Künstliche Intelligenz I - Zusammenfassung

Autoren: Linda Schneider und Julian Kotzur

What is AI?

Definition (AI).
- AI is a subfield of Computer Science that's concerned with the automation of intelligent behavior.
- AI studies how we can make the computer do things that humans can still do better at the moment.

Definition (Components of AI).
- Ability to learn
- Making conclusions
- Perception
- Language understanding
- Emotions

Definition (Narrow AI).
Use of a software to study or accomplish a specific problem solving or reasoning tasks.

Definition (Strong AI).
Software that performs full range of human cognitive abilities.
⇒ Doesn't exist yet!
→ Problems requiring strong AI are called **AI complete**.

Rational Agents

Definition (What is AI here?).
- Systems that think like humans.
  → Approximated by Neural networks.
- Systems that act like humans.
  → Turing Test (Can systems fool a human?)
  → Problem: Turing test is not reproducible, constructive, or amenable to mathematical analysis.
- Systems that think rationally.
  → Logics, Formalization of knowledge and inference.
  → Problem: Not all intelligent behavior is mediated by logical deliberation.
- Systems that act rationally.
  → The question how to make good action choices.
  ⇒ Rational behavior consists of always doing what is expected to maximize goal achievement given the available information.

Definition (Agent and rationality).
An **agent** is anything that perceives its environment via sensors and acts on it with actuators. Here we designing agents that exhibit **rational behavior**.
→ For given class of environments and tasks, we seek the agent with the best performance.
→ Computational limits make perfect rationality unachievable.
⇒ Design best program for given machine resources.

Definition (Modeling Agents).
- An **agent** is an entity that perceives and acts.
- A **percept** is the perceptual input of an agent at a specific instant.
- Any recognizable, coherent employment of the actuators of an agent is called an **action**.
- The **agent function** of an agent a maps from percept histories to actions: $f_a : \mathcal{P}^* \rightarrowtail \mathcal{A}$
- An agent function can be implemented by an **agent program** that runs on a physical **agent architecture**.

⇒ Problem: Agent function can become very big.

Definition (Rational Agent).
An **agent** is called **rational**, if it chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date.

Consequences of rationality

Definition (Performance measure).
A **performance measure** is a function that evaluates a sequence of environments.

Note. Rational don't need to be perfect. Agent only needs to maximize expected value.Percepts may not supply all infos.
⇒ Rationality is exploration, learning and autonomy.

Definition (Autonomous).
An agent is called **autonomous**, if it does not rely on the prior knowledge of the designer.
⇒ Autonomy avoids fixed behaviors that can become unsuccessful in a changing environment.

Definition (PEAS for describing task environment).
Specify the task environment in terms of
- **Performance measure**: how should he behave?
- **Environment**: where is our agent?
- **Actuators**: what can he do, move or change?
- **Sensors**: how can he perceive?

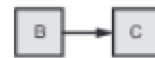
Classifying Environments

Definition (Environment properties).
- **Fully observable**, if sensors give access to complete state of environment at any time, else **partially observable**.
- **Deterministic**, if the next step is completely determined by current state and the agents action, else **stochastic**.
- **Episodic**, if experience is divided into atomic episodes with perceiving and then performing a single action. It is also important that the next action does not depend on previous ones. Otherwise we call it **sequential**.
- **Dynamic**, if the environment can change without an action of the agent, else **static**. If the environment does not change but the performance measure og the agent it is called **semidynamic**.
- **Discrete**, if the environment and agent states are countable, else **continuous**.
- **Single-agent**, if just one agent acts in the environment, else **multi-agent**.

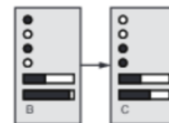
Representing the environment in agents

Note. Questions of an agent could be like
- What is the world like now?
- What action should I do?
- What do my actions do?

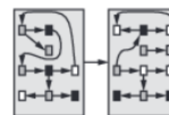
Definition (State representations).
We call a state representation
- **Atomic**, if it has no internal structure. (Black Box)



- **Factored**, if state is characterized by attributes, values.

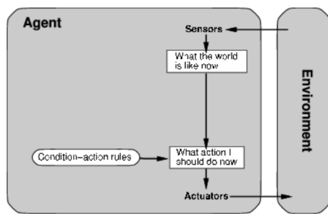


- **Structured**, if the state includes objects and relations.



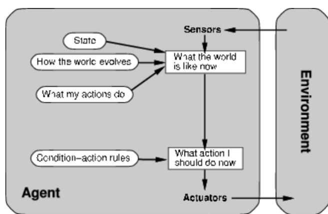
Types of Agents

Definition (Simple reflex agents)**.**
A **simple reflex agent** is an agent a that only bases its actions on the last percept.



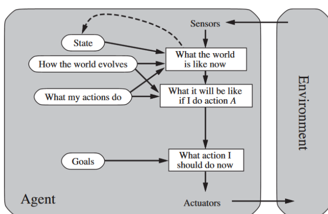
→ Problem: Can only react to the perceived state of the environment, not to changes.

Definition (Reflex agents with state)**.**
A **stateful reflex agent** is an agent whose agent function depends on a model of the world.



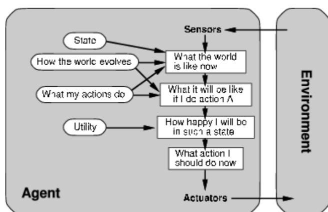
→ Problem: Having a model of the world does not always determine what to do.

Definition (Goal-based agent)**.**
A **goal-based agent** is a stateful reflex agent that deliberates actions based on goals and a world model.



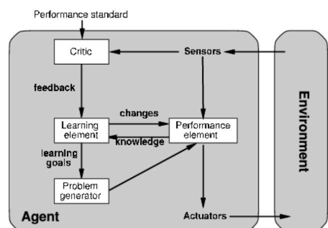
→ Observation: More flexible in the knowledge it can utilize.

Definition (Utility-based agent)**.**
A **utility-based agent** uses a world model with a utility function that influences its preferences among the states of that world. It chooses the action that leads to the best expected utility.



→ Allows rational decisions where goals are inadequate.

Definition (Learning Agent)**.**
A **learning agent** is an agent that augments the performance element, which determines actions from percept sequences with a learning element, a critic and a problem generator.



Problem solving and search

Problem Solving

Definition (Describing Problems)**.**
We want to describe problems in a standardized way, so we may find a **general algorithm**. We will use the following two concepts:

- **States**: A set of possible situations in our problem domain
- **Actions**: Get us from one state to another

Sequences of actions, which brings us to a situation, where the problem is solved, is a solution.

Definition (Offline problem solving)**.**
In **offline problem** solving an agent computing an action sequence based complete knowledge of the environment.
→ Remark: Offline problem solving only works in fully observable, deterministic, static, and episodic environments.
→ We restrict ourselves to offline problem solving.

Definition (Problem formulation)**.**
A **problem formulation** models the situation (states and action) at an appropriate level of abstraction. So it describes the initial state and also limits the objectives by specifying goal states.

Definition (Mathematical description of a Search Problem)**.**

- A **search problem** $P = (S, A, T, I, G)$ consists of a set S of **states**, a set A of **actions** and a **transition model** $T : A \times S \to P(S)$ that assigns to $a \in A,\ s \in S$ a set of **successor states**.
  Some states in S are **goal states** G and **initial states** I.
- A **solution** for a search problem consists of a sequence of actions $a_1, \ldots a_n$, such that action a_i can be done in s_{i-1}, $s_0 \in I$ and in the end we reach $s_n \in G$.
- Often we add a **cost function** e, that associates a step cost $e(a)$ to an action. The cost of a solution is the sum of the step costs.
- The predicate that tests for goal states is called a **goal test**.

→ The definition for problem formulation we made is called **blackbox description**.
⇒ Gives the algorithm no information about the problem.

Definition (Whitebox description)**.**
A **declarative description** (also called **whitebox description**)describes the problem itself.

→ Problem description language
→ Declarative descriptions are strictly more powerful than blackbox descriptions: they induce blackbox descriptions, but also allow to analyse/ simplify the problem.

Problem types

Types:

- **Single-state problem**: observable, deterministic, static, discrete.
- **Multiple-state problem**: initial state not or just partially observable, deterministic, static, discrete.
- **Contingency problem**: non-deterministic, unknown state space (like a baby).

→ Often we need to select a state space, because the real world is absurdly complex.

Definition (Single-state problem formulation)**.**

- Initial state
- Successor function S
- Goal test
- Optional: Path cost

Search

Definition (Tree Search Algorithms).
We observe, that the state space of a search problem $P = (S, A, T, I, G)$ forms a graph (S, T_A). The tree search algorithm consists of the simulated exploration of state space (S,A) in a search tree formed by successively generating successors of already-explored states.
We define the path cost of a node n in a search tree T to be the sum of the step costs on the path from n to the root of T. The cost of a solution is defined analogously.

→ It is helpful to implement a fridge.
→ The **fringe** is a list nodes not yet considered. It is ordered by the search strategy.

Definition (Search strategies).
A **search strategy** is a function that picks a node from the fringe of a search tree.
→ Equivalently: orders the fringe and picks the first

→ Important properties of strategies: completeness, time complexity, space complexity and optimality.

