ARTIFICIAL INTELLIGENCE

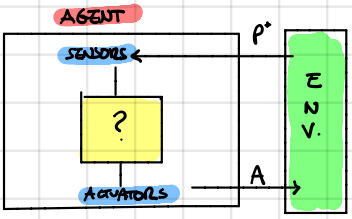
Edoardo Puglisi

CTN 0452-9

Artificial Intelligence is a bunch of techniques that includes machine learning too
There are different definitions of AI depending on the PoV of study

• let machine think like humans	• let machine think
• Act like humans	• Act
↑human-like model↑	↑rational model↑

Rational Agents



Agent is an entity that perceives and act (like a computer program)
Rational agents act to achieve the best goal possible → we try to find the best rational agent

$f: P^* \rightarrow A$ (percept histories → actions; f: agent function)

ie. vacuum-cleaner: P^* = [pos, dirty] (location and contents) → A = [left, right ...] (movement direction)

Rationality means act to maximize the expected value given a performance measure → learning, exploration

PEAS = performance measure, environment, actuators, sensors.

ENVIRONMENT

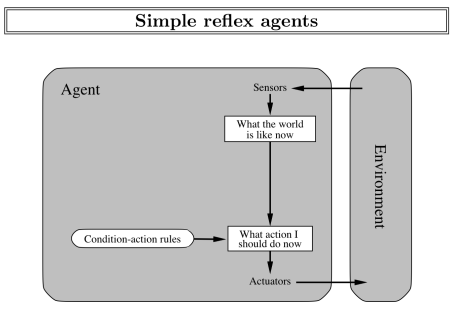
- Observable: fully, partially or not
- Deterministic / not or Stochastic: deterministic guarantee the actions
- Episodic or Sequential: if needs (or not) keep track of events
- Static or (Semi)Dynamic
- Discrete or Continuous
- Single or Multi agent

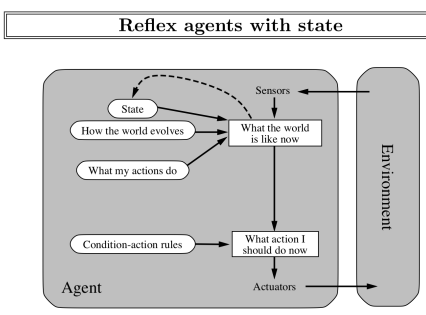
→ influences agent design

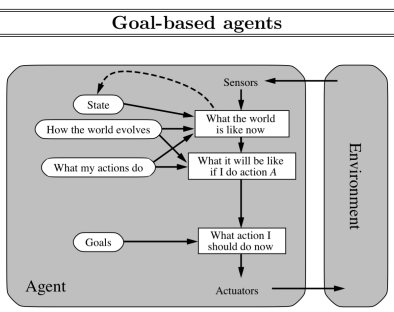
AGENTS

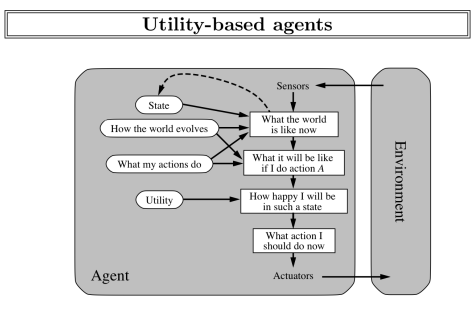
- Simple Reflex: given a percept and a condition-action rule it does an action
- Reflex with state: before applying condition-action rule it "thinks" how the environment evolves and it is in that time
- Goal-based: "what will happen if i do this action?"
- Utility-based: "After that action, the next state will be fine?"

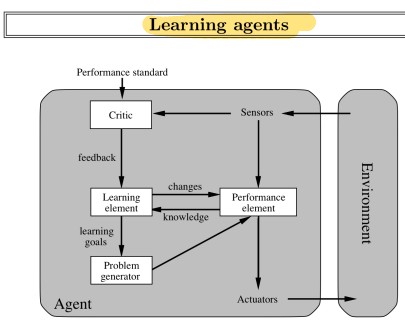
Previous agents can be evolved in Learning Agents











THE PROBLEM SOLVING AGENT

- Problem formulation: state definition, initial state and operators (actions) → path = sequence of operators
- Goal formulation: set of states or goal test
- Search: process to reach the solution
- Execution: when the agent reaches the solution

Search strategies are evaluated along completeness, time complexity, space complexity and optimality

Completeness evaluate if it always find a solution, optimality if the one it finds is the least-cost solution

b = maximum branching factor of the tree AKA # children nodes

d = depth of the least-cost solution

m = maximum depth of the state space (may be ∞)

EXPANSION: the process that builds successor nodes by applying the operators.

