – Horrible Space: O(b^d) – Horrible **DFS: Depth First seartch** Complete: No, fails in infinite-depth spaces and with loops Optimal: No Time: O(b^d) – Horrible Space: O(V) – Liniar in space -Can be made with depth limit **IDDFS:** Complete: Yes Optimal: Yes, if step cost = 1 Time: O(b^d) Space: O(bd)
Uniform cost search	Uniform-cost search (expands lowest cost) – like BFS but the queue is ordered by path cost (g(n)), lowest first Goal test when node frontier gets expanded, Sorts on total cost, not number of steps

Iterative deepening depth first search	IDDFS combines depth-first search's space-efficiency and breadth first search completeness (when branching factor is finite). It is optimal when the path cost is a non-decreasing function of the depth of the node Iterative increase in depth limit – several search.
Bidirectional seartch	Complete: Yes Optimal: Usually, but finds information used for optimization. Optimal if both direction uses BFS Time and Space O(b^d/2)
Informed (Heuristic) search strategies	Greedy best-first search Expansion based on evaluation function f(n) A* search – generalization of Dijkstra Use as little as possible memory search: Iterative-deepening A* (IDA*) Recursive best-first search
Heuristic functions	Admissible – Never overestimates, optimistic - Manhatten distance Consistency – $h(n) \le c(n,n+1) + h(n+1)$
Best-first search	Idea: use an evaluating function for each node – estimate of "desirability" → Expand most desirable unexpanded node

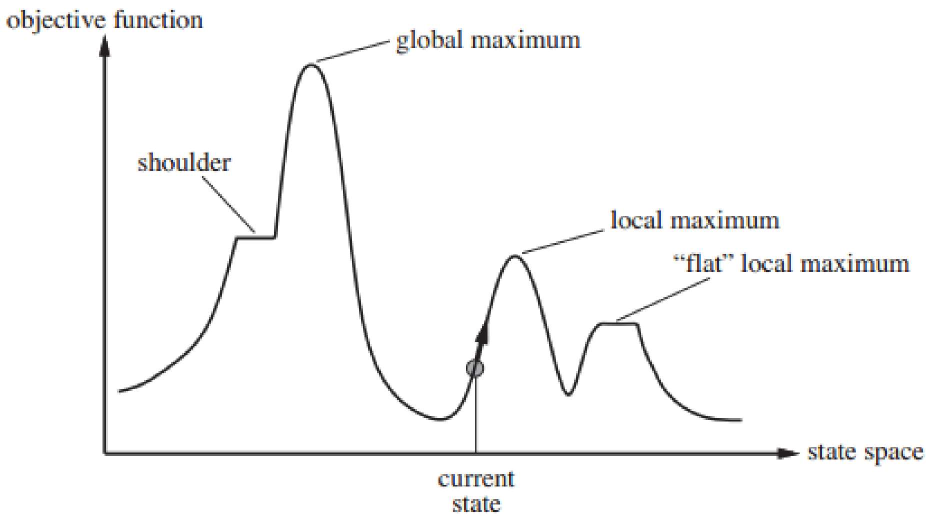
	Implementation: fringe is a queue sorted in decreasing order of desirability Special cases: Greedy search A* search
Greedy best-first search	Evaluation function h(n) heuristic = estimate of cost from n to the closes goal. Example: straight line from n to goal. A heuristic is problem specific Greedy search expands node that appears to be closest to the goal

Preoperties of greedy search	Evaluation function: f(n) = h(n) Complete?? No can get stuck in loops Time?? O(b^d) – but a good heuristic function can give dramatic improv. Space?? O(b^d) – keep all nodes in memory Optimal?? No
A* search	Idea: Avoid expanding paths thar are already expensive: Evaluating function f(n) = g(n) + h(n) f(n) = estimated total cost of path through n to goal Admissible heuristic function Theorem: A* search is optimal
Optimality of A* (standard proof)	Suppose some suboptimal goal G2 has been generated and is in the queue. Let n be an unexpended node on a shortest path to an optimal goal G1 F(G2) = g(G2) + (h(g2) = 0) > g(G1) since G2 is suboptimal >= f(N) since h is admissbile Since f(G2) > f(n), A* will never select G2 for expansion.
Optimallity of A* (more useful)	Lemma: A* expands nodes in order of increasing f value* Gradually adds"f-contours" of nodes(cf. Breath first add layers). Contour I has all nodes with f = fi where fi < fi+1
Properties of A*	Complete = Yes, unless there are infinitely many nodes with f<= f(G) Time = Exponential in [relative error in h x length of solution] - Worst-case 0(b^d) Space = Keeps all nodes in memory →O(b^d) Optimal= Yes – connor expand fi+1 until fi is finished A* is optimally efficient for any given consistent heuristic function.
Memory bounded search	Memory-bound A* and Simple MA* - Adds nodes up to memory limit - Removes (oldest) worst nodes (highest f) - Trashing – loops between states due to optimal pruning - May be intractable (time grows exponentially)
Heuristic: Proof of lemma: Consistency/Monotonicity	A heuristic is consistent/monotonous if h(n) <= c(n,a,n´) + h(n´) If h is consistent we have f (n') = g(n') + h(n') = g(n)+c(n,a,n')+h(n') ≥ g(n)+h(n) = f(n) I.e., f (n) is nondecreasing along any path (triangle inequality).

Admissible heuristic	Never overstimates the cost to reach the goal E.g. the 8-puzzle H1(n) = number of misplaced tiles = 6 H2(n) total Manhatten distance (no of squares from desired loacation of each tile) = 14 (slides) H3(n) = Euclidean, the length one would measure with a ruler. One can clasify a heuristic with branching factor, and find effective branching factor b*
Dominance	If h2(n) >= h1(n) for all n (both admissble) then h2 dominates h1 and is better for search. Typical search for solution length d: Typical search costs for a solution length d: d = 14 IDS = 3,473,941 nodes A ∗ (h1) = 539 nodes A ∗ (h2) = 113 nodes d = 24 IDS ≈ 54,000,000,000 nodes A ∗ (h1) = 39,135 nodes A ∗ (h2) = 1,641 nodes Given any admissible heuristics ha, hb, h(n) = max(ha(n), hb(n)) is also admissible and dominates ha, hb IDDS: Iterative deepening depth-first search
Relaxed problems	Fewer restrictions on actions, allowing for “optional” pats Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then h1(n) gives the shortes solution If the rules are relaxed so that a tile can move to any adjacent square, then h2(n) gives the shortest solution Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Summary	Heuristic functions estimate costs of shortest path Good heuristic can dramatically reduce search cost Greed best-first search expands lowest h – incomplete and not always optimal A* search expands lowest g+h - Complete and optimal - Also optimally efficient

	Admissible heuristics can be derived from exact solution of relaxed problems Iterative improvement algorithms can be used when the path to the goal state is irrelevant
Chapter 4	**Beyond classical search (Not covered that much in class, only 4.1)**
	Relaxed
Local search	LS algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbours of that node. Goal is to locate goal state, not the path to it (highest peak etc) “Find best state according to objective function” - not optimization problem Key advantages: 1. Memory efficient 2. Can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable Local search doesn’t use heuristics, but a similar concept called objective function. The objective function answer the question ”How optimal are your solution. We operate in state-space landscape <u>Objective function:</u> If we dont’t have a goal state – what are we searching for? Objective function f: State → Value (give a max or min value at the goals).

Iterative improvement algorithms	In many optimization problems, path is irrelevant; tho goal state itself is the solution. Then state space = set of ”complete” configurations; find optimal configuration of TSP, or find configuration satisfying constraints – timetable. In such cases, can use iterative improvement algorithms. Constant space. E.g. Start with TSP solution and swap cities. Start with all Queens on board and move to reduce number of conflicst. Constant space, suitable for online and offline search
State Space	Location defined by state Elevation equals value of objective function: max value is peak Local vs global peaks, shoulders, flat locals etc are all problems Local search engines explore this landscape 
Hill climbing	Greedy: ”Likelimbing Everest in thick fog with amnesia” . Walks uphill Can get stuck in local maxima when trying to find the global. Random-restart hill climbing overcomes local maxima. Trivially complete. Random sideways moves escape from shoulders, but loop on flat maxima.
Simulated annealing	Idea: Escape local maxima by allowing some ”bad” moves but gradually decrease their size and frequency: Gradient Descent (ping pong)

	It prioritizes neighbors that improve on the situation, but with some probability, will accept other neighbors. Uses an objective function
Local beam search	Idea: keep k states instead of 1; choose top k of all their successors. Searches that find good states recruit other searches to joint them, sharing the information. Problem: quite ofte, all k states end up on the same local hill. Idea: choose k successors randomly, biased towards good ones increasing (stochastic beam search). Oberve the close analogy to natural selection
Genetic algorithms	GA = stochastic local beam search + generate successors from pairs of states Sexual (vs asexual) reporduction Begin with a set of k randomly generated states, represented by bit string, called the *population*. Each state is rated by the fitness function, which returns higher values for better states. The successor states aret hen generated by combining two of the states to produce a child state – by choosing a random crossover point. The new states aret hen subject to random mutations. Gas require encoded as strings (GPs use programs) Crossover helps iff substrings are meaningful components. Appealing because of its simplicity and connection with theory of evolution; however does require the fitness function and genome to be well defined (els destructive)
GA	1. Fitness function (rates) 2. Selection (best) 3. Crossover 4. Mutation (small)
Summary	Same as chap 3
Chapter 5	Adversarial search = " characterises by conflict or opposition"
Games vs. Search problems	A search three is different from standard search. Here it is a two-ply tree where each alternating level one of the players has the control/decisions. "Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply. Time limits → unlikely to find a goal, must approximate World we search in is changing The search is about maximizing the utility of the computer (MAX) - Assuming involved parts plays optimally

Tpes of games		Deterministic	Chance(Stokastic)
	Perfect info	Chess, checkers, go, othello	Backgammon, monopoly
	Imperfect info	Battleships, blind tictac-toe	Bridge, poker, scrabble, nuclear war
Adversarial search threes	Opponent – my – opponent action. Nodes are “OR” choices Solution is a strategy, sometimes time limits force an approximate solution Evaluation function		

<table>
<tr><td>Utility Function</td><td>Internalization of Performance Measure
- Measures Preferences over set of properties
- Formalized satisfaction
- Vector of utilities
Alliances emerges from selfish behaviour
Utilized on terminal states, defined by the terminal-test</td></tr>
<tr><td>Minmax</td><td>Perfect play for determinstic, perfect information games.
Idea: Choose move to position with the highest minmax value = best achievable payoff against best play. Full depth-first analysis

Use a heuristic/eval function to evaluate promise at bottom-level nodes, propagate upwards.
Example of multiplayer. Two players has initial values [-inf, inf]