Time and space complexity measured in terms of:

b	maximum branching factor of the search tree
d	minimal depth of a solution in the search tree
m	maximum depth of the search tree (may be ∞)

Uniformed search strategies

Definition (Uniformed search).
Use only the information **available** in the problem definition.

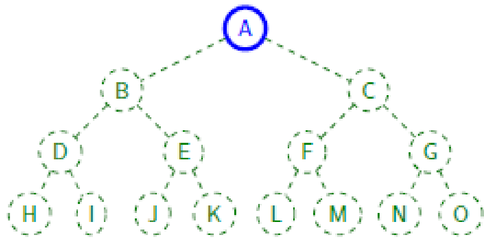


Abb. 1: Example tree for the uniform searches

Definition (Breadth-first search).
Expand shallowest unexpanded node. So, the fringe is a **FIFO** queue, i.e. successors go in at the end of the queue.
Properties:
- Its complete, if b is finite.
- Time and space: $\mathcal{O}(b^d)$
- Optimal, just if cost = 1 per step.
- Disadvantage: Space!
- One can always get an optimal solution, when all solutions are created an then the optimal one is picked. This works only, if m is finite.

→ Visiting order: A-B-C-D-E-F-G-H-I-J-K-L-M-N-O

Definition (Uniform-cost search).
Expand least-cost unexpanded node. So, the fringe is ordered by **increasing path cost**. It's equivalent to breadth-first search if all step costs are equal.
Properties:
- Its complete, if step costs greater than zero.
- Time and space: number of nodes with path-cost less than that of optimal solution.
- Optimal: Yes!

→ Visiting order: A-B-C-D-E-F-G-H-I-J-K-L-M-N-O

Definition (Depth-first search).
Expand deepest unexpanded node. So, the fringe is organized as a **LIFO** queue (a stack), i.e. successors go in at front.
Note: It can perform infinite cyclic excursions.
⇒ Finite, non-cyclic search space (or repeated-state checking)
Properties:
- Its complete, if state space is finite and does not contain infinite paths or loops.
- Time: $\mathcal{O}(b^m)$
- Space: $\mathcal{O}(b \cdot m)$
- Optimal: No
- Disadvantage: Time terrible if m much larger than d.
- Advantage: Time may be much less than breadth-first search if solutions are dense.

→ Visiting order: A-B-D-H-I-E-J-K-C-F-L-M-G-N-O

Definition (Iterative deepening search).
Iterative deepening search is **depth-limited search** with ever **increasing limits**, where depth-limited search is depth-first search with a depth limit.
Properties:
- It's complete!
- Time: $\mathcal{O}(b^{d+1})$
- Space: $\mathcal{O}(b \cdot d)$
- Optimal: Yes, if step cost are 1.

Tree Search vs. Graph Search
→ Nobody uses Tree search in practice, because states duplicated in nodes are a huge problem for efficiency and it is quite memory-intensive.
→ A **graph search algorithm** is a variant of a tree search algorithm that prunes nodes whose state has already been considered, essentially using a DAG data structure.

Informed Search Strategies

Definition (Best-first search).
Basic Idea: Use an evaluation function for each node. Expand most desirable unexpanded node. The fringe is a queue sorted in **decreasing order of desirability**.
Special cases are Greedy search and A^*-search.

→ This is like UCS, but with evaluation function related to problem at hand replacing the path cost function.

Definition (Greedy search).
Greedy search is a heuristic, which expands the node that appears to be closest to the goal. In greedy search we replace the objective cost to construct the current solution with a heuristic or subjective measure from which we think it gives a good idea how far we are from a solution.
So instead of measuring the cost to build the current partial solution, we estimate how far we are from the desired goal.
Properties:
- No, it can get stuck in loops.
- Time and Space: $\mathcal{O}(b^m)$
- Optimal: No, it's a heuristic!

→ Worst-case time same as depth-first search.
→ Worst-case space same as breadth-first.

Two properties of Heuristic functions:
1. h is admissible if it is a lower bound on goal distance.
2. h is consistent if, when applying an action a, the heuristic value cannot decrease by more than the cost of a.
   ⇒ Consistency is a sufficient condition for admissibility.
3. If h_2 dominates h_1, then h_2 is better for search than h_1.

→ For good heuristics,try the **relaxed version** of the problem.
→ h = 0: no overhead at all, completely un-informative

→ $h = h^*$: perfectly accurate, overhead, olving the problem in the first place.

Definition (A^*-search)**.**
Basic idea: Avoid expanding paths that are already expensive. The simplest way to **combine heuristic and path cost** is to simply **add** them. Thus the evaluation function f(n) is the estimated total cost of path through n to goal.
→ A^*-search with admissible heuristic is optimal.
→ In A^*, node values always increase monotonically.
Properties:
- Complete: Yes, unless there are infinitely many nodes n with $f(n) \leq f(0)$
- Time and Space: Exponential in (relative error in h × length of solution)
- Optimal: Yes.

→ The run-time depends on how good we approximated the real cost h^* with h.

Local Search

Definition (Systematic Search)**.**
We call a search algorithm **systematic**, if it considers all states at some point. Systematic search procedures are complete. There is no limit of the number of search nodes that are kept in memory at any time systematic search procedures. For example, all tree search algorithms are systematic.

Definition (Local Search Problems)**.**
Basic Idea: Sometimes the path to the solution is irrelevant.
A **local search algorithm** is a search algorithm that operates on a single state, the current state. Advantage is the constant space.
Popular example (n-queens problem): Put n queens on n×n board such that no two queens in the same row, columns, or diagonal.

Definition (Hill-climbing)**.**
Start anywhere and go in the direction of the steepest ascent. Doing this more than once leads to a good heuristic.
Works, if solutions are dense and local maxima can be escaped.
⇒ Depth-first search with heuristic.

Definition (Simulated annealing)**.**
Escape local maxima by allowing some bad moves, but gradually decrease their size and frequency. If it decreases slowly enough, we will always reach the best state.

Definition (Local beam search)**.**
Keep **k states instead of 1** and choose top k of all their successors. Choose k successors randomly, biased towards good ones, so maybe not all k states end up on the same local hill.

Definition (Genetic algorithms)**.**
Basic idea: Use local beam search, **randomly modify population** an generate successors from pairs of states. Optimize fitness function.
Problem: Genetic Algorithms require states encoded as strings.

Adversarial search for game playing

→ Adversarial search = Game playing against an opponent.
Restrictions for game playing:
- Game states discrete, number of game states finite.
- Finite number of possible moves.
- The game state is fully observable.
- The outcome of each move is deterministic.
- Just two players (call them Max and Min).
- Turn-taking: Alternatingly (Max begins).
- There are no infinite runs of the game.
- Terminal game states have a utility u. Max tries to maximize u, Min tries to minimize u.

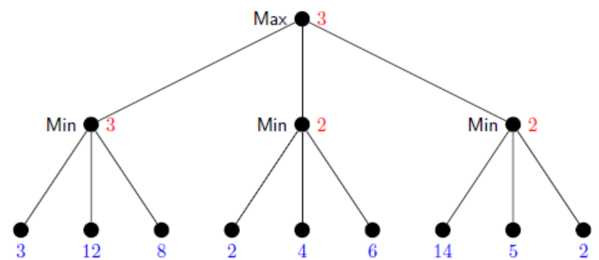
Definition (Strategy and optimality)**.**
Let Θ be a game state space. A **strategy** for X is a function, so that a is applicable to s.A strategy is optimal if it yields the best possible utility for X assuming perfect opponent play.
→ In almost all games, computing a strategy is infeasible.

Three possibilities to describe a game state space:
1. Explicit: Sometimes used by humans, but nor good for computers.
2. Blackbox/API: Used by both. Assumed description in
   - Method of choice for all those game players out there in the market.
   - Programs designed for, and specialized to, a particular game..
   - Human knowledge is key!
3. Declarative: General Game playing, based on logic. Active area of research in AI.

Minimax Search

→ We want to compute an optimal strategy for Max.

Definition (Minimax)**.**
We alternating between max and min, using the **following rules**:
1. Depth-first search in game tree, with Max in the root.
2. Apply utility function to terminal positions.
3. Bottom-up for each inner node n in the tree, compute the utility u(n) of n as follows:
   - If it's Max's turn: Set u(n) to the maximum of the utilities of n's successor nodes.
   - If it's Min's turn: Set u(n) to the minimum of the utilities of n's successor nodes.
4. Selecting a move for Max at the root: Choose one move that leads to a successor node with maximal utility.



Minimax advantages:
- Minimax is the simplest possible (reasonable) game search algorithm.
- Returns an optimal action, assuming perfect opponent play.
- There's no need to re-run Minimax for every game state.

Minimax disadvantages:
- It's completely infeasible in practice.
- When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut-off states.

Evaluation functions

Definition (Evaluation function)**.**
An **evaluation function** f maps game states to numbers:
- f(s) is an estimate of the actual value of s (as would be computed by unlimited-depth Minimax for s).
- If cut-off state is terminal: Just use u instead of f.