A state is a (representation of) a physical configuration while a node is a data structure costituing part of a search tree (works on graphs too!)

UNIFORMED SEARCH

BFS: FIFO, parent node before its children.

Complete: if b finite, Time: $O(b^{d+1})$, Space: $O(b^{d+1})$ cause keeps every node in memory, Optimal: in general not, yes if cost per step = 1

UNIFORM-COST: expand least-cost unexpanded node (equivalent to bfs if all equal step costs)

Complete: if step cost $\geq \epsilon$, Time: $O(b^{\lceil 1+C^*/\epsilon \rceil})$ where C^* is the cost of optimal solution, Space: $O(b^{\lceil C^*/\epsilon \rceil})$, Optimal: yes

DFS: LIFO, children (successors) before parents

Complete: no cause fails in infinite-depth spaces, Time: $O(b^m)$ terrible if $m \gg d$ otherwise may be better than bfs, Space: $O(bm)$ linear, Optimal: no

↳ Depth limited Search: dfs within a determinated depth l

Iterative Deeping: DLS repeated multiple times untill d (= minimal depth in which the goal is) is reached.

Complete: yes, Time: $O(b^d)$, Space: $O(bd)$, Optimal: if step cost = 1

SEARCH DIRECTION

FORWARD: data driven ↝ BIDIRECTIONAL: mixed, if reversible → start from initial state in forward and from goal backward and try to

BACKWARD: goal driven ↗ meet in "the middle".

Time: $O(b^{d/2})$ Space: $O(b^{d/2})$ Completeness and Optimality depend on tecniques used

Problems: ① check the next nodes $O(b^{d/2})$ ② goal state implicitly known.

Note: avoid cyclic paths and generation of already generated paths.

Summary of algorithms

Crit	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iter Deep	Bid (if app)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optim	Yes	Yes	No	No	Yes	Yes
Compl	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

BEST-FIRST SEARCH: use an evaluation function for each node to estimate "desirability" and expand the most desirable one.

$h(n)$ = heuristic function that estimate the cost from n to nearest goal → Knowing ideal $h(n)$ mean knowing the solution → not optimal

Time: $O(b^m)$, Space: $O(b^m)$ keeps all nodes in memory, Optimal: No, Complete: No, can get stuck in loops; yes in finite space with repeated-state checking

A* SEARCH

$f(n) = g(n) + h(n)$ with $g(n)$ = cost so far to reach n : avoid expanding paths that are already expensive → optimal

It never overestimates the costs. Too much memory! A* uses **admissible heuristics** = heuristics which never overestimate the costs.

For some admissible heuristics, f can decrease in a path → bad in graphs. Heuristic is **consistent** if $h(n) \leq c(n,a,n') + h(n')$

This guarantees that f is not decreasing: $f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n)$

Usually if admissible also consistent

Complete: yes (unless infinite node with $f \leq f(G)$), Time: exponential, Space: all nodes in memory, Optimal: Yes (f_{i+1} after f_i always)

How to obtain heuristics?

1. From Relaxed Problem: heuristics that can solve a problem with less constraints than the original one → of course admissible
2. From Pattern Database: find heuristics from every subproblem instance solution costs
3. From Experience: use learning algorithm to construct an heuristic solving the problem multiple times

DOMINANCE

If $h_2(n) > h_1(n)$ for all n (both admissible) then h_2 dominates h_1 and is better for search (expands less nodes)

Nodes generated by A* search are always less than other search types

MISSIONARIES & CANNIBALES PROBLEM

We must specify all possible action such as the "carry" from a river side to the other

eg carry MM = move 2 missionaries

M = move 1 missionary

MC = move 1 missionary and 1 cannibale

C = move 1 cannibale

CC = move 2 cannibales

} ⇒ carry(nM, nC)