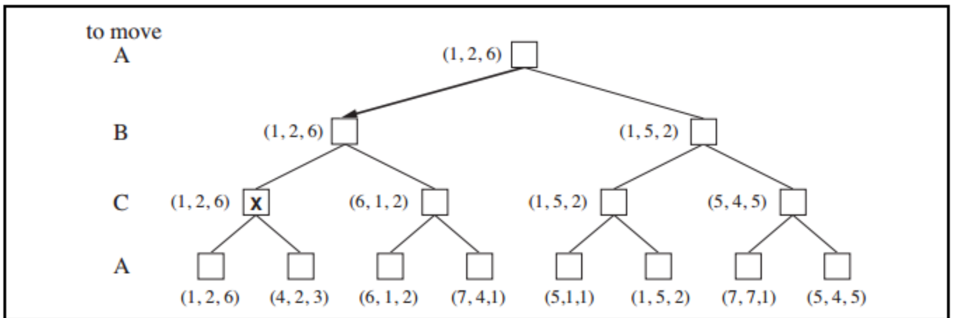
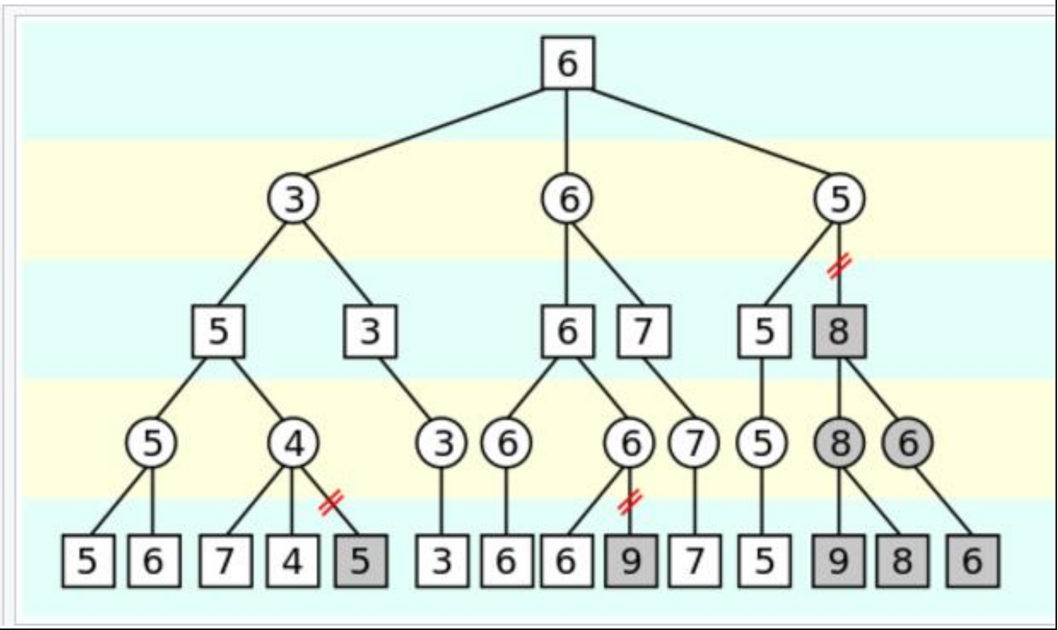
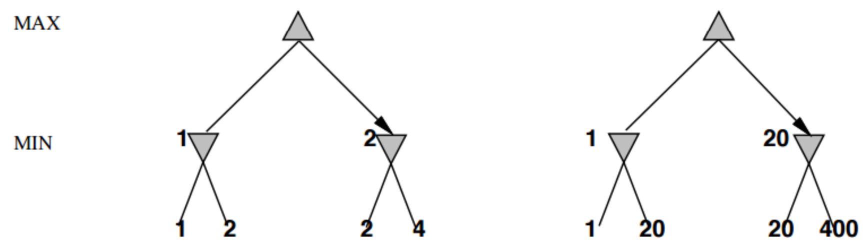
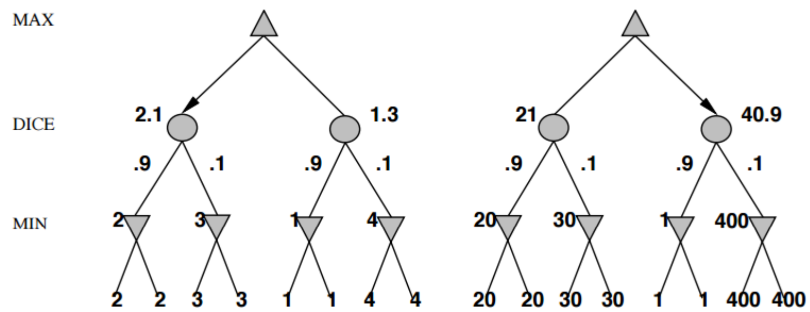


Figure 5.4 The first three plies of a game tree with three players (*A*, *B*, *C*). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.</td></tr>
<tr><td>Popertie s of MinMax</td><td>Complete?? Only if three is finite (chess has spesific rules for this)
Optimal?? Yes, against an optimal opponent
Time complexity?? O(b^m)
Space complexity?? O(m) (depth first)
For chess b = 35, m = 100 (ca), exact solution completely infeasible. But do we need to explore every path?</td></tr>
<tr><td>α-β pruning</td><td>α is the best value (to max) found so far off the current path. If V is worse than α, Max will avoid it → prune that branch. Define β similarly for min.</td></tr>
</table>

	
Properties of αβ alpha-beta	Pruning does not affect final result Good move ordering impoves effectiveness of pruning Best moves based on history, known as killer moves. Squareroots moves per node. With ”perfect ordering” time complexity = O(b^m/2) → doubles solvable depth (Killer move heuristic) A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning). Unfortunately 35^50 is still impossible,
Resource limits (imperfect realtime decisions)	Standard approach: -Use Cutoff-Test(state, depth) instead of Terminal-Test – how far down am I. e.g. depth limit (perhaps add quiescence search) Use Evaluation instead of Utility - expected utility i.e. evaluation function that estimates desireability of position Quiescence search – will the outcome swing wildly? Horizon effect – Unavoidable, damaging move - Delaying tactics - Singular extension → Allow awesome moves after depth Suppose we have 180 seconds, explore 10^6 nodes/second → 18^8 nodes per move = 35^10/2 → α–β reaches depth 10 ⇒expert level chess program

Evaluation functions	Expected utility of state – quick assumptions Correlated with chance of winning, based on features and history For chess, typically linear weighted sum of features - Features ain't independent Eval(s) = w1f1(s) + ... + wnfn(1) e.g. w1 = 9 (=value of queen) with f1(s) = (number of white queens)(number of black queens), etc.
Efficiency	Forward pruning – Prune nodes immediately without further consideration - Dangerous to prune away good nodes Lookup in database made by experts, history - Chess openings and closings - Policies -mapped solutions
Properties of actions	Deterministic games in practise: Checkers, Chess, Othello Non-deterministic Chance introduced by dice, card-shuffeling etc – edges with chance. Nodes are circles – chance nodes

<table>
<tr>
<td>Digression:
(Deterministic)</td>
<td>In deterministic games exact values don't matter, only order does
Behaviour is preserved only by positive linear transformation of Eval.
Hence Eval should be proportional to the expected payoff.


</td>
</tr>
<tr>
<td>Digression:
Non-determ</td>
<td>In games of probability exact values DO matter, due to final value of node in accordance with probabilities*values of said node


</td>
</tr>
</table>

Games of imperfect information	E.g. Card games where opponent's initial cards are uknown. Typically we can calculate possibility for ech possible deal. Seems just like having one big dice roll at the beginning of the game* Idea: compute minmax value of each action in each deal, then choose the action with highest expected value over all deals, it's optimal* HIB, current best bridge program, approximates this idea by 1) Generating 100 deals consistent with bidding information 2)Picking the one winning thos most tricks in avrage
AI Critics (Wiki)	Critic = a system that avaluates search states. Very similar to both heuristics and objective function. Actor = a system for mapping search states to action. Actor-Critic systems use AI to both compute actions and avaluate states – in some version best action is the one that leads to a state with highest evaluation
Summary	Games are fun, they illustrate several important points about AI: - Perfection is unatteinable → must approximate - Good idea to think about what to think about - Uncertainty constrains the assignment of values to states games are to AI as grand prix racing is to automobile design.
Chapter 6	**Constraint Satisfaction problem**
CSPs	Several variables that must be satisfied, defined by constraints. Compare state to goal in statespace Standard search problem: state is a "black box" – any old data structure that supports goal test, eval, successor CSP: - State is defined by variables Xi with values from domain Di - goal test is a set of constraints specifying allowable combinations of values for for subset of variables Allows useful general-purpose algorithms with more power than standard search algorithms
Properties	X – set of variables D – set of domains, consist of allowable values for each variable C – constraints: <scope, relation> = <particpants, relation> Assignment: Set of valued variables of state - Consistent: Leagal, does not violate constraint - Complete: Every variable is assigned (not partially) - Looking for complete and consistent assignment

Constraint graph	Binary CSP: each constraint relates at most two variables. Constraint graph: nodes are variables, arcs show constraints – by doing this we can divide the problem into subproblems . Domains represented as remaining possible colours/numbers for a node. General-purpose CSP algorithms use the graph structure to speed up search
Example map coloring	Variables: WA; NT; Q, NW, V, SA, T Domains Di = {red, green, blue} Constraints: adjacent regions must have different colors

Varieties of CSPs	Discrete variables: - finite domains; size d ==⇒O (d^n) complete assignments e.g. Boolean CSP, inc. Boolean satisfiability (NP – complete) - Infinite domains (integers, strings, etc): e.g. job scheduling, variables are start/end days for each job need a *constraint language* e.g. StartJob1 + 5 <= StartJob3 linear constraints solvable, nonlinear undecidable Continuous variables - Start/end times for Hubble Telescope observations linear constraints solvable in poly time by LP methods
Varieties of constraints	- Unary: single variable: SA != Green - Binary: two variables: SA != WA - Higher.order: constraints involve 3 or more variables - “Alldiff constraint” – all variables must be different (Sudoku)
Constraint types	- Absolute (hard) – must be satisfied in valid solution - Preference (soft) – should be satisfied. Better solutions satisfy more, often implemented with cost - A CSP becomes a **COP (Constraint optimization problem)** when soft constraints are involved
Node consistency	A single variable is node-concistent if all the values in the variable’s domain satisfy the variable’s unary constraints Network is node-consistent if all nodes are
Arc consistency	A variable in CSP is arc-consistent if every value in its domain satisfies the variable’s binary constraints. (AC-3 – checks all arcs(X, Y), and alters domain Dx if nessecary – then checks all arcs (X,Z) and see if there is any difference with reduced Dx and so on)

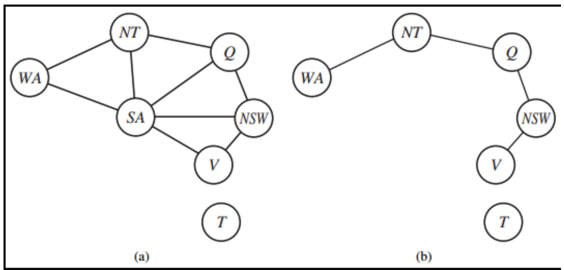
Path Consistency	Tightens binary constraints by looking at mupltiple (triplets) of variables at once.
Real-world CSPs	Assignment problems e.g. who teaches what class Timetabling problems e.g. which class is offered when and where? Hardware configuration, spreadsheets, transportation scheduling, factory scheduling, floot planning Notice that many real-world problems involve real-valued variables
Standard search formulation (incremental)	Let's start with the straightforward dumb approach, then fix it. States are defined by the values assigned so far. (Common approach) Initial state: the empty assignment {} Successor function: assign a value to an unassigned variable that does not conflict with current assignment → fail if no legal assignments (not fixable) Goal test: the current assignment is complete: 1) This is the same for all CSPs! 2) Every solution appears at depth n with n variables → use depthfirst search 3) Path is irrelevant, sp can also use complete-state formulation 4) B = (n-1)d at depth l, hence n!d^n leaves.

