(Yet another)^2 *Artificial Intelligence I* Summary

Written by Lorenz Gorse[*]
Updated by Marius Frinken[†] Philip K.[‡]

Last updated for the Winter Semester 2020

1 Agents

Def. 1 (Agent)**.** An agent a is an entity that perceives (via sensors) and acts (via actuators). It can be modelled as an *agent function* $f_a : \mathcal{P}^* \mapsto \mathcal{A}$, mapping from percept histories to actions.

Def. 2 (Performance measure)**.** A function that evaluates a sequence of environments.

Def. 3 (Rationality)**.** An agent is "rational", if it chooses the actions that maximizes the expected value of the performance measure given the percept history.

Def. 4 (Autonomy)**.** An agent is called autonomous, if it does not rely on the prior knowledge of the designer. Autonomy avoids fixed behaviour in changing environments.

Def. 5 (Task environment)**.** The combination of a performance measure, **e**nvironment, **a**ctuators and **s**ensors (PEAS) describes a (task) environment e.

Def. 6 (Environment properties)**.** An environment is called...

fully observable, iff a's sensors give it access to the complete state of e at any point in time, else **partially observable**.
deterministic, iff the next state of e is completely determined by a's action and e's current state, else **stochastic**.
episodic, iff a's experience is divided into atomic, independent episodes, where it perceives and performs a single action. Non-episodic environments are called **sequential**.
dynamic, iff e can change without an action performed by a, else **static**.
discrete, iff the sets of e's states and a's actions are countable, else **continuous**.
single-agent, iff only a acts on e.

Def. 7 (Simple reflex agent)**.** An agent that bases its next action only on the most recent percept, $f_a : \mathcal{P} \mapsto \mathcal{A}$.

Def. 8 (Model-based agent)**.** Like simple reflex agent, but additionally maintains a (world) model to decide it's next move.

Def. 9 (Goal-based agent)**.** A model-based agent that also takes it's goals into consideration when deciding.

Def. 10 (Utility-based agent)**.** An agent that combines a world model and a utility function, measuring state-preferences. Its choices attempt to maximize the expected utility, allowing rational decisions where goals are insufficient.

Def. 11 (Learning agent)**.** An agent that augments the performance element, which chooses actions from percept sequences, with a...

learning element making improvements to the agent's performance element.
critic giving feedback to the *learning element* based on an external performance standard.
problem generator suggesting actions that can lead to new, informative experiences.

Def. 12 (State representation)**.** We call a state representation:

atomic if it has no internal structure.
factored if each state is characterized by attributes and their values.
structured if the state includes objects and their relations.

2 Search

Def. 13 (Search problem)**.** A search problem $\Pi := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{G} \rangle$ consists of a set $\mathcal{S}$ of states, a set $\mathcal{A}$ of actions, a transition model $\mathcal{T} : \mathcal{A} \times \mathcal{S} \mapsto \mathfrak{P}(\mathcal{S})$ that assigns any action and state to a set of successor states. Certain states in $\mathcal{S}$ are labelled as "goal states" $\mathcal{G}$ and "initial states" $\mathcal{I}$. A cost function $c : \mathcal{A} \mapsto \mathbb{R}_0^+$ may assigns costs to actions.

Def. 14 (Solution)**.** A sequence of applicable actions that lead from a initial state $\mathcal{I}$ to a goal state $g \in \mathcal{G}$ is a solution.

Def. 15 (Problem types)**.** Problems come in many variations:

Single-state problem: state is always known with certainty (observable, deterministic, static, discrete)
Multiple-state problem: know which states might be in (initial state not/partially observable)
Contingency problem: constructed plans with conditional parts based on sensors (non-deterministic, unknown state space)

Def. 16 (Tree search)**.** An algorithm that explores state spaces, forming a search tree of already-explored states, modelled as nodes. It's fringe are the nodes that have not yet been considered.

Def. 17 (Search strategy)**.** A search strategy picks a node from the fringe of a search tree. It's properties are:

Completeness: Does it always find a solution if one exists?
Time complexity: Number of nodes generated/expanded.
Space complexity: Maximum number of nodes held in memory.
Optimality: Does it always find the least-cost solution?

Def. 18 (Uninformed search)**.** Search strategies that only employ information from the problem definition yield uninformed searches. Examples are breadth-first-search (BFS), uniform-cost-search (UCS, also called "Dijkstra's algorithm"), depth-first-search (DFS), depth-limited search and iterative-deepening-search (IDS).