State(L,M):

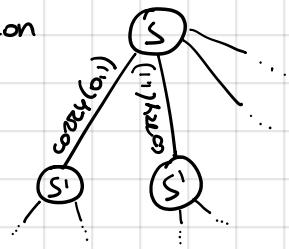
M	C	
3	3	Left
0	0	Right

action must have conditions! CARRY(nM,nC)

- → $1 \leq nM + nC \leq 2$ ⟹ $nM \leq S(L,M)$, $nC \leq S(L,C)$
- → $S(L,M) - nM \geq S(L,C) - nC$
- → $S(R,M) + nM \geq S(L,C) + nC$
- → boat = 0

→ simmetrically (almost) we do "carryback" action

↳ Solution: carry(0,2), carryb(0,1), carry(0,2), carryb(0,1), carry(2,0), carryb(1,1), carry(2,0), carryb(0,1), carry(0,2), carryb(0,1), carry(0,2)



$S'(L,M) = S(L,M) - nM$ & $S'(R,M) = S(R,M) - nM$ boat = 1

$S'(L,C) = S(L,C) - nC$ $S'(R,C) = S(R,C) - nC$

S' is the state after action!

Effective Branching Factor: the closest to 1 the better.

We can "combine" not dominating heuristic to find a common dominating heuristic $h(n) = \max(h_1(n), h_2(n) \dots h_n(n))$

IDA*: Iterative Deeping A*, reduce space usage → linear memory

RECURSIVE BFS: when it finds a greater value of f in one branch, removes others from (linear) memory $O(bd)$

SMA*: simplified Memory-bounded A*. can use all avible memory for the search (opposite of IDA* or RBFS)

like A* until we have memory, then when new node is generated forgets the node with highest f and keep its cost f in the parent. Forgotten node re-generated when all other paths are discarted

Typical best heuristic.

Ex

2 robots M and T. M grab objects from a bin and put them in a counter. The bin is unlimited. T grab objects from counter to assembly line. 3 types of objects A, B, C. Counter can store max 4 objects, no more than 1 B and no more than 1 C. M can takes 2 or 3 objects in a single grab and if it grabs 3, two of them must be equal. When A, B, C are in counter T grabs them and puts them in assembly line. No conflicts between M and T. Stop after 2 completed sets A, B, C.

① State Space

State S <[ ], n> : list of objects on counter + n = completed sets

Initial state <[ ], 0>

Final state <[*], 2>

② Operators

Note: a, b = *

Tproceed(){
- A, B, C in [ ]

}

⇓

<[x, A, B, C], n> → <[x], n+1>

<[A, B, C], n> → <[ ], n+1>

Mmove2(u, v){
- [ ] ≤ 2
- #B ≤ 1
- #C ≤ 1

}

⇓

<[a, b], n> → <[a, b, u, v], n>

<[a], n> → <[a, u, v], n>

<[ ], n> → <[u, v], n>

Mmove3(u, v, z){
- [ ] ≤ 1
- #B ≤ 1
- #C ≤ 1
- v = z = A

}

⇓

<[a], n> → <[a, u, v, z], n>

<[ ], n> → <[u, v, z], n>

③ Search Tree (only solutions)

<[ ], 0> → Mmove2(B,C) → <[B,C], 0> → Mmove2(A,A) → <[A,A,B,C], 0> → Tproceed() → <[A], 1> → Mmove2(B,C) → <[A,B,C], 1> → Tproceed() → <[ ], 2>

<[ ], 0> → Mmove2(AB) → <[A,B], 0> → Mmove2(A,C) → <[A,A,B,C], 0> → Tproceed() → <[A], 1> → Mmove2(B,C) → <[A,B,C], 1> → Tproceed() → <[ ], 2>

<[ ], 0> → Mmove2(A,C) → <[A,C], 0> → Mmove2(A,B) → <[A,A,B,C], 0> → Tproceed() → <[A], 1> → Mmove2(B,C) → <[A,B,C], 1> → Tproceed() → <[ ], 2>

<[ ], 0> → Mmove2(A,A) → <[A,A], 0> → Mmove2(B,C) → <[A,A,B,C], 0> → Tproceed() → <[A], 1> → Mmove2(B,C) → <[A,B,C], 1> → Tproceed() → <[ ], 2>

e.g.