Definition (Linear evaluation function).
A common approach is to use a weighted linear function for f:

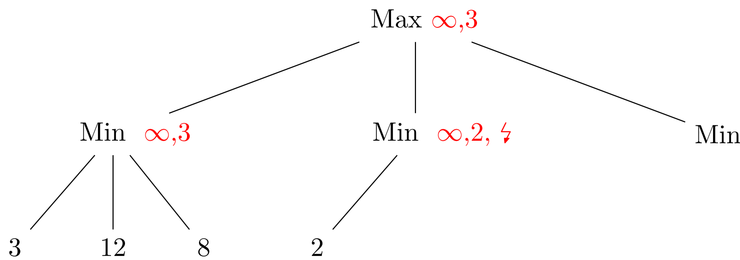
$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$$

where the wi are the weights, and the fi are the features.
Weights wi can be learned automatically, but the features have to be designed by humans.

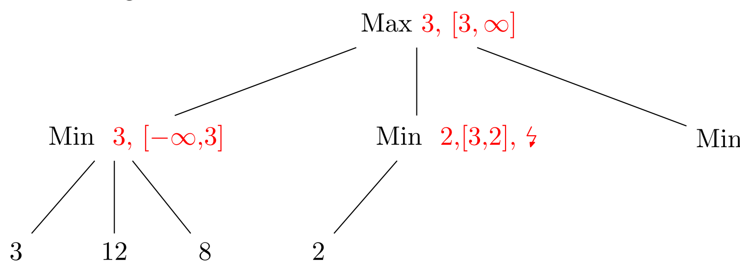
Definition (The horizon problem).
Critical aspects of the game can be cut-off by the horizon.
The solution how deeply to search is to use iterative deepening. There we search with a depth limit and try to search more deeply in good positions.

Alpha-Beta Search

Definition (Alpha Pruning).
Basic idea: Save some work by doing minimax with stopping if we know that some part is worse than our best one!



Definition (Alpha-Beta Pruning).
Basic idea: Use **Alpha Pruning** and give an interval. For Max, its an interval from the lowest he could reach to ∞. For Min, the interval goes from Max above to the minimum he could reach.



→ Choosing the best moves first yields most pruning in alpha-beta search.

Monte-Carlo Tree Search

Definition (Monte Carlo).
For **Monte-Carlo sampling** we evaluate actions through sampling. For **Monte-Carlo tree search** we maintain a search tree T. Pro in contrast to alpha-beta search is runtime and memory, contra is the guardiance needed.
⇒ Samples game branches, and averages the findings.

Constraint satisfaction problems

Definition (CSP).
A **constraint satisfaction problem** (CSP) is a search problem, where the states are given by a finite set $V = \{X_1, \ldots, X_n\}$ of variables and domains $\{D_v \mid v \in V\}$ and the goal test is a set of constraints.
Example (Map Coloring): Variables are states, domains are colors and constraints are that neighbors must be colored differently.

→ CSP is NP-complete.

The Waltz Algorithm

Motivation: Interpret line drawings of solid polyhedra.
Problem: Are intersections convex or concave?
Main idea: Neighbored intersections impose constraints in each other. So use CSP to find a unique set of labelings.

Formal definition of CSP

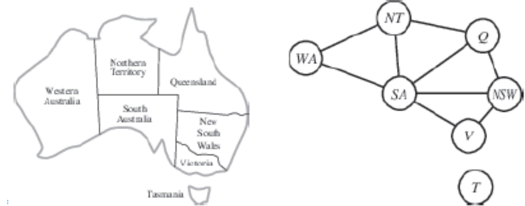
Various types of CSPs:

- n discrete variables:
  - finite domains, eg. Boolean CSPs
  - infinite domains, eg. job scheduling, here a linear constraint language is wanted
- Continuous variables:
  - Linear constraints solvable in poly time, but no optimal algorithms for nonlinear constraint systems.
  - Eg. start and end times for Hubble Telescope observations

Types of constraints:

- **Unary constraints** involve a single variable.
- **Binary constraints** involve pairs of variables.
- **Higher-order constraints** involve 3 or more variables.
- **Preferences** often represented by a cost for each variable assigned.

Definition (Constraint Graph).
We need a binary CSP for a constraint graph. A **binary CSP** is a CSP where each constraint relates at most two variables.
A constraint network forms a graph whose nodes are variables, and whose edges represent the constraints.
Example (Map coloring):



Definition (Constraint network).
A **constraint network** is a triple (V, D, C), where

- V is a finite set of variables.
- D the set of their domains.
- C is a binary constraint.

Unary constraints can be expressed by restricting the domain and higher-order constraints can be transformed into equisatisfiable binary constraints using auxiliary variables.
⇒ Any CSP can be represented by a constraint network.

Short definitions:

- A **partial assignment** a maps some variables to values, a **total assignment** does so for all variables.
- a is **consistent** if it complies with all constraints.
- A consistent total assignment is a solution.

CSP as search

Definition (Standard search formulation).
States are defined by the values assigned so far. Every solution appears at depth n with n variables! This works for all CSPs.
BUT: There would be d^n leaves.

Definition (Backtracking search).
Depth-first search for CSPs with single-variable assignments is called **backtracking search.**
Variable assignments are commutative, so you only need to consider assignments to a single variable at each node.
⇒ b = d, so there are d^n leaves!

Improving backtracking efficiency

Definition (Minimum Remaining Values Heuristic).
The **Minimum remaining values (MRV)** heuristic for backtracking search always chooses the variable with the fewest legal values, i.e. such that $\#(\{d \in D_v \mid a \cup \{v \rightarrowtail d\}$ is consistent$\})$ is minimal.
→ So we choose the most restricted variable first to reduce the branching factor.
→ Extreme Case: $\#(\{d \in D_v \mid a \cup \{v \rightarrowtail d\}$ is consistent$\}) = 1$, then the value to take is forced.

Definition (Degree heuristic).
The **degree heuristic** in backtracking search always chooses the variable with the most constraints on remaining variables. By choosing a most constraining variable first, we detect inconsistencies earlier on and thus reduce the size of our search tree.
→ Commonly used strategy combination: From the set of most constrained variables, pick a most constraining variable.

Definition (Last Constraining Value Heuristic).
Given a variable, the **least constraining value** heuristic chooses the least constraining value: the one that rules out the fewest values in the remaining variables.
By choosing the least constraining value first, we increase the chances to not rule out the solutions below the current node.

Constraint Propagation

Inference

Definition (Equivalent constraint network).
Two constraint networks are **equivalent**, if the have the same set of variables and the same solutions.

Definition (Tightness).
Let γ, γ' be two constraint networks sharing the same set of variables. We say that γ' is **tighter** that γ, if γ' has the same constraints as γ plus some.

Definition (Inference).
Inference consists in deducing additional constraints (unary or binary), that follow from the already known constraints, i.e. that are satisfied in all solutions.
So, if two networks are equivalent and tight, we have inference. Then the tighter network has fewer consistent partial assignments, so it is better for the underlying problem.

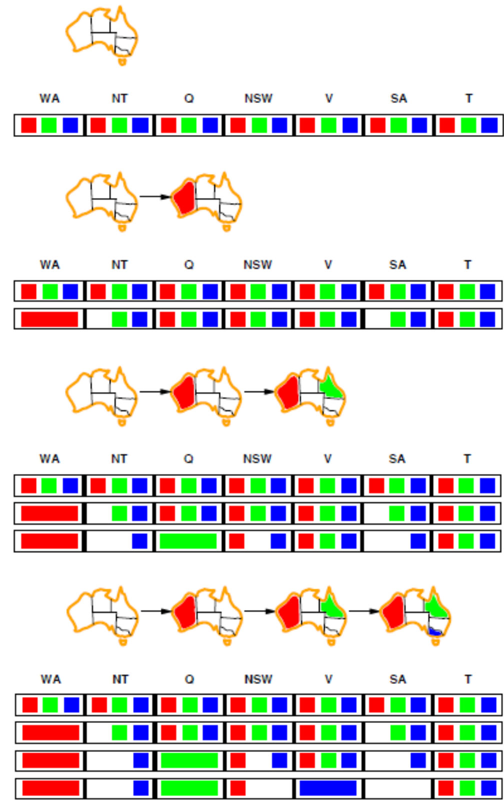
Use of Inference:

- Inference as **pre-process**: Just once before search starts. Then it has little runtime overhead and pruning power.
- Inference **during search**: At every recursive call of backtracking. Then it has strong pruning power, but may have larger runtime overhead.

Search vs. Inference:
The more complex the inference, the smaller the number of search nodes, but the larger the runtime needed at each node.

Forward Checking

→ One method for inference.
→ Cheap and useful.
→ Forward checking removes values conflicting with an assignment already made.

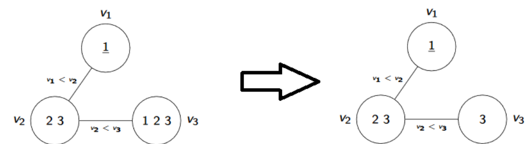


→ Forward checking makes inferences only from assigned to unassigned variables.

Arc consistency

→ Better method for inference!