Def. 19 (Informed search)**.** Search strategies that use information about the real world beyond the problem statement yield informed searches. The additional information about the world is provided in form of heuristics. Examples are greedy-search and $A^\star$-search.

Def. 20 (Heuristic)**.** A heuristic is an evaluation function $h : \mathcal{S} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ that estimates the cost from a state n to the nearest goal state. If $s \in \mathcal{G}$, then $h(s) = 0$. All nodes for the same states must have the same h-value.

Def. 21 (Goal distance function)**.** A function $h^* : \mathcal{S} \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ determining the cheapest path from any nodes to a goal state, or ∞ if no path exists.

Def. 22 (Admissibility and consistency)**.** A heuristic h is admissible if $h(s) \leq h^*(s)$ for all states $s \in \mathcal{S}$, i.e. forming a lower bound. h is consistent if $h(s) - h(s') \leq c(a)$ for all $s \in \mathcal{S}$, $a = (s, s') \in \mathcal{A}$ and a cost function c.

Def. 23 (Greedy-search)**.** Greedy-search always expands the node that appears to be closest to a goal state, as determined by a heuristic.

[*]https://gitlab.cs.fau.de/snippets/15
[†]https://gitlab.cs.fau.de/tyr/ki-1-summary
[‡]https://gitlab.cs.fau.de/oj14ozun/ai1-summary

Def. 24 ($A^\star$-search)**.** Expands the node with the minimum evaluation value $f(s)=g(s)+h(s)$, where $g(s)$ is the path cost. $A^\star$-search is optimal if it uses an admissible heuristic h.

Def. 25 (Dominance)**.** Let h_1 and h_2 be two admissible heuristics we say that h_1 dominates h_2 if $h_1(s)\geq h_2(s)$ for all $s\in\mathcal{S}$. The dominant heuristic is better for search.

Def. 26 (Relaxation)**.** A search problem $\Pi:=\langle\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{I},\mathcal{G}\rangle$ has a relaxed problem $\Pi^r:=\langle\mathcal{S},\mathcal{A}^r,\mathcal{T}^r,\mathcal{I}^r,\mathcal{G}^r\rangle$ iff $\mathcal{A}\subseteq\mathcal{A}^r$, $\mathcal{T}\subseteq\mathcal{T}^r$ $\mathcal{I}\subseteq\mathcal{I}^r$ $\mathcal{G}\subseteq\mathcal{G}^r$. This means that any solution for Π is a solution for Π^r.

Def. 27 (Local search)**.** A search algorithm that only operates on a single space at a time is called a local search. Local search algorithms need constant space, because it doesn't have to remember multiple paths. Examples are hill-climbing, simulated annealing or genetic algorithms.

2.1 Adversarial Search

Def. 28 (Game state space)**.** A 6-tuple $\Theta=\langle S,A,T,I,S^T,u\rangle$ is a game state space, for two players "Max" and "Min" consists of:

- S is the disjoint union of S^{Max}, S^{Min} and S^T (respectively the sets of "Max" 's, "Min" 's and terminal states).
- A is the disjoint union $A^{\text{Max}}\subseteq S^{\text{Max}}\times(S^{\text{Min}}\cup S^T)$ and $A^{\text{Min}}\subseteq S^{\text{Min}}\times(S^{\text{Max}}\cup S^T)$
- I is the initial state.
- $u:S^T\mapsto\mathbb{R}$ is the utility function.

Def. 29 (Strategy)**.** Let Θ be a game state space, and $X\in\{\text{Max},\text{Min}\}$. A strategy for X is a function $\sigma^X:S^X\mapsto A^X$ so that a is applicable to s whenever $\sigma^X(s)=a$. A strategy is optimal if it yields the best possible utility for X assuming perfect opponent play.