<[ ], 0> → Mmove2(A,A) → <[A,A], 0> → Mmove2(A,A) → <[A,A,A,A], 0> Failure!

BEYOND CLASSICAL SEARCH

LOCAL SEARCH

Instead of transition between state we "guess" how far we are from solution. In this kind of algorithms path is irrelevant = cheap in memory.
The "goodness" of state decides next states → fitting function

Hill-climbing (HC): Starting from an arbitrary solution attempts to find a better solution with incremental change. Continue until no more better solution can be found. Not always optimal (local maximum, not global)
If next step doesn't change situation in term of fairness from solution → side / random moves, first choice = random generation of successor, random restart = try again!

Simulated Annealing: allow "bad" choices with decreasing probability. Bad choices not excluded a priori.

BEAM SEARCH

Run multiple local searches in parallel: at each step use the best k successors from all "beam" → not properly parallel
Too quick convergence in some region → add randomness or weighting!

GENETIC ALGORITHMS

At every step the genetic algorithm select individual solutions at random from current "population" to be parents and use them to produce children for next gen. Over generation, population evolves toward optimal solution

CONSTRAINT SATISFACTION PROBLEM

Create standard representation of the problem with restriction on value of variables. → solution satisfy all constraints.
CSP = {X, D, C} = {variables, domains, constraints}
Discrete Variables: finite domain e.g. Boolean, most common
Infinite variables, Continous Variables ...
Unary constraints: involves a single variable → Binary, higher-order depending on variable number.
Preferences (soft constraints): constrained optimization problem (COP) e.g. red better tho green → cost representation
Choose variable value if doesn't conflict with (current assignment) constraints: all solutions are at depth n = #variables → use DFS
$b = (n-\ell)d$ at depth ℓ → $n!d^n$ leaves.

- Variable assignment are commutative: only need to consider assignment to a single variable at each node
- DFS for CSP with single-variable assignment is called Backtracking search: go back if get stucked!

To choose target variable: Minimum Remaining Values (MRV), choose variables with fewest legal value.; or
Degree Heuristic: choose the one with the most constraints on remaining variables.
To choose value: least Constraining Value: given variable choose least constraining value, the one that rules out the fewest values in the remaining variables.
Improve Backtracking keeping track of sets of conflicting variables → Intelligent backtracking.
FORWARD CHECKING: Keep track of legal value for unassigned variables. Stop search when a variable has no legal value.
ARC CONSISTENCY: arcs between variable's legal value sets. $X \to Y$ consistent iff $\forall x \in X$ values there is some allowed y
If X lose a value we need to check all variable which have constraints with X. Detect failure before forward checking. Can be run as preprocessor or after each assignment. K-consistency = constraints that involves k variables.
The main difference between Forward Checking and Arc Consistency is that the first checks a single unassigned variable at time while the second also check pairs of unassigned variables for mutual consistency
Forward checking doesn't provide early detection for all failures.

Ex. Put following numbers in ascending order: 4, 1, 3, 2

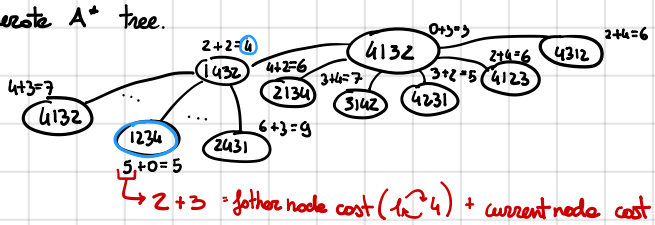
- At each step, while performing reordering, its possible to swap number in pos. i with number in pos J
- Cost for each move $|J-i|+1$
- heuristic $h(n)$ = # missplaced number

① $h(n)$ is admissible?

If only one number misplaced (impossible in real), $h(1)=1$ while the real cost (move two numbers) is at least 2

$h(n)$ doesnt overestimates the coast of reaching the goal so is admissible

② Generate A* tree.



↳ 2+3 = father node cost (1→4) + current node cost (2→4)

Ex Given h_1 and h_2 admissible and neither one dominating the othe:

① How to construct h_3 from h_1 and h_2 which dominates both and is admissible?