Definition (Arc consistency).
A variable $u \in V$ is **arc consistent** relative to another variable v, if either no constraint between them exist or we can find a value for both that's not a contradiction.
The network is arc consistent if every variable is arc consistent.
→ Enforcing arc consistency ⇒ removing variable domain values until network is arc consistent.
→ Arc consistency uses forward checking!
→ Run time: $\mathcal{O}(mk^3)$ with m constraints and maximal domain size k.



→ Algorithm AC-3: Add all arcs to a queue, take (X_i, X_j) and make them arc consistent. If you changed X_i, add all adjacent arcs (X_k, X_i) with $k \neq j$ to the queue.
The algorithm ends, if the queue is empty or the domain of one node is empty.

Decomposition

Aim: Decompose the graph into smaller parts, which may be easier to solve.

Theorem (Disconnected constraint graph).
If we combine all partial solutions of the disconnected networks, we also have a solution for the whole constraint network.

Theorem (Acyclic constraint graph).
For a acyclic constraint network with n variables and maximal domain size k we can find a solution or prove that the network is inconsistent in time $\mathcal{O}(nk^2)$.

Algorithm for Acyclic constraint graphs:

1. Obtain a directed tree from the networks constraint graph, picking an arbitrary variable v as root and directing arcs outwards.
2. Order the variables, such that each vertex is ordered before its children (denote by $v_1, \dots, v_n$).
3. For $i = n, \dots, 2$, enforcing consistency of parents relative to v_n and if the parents have no domain left, return inconsistent.
   ⇒ Now, every variable is arc consistent relative to its children.
4. Run Backtracking With Inference with forward checking, using the variable order.

Cutset Conditioning

Definition (Cutset)**.**
Let γ be a constraint network and $V_0 \subseteq V$. V_0 is a cutset for γ, if the subgraph of γ's constraint graph induced by $V \setminus V_0$ is acyclic. V_0 is optimal, if its size is minimal among all cutsets.
→ Which subset has to be removed, such that the graph is acyclic?
⇒ Finding optimal cutsets is NP-hard.

→ Cutset decomposition backtracks only on a cutset, and solves a sub-problem with acyclic constraint graph at each search leaf.

Propositional Reasoning - Principles

Propositional logic

Definition (Syntax)**.**
Propositional Logic (PL^0) is made up from propositional variables and connectives. We define a set $wff_0(V_0)$ of **well-formed propositional** formulas as

- negations $\neg A$
- conjunctions $A \wedge B$
- disjunctions $A \vee B$
- implications $A \Rightarrow B = \neg A \vee B$
- equivalences $A \Leftrightarrow B$

where $A, B \in wff_0(V_0)$ themselves.
A propositional formula without connectives is called **atomic** and **complex** otherwise.

Definition (Semantics)**.**
A **model** $M = (D_0, I)$ for PL^0 consists of the Universe $D_0 = \{\top, \bot\}$ and the Interpretation I, that assigns values to essential connectives. A **variable assignment** $\varphi : V_0 \rightarrowtail D_0$ assigns values to propositional variables.
The **value function** I_φ assigns values to formulae.
How to work with this:

- Recursive defined base case: $I_\varphi(P) = \varphi(P)$
- $I_\varphi(\neg A) = I(\neg)(I_\varphi(A))$
- $I_\varphi(A \wedge B) = I(\wedge)(I_\varphi(A), I_\varphi(B))$

Notation:
→ A is **true under** φ, if φ satisfies A.
→ A is **false under** φ, if φ falsifies A.
→ A is **satisfiable** in M, if $\exists \varphi : I_\varphi(A) = \top$.
→ A is **valid** in M, if it is true for all φ.
→ A is **falsifiable**, if $\exists \varphi : I_\varphi(A) = \bot$.
→ A is **unsatisfiable** in M, if $\nexists \varphi : I_\varphi(A) = \top$.
→ We say that A entails B (A $\models$ B), if all assignments that make A true also make B true.

Formal Systems

Definition (Logical System)**.**
A **logical system** is a triple $S = (L, K, \models)$, where L is a formal language, K is a set and $\models \subseteq K \times L$. Members of L are called formulae of S, members of K models for S, and $\models$ the satisfaction relation.

Notation:
→ A is **satisfied** by M, if $M \models A$.
→ A is **falsified** by M, if $M \not\models A$.
→ A is **satisfiable** in K, if $\exists M \in K : M \models A$.
→ A is **valid** in K, if $M \models A$ for all $M \in K$.
→ A is **falsifiable** in K, if $\exists M \in K : M \not\models A$.
→ A is **unsatisfiable** in K, if $\nexists M \in K : M \models A$.

Definition (Derivation and Inference Rules)**.**
Let S be a logical system, then we call a relation $\vdash$ a **derivation relation** (syntaktisch ableitbar) for S, if it

- is proof-reflexive, i.e. $H \vdash A$, if $A \in H$
- is proof-transitive, i.e. $H \vdash A$ and $H' \cup \{A\} \vdash B$, then $H \cup H' \vdash B$
- monotonic, i.e. $H \vdash A$ and $H \subseteq H'$ imply $H' \vdash A$

We call $(L, K, \models, \vdash)$ a **formal system**.
An inference rule over L is written

$$\frac{A_1, \dots, A_n}{C} N$$

where A_i are called assumptions, C is called conclusion and N is a name.
→ An inference rule without assumptions is called an axiom.
→ A set $\mathcal{C}$ of inference rules over L is called calculus for S.

Definition ($\mathcal{C}$-derivation)**.**
Let S be a logical system and $\mathcal{C}$ a calculus for S, then a **$\mathcal{C}$-derivation** of a formula C from a set H of hypotheses (H $\vdash_\mathcal{C}$ C) is a sequence $A_1, \dots, A_m$ of l-formulae, such that $A_m = C$ and for all i, either $A_i \in H$ or there is an inference rule.
⇒ Derivation can be seen as a tree!

Definition (Proof)**.**
A derivation $\emptyset \vdash_\mathcal{C} A$ is called a **proof** of A. If one exists, the A is called a **$\mathcal{C}$-theorem**.

Definition (Admissible)**.**
An inference rule I is called **admissible** in $\mathcal{C}$, if the extension of $\mathcal{C}$ by I does not yield new theorems.

Definition (Propositional Logic with Hilbert-Calculus)**.**
Rules:
$K = P \Rightarrow Q \Rightarrow P$, $S = (P \Rightarrow Q \Rightarrow R) \Rightarrow (P \rightarrow Q) \Rightarrow P \Rightarrow R$
and $\frac{A \Rightarrow B \quad A}{B} MP$, $\frac{A}{[B/X](A)} Subst$

Definition (Soundness and Completeness)**.**
Let S be a logical system, then we call a calculus $\mathcal{C}$ for S

- **sound** or **correct**, if $H \models A$, whenever $H \vdash_\mathcal{C} A$
- **complete**, if $H \vdash_\mathcal{C} A$, whenever $H \models A$.

Propositional Natural Deduction Calculus

Definition (Natural Deduction ND^0)**.**
Natural Deduction has the following rules:

Introduction	Elimination	Axiom
$\frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \wedge I$	$\frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{A}} \wedge E_l$ $\frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{B}} \wedge E_r$	$\frac{}{\mathbf{A} \vee \neg \mathbf{A}}$ TND
$\frac{\genfrac{}{}{0pt}{}{[\mathbf{A}]^1}{\mathbf{B}}}{\mathbf{A} \Rightarrow \mathbf{B}} \Rightarrow I^1$	$\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}} \Rightarrow E$	

$$\frac{A}{A \lor B} \lor I_l \quad \frac{B}{A \lor B} \lor I_r \quad \frac{A \lor B \quad \begin{matrix}[A]^1 \\ \vdots \\ C\end{matrix} \quad \begin{matrix}[B]^1 \\ \vdots \\ C\end{matrix}}{C} \lor E^1$$

$$\frac{\begin{matrix}[A]^1 \\ \vdots \\ F\end{matrix}}{\neg A} \neg I^1 \quad \frac{\neg\neg A}{A} \neg E$$

$$\frac{\neg A \quad A}{F} FI \quad \frac{F}{A} FE$$

Definition (Sequent Calculus formulation).
A **judgment** is a meta-statement about the provability of propositions.
A **sequent** is a judgment of the form $H \vdash A$ about the provability of the formula A from the set H of hypotheses.
Rules for **propositional sequent-style natural deduction calculus** $ND^0_\vdash$:

$$\frac{}{\Gamma, A \vdash A} Ax \quad \frac{\Gamma \vdash B}{\Gamma, A \vdash B} \text{weaken} \quad \frac{}{\Gamma \vdash A \lor \neg A} TND$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B} \land I \quad \frac{\Gamma \vdash A \land B}{\Gamma \vdash A} \land E_l \quad \frac{\Gamma \vdash A \land B}{\Gamma \vdash B} \land E_r$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \lor B} \lor I_l \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \lor B} \lor I_r \quad \frac{\Gamma \vdash A \lor B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \lor E$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow I \quad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow E$$

$$\frac{\Gamma, A \vdash F}{\Gamma \vdash \neg A} \neg I \quad \frac{\Gamma \vdash \neg\neg A}{A} \neg E$$

$$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash F} FI \quad \frac{\Gamma \vdash F}{\Gamma \vdash A} FE$$

Linearized notation for ND-proofs:

#	hyp	formula	ND just
1	1	A	Ax
2	2	B	Ax
3	1,2	A	weaken 1,2
4	1	$B \Rightarrow A$	$\Rightarrow$ I(3)
5		$A \Rightarrow B \Rightarrow A$	$\Rightarrow$ I(4)

Machine-oriented calculi for propositional logic

Theorem (Unsatisfiability Theorem).
$H \models A$, if $H \cup \{\neg A\}$ is unsatisfiable.