Def. 30 (Minimax Algorithm)**.** The minimax algorithm is given by the following function whose input is a state $s\in S^{\text{Max}}$, in which Max is to move. It attempts to find the best move for "Max":

```
function 1 MinimaxDecision($s$) returns an action
1: $v:=$MaxValue($s$)
2: return an action yielding value $v$ in the previous function call
```

```
function 2 MaxValue($s$) returns a utility value
1: if TerminalTest($s$) then return $u(s)$
2: $v:=-\infty$
3: for each $a\in$Actions($s$) do
4:     $v:=\max(v,$MinValue(ChildState($s,a$)))
5: return $v$
```

```
function 3 MinValue($s$) returns a utility value
1: if TerminalTest($s$) then return $u(s)$
2: $v:=+\infty$
3: for each $a\in$Actions($s$) do
4:     $v:=\min(v,$MaxValue(ChildState($s,a$)))
5: return $v$
```

Def. 31 (Alpha-Beta Search)**.** To avoid evaluating states that are not of interest, *Alpha-Beta* Pruning can be used to accelerate Minimax search:

```
function 4 AlphaBetaSearch($s$) returns an action
1: $v:=$MaxValue($s,-\infty,+\infty$)
2: return an action yielding value $v$ in the previous function call
```

```
function 5 MaxValue($s,\alpha,\beta$) returns a utility value
1: if TerminalTest($s$) then return $u(s)$
2: $v:=-\infty$
3: for each $a\in$Actions($s$) do
4:     $v:=\max(v,$MinValue(ChildState($s,a$)$,\alpha,\beta$))
5:     $\alpha:=\max(\alpha,v)$
6:     if $v\geq\beta$ then return $v$        ▷ $v\geq\beta\iff\alpha\geq\beta$
7: return $v$
```

```
function 6 MinValue($s,\alpha,\beta$) returns a utility value
1: if Terminal-Test($s$) then return $u(s)$
2: $v:=+\infty$
3: for each $a\in$Actions($s$) do
4:     $v:=\min(v,$MaxValue(ChildState($s,a$)$,\alpha,\beta$))
5:     $\beta:=\min(\beta,v)$
6:     if $v\leq\alpha$ then return $v$        ▷ $v\leq\beta\iff\alpha\geq\beta$
7: return $v$
```

Def. 32 (Monte-Carlo Tree Search)**.** If there is no good known evaluation function, *Monte-Carlo Tree Search* decides on an action through sampling average $u(t),t\in S^T$. For Monte-Carlo tree search we maintain a search tree T:

```
function 7 MonteCarloTreeSearch($s$) returns an action
1: while time not up do
2:     apply actions within $T$ to select a leaf state $s'$
3:     select action $a'$ applicable to $s'$
4:     run random sample from $a'$
5:     add $s'$ to $T$, update averages etc.
6: return an $a$ for $s$ with maximal average $u(t)$
7: When executing $a$, keep the part of $T$ below $a$.
```

3 Constraint Satisfaction Problems

Def. 33 (Constraint Satisfaction Problem, CSP)**.** This is a search problem where the states are given by a finite set of variables $V:=\{X_1,...,X_n\}$ over domains $D:=\{D_v\,|\,v\in V\}$ and a goal test, giving legal combinations of values for subsets of variables. The CSP is called...

binary iff all constraint relate at most two variables.
discrete iff all of the variables have countable domains.
continuous iff it is not discrete.

A CSP has a factored world representation. Examples include SuDuKo, Map-Colouring, Timetabling and Scheduling.

Def. 34 (Constraint network)**.** A triple $\langle V,D,C\rangle$ is called a constraint network, where V and D as the same as for CSPs, and a set of binary constraints

$$C:=\{C_{uv}=C_{vu}\subseteq D_u\times D_v\,|\,u,v\in V \text{ and } u\neq v\}.$$

Any CSP can be represented by a constraint network.

Def. 35 (Constraint Network Graph)**.** For a constraint network $\gamma=\langle V,D,C\rangle$, the graph formed by $\langle V,C\rangle$ is called the "constraint graph" of γ.

Def. 36 (Assignment)**.** A partial assignment for a constraint network is a partial function $a: V \mapsto \bigcup_{v\in V} D_v$ if $a(v) \in D_v$ for all $v \in V$. If a is total a, is just called an "assignment".

Def. 37 (Consistency)**.** A partial assignment a is inconsistent, iff there are variables $u,v \in V$ and a constraint $C_{uv} \in C$ and $(a(u),a(v)) \notin C_{uv}$. Otherwise a is called consistent. A consistent, total assignment is a solution.

Def. 38 (Backtracking on CSPs)**.** A straightforward approach to solve a CSP is to incrementally try assigning variables until a consistent solution is found, backtracking if necessary. To improve the efficiency of this approach, the following heuristics can be applied:

Minimum remaining values Assign the variable with the fewest remaining legal values. This is done to reduce the branching factor of the search tree.

Degree heuristic Assign the variable with the most constraints on remaining variables. This is done to detect inconsistencies early on.