$h_3 = h_1 + h_2$ may Overestimate ⟶ $\forall n \max\{h_1, h_2\} = h_3$

② h_3 is preferred cause exponds less nodes

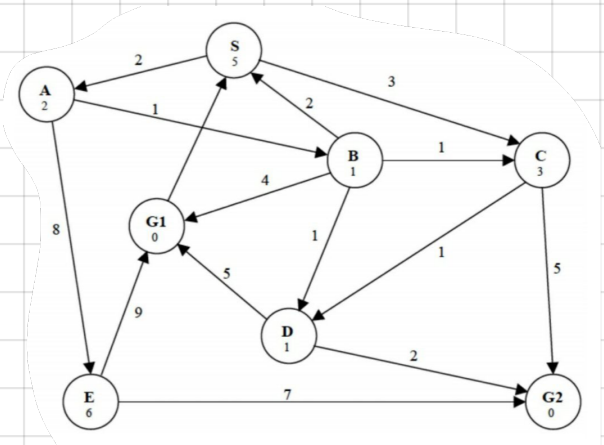
Ex Consider the search space below with S= stard node, G_1 and G_2 goal nodes. Arc labels are cost of traversing and number insides nodes are estimated cost to goal

① Draw A* tree

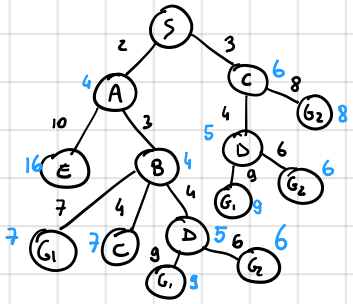
② List - inorder - all states popped of open list

= mark nodes with incremental number as they are exponded.

Alphabetic order when equals.



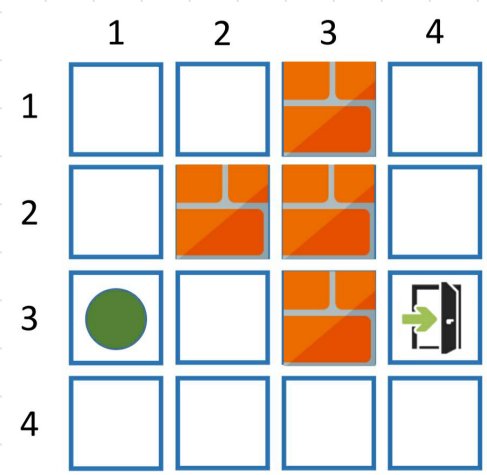
①



② S-A-B-D-C-D...

Ex Agent ● hasto reach the door avoid Walls. Cost to go up or right is 1, down or diagonal is 2. Cant go left.

Initial state is (3,1), goal state (3,4)



① Characterize state space

② Specify operators

③ Is Manhatton Distance an appropriate heuristic? Is admissible?

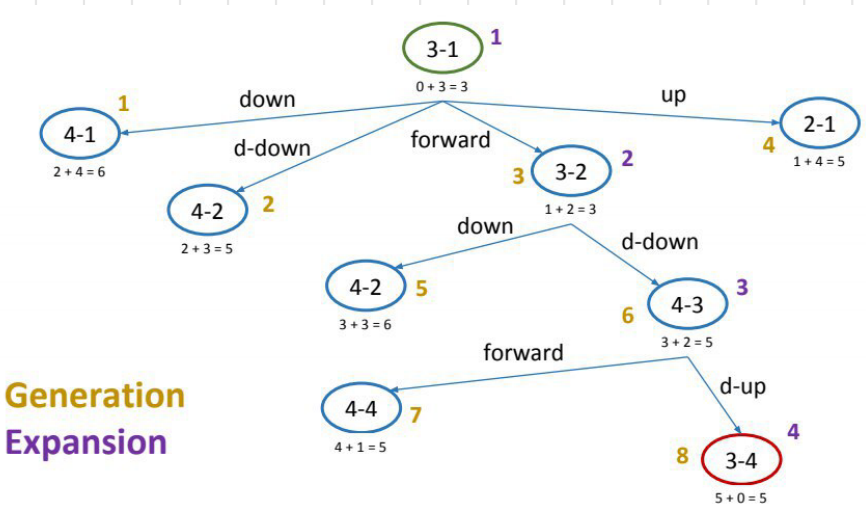
④ List nodes in expansion order

⑤ Use A* with above heuristic function to find a solution. List all generated nodes with the order of generation and value of g, h and f. When more nodes have same min f value expand the most recently generated.