Definition (Normal Forms).
A formula is in **conjunctive normal form** (CNF) if it consists of a conjunction of disjunctions of literals: $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} l_i, j$.
formula is in **disjunctive normal form** (DNF) if it consists of a disjunction of conjunctions of literals: $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_i, j$.
→ Every formula can be written in CNF/DNF. But finding the DNF is NP-complete.

Analytical Tableaux

→ Idea: Show about negation, that a formula is true or false.
→ Formula is analyzed in a tree to determine satisfiability.
Rules for $\mathcal{T}_0$ (derived rules of inference):

$$\frac{A \land B^T}{\begin{matrix}A^T \\ B^T\end{matrix}} \mathcal{T}_0\land \quad \frac{A \land B^F}{A^F \mid B^F} \mathcal{T}_0\lor \quad \frac{\neg A^T}{A^F} \mathcal{T}_0{\neg}^T \quad \frac{\neg A^F}{A^T} \mathcal{T}_0{\neg}^F \quad \frac{\begin{matrix}A^\alpha \\ A^\beta\end{matrix} \quad \alpha \neq \beta}{\bot} \mathcal{T}_0\text{cut}$$

$$\frac{A \Rightarrow B^T}{A^F \mid B^T} \quad \frac{A \Rightarrow B^F}{\begin{matrix}A^T \\ B^F\end{matrix}} \quad \frac{\begin{matrix}A^T \\ A \Rightarrow B^T\end{matrix}}{B^T}$$

$$\frac{A \lor B^T}{A^T \mid B^T} \quad \frac{A \lor B^F}{\begin{matrix}A^F \\ B^F\end{matrix}} \quad \frac{A \Leftrightarrow B^T}{\begin{matrix}A^T \\ B^T\end{matrix} \mid \begin{matrix}A^F \\ B^F\end{matrix}} \quad \frac{A \Leftrightarrow B^F}{\begin{matrix}A^T \\ B^F\end{matrix} \mid \begin{matrix}A^F \\ B^T\end{matrix}}$$

Definition (Derived inference rule).
Let $\mathcal{C}$ be a calculus, a rule of inference $\frac{A_1, \ldots, A_N}{C}$ is called a derived inference rule in $\mathcal{C}$, if there is a $\mathcal{C}$-proof of $A_1, \ldots, A_n \vdash C$.

Definition (Saturated and closed branch).
Call a tableau **saturated**, if no rule applies. Call a branch closed, if it ends in $\bot$, else open.

Definition ($\mathcal{T}_0$-theorem).
A is a $\mathcal{T}_0$**-theorem**, if there is a closed tableau with rooted A^F.

Example:

0	$((P \land Q) \Rightarrow (P \lor Q))^F$		
1	$(P \land Q)^T$		from (0)
2	$(P \lor Q)^F$		from (0)
3	P^T		from (1)
4	Q^T		from (1)
5	P^F		from (2)
6	$\bot$		

Definition (Derivability).
$\Phi \subseteq wff_0(V_0)$ **derives** A in $\mathcal{T}_0$ ($\Phi \vdash_{\mathcal{T}_0} A$), if there is a closed tableau starting with A^F and Φ^T

Definition (Soundness for Tableau).
Idea: A test calculus is sound, if it preserves satisfiability and the goal formulae are unsatisfiable.
A tableau T is satisfiable, if set of formulae in P is satisfiable.
Important: Tableau rules transform satisfiable tableaux into satisfiable ones.
Soundness: A set Φ of propositional formulae is valid, if there is a closed tableau T for Φ^F.

Theorem (Termination for Tableaux).
The tableau procedure terminates, i.e. after a finite set of rule applications, it reaches a tableau, so that applying the tableau rules will only add labeled formulae A^α that are already present on the branch.

Resolution

Definition (Resolution).
The **resolution calculus** operates a clause sets via a single inference rule:

$$\frac{P^T \lor A \quad P^F \lor B}{A \lor B}$$

Let S be a clause set, then we call a R derivation $D : S \vdash_R \Box$ **resolution refutabtion**.
→ $\Box$ is called empty clause.
We call a resolution refutation of $CNF^0(A^F)$ a **resolution proof** for $A \in wff_0(V_0)$.

Transformation into Clause Normal Form:

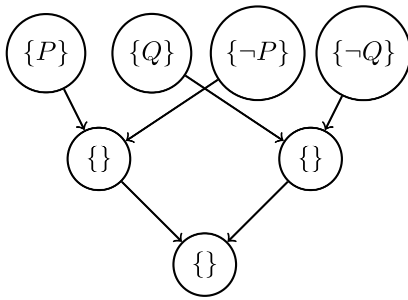
$$\frac{C \lor (A \lor B)^T}{C \lor A^T \lor B^T} \quad \frac{C \lor (A \lor B)^F}{C \lor A^F; C \lor B^F} \quad \frac{C \lor \neg A^T}{C \lor A^F} \quad \frac{C \lor \neg A^F}{C \lor A^T}$$

$$\frac{C \lor (A \Rightarrow B)^T}{C \lor A^F \lor B^T} \quad \frac{C \lor (A \Rightarrow B)^F}{C \lor A^T; C \lor B^F} \quad \frac{C \lor A \land B^T}{C \lor A^T; C \lor B^T} \quad \frac{C \lor A \land B^F}{C \lor A^F \lor B^F}$$

Example:
CNF-Transformation on negative formula:

$$\neg((P \land Q) \Rightarrow (P \lor Q))$$
$$(P \land Q) \land \neg(P \lor Q)$$
$$(P \land Q) \land \neg P \land \neg Q$$
$$\Rightarrow \quad P \land Q \land \neg P \land \neg Q$$

Resolution Algorithm:



Propositional Reasoning - SAT solvers

Definition (SAT).
The **SAT Problem**: Given a propositional formula A (normally in CNF), decide whether or not A is satisfiable.
→ This problem is NP-complete.
⇒ Deduction can be performed using SAT solvers.

SAT and CSP:
SAT can be viewed as a CSP problem in which all variable domains are Boolean, and the constraints have unbounded arity.
Encoding CSP as SAT:
Given a constraint network γ, we can construct a CNF formula $A(\gamma)$ that is satisfiable, if γ is solvable.

Davis-Putnam-(Logemann-Loveland) Proc.

Properties:
- Unsatisfiable case:
  In this case, we know that Δ is unsatisfiable: Unit propagation is sound, in the sense that it does not reduce the set of solutions.
- Satisfiable case:
  Any extension of I to a complete interpretation satisfies Δ.

⇒ DPLL is equivalent to Backtracking with Inference and so it is equivalent to unit propagation.
→ Worst case running time for SAT: 2^n
⇒ Unit propagation is sound.
Example: $\Delta = P \vee Q \vee \neg R,\ \neg P \vee \neg Q,\ R,\ P \vee \neg Q$
1. UP-Rule: $R \to \top$
   $\Rightarrow P \vee Q,\ \neg P \vee \neg Q,\ P \vee \neg Q$
2. Splitting Rule:

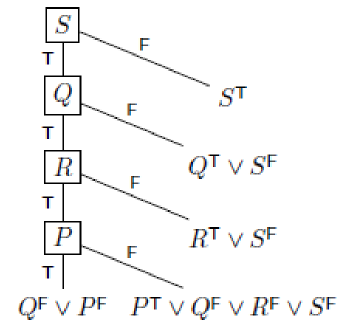
	2a	2b
	$P \to \bot$	$P \to \top$
	$\Rightarrow Q,\ \neg Q$	$\Rightarrow \neg Q$
	3a	3b
3.	UP-Rule: $Q \to \top$	UP-Rule: $Q \to \bot$
	$\Box$ ↯	✓

Definition (Unit resolution).
Unit resolution (UR) is the calculus consisting of the following inference rule:

$$\frac{C \vee \neg P \quad P}{C}$$

⇒ UR is sound.
⇒ UR is not refutation complete (not every unsatisfiable formula can be derived).

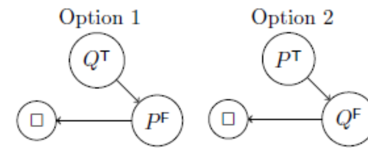
Definition (DPLL vs. Resolution).
We define the **number of decisions** of a DPLL run as the total number of times a truth value was set by either unit propagation or the splitting rule.
⇒ If DPLL returns unsatisfiable on Δ, then $\Delta \vdash_R \Box$ with a resolution derivation whose length is at most the number of decisions.
→ DPLL = tree resolution.
⇒ DPLL makes the same mistakes over and over again.
⇒ There exists Δ whose shortest tree-resolution is exponentially longer than their shortest resolution proof.