Least constraining value When assigning a variable, choose the value that rules out the fewest values from the neighbouring domains.

Def. 39 (Equivalent constraint networks)**.** Two constraint networks $\gamma = \langle V,D,C\rangle$ and $\gamma' = \langle V,D',C'\rangle$ are equivalent $(\gamma \equiv \gamma')$ iff they have the same solutions.

Def. 40 (Tightness)**.** Let $\gamma = \langle V,D,C\rangle$ and $\gamma' = \langle V,D',C'\rangle$ be two constraint networks. γ' is "tighter" than γ $(\gamma' \sqsubseteq \gamma)$ iff
1. For all $v \in V$, $D'_v \subseteq D_v$
2. For all $u,v \in V, u \neq v$ and $C'_{uv} \in C'$, $C'_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$

If at least one of these inclusions are strict, γ' is "strictly tighter".

An equivalent but tighter constraint network is preferable, because it has fewer consistent partial assignments.

Def. 41 (Backtracking with Inference)**.** The general algorithm for backtracking with inference, where Inference(γ) is any procedure that delivering a (tighter) equivalent network.

function 8 BacktrackingWithInference(γ,a) **returns** a solution, or "inconsistent"

```
1: if $a$ is inconsistent then return "inconsistent"
2: if $a$ is a total assignment then return $a$
3: $\gamma' :=$ a copy of $\gamma$                  ▷ $\gamma' := (V,D',C')$
4: $\gamma' :=$ Inference($\gamma'$)
5: if exists $v$ with $D'_v = \emptyset$ then return "inconsistent"
6: select some variable $v$ for which $a$ is not defined
7: for each $d \in$ copy of $D'_v$ in some order do
8:     $a' := a \cup \{v = d\}$        ▷ makes $a$ explicit as a constraint
9:     $D'_v := \{d\}$
10:    $a'' :=$ BacktrackingWithInference($\gamma'$,$a'$)
11:    if $a'' \neq$ "inconsistent" then return $a''$
12: return "inconsistent"
```

Def. 42 (Forward checking)**.** For a constraint network γ and a partial assignment a, propagate information about values from the domains of unassigned variables that are in conflict with the values of already assigned variables to obtain a tighter network γ'.

function 9 ForwardChecking(γ,a) **returns** modified γ

```
1: for each $v$ where $a(v) = d'$ is defined do
2:     for each $u$ where $a(u)$ is undefined and $C_{uv} \in C$ do
3:         $D_u := \{d \in D_u \mid (d,d') \in C_{uv}\}$
4: return $\gamma$
```

Def. 43 (Arc consistency)**.** A variable pair $v,u \in V, v \neq u$ is arc consistent, if $C_{uv} \notin C$ or for every value $d \in D_v$ there exists a $d' \in D_u$ such that $(d,d') \in C_{uv}$.

A constraint network γ is act consistent, if every variable pair $v,u \in V, v \neq u$ is arc consistent.

Arc consistency can be "enforced" by reducing domains. Revise(γ,v,u) enforces arc consistency for v relative to u.

function 10 Revise(γ,v,u) **returns** modified γ

```
1: for each $d \in D_v$ do
2:     if there is no $d' \in D_u$ with $(d,d') \in C_{vu}$ then
3:         $D_v := D_v \setminus \{d\}$
4: return $\gamma$
```

The AC-3 Algorithm ($\mathcal{O}(mk^3)$, for m constraints and maximal domain size k) applies Revise(γ,u,v) up to a fixed point, remembering potentially inconsistent variable pairs:

function 11 AC$-$3(γ) **returns** modified γ

```
1: $M := \emptyset$
2: for each constraint $C_{uv} \in C$ do
3:     $M := M \cup \{(u,v),(v,u)\}$
4: while $M \neq \emptyset$ do
5:     remove any element $(u,v)$ from $M$
6:     Revise($\gamma$,$u$,$v$)
7:     if $D_u$ has changed in the call to revise then
8:         for each constraint $C_{wu} \in C$ where $w \neq v$ do
9:             $M := M \cup \{(w,u)\}$
10: return $\gamma$
```

To solve an acyclic constraint network, enforce arc consistency with AC$-$3(γ) and run backtracking with inference on the arc consistent network. This will find a solution without having to backtrack.

A simpler algorithm, AC$-$1(γ) has a runtime of $\mathcal{O}(mk^3n)$, where n is the number of variables.