Backtracking search	Variable assignments are commutative, e.e., [WA = red then NT = green] same as NT = green then WA = red] Only need to consider assignments to a single variable at each node → b = d and there are d^n leaves. Depth-first search for CSPs with single-variable assignments is called backtracking search – backtrack if no variable can be assigned. Backtracking search is the basic uninformed algorithm for CSPs. Can solve n-queens for n = 25 (ca)
Improving backtracking efficiency	General-purpose methods can give huge gains in speed: - Which variable should be assigned next? - In what order should its values be tried - Can we detect inevitable failure early? - Can we take advantage of problem structure.
MRV: (improving)	Minimum remaining values Choose the variable with the **fewest legal values**
Degree heuristic (improving)	Tie-breaker among MRV variables: Degree heuristic: choose the variable with the **most constraints** on remaining variables

Least constraining value(imprv)	Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables (neighbouring values)
Forward checking (improving)	Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values
Constraint propagation	Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures. Constraint propagation repeatedly enforces constraints locally
Arc consistency check	Simples form of propagation makes each arc consistent. X → Y is consistent if for every value x of X there is some allowed y. If X loses a value, neighbours of X need to be rechecked. Arc consistency detects failure earlier than forward checking. Can be run as a prepocessor or after each assignment (AC -3) Can with revised algorithm reduce O(n^2d^3) to O(n^2d^2) (but detecting all is NP-hard)
Path consistency check	A two variable set X;Y is path-consistent with respect to Z, if for every assignment (X = a, Y = b) consistent with constraints on X,Y there is an assignment to Z that satisfy (X,Z) and (Y,Z)
Approaches to Solving CSPs	- Inference (via constraint propagation) – values of some variables force the values of other. Easy Suoku can be solved y AC-3 (Arch-consistency #3) only - Incremental Search + Inference: Makes assumption about values for certain variables then run constraint-prop to infer consequences. If comvinations of assumptions lead to inconsistencies then remove some of the (i.e. backtrack) and try other combinations. Otherwise, make additional assumptions and infer further consequences backtrack + AC-3 - Local search + assumption modification: Generate complete assignments: assume/guess a value for each variable. Evaluate the assignments w.r.t. violated constraints. Modify the assignments to reduce the number of violations

Adding Intelligence to State Generation	Depth-1st, Breadth-1-st and Best-1st search: - These normally generate all child states automatically, then visit/explore them using different strategies. The details of a parent state are largely ignored when generating child states Constraint-satisfaction search: - The details of the parent state are carefully analysed to determine the best child states to generate, in terms of the variable assignments that have the most promise, the constraints that would be satisfied or violated, etc. - This involves inference and appears more intelligent - Not possible with many other search techniques due to lack of knowledge in a canonical, operational form
Iterative algorithms for CSPs (Local search) – utrolig rask (se lenger nede)	All local search techniques are candidates for application to CSPs - Hill-climbing simulated annealing typically work with "complete" states, e.e., all variables assigned - To apply to CSPs: Allow states with unsatisfied constraints, operator reassign variable values. - Variable selection: randomly select any conflicted variable - Value selection by min-conflicts heuristic: choose value that violates the fewest constraints, e.e. hillblim with h(n) = total number of violated constraints
Performance of min-conflicts	Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10 000 000) The same appears to be true for any randomly-generated CSP ecept in a narrow range of the ratio R = number of constraints/number of variables
Problem structure – The complexity of solving a CSP is strongly related to the structure of its constraint graph	Tasmania and mainland are independent of **subproblems**, identifiable as **connected components** of constraint graph Suppose each sub-problem has c variables out of n total. Worst-case solution cost is n/c x d^c, linear in n.
Tree-structured CSPs	Theorem: if the constraint graph has no loops, the CSP can be solved in O(nd^2) time Compare to general CSPs, where worst-case time is O(d^n)

Nearly treestructured CSPs	Conditioning: instantiate a variable, prune its neighbors' domains. **Cutset conditioning**: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree Cutset size c =⇒runtime O(d^c · (n − c)d^2), very fast for small c

	 **Figure 6.12** (a) The original constraint graph from Figure 6.1. (b) The constraint graph after the removal of *SA*.
Summary	CSPs are a special kind of problem - States defined by values of a fixed set of variables - Goal test defined by constraints on varable values Backtracking = depth first search with one variable assigned/node Variable ordering and value selection heuristic help significantly Forward checking prevents assignments that guarantee later failure Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies The CSP representation allow analysis of problem structure Tree structured CSPs can be solved in linear time Iterative min-conflicts is usually effective in practise
Chapter 7	**Logical Agents**
Definitions: Inference algorithms	**Inference:** Steps in reasoning, moving from premises to conclusion **Resolution:** Inference rule: $\frac{a \vee b, \quad \neg a \vee c}{b \vee c}$ - Technique for solving logic **Syntax:** Defines meaning of symbols (relation between signs) **Semantics:** Meaning, interpretation of language/sentence **Entailment:** Describes relationship between statements: - Naturally follows, **KB $\models$ a** **Predicate:** Property or function holding Boolean for variable: - Apples(x) **Proposition** or assertion that don't depent on parameters - statements - Apple Sound: Truth-preserving. "Cant prove false" Completeness: Ability to derive any entailed sentence. "Prove truth" - Derive KB $\vdash_i$ a means KB $\models$ a
Knowledge bases	Inference engine ← domain-independent algorithms Knowledge base ← Domains-spesific content KB = set of sentences in a formal language. Declarative approach to building an agent (or other system). Tell it what it needs to know. Then it can ask itself what to do – answers should follow from KB. Agents can be viewed at the **knowledge level** – what they know, regardless of how they are implemented.

	Or at the **implementation level** – data structures in KB and algorithms that manipulate them
A simple knowledgebased agent	The agent must be able to: Represent states, actions etc Incorperate new percepts Update internal representation of the world, deduce hidden properties of the world, deduce appropriate actions Communicating with knowledge base: - **Tells** perceptions - **Asks** for action with reasoning - **Tells** action chosen

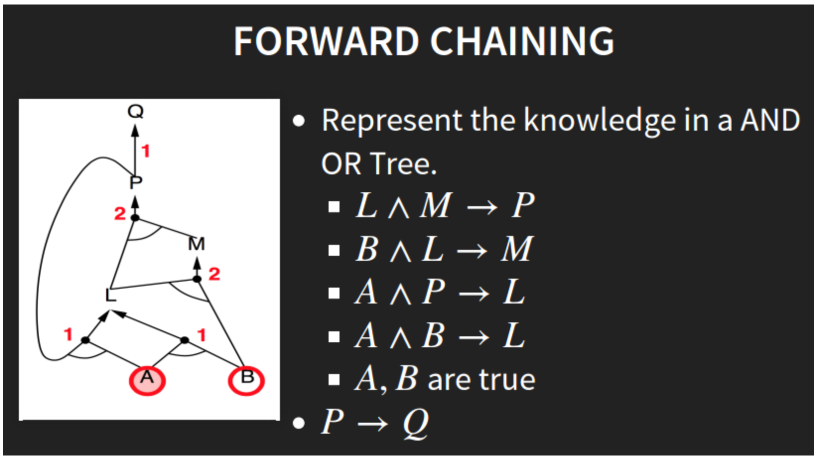
Wumpus World PEAS description	Performance measure: Gold + 1000, death – 1000, -1 per step, - 10 for using the arrow Environment: - Squares adjacent to wumpus are smelly - Squares adjacent to pi tare breezy - Gltter if god is in the same square - Shooting kills wumpus if you are facing it - Shooting uses up the only arrow - Grabbing pick up gold in the same square - Realeasing drops the god in the same square Actuators: Left turn, right turn, forward, grab, release, shoot, back Sensors: Breeze, Glitter, Smell.
Wumpus world characterization:	Observable?? No, only local perception Deterministic?? Yes, outcome exactly specified Episodic?? No, sequential at the level of action Static? Yes-Wumpus and Pits do not move Discrete?? Yes Singe-agent? Yes-Wumpus is essentially a natural feature.
Logic in general	Logic are formal languages for representing information such that conclusions can be drawn. *Syntax* defines the sentences in the language (How can the sentences be constructed to add sense). *Semantics* define the ”meaning” og sentences – define truth of a sentence in a world.
Entailment	Entailment means that one thing follow from another: KB |= a Knowledge base KB entails sentence a if a is true in all worlds where KB is true. KB |= a if and only iff (iff) M(KB) □ M(a) E.g. the KB containing ”the Giants won” and ”the Red won” entails ”Either the Giants won or the Red won”. E.g. x + y = 4 entails 4 = x + y Entailment is a relationship between sentences that is based don semtics.

Models (world)	Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be avaluated. We may m is a model of a sentence a if a is true in m M(a) is the set of all models of a. Then KB |= a if and only if M(KB) is a part of M(a) KB = Giants won and Red won, a = Giants won
Inference	KB $\|-_i$ a = sentence a can be derived from KB by procedure i. Consequence of KB are a haystack: a is a needle. Entailment = needle in haystack, inference = finding it. *Soundness:* i is sound if whenever KB |- a, it is also true that KB |= a *Completeness:* i is complete if whenever KB |= a it is also true that KB |- a Preview: we will define a logic(first –order logic) which is expressive enough to say almost anything of interest, and for which there exist a sound and complete inference procedure.