UP conflict analysis

Definition (Different literals).
Let β be a branch in a DPLL derivation, and P a variable on β then we call
- P^α a **choice literal**, if its value is set by a splitting rule.
- P^α a **implied literal**, if its value is set by UP-rule.

Definition (Implication Graph).
The **implication graph** G_β^{impl} is the directed graph whose vertices are labeled with the choice and implied literals along β , as well as a separate **conflict vertex** $\Box_C$ for every clause C that became empty on β.
→ The implication graph is not uniquely determined by the choice literals, it depends on ordering decision.
Example: $\Delta = \neg P \vee \neg Q,\ Q,\ P$



Definition (Conflict Graph).
A sub-graph C of G_β^{impl} is a **conflict graph** if:
- C contains exactly one conflict vertex $\Box_C$.
- If l' is a vertex in C, then all parents of l', are vertices in C as well.
- All vertices in C have a path to $\Box_C$.

→ Conflict graph ⇔ Starting at a conflict vertex, backchain through the implication graph until reaching choice literals.

Clause Learning

Observation: Conflict graphs encode logical entailments.

Lemma (Clause Learning).
Let Δ be a set of clauses and C the conflict graph at some time point during a run of DPLL. Let L be the choice literals in C. Then $\Delta \models \bigvee_{l \in L} \neg l$. ⇒ The negation of the choice literals in a conflict graph is a valid clause.

After we learned a new clause:
1. We add Clause $C = \bigvee_{l \in L} \neg l$ into δ.
2. We retract the last choice.
3. We set the opposite choice $\neg l'$ as an implied literal.
4. We run UP and analyze conflicts.

Observation: Given earlier choices $l_1, \ldots, l_k$, after we learned the new clause $C = \neg l_1 \vee \cdots \vee \neg l_k \vee \neg l'$, $\neg l'$ is now set by UP.
Clause learning vs. resolution
1. We add each learned clause to δ.
2. Clause learning renders DPLL equiv. to full resolution.

Note: Selecting different variables/values to split on can provably not bring DPLL up to the power of DPLL+Clause Learning.

Which clause to learn?
→ While we only select choice literals, much more can be done.
→ For any cut through the conflict graph, with choice literals on the "left-hand" side of the cut and the conflict literals on the right-hand side, the literals on the left border of the cut yield a learnable clause.
⇒ But we must care not to learn too many clauses.

→ DPLL with clause learning is called CDCL.

First order predicate logic

→ Predicate Logic (PL^1) extends propositional logic with the ability to explicitly speak about objects and their properties.

First-order Logic

In PL^1, we can talk about:

- individual thinks and denote them by variables or constants
- properties of individuals
- relations of individuals
- functions on individuals
- state the existence of an individual with a certain property, or the universality of a property

Definition (Syntax)**.**
We are talking about two kinds of objects:

- **truth values**; sometimes annotated by type o
- **individuals**; sometimes annotated by type l

A **first-order signature** consists of

- connectives σ^o
- function constants σ^f_k
- predicate constants σ^p_k

→ We assume a set of **individual variables** V_l.
→ We call formulae without connectives or quantifiers atomic, else complex.

Definition (Free and Bound Variables)**.**
We call an occurrence of a variable X **bound** in a formula A, if it occurs in a sub-formula $\forall X.B$ of A. Otherwise, the variable occurrence is **free**. We will notate **BVar(A)(free(A))** for the set of bound(free) variables of A.
Definition of the set free(A):

$$\begin{aligned} free(X) &= \{X\} \\ free(f(A_1,\dots,A_n)) &= \bigcup_{1\leq i\leq n} free(A_i) \\ free(p(A_1,\dots,A_n)) &= \bigcup_{1\leq i\leq n} free(A_i) \\ free(\neg A) &= free(A) \\ free(A\wedge B) &= free(A)\cup free(B) \\ free(\forall X.A) &= free(A)\setminus\{X\} \end{aligned}$$

We call a formula A **closed or ground**, if free(A) = ∅. We call a closed proposition a **sentence**.
→ Note: Bound variables can be renamed!

Definition (Semantics)**.**
At first, we fix the **universe** D_o of truth variables, the we assume an arbitrary universe $D_l \neq \emptyset$ of individuals.
An **interpretation** I assigns values to constants. A **variable assignment** $\varphi : V_l \rightarrowtail D_l$ maps variables into the universe.
Then, a first-order **model** M = (D_l, I) consists of a universe D_l and an interpretation I.
Given a model, the **value function** I_φ is recursively defined:

- Terms: $I_\varphi : wff_l(\sigma_l) \rightarrowtail D_l$

$$\begin{aligned} I_\varphi(X) &= \varphi(X) \\ I_\varphi(f(A_1,\dots,A_k)) &= I(f)(I_\varphi(A_1),\dots,I_\varphi(A_k)) \end{aligned}$$

- Propositions: values assigned to formulae like in PL^0.

Definition (Substitutions on Terms)**.**
We call σ a **substitution**, if $\sigma(f(A_1,\dots,A_n)) = f(\sigma(A_1),\dots,\sigma(A_n))$ and the **support** $supp(\sigma) = \{X \mid \sigma(X) \neq X\}$ is finite.
So, if B is a term and X is a variable, then we denote the result of systematically replacing all occurrences of X in a term A by B with $[B/X](A)$.

Definition (Additional definitions to substitution)**.**
We call $intro(\sigma) = \bigcup_{X\in supp(\sigma)} free(\sigma(X))$ the set of variables **introduced** by σ.
If σ is a substitution, then we call $\sigma, [A/X]$ the **extension** of σ by [A/X].
We can **discharge** a variable X from a substitution σ by $\sigma_{-X} = \sigma, [X/X]$.

Lemma (Substitution Value Lemma for Terms)**.**
Let A and B be terms, then $I_\varphi([B/X]A) = I_\psi(A)$, where $\psi = \varphi, [I_\varphi(B)/X]$.

Lemma (Substitution Value Lemma for Propositions)**.**
$I_\varphi([B/X]A) = I_\psi(A)$, where $\psi = \varphi, [I_\varphi(B)/X]$

First-order Calculi

Propositional Natural Deduction Calculus

→ Use the already given rules for natural deduction.
→ The first-order natural deduction calculus ND^1 extends ND^0 by the following four rules

$$\frac{\mathbf{A}}{\forall X.\mathbf{A}}\forall I^* \qquad \frac{\forall X.\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\forall E$$

$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X.\mathbf{A}}\exists I \qquad \frac{\exists X.\mathbf{A} \quad \begin{matrix}[[c/X](\mathbf{A})]^1 \\ \vdots \\ \mathbf{C}\end{matrix}}{\mathbf{C}}\exists E^1$$

→ The intuition behind the rule ∀I is that a formula A with a (free) variable X can be generalized to ∀X.A, if X stands for an arbitrary object
→ Quantifier Rules:

$$\frac{\Gamma\vdash\mathbf{A} \quad X\notin \text{free}(\Gamma)}{\Gamma\vdash\forall X.\mathbf{A}}\forall I \qquad \frac{\Gamma\vdash\forall X.\mathbf{A}}{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}\forall E$$

$$\frac{\Gamma\vdash[\mathbf{B}/X](\mathbf{A})}{\Gamma\vdash\exists X.\mathbf{A}}\exists I \qquad \frac{\Gamma\vdash\exists X.\mathbf{A} \quad \Gamma,[c/X](\mathbf{A})\vdash\mathbf{C} \quad c\in\Sigma_0^{sk}\text{ new}}{\Gamma\vdash\mathbf{C}}\exists E$$

Definition (Natural deduction with equality)**.**
We add a new logical symbol for equality $=\in\sigma^p_2$ and fix its semantics to $I(=) = \{(x,x) \mid x\in D_l\}$. We call the extended logic **first-order logic with equality** ($PL^1_=$).
New rules:

$$\frac{}{A=A} =I \qquad \frac{A=B \quad C[A]_p}{[B/p]C} =E$$

$$\frac{}{A\Leftrightarrow A} \Leftrightarrow I \qquad \frac{A\Leftrightarrow B \quad C[A]_p}{[B/p]C} \Leftrightarrow\!= E$$

→ $C[A]_p$ if the formula C has a subterm A at position p and [B/p]C is the result of replacing that subterm with B.

Definition (Positions in formulae)**.**
Idea: Formulae are (naturally) trees, so we can use tree positions to talk about subformulae.
A **formula position** p is a list of natural number that in each

node of a formula (tree) specifies into which child to descend. For a formula A we denote the **subformula at p** with $A|_p$.
⇒ We will sometimes write a formula C as $C[A]_p$ to indicate that C the subformula A at position p.
Let p be a position, then [A/p]C is the formula obtained from C by **replacing** the subformula at position p by A.

First-Order Inference

Tableaux

Note: There are two possible kinds of Tableaux-proofs:

1. **Tableau Refutation:**
   Negate the formula and derive everything to false.
2. **Model Generation:**
   Derive given formula to satisfiable assignment.

Additional rules to T_0 in T_1:

$$\frac{\forall X.A^T \quad C \in cwff_l(\Sigma_l)}{[C/X](A)^T} T_1:\forall \qquad \frac{\forall X.A^F \quad c \in (\Sigma_0^{sk} \setminus H)}{[c/X](A)^F} T_1:\exists$$

T_1:∀: A universally quantified formula is true, if all of the instances of the scope are. (Note: we have to guess C!)
T_1:∃: Remind, that ∃X.A ⇔ $\forall X.A^F$.
In other words: There exists no object with property A^F. We name this object c and take it from our witness constants Σ_0^{sk}.