③ Yes ×2 ④ & ⑤

① Almost given by text: (1,3), Goal (3,4)
I.S. (3,1)

② ↑ → ↓ ↘



Ex. Environment: 2×2 grid to be covered by two kind of devices (A, B) initially empty.
Devices can be placed only in empty cells and can't have some type of devices in some row/column.
"A" can be placed on top of "B". Goal: cover all cells.

- State Space: $S: D \times D \times D \times D, \; D \in \{A, B, e\}$

h_0	h_1
h_2	h_3

$\{h_0, h_1, h_2, h_3\} \in S$

- Initial State: $[e, e, e, e]$
- Goal State: $[A, B, B, A]$ (or $[B, A, A, B]$)
- Operators:
  - ① PlaceBh_0 → PRE: $h_0 = e \quad h_1 \neq B \quad h_2 \neq B$
    EFF: $S' = [B, *, *, *]$
  - ② PlaceBh_1 → PRE: $h_1 = e \quad h_0 \neq B \quad h_3 \neq B$
    EFF: $S' = [*, B, *, *]$
  - ③ PlaceBh_2 → ...
  - ④ PlaceBh_3 → ...
  - ⑤ PlaceAh_0 → PRE: $h_0 \neq A \quad h_1 \neq A \quad h_2 \neq A$ Note NO $h_0 = e$ but $h_0 \neq A$
    EFF: $S' = [A, *, *, *]$
  - ⑥⑦⑧ ...

PLANNING

General approach:
- open up action and goal representation to allow selection
- divide & conquer
- relax requirement for sequential solutions

A plan is a sequence of actions from initial state to goal

Environment:
- fully observable
- deterministic
- static
- discrete (and finite)

STRIPS

Stanford language for action description: $P(x_1 \dots x_n)$ or $\neg P(x_1 \dots x_n)$ $\quad n \geq 0$ $\quad x_i$ = variables or constants

$P(x_1 \dots x_n)$ is called literals

States are set of instantiated literals (properties) that must be satisfied (True v False)

close-world assumption: state with only positive literals

function free: domain assumed finite

ACTION SCHEMA:

Precondition: set of positive literals

Effect: set of literals

Effects specify change on literals list → ADD and DELETE list

e.g. Action: Buy(x)
Prec.: At(P), Sells(P,x)
Effect: Have(x)

Action a is applicable in a state s iff $s \models \text{Preconditions}(a)$

$\text{Result}(s,a) = (s - \text{Del}(a)) \cup \text{ADD}(a)$ ← computing the state resulting from action execution.

Note: persistence assumption, if not changed / deleted → persists

PLANNING PROBLEM

- Specification of actions
- Initial state $I \subseteq S$
- Goal state/s $G \subseteq S$

Plansat: find if exists or not a sequence of actions to reach goal state.

Bounded Plansat: ↑ with length $\leq k$ → allow to determinate optimal plans.

PROGRESSION

Search forward, apply actions until goal or all states explored.

REGRESSION

Backward search. Strips operators invertible, goal spec. represents set of states. Focuses on actions that make some of goal literals true.

$\text{Result}^{-1}(s,a) = (s - \text{EFF}(a)) \cup \text{PREC}(a)$ ← regression step.

Actions in this case can be relevant if add a goal conjunct or consistent if don't undo any other goal conjunct.

HEURISTIC

Regression >> progression but both unsatisfactory. Possible solutions:
- Relaxing the problem: remove preconditions → set covers, remove negated effects → empty-delete-list
- Abstracting set of states: subgoal independence assumption → cost of solving a conjunction estimated based on cost of solving each of the conjuncts (MAX or SUM)

Difference between Search and Planning:

Search: given a space of possibilities (huge) we want to locate one (goal) inspecting a subset using all sort of algorithms and techniques. Particular action in particular situation gets to a second particular situation. Almost any problem can be posed as search problem.