	That is, the procedure will answer any question whose answer follows from what is know by the KB.
Propositional logic: Syntax	Simple, powerful logic: "Truthbearing" ¬S (negation) S1 ∧ S2 (conjunction) S1 ∨ S2 (disjunction) S1 ⇒S2 (implication) S1 ⇔S2 (biconditional)
Logical euivalence	A ≡ B = A|=B and B|=A (α∧β) ≡ (β ∧ α) commutativity of ∧ (α∨β) ((α∧β)∧γ) ≡ (β ∨ α) commutativity of ∨ ((α∨β)∨γ) ≡ (α ∧ (β ∧ γ)) associativity of ∧ (α ∨ (β ∨ ¬(¬α) ≡ γ)) associativity of ∨ α double- (α ⇒β) ≡ negation elimination (¬β =⇒¬α) (α⇒β) ≡ contraposition (α ⇔β) ≡ (¬α ∨ β) implication elimination ((α =⇒β) ¬(α∧β) ≡ ∧ (β =⇒α)) biconditional elimination (¬α ¬(α∨β) ≡ ∨ ¬β) De Morgan (α∧(β∨γ)) ≡ (¬α ∧ ¬β) De Morgan (α∨(β∧γ)) ≡ ((α∧β)∨(α∧γ)) distributivityof ∧ over∨ ((α∨β)∧(α∨γ)) distributivityof ∨ over∧

Validity and satisfiability	A sentence is ***valid*** if it is **true in all** models E.g. True A V –A Validity is connected to inference via *the Deduction Theorem* KB |= a if and only if KB → a) is valid A sentence is ***satisfiable*** if it is true in **some model** e.g. A V B, C A sentence is *unsatisfiable* if its true in no models: A and not A Satisfiability is connected to inference via the following: KB |= a if and only if (KB and not a) is unsatifsfiable. e.e. prove a by reductio ad absurdum
Proof of methods	Proof methods divide into (roughly) two kinds: 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 alg. - Typically require translation of sentence into a normal form Model checking - Truth table enumeration (always exponential in n) - Improved backtracking: e.g. David.Putnam-Logenam-Loveland - Heuristic search model space (sound but incomplete) e.g. minconflics-like hill-climbing algorithms.

Horn Clauses	Disjuntion of literals (i.e. clauses) in which at most one of the literals is positive; the rest is negative **¬A V ¬B V ¬C V D** = [¬A V ¬B V ¬C] v D (associativity) = ¬[A and B and C] V D (de Morgan) **= [A and B and C] → D** Horn clauses are in the ideal format for logical rules.
Forward and backward chaining	Horn form (restricted) KB = conjunction of Horn clauses Horn clause= - Proposition symbol, or - Conjunction of symbol =→ symbol E.g., C ∧ (B =⇒ A)∧ (C ∧ D =⇒ B) Modus Ponens (for Horn Form): complete for Horn KBs α1,...,αn, α1 ∧ · ··∧ αn =⇒ β / β Can be used with forward chaining or backward chaining. Tthese algorithms are very natural and run in linear time

Forward chaining	Idea: Fire any rules whose premises are satisfied in the KB, add its conculusion to the KB, until query is found 
Proof of completeness (also saound)	FC derives every atomic sentence that is entailed by KB 1. FC reaches a fixed point where no new atomic sentences are derived 2. Consider the final state as a model m, assigning true/false to symbols Every clause in the original KB is true in m Proof: Suppose a clause a1 and .. and Ak → b is false in m. Then a1 and ... and Ak is true in m and b is false in m. Therefore the algorthm has not reached a fixed point. 4. Hence m is a model of KB 5. If KB |= q, q is true in very model of KB, including m. General idea: construct any model of KB by sound inference, check a.
Backward chaining	Idea: work backwards from the query q: to prove q by BC, chck if q is known already, or prove by BC all premises of some rule concluding q. Avoid loops: check if new sub-goal is already on the goal stack Avoid repeated work: check if new su-goal 1. Has already been proven true, or 2. Has already failed
FC vs BC	FC is data-driven, cf. Automatic, unconscious processing e.g. object recognition routine decisions May do lots of work that is irrelevant to the goal BC is goal driven. Appropriate for problem solving. E.g. where are my keys? How do i get into a PhD program Complexity of BC can be much less than liniear in size of KB
CNF – conjunctive normal form	Conjunction of disjunction of literals. Where disjunctions of literals = clauses

Convertion to CNF	$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ **1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Longrightarrow \beta) \land (\beta \Longrightarrow \alpha)$.** $(B_{1,1} \Longrightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Longrightarrow B_{1,1})$ **2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \lor \beta$.** $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$ **3. Move ¬ inwards using de Morgan's rules and double-negation:** $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$ **4. Apply distributivity law ($\lor$ over $\land$) and flatten:** $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$
Resolution	**Conjunctive Normal From** **E.g. $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$** **Resolution inference rule (for CNF) complete for propositional logic** **L1 V ... V Lk M1 V .... V Mn** **-- L1 V ... V Lk V M1 V ... VMn** **Resolution is sound and complete for propositional logic** **If you know alpha or beta and you know "not beta or gamma – you can conclude "aplpha or gamma"**

Summary	Logical agents apply inference to a knowledge base to derive new information and make decisions Basic concepts of logic: - **Syntax:** Formal structure of sentences - **Semantics:** Truth of sentenced in a model - **Entailment**: Necessary truth of one sentence given another - **Inference:** Deriving sentences from other sentences - **Soundness:** Derivation produce only entailed sentences - **Completeness** Derivations can produce all entailed sentences

	Wumpus world requires the ability to represent partial and negated information, reason by cases etc. Forward, backward chaining are **linear-time, complete** for Horn clauses. **Resolution is complete for propositional logic**. Propositional logic lacks expressive power
Chapter 8	**First-Order Logic**
Pros and cons of propositional logic	+ Propositional logic is declarative: pieces of syntax responds to facts + Propositional logic allow partial/disjunctive/negated information (unlike most data structures and databases) + Propositional logic is compositional: Meaning of (B and P) is derived from meaning of B and of P + Meaning in propositional logic is **context-independent** (Unlike natural language, where meaning depends on context) - Propositional logic **has very limited express**ive power (unlike natural language) - Eg. Cannot say ”pits cause breezes in adjacent squares” except by writing one sentence for each square
First order logic	Wheras propositional logic assumes world contains facts, first order logic (like natural language) assumes the world contains: - Objects: People, houses, numbers, theories, colors etc - Relations: Red, round, prime - Functions: father of, best friend, more than. Relations can be properties of more general n-ary relation Some relations are functions where a given object must be related to exactly one object in this way. There is only one value for a given input

<table>
<tr><td>Definitions</td><td colspan="3">Ontological Commitment:
- What it assumes about the nature of reality
Epistemological commitment; Possible states of knowledge for estatements</td></tr>
<tr><td rowspan="2">Logic in general</td><td>Language</td><td>Ontological Commitment</td><td>Epistemological commitment</td></tr>
<tr><td>Propositional logic
First order logic
Temporal logic
Probability theory
Fuzzy logic</td><td>Facts
Facts, objects, relation
Facts, objects
Relation, times facts
Facts + degree of truth</td><td>True/false/unknown
True/false/unknow
True/false/unknown
Degree of belief (0.1)
Known interval value</td></tr>
<tr><td>Syntax of FOL:
Basic elements</td><td colspan="3">Constants KingJohn, 2, UCB …
Predicates Brother, >, <, … (relations)
Functions Sqrt, LeftLegOf, …
Variables x, y, a, b, …
Connectives ∧ ∨ ¬ ⇒⇔
Equality =
Quantifiers ∀ ∃</td></tr>
<tr><td>Atomic sentences</td><td colspan="3">Atomic sentence = predicate(term1, ... , Term n) or term1 = term2.
Term = function(term 1, ... , term n) or constant or variable. A term is ”complicated name”; we can use a function to describe a symbol instead of naming all the symbols, i.e. we dont need to name all leftleg, we can use LeftLeg(John)

Term = logical expression that refers to an object – constant symbol</td></tr>
</table>

<table>
<tr><td></td><td>Predicate = relation

God oversikt på s.293</td></tr>
<tr><td>Complex sentences</td><td>Complex sentences are made from atomic sentences using connectives.
Disjunctions or conjunctions</td></tr>
<tr><td>Truth in first order logic</td><td>Sentences are true with respect to a model and an interpretation.
Model contains >= 1 objects (domain elements) and relations among them. Interpretation specifies referents for
Constant symbol → objects
Predicate symbols → relations (adjectives, adverbs)
Function symbols → functional relations, dependant on variable</td></tr>
</table>

	An atomic sentence predicate (term1, ... , term n) is true iff the objects referred to by term 1, ... , term n are in the relation referred to by predicate.
Universal quantification	$\forall \langle variables \rangle \langle sentence \rangle$ Everyone at NTNU is smart: $\forall x\ At(x,NTNU) \Longrightarrow Smart(x)$ $\forall x$ P is true in a model m iff P is true with x being each possible object in the model Main connectivity is for $\forall$ is $\rightarrow$
Ski race: Firstorder logic syntactic primitives	- Constants (A, B, C, D) - Variables – x, y, z - Functions – Best10K(person) -$\rightarrow$ time - Predicates – Greater(X, Y) - Computing entailment by enumerating FOL models is not easy (...)
Existensial quantification	$\exists \langle variables \rangle \langle sentence \rangle$ Someone at NTNU is smart: $\exists x\ At(x,NTNU) \wedge Smart(x)$ $\exists x$ P is true in a model m iff P is true with x being some possible object in the model Main connector for $\exists$ is $\wedge$
Properties of quantifiers	$\forall x\ \forall y$ isthesameas $\forall y\ \forall x$ $\exists x\ \exists y$ isthesameas $\exists y\ \exists x$ $\exists x\ \forall y$ isnotthesameas $\forall y\ \exists x$ $\exists x\ \forall y\ Loves(x,y)$ “There is a person who loves everyone in the world” $\forall y\ \exists x\ Loves(x,y)$ “Everyone in the world is loved by at least one person” Quantifier duality: each can be expressed using the other $\forall x\ Likes(x,IceCream)\ \neg\exists x\ \neg Likes(x,IceCream)$ $\exists x\ Likes(x,Broccoli)\ \neg\forall x\ \neg Likes(x,Broccoli)$

	But for simplicity and readability, we keep both Tip: if you are in doubt, it helps to put parenthesis in your expression, e.g. $\exists x\ (\forall y\ Loves(x,y))$
Equality	Term 1 = term 2 is true under a given interpretation if and only if term1 and term2 refer to the same object E.g., 1 = 2 and $\forall x \times(Sqrt(x), Sqrt(x)) = x$ are satisfiable 2 = 2 is valid E.g., definition of (full) Sibling in terms of Parent: $\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x=y) \wedge \exists m,f\ \neg(m=f) \wedge$ $Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$