Free Variable Tableaux (T_1^f):
This is refutation calculus bases on:

- T_0- rules
- Quantifier rules:

$$\frac{\forall X.A^T \quad Y\ new}{[Y/X](A)^T} T_1^f:\forall$$

$$\frac{\forall X.A^F \quad free(\forall X.A) = \{X^1,\ldots,X^k\} \quad f \in \Sigma_k^{sk}}{[f(X^1,\ldots,X^k)/X](A)^F} T_1^f:\exists$$

- Generalized cut rule: T_1^f:⊥ instantiates the whole tableau by σ

$$\frac{\begin{matrix}A^\alpha \\ B^\beta \quad \alpha \neq \beta \quad \sigma(A) = \sigma(B)\end{matrix}}{\bot : \sigma} T_1^f:\bot$$

→ Instead of guessing (T_1:∀), we instantiate with a new **meta-variable** Y (T_1^f:∀)
⇒ All T_1^f rules except T_1^f:∀ only need to be applied once!
→ Number of times T_1^f:∀ occurs is called **multiplicity**.
→ There might be more than one opportunity to use T_1^f:⊥ on a branch.
→ There are two ways to find the right one: backtracking over T_1^f:⊥ opportunities or saturate without T_1^f:⊥ and find spanning matings.

Definition (Spanning matings)**.**
Idea: Saturate without T_1^f:⊥ and treat all cuts at the same time.
Let T be a T_1^f tableau, then we call a **unification** problem $\varepsilon = A_1 =^? B_1 \wedge \cdots \wedge A_n =^? B_n$ a **mating** for T, if A_i^T and B_i^F occur in the same branch in T.
We say that ε is a **spanning mating**, if ε is unifiable and every branch B is a mating.

⇒ A T_1^f-tableau with a spanning mating induces a closed T_1-tableau.

First-Order unification:
Problem: Find a substitution σ, such that two terms are equal.
→ Solutions are called unifiers.

Definition (Most general unifier)**.**
σ is called a **most general unifier** of A and B, if it's minimal in $U(A =^? B)$.

→ Unification can be written as an equational system:

$$A_1 =^? B_1 \wedge \cdots \wedge A_n =^? B_n$$

Lemma (Unique most general unifier)**.**
If ε is a solved form, then ε has the unique most general unifier $\sigma_\varepsilon = [B^1/X^1],\ldots,[B^n/X^n]$.

Unification Algorithm with inference system U:

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1,\ldots,\mathbf{A}^n) =^? f(\mathbf{B}^1,\ldots,\mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 =^? \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^? \mathbf{B}^n} \mathcal{U}\,\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathbf{A} =^? \mathbf{A}}{\mathcal{E}} \mathcal{U}\,\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^? \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad X \in \mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =^? \mathbf{A}} \mathcal{U}\,\mathrm{elim}$$

→ U is **correct**: $\varepsilon \vdash_U F$ implies $U(F) \subseteq U(\varepsilon)$
→ U is **complete**: $\varepsilon \vdash_U F$ implies $U(\varepsilon) \subseteq U(F)$
→ U is **confluent**: The order of the derivations does not matter.
→ If ε is unifiable but not solved, then it is **U-reducible**.

First-order Resolution

Extension of the rules for the CNF (CNF^1):

$$\frac{(\forall X.A)^T \vee C \quad Z \notin (free(A) \cup free(C))}{[Z/X](A)^T \vee C}$$

$$\frac{(\forall X.A)^F \vee C \quad \{X_1,\ldots,X_k\} = free(\forall X.A)}{[f_n^k(X^1,\ldots,X^k)/X](A)^F \vee C}$$

→ $CNF^1(\Phi)$ is called the set of all clauses that can be derived from Φ.

Two inference rules for first-order resolution R^1:

$$\frac{A^T \vee C \quad B^F \vee D \quad \sigma = mgu(A,B)}{\sigma(C) \vee \sigma(D)}$$

$$\frac{A^\alpha \vee B^\alpha \vee C \quad \sigma = mgu(A,B)}{\sigma(A) \vee \sigma(C)}$$

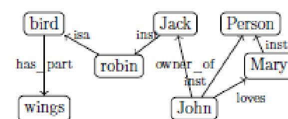
Knowledge Representation

→ Knowledge is the information necessary to support intelligent reasoning.
→ Representation as structure and function:
⇒ Representation determines the content theory
⇒ Function determines the process model

Definition (Knowledge Representation evaluation criteria)**.**

- **Expressive Adequacy**: What can be represented, what distinctions are supported.
- **Reasoning Efficiency**: Results in acceptable speed?
- **Primitives**: Are the primitives intuitive?
- **Meta-representation**: Knowledge about knowledge.
- **Incompleteness**: Knowledge is known to be incomplete.

Definition (Semantic Networks)**.**
A **semantic network** is a directed graph for representing knowledge. Nodes represent concepts (i.e. classes of objects) and links represent the relation between these.



Observation: There is more knowledge in a semantic network that is explicitly written down

Definition (Inference in semantic networks)**.**
We call all link labels except inst (instance) and isa (is a) in a semantic network **relations**.
Let N be a semantic network and R a relation in N, such that

$A \overset{isa/inst}{\rightarrow} B \overset{R}{\rightarrow} C$, then we can derive a relation $A \overset{R}{\rightarrow} C$ in N.
⇒ Derived relations represent knowledge that's implicit in network.
We call the subgraph of a semantic network N spanned by the isa relations the terminology and the subgraph spanned by the inst relation the assertions.

Definition (Function/argument notation).
In this notation for semantic networks, nodes are interpreted as arguments and links as functions.
→ inst: $A \subseteq B \Rightarrow$ FOL: $\forall X.A(X) \Rightarrow B$
→ isa: $a \in A \Rightarrow$ FOL: $S(a)$

Definition (Semantic Web).
The **semantic web** is a collaborative movement led by the W3C that promotes the inclusion of semantic content in web pages with the aim of converting the current web, dominated by unstructured and semistructured documents into a machine-understandable "web of data".
→ For better computational reading, add XML markup with meaningful tags.
→ The current web consist of links with less meaning.
→ In semantic web, our aim is to get references and links, such that we can do inference with that.
⇒ Inference with annotations and ontologies.

Logic-based knowledge representation

→ Logic have a well-defined semantics because it is explicitly, transparently and systematically.
Possible problems with logic-based approaches:

- Ontology problem: Where does world knowledge come from?
- Combinatorial explosion: How to guide search?

Propositional Logic as a Set Description Language

Definition (Formal Semantics).
Let domain D bei a given set and $\varphi : V_0 \rightarrowtail P(D)$ then

- $[[P]] = \varphi(P)$
- $[[A \sqcup B]] = [[A]] \vee [[B]]$
- $[[\bar{A}]] = D \setminus [[A]]$

Example:

$$\overline{\text{son} \sqsubseteq \text{child}}^{fo} = \forall x.\text{son}(x) \Rightarrow \text{child}(x)$$
$$\overline{\text{daughter} \sqsubseteq \text{child}}^{fo} = \forall x.\text{daughter}(x) \Rightarrow \text{child}(x)$$
$$\overline{\overline{(\text{son} \sqsubseteq \text{daughter})}}^{fo} = \forall x.\overline{\text{son}(x) \wedge \text{daughter}(x)}$$
$$\overline{\text{child} \sqsubseteq (\text{son} \sqcup \text{daughter})}^{fo} = \forall x.\text{child}(x) \Rightarrow \text{son}(x) \vee \text{daughter}(x)$$

Ontologies and Description Logics

Definition (Ontology).
An **ontology** is a representation of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse. It consists of a representation format L and statements about individuals, concepts and relations.
→ Example: PL^1 is an ontology format

Definition (Description logic).
A **description logic** is a formal system for talking about sets and their relations that is at least as expressive as PL^0 with set-theoretic semantics and offers individuals and relations.

Definition (D-ontology). Given a description logic D, D-ontology consists of

- a terminology: concepts and roles and a set of concept axioms that describe them.
  → A concept definition is a pair c = C, where c is a new concept name and C ∈ C is a D-formula.
  → c = C is called recursive, if c occurs in C.
  → An TBox is a finite set of concept definitions and concept axioms. It is called acyclic, if it does not contain recursive definitions.
- assertions: a set of individuals and statements about concept membership and role relationships for them.

Example T-Box:

woman	=	$\text{person} \sqcap \overline{\text{has_Y}}$	person without y-chromosome
man	=	$\text{person} \sqcap \text{has_Y}$	person with y-chromosome
hermaphrodite	=	$\text{man} \sqcap \text{woman}$	man and woman

Definition (Subsumption).
A **subsumes** B, if $[[B]] \subseteq [[A]]$ for all interpretations D, that satisfy A and if Axioms ⇒ B ⇒ A is valid.
This is valid, if $Axioms \wedge A \wedge \neg B$ is inconsistent.
Classification is the computation of the subsumption graph.