Def. 44 (Acyclic Constraint Graph)**.** Let $\gamma = \langle V,D,C\rangle$ be a constraint network with n variables and maximal domain size k, whose constraint graph is acyclic. Then we can find a solution for γ, or prove γ to be inconsistent, in time $\mathcal{O}(nk^2)$:

function 12 AcyclicCG(γ) **returns** solution, or "inconsistent"

```
1: Obtain a directed tree from $\gamma$'s constraint graph, picking an
   arbitrary variable $v$ as the root, and directing arcs outwards.
2: Order the variables topologically, i.e., such that each vertex
   is ordered before its children; denote that order by $v_1,\ldots,v_n$.
3: for $i := n, n-1, \ldots, 2$ do
4:     Revise($\gamma$,$v_{\text{parent}(i)}$,$v_i$)
5:     if $D_{v_{\text{parent}(i)}} = \emptyset$ then return "inconsistent"
6: Run BacktrackingWithInference with forward checking,
   using the variable order $v_1,\ldots,v_n$.
```

Def. 45 (Cutset conditioning)**.** Let $\gamma = \langle V,D,C\rangle$ be a constraint network, and $V_0 \subseteq V$. V_0 is a "cutset" for γ if the sub-graph of γ's constraint graph-graph induced by $V \setminus V_0$ is acyclic. V_0 is optimal if its size is minimal among all cutsets for γ. The cutset conditioning algorithm computes an optimal cutset:

function 13 CutsetConditioning(γ,V_0,a) **returns** a solution, or “inconsistent”

```
$\gamma' := $ForwardChecking(a copy of $\gamma$,$a$)
if ex. $v$ with $D'_v = \emptyset$ then return “inconsistent”
if ex. $v \in V_0$ s.t. $a(v)$ is undefined then
    select such $v$
else
    $a' := $AcyclicCG($\gamma'$)
    if $a' \neq$ “inconsistent” then return $a \cup a'$
    else return “inconsistent”
for each $d \in$ copy of $D'_v$ in some order do
    $a' := a \cup \{v = d\}$; $D'_v := \{d\}$
    $a'' := $CutsetConditioning($\gamma'$,$V_0$,$a'$)
if $a' \neq$ “inconsistent”  then return $a''$
else return “inconsistent”
```

4 Logic

Def. 46 (Syntax)**.** Rules to decide what are legal statements (formulas).

Def. 47 (Semantics)**.** $\phi \models A$: Rules to decide whether a formula A is true for a given assignment ϕ .

Def. 48 (Model)**.** Consists of a universe and an interpretation (what connectives “do” and assignments).

Def. 49 (Entailment)**.** If for every model ϕ

$$\phi \models A \Rightarrow \phi \models B$$

B is entailed by A, written $A \models B$.

Def. 50 (Calculus)**.** A set of inference rules.

Def. 51 (Deduction)**.** Statements that can be derived from A using a calculus $\mathcal{C}$ (calculus), written $A \vdash_{\mathcal{C}} B$.

Def. 52 (Soundness)**.** A calculus $\mathcal{C}$ is sound if for all formulas A,B it is true that $A \vdash_{\mathcal{C}} B \Rightarrow A \models B$.

Def. 53 (Complete)**.** A calculus $\mathcal{C}$ is complete if for all formulas A,B it is true that $A \models B \Rightarrow A \vdash_{\mathcal{C}} B$.

Def. 54 (Logical Systen)**.** A logical system is a triple $\langle \mathcal{L}, \mathcal{K}, \models \rangle$, where $\mathcal{L}$ is a formal language, $\mathcal{K}$ is a set and $\models \subseteq \mathcal{K} \times \mathcal{L}$.

For a model $\mathcal{M} \in \mathcal{K}$ and formula $A \in \mathcal{L}$, we call A…

satisfied by $\mathcal{M}$, iff $\mathcal{M} \models A$
falsified by $\mathcal{M}$, iff $\mathcal{M} \not\models A$
satisfiable in $\mathcal{K}$, iff “$\exists \mathcal{M} \in \mathcal{K}. \mathcal{M} \models A$”
valid in $\mathcal{K}$ (written $\models \mathcal{M}$), iff “$\forall \mathcal{M} \in \mathcal{K}. \mathcal{M} \models \mathcal{A}$”
falsifiable in $\mathcal{K}$, iff “$\exists \mathcal{M} \in \mathcal{K}. \mathcal{M} \not\models A$”
unsatisfiable in $\mathcal{K}$, iff “$\forall \mathcal{M} \in \mathcal{K}. \mathcal{M} \not\models A$”