Deducing hidden properties (with FOL) WUmpus world	Properties of locations: ∀x,t At(Agent, x,t) ∧ Smelt(t) =⇒Smelly(x) ∀x,t At(Agent, x,t) ∧ Breeze(t) =⇒Breezy(x) Squares are breezy near a pit: Diagnostic rule—infer cause from effect ∀y Breezy(y) =⇒∃ x Pit(x) ∧ Adjacent(x, y) Causal rule—infer effect from cause ∀x, y Pit(x) ∧ Adjacent(x, y) =⇒Breezy(y) Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy Definition for the Breezy predicate: ∀y Breezy(y) ⇔∃ x Pit(x) ∧ Adjacent(x, y) Propositional logic would require an axiom for each square!
Summary	FOL: - Objects and relations are semantic primitives - Syntax: constants, function, predicates(relations), equality, quantifiers Increased expressive power: sufficient to define wumpus world Developing a knowledge base in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences
	-
Chapter 9	**Inference in First-Order Logic.**
Universal quantification	Quantification express properties of collections of subjects ∀x P is true in a model m iff P is true with x being each possible object in the model
Existensial quantification	∃ (variable) (sentenc) i Someone at NTNU is smart: ∃ x At(x, NTNU) ∧ Smart(x) ∃ x P is true in a model m iff P is true with x being some possible object in the model

Universal instantiation (UI)	Every instantiation of a universally quantified sentence is entailed by it: $\forall v\ \alpha$ Subst({v/g}, α) for any variable v and ground term g (a term without variables) in the sentence α E.g., $\forall$ x King(x) $\wedge$ Greedy(x) =⇒Evil(x) yields King(John) $\wedge$ Greedy(John) =⇒Evil(John) King(Richard) $\wedge$ Greedy(Richard) =⇒Evil(Richard) King(Father(John)) $\wedge$ Greedy(Father(John)) =⇒Evil(Father(John))
Existensial instantiation (EI)	For any sentence α, variable v, and new constant symbol k that does not appear elsewhere in the knowledge base: $\exists v\ \alpha$ Subst({v/k}, α) E.g., $\exists$ x Crown(x) $\wedge$ OnHead(x, John) yields Crown(C1) $\wedge$ OnHead(C1, John) provided C1 is a new constant symbol, called a Skolem constant UI can be applied several times to add new sentencec; the new KB is logically equivalent to the old EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable, i.e. inferentially equivalent EI and UI allow to ”propositionalize” any FOL sentence or KB EI produce one instantiation per EQ sentence UI produces a whole set of instantiated sentences per UQ sentence
Instantiation	For Universal: Write out all cases For Existential: replace by some variable that could represent anyone Used to reduce to propositional form.

Reduction to propositional inference	Suppose the KB contains just the following: ∀ x King(x) ∧ Greedy(x) =⇒Evil(x) King(John) Greedy(John) Brother(Richard, John) Instantiating the universal sentence in all possible ways, we have King(John) ∧ Greedy(John) =⇒Evil(John) King(Richard) ∧ Greedy(Richard) =⇒Evil(Richard) King(John) Greedy(John) Brother(Richard, John) The new KB is propositionalized: proposition symbols are King(John), Greedy(John), Evil(John),King(Richard) etc. Claim: a ground sentence is entailed by new KB iff entailed by original KB

	Claim: every FOL KB can be propositionalized so as to preserve entailment Idea: propositionalize KB and quere, apply resolution, return result Problem: with function symbols, there are infinitely many ground terms e.g. Father(Father(Father(John)))) Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB Idea: For n = 0 to ∞ do: Create a propositional KB by instantiating with depth-n terms see if a is entailed by this KB Problem: works if a is entailed, loops if a is not entailed Hence, entailment in FOLK is Turing(1936)semidecidable: algorithms exist that say yes to every entailed sentence. But no algorithm exist that also says no to every nonentailed sentence.

Problems with propositionaliz a tion	Propositionalization seems to generate lots of irrelevant sentences E.g. from ∀ x King(x) ∧ Greedy(x) =⇒Evil(x) King(John) ∀ y Greedy(y) Brother(Richard, John) It seems obvious that Evil(John) but prop... produces a lots of facts such that Greedy(Richard) that are irrelevant. With p k-ary predicates and n constants, there are p * n^k instantiations. With function symbols, it gets much worse. Direct inference with FOL sentences? Its complete – that is every entailed sentence can be proved – but the space of possible model is infinite. We don't know until the proof is done that a sentence is entailed
Unification	Make a substitution that makes the premise of the implication identical to the sentences in the KB, so we can assert the conclusion. In other words: 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) θ = {x/John, y/John} works Unification find substitution that make different logical expressions look identical UNIFY takes two sentences and returns a unifier for them, if one exist Unify(α, β) = θ if αθ = βθ Basically, find a θ that makes the two clauses look alike $\text{UNIFY}(Knows(John, x),\ Knows(John, Jane)) = \{x/Jane\}$ $\text{UNIFY}(Knows(John, x),\ Knows(y, Bill)) = \{x/Bill, y/John\}$ $\text{UNIFY}(Knows(John, x),\ Knows(y, Mother(y))) = \{y/John, x/Mother(John)$ $\text{UNIFY}(Knows(John, x),\ Knows(x, Elizabeth)) = fail$.

Standarizing	Pretty straight forward but UNIFY (Knows(John,x), Knows (x, Elisabeth)) = fail Fails because x cannot be both John and Elisabeth We can avoid this problem by standardizing: UNIFY (Knows(John,x), Knows (z, Elisabeth))={x/Elisabeth, z/John}

Generalized Modus Ponens (GMP)	$P'1, p'2, \ldots, p'n, (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)$ --- $Q\theta$ **"If some facts are known, sentences can be reduced to the obvious entailment"** where $p'i\ \theta = pi\ \theta$ for all i p1´ is King(John) p1 is King(x) p2´ is Greedy(y) p2 is Greedy(x) θ is {x/John, y/John} q is Evil(x) qθ is Evil(John) GMP used with KB of definite clauses (exactly one positive literal) All variables assumed universally quantified Must show Soundness: Need to show that $p1', \ldots, pn', (p1 \wedge \ldots \wedge pn \Rightarrow q) \models q\theta$ Provided that $pi'\ \theta = pi\ \theta$ for all i Lemma: For any definite clause p, we have $p \models p\theta$ by UI 1. $(p1 \wedge \ldots \wedge pn \Rightarrow q) \models (p1 \wedge \ldots \wedge pn \Rightarrow q)\theta = (p1\theta \wedge \ldots \wedge pn\theta \Rightarrow q\theta)$ 2. $p1', \ldots, pn' \models p1' \wedge \ldots \wedge pn' \models p1'\ \theta \wedge \ldots \wedge pn'\ \theta$ 3. From 1 and 2, qθ follows by ordinary Modus Ponens
Example Knowledge Base (from forward chaining)	The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American. Prove that Col. West is a criminal … it is a crime for an American to sell weapons to hostile nations: American(x)∧Weapon(y)∧Sells(x,y,z)∧Hostile(z) ⇒Criminal(x) Nono … has some missiles, i.e., ∃ x Owns(Nono, x) ∧ Missile(x): Owns(Nono, M1) and Missile(M1) … all of its missiles were sold to it by Colonel West ∀x Missile(x)∧Owns(Nono,x) ⇒Sells(West,x,Nono) Missiles are weapons: Missile(x) ⇒Weapon(x)

	An enemy of America counts as “hostile”: Enemy(x,America) ⇒Hostile(x) West, who is American . . . American(West) The country Nono, an enemy of America . . . Enemy(Nono,America)
Forward chaining	Idea: Start with atomic sentences in the KB Apply Modus Ponens Add new atomic sentences until no further inferences can be made Works well for a KB consisting of Situation → Response clauses when processing newly arrived data First Order Definite Clauses: Disjunctions of literals of which exactly one is positive $\forall$ x King(x) $\wedge$ Greedy(x) =⇒Evil(x) King(John) $\forall$ y Greedy(y) First Order Definite Clauses can include variables Variables are assumed to universally quantified Greedy(y) means $\forall$ y Greedy(y) Not every KB can be converted into definite clauses
Properties of forward chaining	Sound and complete for first-order definite clauses (proof similar to propositional proof) Datalog = first-order definite clauses + no functions (e.g. crime KB). FC terminates for datalog in poly iteration: at most p * n^k literals. May not terminate in general if a is not entailed. This is unavoidable: entailment with definite clauses is semidecidable.
Efficiency of forward chaining	Simple observation: No need to match a rule on iteration k if a premise wasn’t added on iteration k-1 → Match each rule whose premise constains a newly added literal. Matching itself can be expensive. Database indexing allow O(1) retrieval of know facts e.g. query Missile(x) retrieves Missile(M1) Matching conjunctive premises against know facts is NP-hard. Forward chaining is widely used in deductive databases
Backward chaining	Idea: Given a query, find all substitutions that satisfy the query Algorthm: Works on list of goals, starting with original query Algorithm find every clause in the KB that unifies with the positive literal and adds remainder to list of goals

Properties of backward chaining	Depth-first recursive proof search. Space is linear in size of proof. Incomplete due to infinite loops → fix by checking current goal against every goal on stack Inefficient due to repeated subgoals(both success and failure)

	→ fix using cashing of previous results (extra space!) Widely used(without impovements!) for logic programming
Prolog system	**Basis: Backward chaining with Horn clauses** = set of clauses = head :- $literal_1, ..., literal_n$ criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z) Depth first, left-to-right backward chaining
Prolog examples	Depth-first search from a start state X: dfs(X) :- goal(X). dfs(X) :- successor(X,S),dfs(S). No need to loop over S: successor succeeds for each Appending two lists to produce a third: append([],Y,Y). append([X|L],Y,[X|Z]) :- append(L,Y,Z). query: append(A,B,[1,2]) ? answers: A=[] B=[1,2] A=[1] B=[2] A=[1,2] B=[]
Resolution (and Inference rule)	First order logic requires sentences in CNF (Conjunctive Normal Form) Each clause is a disjuntion of literals, but literals can contain variables, which are assumed to be universally quantified. Example $\forall x$ American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) =⇒ Criminal(x) = ¬American(x)∨¬Weapon(y)∨¬Sells(x, y, z)∨¬Hostile(z)∨Criminal(x) Full first order-version (inference) $l_1 \vee \cdots \vee l_k,\ m_1 \vee \cdots \vee m_n$ $(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$ Where Unify(\`i , ¬mj) = θ. Two standarized clauses can be resolved if they contain complementary literals (one is the negation of the other). FOLK literals are complementary if one unifies with the negation of the other ¬Rich(x) ∨ Unhappy(x) Rich(Ken) ――――――――――――――― Unhappy(Ken) with θ = {x/Ken}