The description logic ALC

Motivation:
→ PL^0 is not expressive enough, but PL^1 is too hard.
→ Allow only restricted quantification, where quantified variables only range over values that can be reached via a binary relation.

Definition (Syntax).
Concepts in DLs name classes of objects like in OOP. We have the top-concept ⊤ for true and the bottom-concept ⊥ for false. We name binary relations like in PL^1
Example: person ⊓ ∃ has_child.student (parents of students)

Definition (TBox Normalization).
Normalization result can be exponential and need not terminate on cyclic TBoxes.

$$\text{german} \mapsto \text{person} \sqcap \exists \text{has_parents}.\text{german}$$
$$\mapsto \text{person} \sqcap \exists \text{has_parents}.(\text{person} \sqcap \exists \text{has_parents}.\text{german})$$
$$\mapsto \ldots$$

Definition (Concept Axioms).
DL formulae that are not concept definitions are called **concept axioms**. They normally contain additional information about concepts.

Definition (Semantic).
A **model** for ALC is a pair, where D is nonempty set called domain and [[.]] a mapping called interpretation, such that

Op.	formula semantics
	$[c] \subseteq \mathcal{D} = [\top]$ $[\bot] = \emptyset$ $[r] \subseteq \mathcal{D} \times \mathcal{D}$
$\overline{\cdot}$	$[\overline{\varphi}] = \overline{[\varphi]} = \mathcal{D} \backslash [\varphi]$
$\sqcap$	$[\varphi \sqcap \psi] = [\varphi] \cap [\psi]$
$\sqcup$	$[\varphi \sqcup \psi] = [\varphi] \cup [\psi]$
∃R.	$[\exists R.\varphi] = \{x \in \mathcal{D} \mid \exists y.\langle x, y\rangle \in [R] \text{ and } y \in [\varphi]\}$
∀R.	$[\forall R.\varphi] = \{x \in \mathcal{D} \mid \forall y.\text{if } \langle x, y\rangle \in [R] \text{ then } y \in [\varphi]\}$

Translation to PL^1:

Definition	Comment
$\overline{p}^{fo(x)} := p(x)$	
$\overline{\overline{A}}^{fo(x)} := \neg \overline{A}^{fo(x)}$	
$\overline{A \sqcap B}^{fo(x)} := \overline{A}^{fo(x)} \wedge \overline{B}^{fo(x)}$	∧ vs. ⊓
$\overline{A \sqcup B}^{fo(x)} := \overline{A}^{fo(x)} \vee \overline{B}^{fo(x)}$	∨ vs. ⊔
$\overline{A \sqsubseteq B}^{fo(x)} := \overline{A}^{fo(x)} \Rightarrow \overline{B}^{fo(x)}$	⇒ vs. ⊑
$\overline{A = B}^{fo(x)} := \overline{A}^{fo(x)} \Leftrightarrow \overline{B}^{fo(x)}$	⇔ vs. =
$\overline{A}^{fo} := (\forall x, \overline{A}^{fo(x)})$	for formulae

$$\overline{\forall R.\varphi}^{fo(x)} := (\forall y.R(x, y) \Rightarrow \overline{\varphi}^{fo(y)}) \quad \overline{\exists R.\varphi}^{fo(x)} := (\exists y.R(x, y) \wedge \overline{\varphi}^{fo(y)})$$

In addition, we have the following identities:

1	$\overline{\exists R.\varphi} = \forall R.\overline{\varphi}$	3	$\overline{\forall R.\varphi} = \exists R.\overline{\varphi}$
2	$\forall R.(\varphi \sqcap \psi) = \forall R.(\varphi \sqcap \forall R.\psi)$	4	$\exists R.(\varphi \sqcup \psi) = \exists R.(\varphi \sqcup \exists R.\psi)$

We have a **negation normal form**, if we have the negation directly in front of concept names. It can be computed with the identity rules.

Definition (ABox Formulae).
→ $a : \varphi$: a is a φ
⇒ $[[a : \varphi]] = \top$, if $[[a]] \in [[\varphi]]$ and $\overline{a : \varphi}^{fo} = \overline{\varphi}^{fo(a)}$
→ aRb : a stands in relation R to b
⇒ [[aRb]]= $\top$, if $([[a]], [[b]]) \in [[R]]$ and $\overline{aRb}^{fo} = R(a, b)$

Definition (Tableau for ALC).
The tableau calculus T_{ALC} acts on ABox assertions (x: φ:) and (xRy :) with the following rules:

$$\frac{x : c \quad x : \overline{c}}{\bot}\mathcal{T}_{\bot} \qquad \frac{x : (\varphi \sqcap \psi)}{x : \varphi \quad x : \psi}\mathcal{T}_{\sqcap} \qquad \frac{x : (\varphi \sqcup \psi)}{x : \varphi \mid x : \psi}\mathcal{T}_{\sqcup} \qquad \frac{x : (\forall R.\varphi) \quad xRy}{y : \varphi}\mathcal{T}_{\forall} \qquad \frac{x : (\exists R.\varphi)}{xRy \quad y : \varphi}\mathcal{T}_{\exists}$$

new rule for $\exists$: $$\frac{x : (\exists R.\varphi) \quad CA = \{a_1, \ldots, a_n\}}{y : \varphi \quad xRy \quad y : \alpha_1 \quad \ldots \quad y : \alpha_n}\mathcal{T}^{\exists}_{CA}$$

Properties:
- Termination: there are no infinite sequences of rule applications
- Correctness: If φ is satisfiable, then C terminates with an open branch.
- Completeness: If φ is in unsatisfiable, then C terminates and all branches are closed.
- Complexity of the algorithm.

Example: Concrete difference between TBox and ABox:

TBox (terminological Box)	ABox (assertional Box, data base)	
woman = $\overline{\text{person} \sqcap \text{has_Y}}$	tony : person	Tony is a person
man = person $\sqcap$ has_Y	tony : has_Y	Tony has a y-chrom

Definition (Realization).
Realization is the computation of all instance relations between ABox objects and TBox concepts.
→ Sufficient to remember the lowest concepts in the subsumption graph

Interactions between TBox and ABox:

property	example
internally inconsistent	tony : student, tony : $\overline{\text{student}}$
inconsistent with a TBox	TBox: $\overline{\text{student}} \sqcap$ prof ABox: tony : student, tony : prof
implicit info that is not explicit	Abox: tony : ($\forall$ has_grad.genius) tonyhas_gradmary $\models$ mary : genius
info that can be combined with TBox info	TBox: cont_prof = prof $\sqcap \forall$ has_grad.genius ABox: tony : cont_prof, tonyhas_gradmary $\models$ mary : genius

Definition (Tableau-based Instance Test and Realization).
Query: do the ABox and TBox together entail a:φ
Solution: test a: φ for consistency with ABox and TBox.
→ Normalize ABox wrt. TBox
→ initialize the tableau with ABox in NNF Example:

Example: add $\overline{\text{mary : genius}}$ to determine $ABox, TBox \models$ mary : genius		
TBox	cont_prof = prof $\sqcap \forall$ has_grad.genius	tony : (prof $\sqcap \forall$ has_grad.genius) TBox tony has_grad mary ABox mary : $\overline{\text{genius}}$ Query tony : prof $\mathcal{T}_{\sqcap}$ tony : ($\forall$ has_grad.genius) $\mathcal{T}_{\sqcap}$ mary : genius $\mathcal{T}_{\forall}$ $\bot$ $\mathcal{T}_{\bot}$
ABox	tony : cont_prof tony has_grad mary	

Description Logics and the Semantic Web

Definition (Resource Description Framework).
The **Resource Description Framework** (RDF) is a framework for describing resources on the web. It is an XML vocabulary developed by the W3C.
RDF is designed to be read and understood by computers, not to be being displayed to people.
→ RDF describes resources with properties and property values.
→ RDF uses Web identifiers (URIs) to identify resources.
⇒ A **resource** is anything that can have a URI.
→ A **property** is a resource that has a name (author).

Definition (RDF statement).
A RDF **statement** (also known as a triple) s consists of a resource (the subject), aproperty (the predicate), and a property value (the object of s). A set of RDF tripless is called an RDF graph.
⇒ RDF is a concrete XML vocabulary for writing statements.
→ Problem: RDF is a standoff markup format (annotate by URIs pointing into other files)
→ Idea: RDF triples are ABox entries hRs or h:φ

Definition (Ontology Language for the Semantic Web).
OWL (the **ontology web language**) is a language for encoding TBox information about RDF classes.
Semantic networks can be expressend in functional syntax:
- ClassAssertion formalizes the inst relation.
- ObjectPropertyAssertion formalizes relations.
- SubClassOf formalizes the isa relation.
- birds has_part wings: SubClassOf (:bird ObjectSomeValuesFrom(:hasPart :wing))

Definition (Sparql).
SPARQL is an RDF query language, able to retrieve and manipulate data stored in RDF. A database that stores RDF data is called a **triple store**. A triple store is called a **SPARQL endpoint**, if it answers SPARQL queries.

Definition (Triplestore).
A **triplestore** or RDF store is a purpose-built database for the storage RDF graphs and retrieval of RDF triples through semantic queries, usually variants of SPARQL.