Def. 55 (Propositional logic, PL^0)**.** $\mathit{wff}_o(\mathcal{V}_o)$ is the set of “well-formed” (syntactically correct) formulas with variables $\mathcal{V}_o$. Its model $\langle \mathcal{D}_o, \mathcal{I} \rangle$ consists of a universe $\mathcal{D}_o = \{\mathsf{T}, \mathsf{F}\}$ and an interpretation $\mathcal{I}$, that assigns connectives values. The value function $\mathcal{I}_\phi : \mathit{wwf}_o(\mathcal{V}_o) \mapsto \mathcal{D}_o$, assigns values to formulas.

PL^0 is an example for a logical system $\langle \mathit{wwf}_o(\mathcal{V}_o), \mathcal{K}, \models \rangle$, where $\mathcal{K}$ is the set of variable assignments, and $\phi \models A \iff \mathcal{I}_\phi(A) = T$.

Def. 56 (First order logic, FOL, PL^1)**.** $\mathit{wff}_\iota(\Sigma_\iota)$ is the set of “well-formed” **terms** over a signature Σ_ι (function and skolem constants — individuals). $\mathit{wff}_o(\Sigma)$ is the set of well-formed **propositions** over a signature Σ (Σ_ι plus connectives and predicate constants — truth values).

Def. 57 (Natural deduction, $\mathcal{ND}^1$)**.** A “natural deduction” calculus for First order Logic:

$$\frac{\mathrm{A} \qquad \mathrm{B}}{A \wedge B} \wedge I \qquad \frac{}{A = A} = I \qquad \frac{\mathrm{A}}{\forall X.A} \forall I$$

$$\frac{A \wedge B}{A} \wedge E_l \qquad \frac{A = B \qquad C[A]_p}{[B/p](C)} = E$$

$$\frac{A \wedge B}{B} \wedge E_r \qquad \frac{[B/X](A)}{\exists X.A} \exists I \qquad \frac{\forall X.A}{[B/X](A)} \forall E$$

$$\frac{\genfrac{}{}{}{}{\mathrm{A}}{\mathrm{B}}}{A \Rightarrow B} \Rightarrow I$$

$$\frac{A \Rightarrow B \qquad \mathrm{A}}{\mathrm{B}} \Rightarrow E \qquad \frac{\exists X.A \qquad \genfrac{}{}{}{}{[c/X](A)}{\mathrm{B}}}{B} \exists E \qquad \frac{}{A \vee \neg A} \mathrm{TND}$$

Def. 58 (Analytical tableaux)**.** A tableau calculus for First order Logic: Every formula is labelled as either true (A^T) or false (A^F). To satisfy a formula A^α, it has to be shown that A has a truth value of α. This is done by branching out using the rules below. A branch is closed if it contains F, else open. A tableau is **closed** ($\neq$ saturated)if all of it’s branches are closed. A is valid iff there is a closed tableau with A^F at the root.

$$\frac{A \wedge B^\mathsf{T}}{A^\mathsf{T} \qquad B^\mathsf{T}} \mathcal{T}_0\wedge \qquad \frac{\forall X.A^\mathsf{T} \qquad C \in \mathrm{cwff}_\iota(\Sigma_\iota)}{[C/X](A)^\mathsf{T}} \mathcal{T}_1\forall$$

$$\frac{A \wedge B^\mathsf{F}}{A^\mathsf{F} \mid B^\mathsf{F}} \mathcal{T}_0\vee \qquad \frac{\forall X.A^\mathsf{F} \qquad c \in (\Sigma_0^{sk} \setminus \mathcal{H})}{[c/X](A)^\mathsf{F}} \mathcal{T}_1\exists$$

$$\frac{\neg A^\mathsf{T}}{A^\mathsf{F}} \mathcal{T}_0\overset{\mathsf{T}}{\neg} \qquad \frac{A \Rightarrow B^\mathsf{T}}{A^\mathsf{F} | B^\mathsf{T}} \mathcal{T}_0\overset{\mathsf{T}}{\Rightarrow}$$

$$\frac{\neg A^\mathsf{F}}{A^\mathsf{T}} \mathcal{T}_0\overset{\mathsf{F}}{\neg} \qquad \frac{A \Rightarrow B^\mathsf{F}}{A^\mathsf{T} \qquad B^\mathsf{F}} \mathcal{T}_0\overset{\mathsf{F}}{\Rightarrow}$$

$$\frac{\genfrac{}{}{0pt}{}{A^\alpha}{B^\beta} \qquad \alpha \neq \beta, \sigma(A) = \sigma(B)}{\mathsf{F} : \sigma} \mathcal{T}_1\mathsf{F} \qquad \frac{A^\mathsf{T} \qquad A \Rightarrow B^\mathsf{T}}{B^\mathsf{T}} \mathcal{T}_0\overset{\mathsf{T}}{\Rightarrow}'$$