)θ

	Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL Resolution proves that KB entails a by proving (KB ∧ ¬α) unsatisfiable – by deriving the empty clause.
Conversion to CNF	Everyone who loves all animals is loved by someone: ∀x [∀y Animal(y) ⇒Loves(x,y)] ⇒[∃y Loves(y,x)] 1. Eliminate biconditionals and implications ∀x [¬∀y ¬Animal(y)∨Loves(x,y)]∨[∃y Loves(y,x)] 2. Move ¬ inwards: ¬∀x,p ≡ ∃x ¬p, ¬∃x,p ≡ ∀x ¬p: ∀x [∃y ¬(¬Animal(y)∨Loves(x,y))]∨[∃y Loves(y,x)] ∀x [∃y ¬¬Animal(y)∧¬Loves(x,y)]∨[∃y Loves(y,x)] ∀x [∃y Animal(y)∧¬Loves(x,y)]∨[∃y Loves(y,x)] 3. **Standardize variables:** each quantifier should use a different one ∀x [∃y Animal(y)∧¬Loves(x,y)]∨[∃z Loves(z,x)] 4. **Skolemize**: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables: ∀x [Animal(**F(x)**) ∧ ¬Loves(x,**F(x))]** ∨ Loves(**G(x),**x) 5. **Drop universal** quantifiers: [Animal(F(x))∧¬Loves(x,F(x))]∨Loves(G(x),x) 6. Distribute ∧ over ∨: [Animal(F(x)) ∨ Loves(G(x),x)] ∧ [¬Loves(x,F(x)) ∨ Loves(G(x),x)]

Skolemization	Removal of existensial quantifiers Each existentially quantified variable is replaced by a Skolem constant or function **Skolem constant**: If the existential variable is not within the scope of any universaly quantified variable. Every instance of the existentially quantified variable is replaced with the same unique constant, a brand new one that does not appear anywhere else. $\exists$ y : (P(x) $\wedge$ Q(x) converted to: P(CC) $\wedge$ Q(CC) **Skolem function**. If any existential quantifier is in the scope of a(or more, e.g, n) universally quantified variable, then replace it with a unique n-ary function over these universally quantified variables: $\forall$ x $\exists$ y : (P(x) $\vee$ Q(y)) converted to: $\forall$ x P(x) $\vee$ Q(f (x)

	Remove then the existensial quantifier
Tableaux method	Syntatic, but based on clear semantic intuition Adatpable, does not depend on human insight/intuition Model building tool Systematic test for validity – Try to falsify – Apply expansion rules for binary connectivity til Tableaux – Closed if T and F apprears for some instance – Else its still open, and it cannot be a balid statement **Conjunctive** $\frac{T(\phi \wedge \psi)}{T\phi \quad T\psi}$ $\frac{F(\phi \vee \psi)}{F\phi \quad F\psi}$ $\frac{F(\phi \rightarrow \psi)}{T\phi \quad F\psi}$ **Disjunctive** $\frac{F(\phi \wedge \psi)}{F\phi \mid F\psi}$ $\frac{T(\phi \vee \psi)}{T\phi \mid T\psi}$ $\frac{T(\phi \rightarrow \psi)}{F\phi \mid T\psi}$ 1 F $(p \vee \neg p)$ √ 2 F p 1,F$\vee$ 3 F $\neg p$ 1,F$\vee$,√ 4 T p 3,F$\neg$

Summary	Inference rules for quantifiers: Universal instantiation and existensial instantiation Unification by making different logical expressions look identical Forward and backward chaining (with a taste of prolog) Resolution using conjunctive normal form
Chapter 10 & 11	**Planning**
What is planning	A plan is a collection of actions for performing some task. There are many programs that help human planners. The goal in AI is to generate plans automatically
Issues in planning	Issues 1. Representation of states, actions, goals and plans 2. Planning mechanism Treatment of change is an important concept in planning
Situation calculus	Facts hold in situations – not eternally Something true in one situation may not be true in another In situation calculus, situations capture the time aspect of facts. Situations **are treated as objects**, and can be referred like objects: ON(A, B, s-1) Result function returns as value a situation
Planning and situation calculus	Representation of states, goal, actions in Situation Calculus, and using Resolution to check whether the Goal can be inferred - **A factored representation**
Representation in Situation calculus	A situation is an argument in the representations of states and action Effects of action on **fluents** (predicates or functions whose values may vary from situation to situation) are represented through - change axioms, e.g., At(M,B,s) → On(M,B,climb(M,B,s)) – means you are on what you climb - frame axioms, e.g., At(B,loc,s) → At(B,loc,climb(M,B,s)) – means climbing does not move B Frame axioms are for handling well known AI-planning problem called frame problem Frame problem: representation of domain/environment rules. What/not changes when an action is executed

<table>
<tr><td>Planning and (more) Logic</td><td>In planning with Situation calculus we can use logic both for representation and reasoning (resolution)
However logic is slow in general as a reasoning system
Additionally there may be need for huge nr of frame axioms in nontrivial domains.</td></tr>
</table>

<table>
<tr><td></td><td>In many cases, the full expressiveness of FOL (as is used in situation calculus) is not needed. Simpler planning mechanism using still logic based but more structured/effective languages

We can then use logic for representation of states and actions, and use forward/backward chaining to find a plan. Planning then involves searching over sets of states,</td></tr>
<tr><td>Search vs planning</td><td>A planning problem is described just like a search problem (states, actions/operators, goal) but the problem representation is more structured.
<table>
<tr><td></td><td>Search</td><td>Planning</td></tr>
<tr><td>States
Actions
Goal
Plan</td><td>Data structures
Code
Goal test
Sequence from S0</td><td>Logical sentences
Preconditions/outcomes
Logical sentence (conjunction)
Constraints on actions</td></tr>
</table>
wo main difficulties arise in more complex search problems:
- Huge branching factor
- Difficulty of finding good heuristic functions</td></tr>
<tr><td>Planning vs.logic</td><td>Planning involves searching over sets of states
We use logic to describe sets of states
Key idea: describe states and actions in propositional logic and use forward/backward chaining to find a plan</td></tr>
<tr><td>STRIPS</td><td>Developed at Stanford in early 1970s (Standford Research institute planning system) for the first ”intelligent” robot.
States are represented as first-order predicates over objects.
Closed-world assumption: everything not stated is false
Actions defined in terms of:
- Precondition: when can the action be applied?
- Effects: What happens after the actions?
(No explicit descriptions of how the action should be executed) Goal: Conjunction of literals</td></tr>
</table>

STRIPS representations	States are represented as conjunctions: In(Robot, room) ∧ ¬In(Charger, r) ∧ . . . Goals are represented as conjunctions: In(Robot,room)∧In(Charger,r) (∃r isimplicit) Actions: - Name: Go(here,there)

	- Preconditions: expressed as conjunctions: At(Robot, here) ∧ Path(here,there) - Effects (postconditions): expressed as conjunctions: At(Robot,there) ∧ ¬At(Robot, here) Variables can only be instantiated with objects of correct type
STRIPS action representation and semantics	Actions hava a **name, preconditions and effects**. Preconditions are conjunctions of positive lierals* If the precondition is false in a world state, the action does not change anything (since it cannot be applied) Effects are represented in terms of: - ADD-list: List of propositions that become true after action - Delete-list: list of propositions that become false after action This is a very restricted language, meaning we can do efficient inference! *Problem domain definition language (PDDL) Accepts negative literals in preconditions and goals
	$Action(Buy(x),$ PRECOND : $At(s) \land Sells(s, x, b) \land HaveMoney(b)$ DELETE-LIST : $HaveMoney(b)$ ADD-LIST : $Have(x))$ Buying action in (AIMA version of) PDDL: $Action(Buy(x),$ PRECOND : $At(s) \land Sells(s, x, b) \land HaveMoney(b)$ EFFECT : $\neg HaveMoney(b) \land Have(x))$

Example: Buy action	In general, note that many importain details are abstracted away! Additional propositions can be added to show that now the store has the money, the stock has decreased etc, PDDL (**Planning Domain Definition Language**) describes the four thing we need to define a search problem: Initial state, actions available in a state, result, and goal test. Accepts negative literals in precondition, effect and goal: **Initial state, actions, result, goal test**
Pros and cons	Pros - Since its restricted, inference can be done efficiently - All actions are additions and deletions of propositions in KB Cons - Assumes only a small number of propositions will change for each action - Limited language, so not apllicable to all domains of interest
Two basic approaches to planning	1. State-space planning works at the level of states and actions – Finding a plan is formulated as search through state space – Most similar to constructive search 2. Plan-space planning works at the level of plans
	-Finding a plan is formulated as search through space of plans - Start with partial incorrect plan, then apply changes to correct it. Most similar to iterative improvement. (Local search)
State-space planners	a) Progression planners reason from the start start state, trying to find the actions that can be applied (match preconditions) b) Regression planners reason from the goal state, trying to find the actions that willl lead to the goal (match effects)
Progression (forward planning)	1. Determine all actions that are applicable in the start state 2. Ground the actions, by replacing any variable with constants 3. Choose an action to apply 4. Determine the new content of the knowledge base, based on the action description 5. Repeat until goal state is reached Note that in this case there are a lot of possible actions to perform

<table>
<tr><td>Regression (backward planning)</td><td>1. Pick an action that satisfied (some of) the goal propositions
2. Make a new goal by:
- Removing the goal conditions satisfied by the action
- Adding the preconditions of this action
- Keeping any unsolved goal propositions
3. Repeat until the goal set is satisfied by the start-state. Note that in this case the order in which we try to achieve these propositions matters!.</td></tr>
<tr><td>Efficiency of state-space search</td><td>Backward search has lower branching factor for most domains. However, difficult to find good heuristic for backward search As for CSPs, planning uses factored representations – Enables good domain-independent heuristics.