Planning is a particular kind of search through space of action sequences for a plan that satisfies some criteria. Given a description of starting state and goal state and set of possible actions find the sequence of actions that get you from start to goal state. In planning we can do subgoaling (divide & conquer).

PDDL

Description language derived by Action Description Language (ADL) on automated planning and scheduling system.

With PDDL we can use disjunctions and quantifiers in preconditions and goals, quantified and conditional effects.

Effect formula may contain:

- conditional effect – (when CONDITION_FORMULA EFFECT_FORMULA)
- universally quantified formula – (forall (?V1 ?V2) EFFECT_FORMULA)

STRIPS Language	ADL Language Note ↓
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

ACTION TEMPLATE

```
(:action drive-truck

    :parameters (?truck  ?loc-from ?loc-to ?city )

    :precondition (and (at ?truck ?loc-from) (in-city ?loc-from ?city) (in-city
    ?loc-to ?city))

    :effect    (and (at ?truck ?loc-to) (not (at ?truck ?loc-from))
                    (forall (?x - obj)
                            (when (and (in ?x ?truck))
                                  (and (not (at ?x ?loc-from))
                                       (at ?x ?loc-to))
                            )
                    ))
)
```

PDDL TEMPLATE (= STRIPS ENHANCED, use this)

Domain file:

```
(define (domain <domain name>)
        <PDDL predicates>
        <PDDL actions>
)
```

```
Action(ActionName,
   PARAM: var1 ... varN
   PRECOND:  list of preconditions
   EFFECT:   list of effects
)
```

Problem file:

```
(define (problem <problem name>)
( :domain <domain name> )
        <PDDL objects>
        <PDDL initial state>
        <PDDL goal specification>
)
```

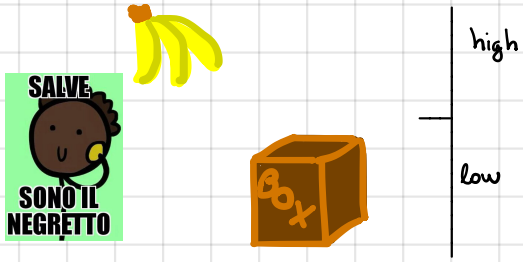
Ex Monkey & Bonanas:

Objects: monkey, bonono, box

Predicates: on, box, holding, on_ceiling etc.

Initial State: as figure Goal State: monkey holding bonono

Actions/Operators: go, push, climb, grab.



Position: A B C

① Model the plan in STRIPS (old STRIPS)

② Progression search tree

③ Regression search tree

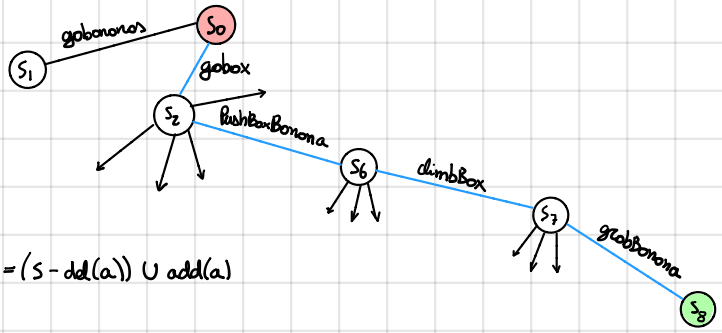
① INITIAL STATE: At(A), level(low), boxAt(C), bononoAt(B)

GOAL STATE: Have(bonono)

ACTIONS:

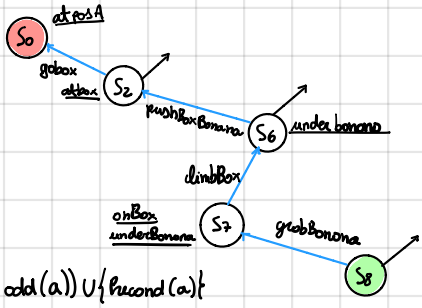
- goBox(x):
  - PRECOND: At(x), level(low)
  - EFFECT: At(c), ¬At(x)
- climbBox(x):
  - PRECOND: At(x), level(low), boxAt(x)
  - EFFECT: level(high), ¬level(low)
- PushBoxBonono(x)
  - PRECOND: At(x), boxAt(x), level(low)
  - EFFECT: At(B), boxAt(B), ¬At(x), ¬boxAt(x)
- GrabBonono(x):
  - PRECOND: At(x) bononoAt(x) level(high)
  - EFFECT: Have(bonono)