Def. 59 (FOL Unification)**.** For two terms **A** and **B**, a unification is the problem of finding a substitution σ, s.t. $\sigma(A) = \sigma(B)$. A substitution σ is “more general” than θ, if there is a substitution φ, s.t. $\theta = \varphi \circ \sigma[W]$. There is no more general unifier than the “most general unifier” (mgu).

Def. 60 (Conjunctive Normal Form (CNF))**.** A formula is in conjunctive normal form if it is a conjunction of disjunction of literals.

For a FOL formula, it can be computed as follows:

1. Rewrite implications $p \Rightarrow q$ into the form $\neg p \vee q$.
2. Move negations inwards, so that only predicates are negated.
3. Rename variables bound by quantifiers making them unique.
4. Replace variables bound by existential quantifiers with new “skolem functions” $f \in \Sigma_k^{\mathrm{sk}}$ over all the free variables $X_1, \ldots, X_k$ in the quantified term:
$$\forall X.A \longrightarrow [f(X_1, \ldots, X^k)/X](A)$$
5. Distribute $\vee$ inwards over $\wedge$:
$$A \vee (B \wedge C) \longrightarrow (A \vee B) \wedge (A \vee C)$$

Def. 61 (FOL Resolution)**.** The resolution calculus for FOL operates on the CNF of a formula.

Like Tableau, it shows shows $\neg T \vdash \mathsf{F}$ to prove T. T is transformed into CNF and manipulated using the rules below. If the empty disjunction (“clause set”, $\Box$) is derived, T has been refuted.

$$\frac{P^\mathsf{T} \vee A \qquad P^\mathsf{F} \vee B}{A \vee B}$$

$$\frac{P^\mathsf{T} \vee A \qquad P^\mathsf{F} \vee B \qquad \sigma = \mathit{mgu}(P, Q)}{\sigma(A) \vee \sigma(B)}$$

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

Def. 62 (DPLL)**.** The DPLL procedure is an algorithm to find an interpretation satisfying a clause set.

4.1 Logic Programming

Def. 63 (Fact)**.** A term that is unconditionally true.

Def. 64 (Rule)**.** A term that is true if certain premises are true.

Def. 65 (Clause)**.** Facts and rules are both clauses.

Def. 66 (Horn clause)**.** A horn clause is a clause with at most one positive literal.

The Prolog rule $H :- B_1,...,B_n$ is the implication $B_1 \wedge \cdots \wedge B_n \Rightarrow H$ can be written as a horn clause $\neg B_1 \vee \cdots \vee \neg B_n \vee H$.

4.2 Knowledge Representation

Def. 67 (Semantic Network)**.** A directed graph representing knowledge. It consist of nodes representing objects/concepts, and edges representing relations between these, also called "links".

Def. 68 (Isa/Inst)**.** Links may be labelled with "isa" (*is a*) or "inst" (*instance*) to designate concept inclusion or concept membership respectively. They propagate propertied encoded by other links.

Def. 69 (TBox)**.** The sub-graph of a semantic network between concepts is called terminology, or TBox. It is spanned by isa links.

Def. 70 (ABox)**.** The sub-graph of a semantic network between objects is called assertions, or ABox. It is spanned by inst links and relations between objects.

Def. 71 (Semantic Web)**.** A collaborative movement led by the W3C promoting inclusion of semantic content into web pages. One example is RDF (Resource Description Framework), used for describing resources on the web.

Def. 72 (Ontology)**.** A logical system $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ and "concept axioms" about individuals, concept and relations. Semantic networks are ontologies.

Def. 73 (Description Logic)**.** A formal system for talking about sets and their relations. A description logic $\mathcal{D}$ has a $\mathcal{D}$-ontology, consisting of a TBox and ABox.