Heuristics can be derived automatically from the action schema</td></tr>
<tr><td>Sussman Anomaly</td><td>Linear planners typically seperate the goal (stack A atop B atop C) into subgoals, such as:
1. Get A atop B
2. Get B atop C
By removing C from A and moving A atop B we accomplish subgoal 1. But the we cannot accomplish subgoal 2 without undoing subgoal 1!</td></tr>
<tr><td>Total vs Partial order</td><td>Total order: Plan is always a strict sequence of actions
Partial order: Plan steps may be unordered</td></tr>
<tr><td>Partial order planning (POP)</td><td>Search in plan space and use least commitment whenever possible

In state space search
- Search space is a set of states of the world
- Actions cause transitions between states
- Plan is a path through state space

In plan space search
- Search space is a set of partially ordered plan
- Plan operators cause transitions
- Goal is a legal plan</td></tr>
</table>

<table>
<tr><td></td><td>Least commitment: Only make choices thar are relevant to solving the current part of the problem
Plan operators: add actions, add causal links, specify orderings, bind variables</td></tr>
</table>

<table>
<tr>
<td>POP process</td>
<td>
Start with an empty plan consisting of:

- Start step with the initial state descriptions as its effect
- Finish step with the goal description as its precondition

Proceed by:

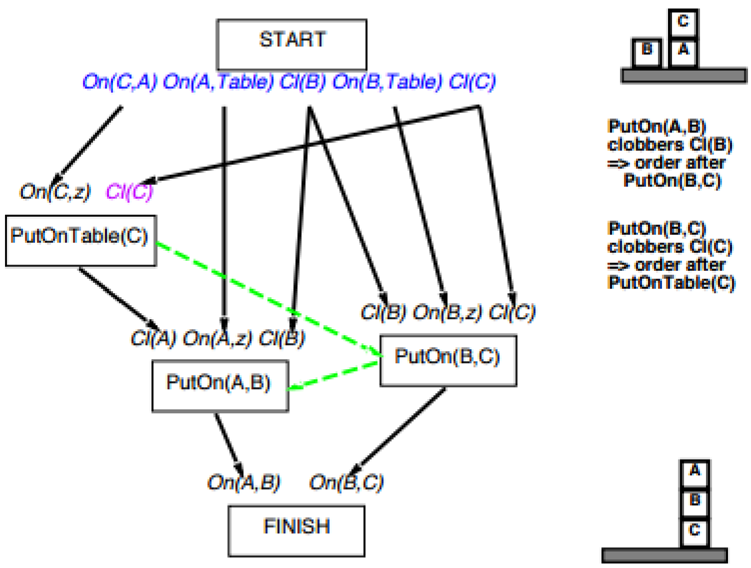
- Adding actions to achieve preconditions
- Adding causal links from an existing action to achieve preconditions
- Order on action w.r.t. another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or if conflict is unresolvable

A plan is complete iff every precondition is achieved

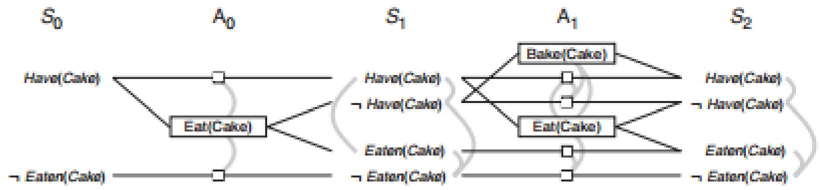
A precondition is achieved iff it is the best effect of an earlier step and no possibly intervening step undoes it



</td>
</tr>
<tr>
<td>Discussion of POP</td>
<td>
Advantages:

- Plan steps may be executed unordered
- Handles concurrent plans
- Least commitment can lead to shorter search times
- Sound and complete
- Typically produces the optimal plan

Disadvantages
-Complex plan operators lead to high cost for generating actions
Larger search space because of concurrent actions
</td>
</tr>
</table>

GraphPlan	Planning graph: a polynomial size approximation of a tree with all possible actions from an initial state S0 to successor states Can be used as an admissble heuristic to determine if G is reachable from S0 GraphPlan extracts a plan directly from a such planning graph Main idea: - Construct a graph that encodes constraints on possible plans – if a valid plan exist it will be part of this planning graph, so search only within this graph List the initial state S0 Until a valid solution is found: - List all actions resulting from the previous literals - List all literals resulting from the actions - Find mutexes between actions - Find mutexes between literals - Check for a valid solution
Planning graph	Two types of noded, arranged in alternating levels: - Propositions - Actions Three types of edges between levels: - Precondition: edge from P to A if P is a precondition of A - Add: edge from A to P if A has P as effect - Delete: edge from A to ¬P if A deletes P Action level includes actions whose preconditions are satisified in the previous level, plus persistence actions (”no-op”
Constructing the planning graph	Level S0 is initialized with all the literals from the initial state. Add an action at level Ai if all its preconditions are present in level Si. Add a proposition in level Si+1 if it is the effect of some action in level Ai (including persistence actions) Maintain a set of exclusion relations (mutexes) to eliminate incompatible propositions and actions **Mutexes define relations where the participants of same level contradict eachother, in either result, condition, or both.** 

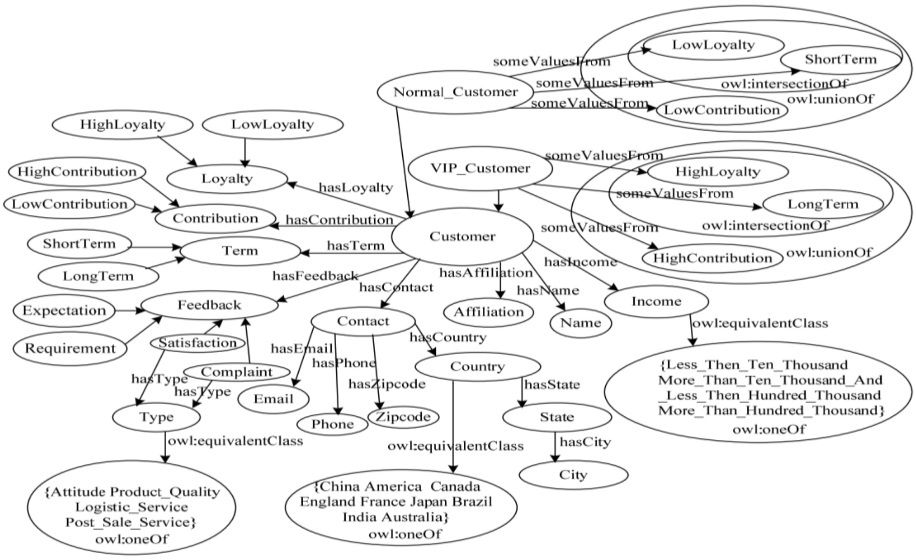
Mutual exclusions	**Two actions are mutually exclusive (mutex) at some stage if no valid plan could contain both at that stage** Two actions at the same level can be mutex of: - Inconsistent effects: an effect of one negates the effect of the other - Interference: One negates a precondition of the other - Competing needs: the actions have mutex preconditions Two propositions at the same level are mutex if: - One is a negation of the other - Inconsistent support: All ways of achieving them are pairwise mutex
Observations	Number of propositions always increases - Because all the ones from the previous level are carried forward Number of action always increases - Because number of satisfied precondition increases Number of propositions that are mutex decrease: - Because there are more ways to achieve same propositions Number of actions that are mutex decreases: - Because of the decrease in mutexes between propositions After some time, all levels become identical: graph "levels off" Because there is a finite number of propositions and actions, mutexes will not reappear
Valid plan	A valid plan is a subgraph of the planning graph such that: - All goal propositions are satisifed in the last level
	- No goal propositions are mutex - Actions at the same level are not mutex - Each actions's preconditions are made true by the plan Basic algorithm: 1. Grow the planning graph until all goal propositions are reachable and not mutex 2. If the graph levels off first, return failure (no valid plan exist) 3. Search the graph for a solution (CSP or backward search) 4. If no valid plan is found, add a level and try again

Planning and acting in the real world	Planners used in the real world are more complex Durations and resource constraints - E.g. limited number of staff or time - Plan first, schedule later: Job shop scheduling Very large state spaces: - Actions are often low level (really long plans) - Decompose problem: hierarchical planning Uncertainty: - Non-correct and non-complete information - Contingency planning, replanning
Hierarchical Task Network (HTN)	State the same as in classical planning Two types of actions - Primitive actions can be executed directly e.g. go forward - High-level action can be refined into a sequence of actions e.g. dock-with-charger High level actions can be refined to primitive actions (an implementation) or other HLAs, both with possible precedence constraints Refinements given by plan library - Defined by domain experts - Learned through experience
Searching for solutions	An instance of a planning problem starts with an initial state (Act) Creating a plan is done by repeatedly applying refinements recursively to the high-level actions, until we reach the level of the primitive tasks (which can be executed directly) Backtracking is done if necessary (e.g. if the internal constraints in the task network cannot be satisfied) Easy if HLAs only contain one implementation (preconditions and effects can be used directly) However, they usually don’t: - Search among each possible implementations - Reason directly about the HLAs
The real world	Things are usually not as exprected Incomplete information: - Unknown preconditions - Disjunctive efects : Inflate(x) causes Inflated(x) according to the knowledge base, but in reality it actually causes

	Inflated(x) ∨ SlowHiss(x) ∨ Burst(x) ∨ BrokenPump ∨ . . . Incorrect information: - Current state incorrect e.g. spare NOT intact - Mssing/incorrect postcondition in operators Qualification problem: can never finish listing all the required preconditions and possible conditional outcomes of actions
Solutions	Conditional planning 1. Plans include obervation actions which obtain information 2. Sub-plans are created for each contingency (each possible outcome of the obervation action) 3. E.g. Check the tire. If it is intact, then we're ok, otherwise there are several possible solutions: inflate, call NAF ... Experie because it plans for many unlikely cases Monitoring/replanning: 1. Assume normal states outcomes 2. Check progress during executions, replan if necessary Unanticipated outcomes may lead to failure (e.g. no NAF card). In general, some monitoring is unavoidable.
Summary	Planning is very related to search, but allows the actions/states have more structure We typically use logical inference to construct solution State-space vs plan-space planning Least-commitment: we build partial plans, order them only as necessary Planning graphs can be used as heuristic or searched directly for a planning solution (GraphPlan) In the real world, it is necessary to consider failure cases – replanning Hierarchy and abstraction make planning more efficient.

<table>
<tr><th>Chapter 12</th><th>Knowledge Representation</th></tr>
<tr><td>Ontological engineering</td><td>
Ontology – The nature of being/existence
How to create more general and flexible representations
- Concepts like actions, time, physical objects, beliiefs
- Operates on a bigger scale than knowledge engineering
Define general framework of concepts
- Upper ontology
Limitations of logic representations
- Exceptions and uncertainty e.g. tomatoes can be red, green, and yellow

To summarize, why would someone want to develop an ontology? Some of the reasons are:
– To share common understanding of the structure of information among people or software agents
– To enable reuse of domain knowledge
– To make domain assumptions explicit
– To analyze domain knowledge

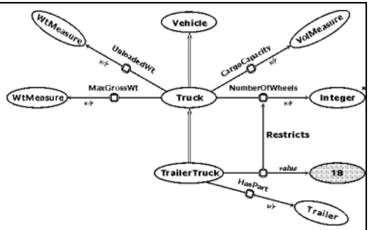
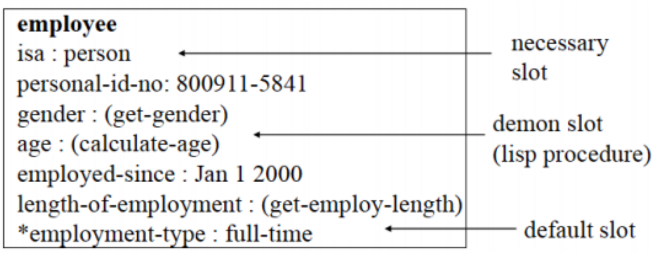
</td></tr>
</table>

Difference with special-purpose ontologies	A general-purpose ontology should be applicable in more or less any special-purpose domain - Add domain-specific axioms In any sufficiently demanding domain different areas of knowledge need to be unified in order to be comparable - Reasoning and problems solving could involve several areas simultaneously What do we need to express? - Categories, measures, composite objects, time, space, change, events, processes, physical objects, substances, mental objects, beliefs
Categories and objects	Knowledge representation requires the organisation of objects into categories - Interaction at the level of objects - Reasoning at the level of categories Categories play a role in the predictions about objects – Based on perceived properties Categories can be represented in two ways by first-order logic - Predicates: apple(x) - Reification ("thingification") of categories into objects: apples Category = set of its members
Category organization	Relation = inheritance - All instances of food are edible, fruits is a subclass of food, and apples is a subclass of fruit, then apple is edible Defines a taxonomy
FOL and categories	An object is a member of a category $BB12 \in Basketballs$ A category is a subclass of another category $Basketballs \subset Balls$ All members of a category have some properties $(x \in Basketballs) \Rightarrow Spherical(x)$ All members of a category can be recognized by some properties $Orange(x) \wedge Round(x) \wedge Diameter(x) = 9.5" \wedge x \in Balls \Rightarrow x \in Basketballs$ A category as a whole has some properties $Dogs \in DomesticatedSpecies$
Relation between categories	A set of categories s constitutes an exhaustive decomposition of a category c if all members of the set c are covered by categories - Set S is covered by category C A disjoint exhaustive decomposition is a partition $Partition(s, c) \Leftrightarrow Disjoint(s) \wedge ExhaustiveDecomposition(s, c)$ $Partition(\{Males, Females\}, Animals)$ - Members of set is included in **one** group of partition

Natural kinds	Many categories have no clear-cut definitions (chair, bush, book) Tomatoes: sometimes green, red, yellow, black. Mostly round. One solution: **category Typical**(Tomatoes)

	x ∈ Typical(Tomatoes) ⇒Red(x) ∧ Spherical(x) We can write down **useful facts** about categories without providing exact definitions What about "bachelor"? Quine challenged the utility of the notion of strict definition. We might question a statement such as "the Pope is a bachelor".
Physical composition	One objects may be **part of another PartOf**(Bucharest, Romania) PartOf(Romania, EasternEurope), PartOf(EaternEurope, Europe) The PartOf predicate is **transitive (and reflexive)** **PartOf(x, x), PartOf (x, y) ∧ PartOf (y, z) ⇒PartOf (x, z)** So we can infer that PartOf(Bucharest, Europe) Composite objects are often characterized by structural relations among parts
Measurements	Objects have height, mass, cost, ..., values we assign to these are measures Combine Unit Function with a number: Length(L1) = inches(1,5) = centimeter(3,81) Conversion between units: Centimeters(2,54 x d) = Inches(d) Some measures have no scale: beauty, defficulty, etc. Most important aspect of measures is that they are **orderable**. Don't care about actual numbers (and apple can have deliciousness .9 or 1)
Event Calculus	Situations calculus deals with discrete and instantaneous events, be we need to address what happens during the action, this is *event calculus.* Based on time instead of situations. Reifies fluents and events - **Fluent:** Condition that changes - **Event:** Fluent at set time interval The fluent At(Knut, NTNU) refers to the fact of Knut being at NTNU, but does not say whether it is true. For this we need the predicate T: T(AT(Knut, NTNU), t) – which is true if Knut is at NTNU at time t I = time interval i = (start,end)
Reasoning systems for categories	Semantic networks - Visualize knowledge base - Efficient algorithms for category membership inference Description logics - Formal language for constructing and combining category definitions

	- Efficient algorithms to decide subset and superset relationships between categories. Formal language for constructing and combining category definitions - Efficient algorithms to decide subset and superset relationships between categories.

Semantic networks, Quillian	Developed by Ross Quillian as ”psychological model of associative memory” (1968)  Associations theories define the meaning of an objects in terms of a network of associations with other objects in a domain or a knowledge base – binary relations A structure that shows relation between concepts can be categories and sets of entities. The relation can be memberships, subclasses and attributes or others Logic vs semantic networks SN: Many variations – all represent individual objects, categories of objects and relationships among objects Main idea: **Knowledge is not a large collection of small pieces of knowledge but larger pieces that are highly interconnected.** The meaning of a concept emerges from how it is connected to other concepts. Allows for inheritance reasoning - Female person inherits all properties from person - Similar to objects-oriented programming Inference of inverse links, e.g. SisterOF vs HasSister Drawbacks: - Links can only be assert binary relations - Can be resolved by reification of the proposition as an even Remeber**: Frame-based representations** – with data 

Description logic	Designed to decribe definitions and properties about categories; a formalization of semantic networks Principal inference task is: - *Subsumption:* checking if one category is the subset of another by comparing their definitions - *Classification:* checking whether an object belong to a category - *Consistency:* checking whether the category membership criteria are logically satisfiable
Reasoning with default information	*Circumscription*: Specify predicates that are "as false as possible", i.e. false for every object except those that are known to be true *Default logic:* a formalism where default rules can be written P : J1, ..., Jn/C
	Where P is the prerequisite, C is the conclusion and Ji are the justifications – if any of the justifications can be proven false, then the conclusion cannot be drawn Bird(x): Flies(x)/Flies(x)
Truth maintenance system	Many of the inferences have default status rather than being absolutely certain - Inferred facts can be wrong and need to be retracted = belif revision - Assume knowledge base contains sentence P and we want to execute Tell(KB, ¬P) - To avoid contradiction: Retract(KB, P) - But, what about sentences inferred from P? Truth maintenance systems are designed to handle these complications
Summary	Large-scale knowledge representation require a general-prupose ontology to organize and tie together the various spesific domains of knowledge A general-purpose ontology needs to cover a wide variety of knowledge and should be capable, in principle of handling any domain We represent an upper ontology based on categories and the event calculus (action, events and time) Special purpose representation systems – semantic netowrks and decription logic have been developed to help in organizing a hierarchy of categories. Inheritance is an important from of inference Mononotonic logics, such as circumscription and default logic are intended to capture default reasoning in general

<table>
<tr><td>(Some pieces from wiki – not sure if this is still in the curriculum</td><td></td></tr>
<tr><td>Knowledge based system</td><td>Is a model of semthing in the real world (outside the agent). It is 3 main players:
- Knowledge engineer – deign, builds and test the ”expert system”
- Domain expert – posesses the skill and knowledge to find a solution
- End user – the final system should meed user needs

The KB represent the knowledge using a KB language, which is a system for encoding knowledge. The inference engine has the ability to find implicit knowledge by reasoning over the explicit knowledge. Decides what kind of conclusions can be drawn</td></tr>
<tr><td></td><td><table>
<tr><td>Declerative knowledge</td><td>Expressed in devlarative sentences or indicative propositions (knowinf of)</td></tr>
<tr><td>Procedural knowledge</td><td>Knowlegde exercised in the performance of some task e.g. shopping list</td></tr>
<tr><td>Domain knowledge</td><td>What we reason about</td></tr>
<tr><td>Strategic knowledge</td><td>How we reason</td></tr>
</table></td></tr>
<tr><td>Five roles of knowlege representation</td><td>- Surrogate: A representation is a substitute for direct interaction with the world
- All representations are approximation to reality and they are invariably imperfect
- Fragmentary theory of intelligent reasoning
- Is a medium for efficient computation
- Is a medium of human expression</td></tr>
</table>

Rule based systems (early KBs expert systems)	*Working memory:* contains facts about the world, observed directly or derived from a rule *Rule base:* Contains rules, where each rule is a step in a problem solving process (if-then format) *Interpreter:* Match rules and the current contents of the working memory
Knowledge representation languages	*Syntatic:* Possible allowed constructions *Semantic:* What the representation mean, mapping from sentence to world *Inferential* Interpreter, what conclusions can be drawn Requirements: - Representation adequacy: Should allow for representing all the required knowledge - Inferential adequacy: Should allow inferring new knowledge - Inferential efficiency: Inferences should be efficient - Clear syntax and semantics - Naturalness: Easy to read and use
Chapter 22	**Natural language processing**
Text Classification	This is known as categorization: given a text of some kind, decide which of predefined set of classes it belongs to: Use cases → Spam detection.

	E.g.: Categorize text as spam or some other thing base don probabileties Ambiguity – A sentence has more than one meaning Redundancy – Some information is expressed more than once Challenges: - Distinguish language, system - Grammar as rules, semantics as meaning - Pragmatics: How its related to people N-grams for chaining words

NLP	Natural language processing (NLP) is about processing human language with computers. Focus on human-computer interaction Goal(with lecture) Introduce some aspects of NLP, without too much technical detail The lecture covers 3 steps: 1. Retrieve candidate documents (information Retrieval) 2. Extract and match parameters from query to those of candidate documents (Information Extraction) 3. Check disagreements (Text Categorization, Sentiment Analysis) Interpretation and generation
Information retrieval - IR	IR: The task of finding documents that are relevant to a user's need for information. Web search engines – Keyword/Text retrieval is not the only way to search IR Setting: ***Source***: **Collection** of documents - Corpus **I*nput***: **query** of topic station what the user wants to know ***Outpu****t* (ranked): subset of relevant, ranked documents
IR – Boolean keyword model	Early IR system used Boolean Keyword Model - Lookup through reverse index Example query: deforestation AND (Amazon OR Sahara) Disadvantages: - Hit or miss: No way to rank documents for relevancy - Hard to use for lay person
IR - Statistics	Modern IR system mostly based on statistics of word counts (terms) Intuition: if the (important) words from the query occur frequently in a documents, it is probably relevant Web search engines are a special case of IR - Can exploit hyperlink structure (PageRank) - Can exploit user behaviour statistics

IR scoring	Based on statistcs of word counts(terms) Document collection D = d1, d2, ..., dm Query Q = t1, t2, ..., tn Scoring function S(d, q) = s Term frquency TF(t,d) First attempt used TF as scoring function (counted for all t) – frequent word that were not informative added a lot of weight. – word filtering Intuition: if a word occurs in most documents, then it is not very relevant → Now use IDF – inverse document frequency Score: IDF x TF – for all terms (solution) (Hundreds of variants: BM25 – is famous – create index for each vocabulary word for faster lookup) Basis for most modern IR-systems Evaluation - Precision: the proportion of returned documents that are truly relevant - Recall: The proportion of truly relevant documents that are returned There is usually a trade off betweeen the two. F-score = 2 x Prec x Rec/(Prec + Rec) Refinements: Case folding, stemming (reduce similar words), synonyms, metadata There is also a lot of linguistic aspects to take into consideration – normalization, stemming, lemmatization
The HITS algorithm	Find pages relevant to query. Get all pages in the link neighbourhood. A page is considered an authority if other pages point to it, and a hib if it points to others. Iterate ground up and build list over authority and hubs. Normalize (Google – pageRank which have common traits)