Def. 74 ($\mathcal{ALC}$)**.** A description logic more expressive than PL^0, but less complex than FOL. It relates "Concepts" (classes of objects, C) with "Roles" (binary relations, R). Its Syntax is as follows:

$$F_{\mathcal{ALC}} := \mathsf{C} \mid \top \mid \bot \mid \overline{F_{\mathcal{ALC}}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid F_{\mathcal{ALC}} \sqcap F_{\mathcal{ALC}} \mid \exists \mathsf{R}.F_{\mathcal{ALC}} \mid \forall \mathsf{R}.F_{\mathcal{ALC}}$$

where $\top$ and $\bot$ are the special concepts designating "all" and "none" respectively.

Def. 75 ($\mathcal{ALC}$ Tableau Calculus)**.** The Tableau calculus for $\mathcal{ALC}$:

$$\frac{x:c \qquad x:\bar{c}}{\bot}\,\mathcal{T}_\bot \qquad \frac{x:\phi \sqcup \psi}{x:\phi \mid x:\psi}\,\mathcal{T}_\sqcup \qquad \frac{x:\exists \mathsf{R}.\phi}{x\,\mathsf{R}\,y \qquad y:\phi}\,\mathcal{T}_\exists$$

$$\frac{x:\phi \sqcap \psi}{x:\phi \qquad x:\psi}\,\mathcal{T}_\sqcap \qquad \frac{x:\forall \mathsf{R}.\phi \qquad x\,\mathsf{R}\,y}{y:\phi}\,\mathcal{T}_\forall$$

5 Planning

Def. 76 (Planning language/task)**.** A logical description of the components of a search problem:

- a set of possible states
- an initial state I
- a goal condition G
- a set of actions A in terms of preconditions and effects.

constituting a planning task. This approach allows a solver to gain insight into the problem structure, resulting in a structured world representation.

Def. 77 (Satisficing planning)**.** A procedure that takes as input a planning problem and outputs a plan or "unsolvable", if no such plan exists.

Def. 78 (Optimal planning)**.** A procedure that takes as input a planning problem and outputs an optimal plan or "unsolvable", if no such plan exists.

Def. 79 (STRIPS planning task)**.** This is a encoding of a planning problem using a quadrupel $\Pi = \langle P,A,I,G \rangle$ where

- P is a finite set of facts
- A is a finite set of actions, each given as a triple of "preconditions", an "add list" and a "delete list".
- $I \subseteq P$ is the initial state
- $G \subseteq P$ is the goal.

Satisficing planning for STRIPS is called "PlanEx", and optimal planning is called "PlanLen", that tries to find the shortest plan.

A heuristic for Π with states S is function $h : S \mapsto \mathbb{N} \cup \infty$ so that $h(g) = 0$ for a goal state g. The perfect heuristic h^* assigns every $s \in S$ the length of the shortest path to g or ∞ if non-existent.

Def. 80 (Partial Order Planning)**.** A partially ordered plan is a collection of causal links $S \xrightarrow{p} T$ and temporal ordering $S \prec T$ where p is an affect of S and precondition of T. If the causal links and temporal ordering induce a partial ordering, it is called "consistent". If every precondition is achieved, it is called "complete".

Partial order planning is the process of computing a complete and consistent partially order plan.

Def. 81 (Delete relaxation)**.** This is a relaxation Π^+ of a given STRIPS task Π all actions have empty delete lists.

Def. 82 (Relaxed plan)**.** For a STRIPS task $\Pi = \langle P,A,I,G \rangle$ and state s, then $\langle P,A,\{s\},G \rangle^+$ is a relaxed plan for I/Π.

PlanEx for relaxed problems is called PlanEx^+.

Def. 83 (h^+-heuristic)**.** For a planning task $\Pi = \langle P,A,I,G \rangle$, the optimal heuristic calculates the length of the optimal relaxed plan for s or ∞ if no plan exists. h^+ is admissible. The heuristic h^{FF} approximates h^+, since calculating h^+ is in NP.

Def. 84 (Real World Planning)**.** When planning in real-world situations, the agent the task environment is partially observable and non-deterministic, which invalidates the previous assumptions. Variations on planning try to overcome this:

Conditional Extend the possible action in plans by *conditional steps* that execute sub-plans conditionally.

Conformant Tries to find a plan without sensing, instead relying on the its (fully observable) belief states.

Contingent Generate a plan with conditional branching based on percepts.

Def. 85 (Online Search)**.** Interleaving of search and actions, basing action on incoming perceptions. A planner P can be turned into an online problem server by adding an action Replan(g), that re-starts P in the current state with goal g.