②



$S' = (S - del(a)) \cup add(a)$

③



$g' = (g - add(a)) \cup \{precond(a)\}$

PDDL example:

current state $\{ car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Rome) \wedge At(c_2, Berlin)$

action: DRIVE $(c_1, Rome, Berlin)$

PRECOND: $At(c_1, Rome) \wedge Car(c_1) \wedge city(Rome) \wedge city(Berlin)$

EFFECT: $\neg At(c_1, Rome) \wedge At(c_1, Berlin)$

$S' = Result(s, a) = (S - DEL(a)) \cup ADD(a)$ → $DEL(a) = \neg At(c_1, Rome)$ $ADD(a) = At(c_1, Berlin)$ → $S' = car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Berlin) \wedge At(c_2, Berlin)$

current state $\{ car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Rome) \wedge At(c_2, Berlin)$

action: DRIVE $(c_1, Rome, Berlin)$

PRECOND: $At(?car, ?from) \wedge Car(?car) \wedge city(?from) \wedge city(?to)$

EFFECT: $\neg At(?car, ?from) \wedge At(?car, ?to)$

$car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Berlin) \wedge At(c_2, Berlin)$ ✓

$car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Rome) \wedge At(c_2, Rome)$ ✓

} i.e. states in which ?car = c_1 or c_2

$car(c_1) \wedge car(c_2) \wedge city(Rome) \wedge city(Berlin) \wedge At(c_1, Berlin) \wedge At(c_2, Rome)$ ×

} wrong cause both cars change location when action only change one

Monkey & Bononos PDDL

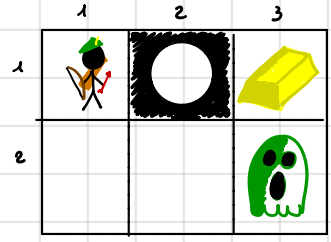
```
(:action go
:parameters (?who ?from ?to)
:precondition (the monkey at ?from )
:effect (the monkey is at ?to and not at ?from))
```

```
(:action push
:parameters (?who ?what ?from ?to)
:precondition (the monkey at ?from, ?what is
at ?from)
:effect (the monkey is at ?to and not at ?from,
?what is at ?to and not at ?from))
```

```
(:action climb
:parameters (?who ?what ?under-what)
:precondition (the monkey at ? under-what,
?what is at ? under-what)
:effect (the monkey is on ?what) )
```

```
(:action grab
:parameters (?who ?what ?under-what)
:precondition (the monkey at ? under-what,
the monkey is on ?what)
:effect (the monkey is holding ?under-what) )
```

WUMPUS WORLD



Agent can move on squares and collect gold if in the same square. Pit (1,2) is not walkable. Agent can shoots to wumpus if in adjacent squares. Cannot move on same square of wumpus if its not dead

- Objects: 6 squares, 1 gold, 1 agent, 1 arrow, 1 wumpus
- Predicates: gold(x), agent(x), wumpus(x), dead(x), at(x,y) etc.
- Initial State ↑
- Goal: collect gold and return to initial position
- Actions: move, collect, shoot

DOMAIN FILE:

```
(define (domain wumpus-c)
(:predicates (at ?what ?square) (adj ?square-1 ?square-2)
(pit ?square) (wumpus-in ?square) (have ?who ?what)
(is-agent ?who) (is-wumpus ?who) (is-gold ?what) (is-arrow ?what) (dead ?who) )

(:action move
:parameters (?who ?from ?to)
:precondition (and (is-agent ?who) (at ?who ?from) (adj ?from ?to) (not (pit ?to)) (not (wumpus-
in ?to)))
:effect (and (not (at ?who ?from)) (at ?who ?to)))

(:action take
:parameters (?who ?what ?where)
:precondition (and (is-agent ?who) (at ?who ?where) (at ?what ?where))
:effect (and (have ?who ?what) (not (at ?what ?where))))

(:action shoot
:parameters (?who ?where ?with-what ?victim ?where-victim)
:precondition (and (is-agent ?who) (have ?who ?with-what) (is-arrow ?with-what) (at ?who
?where) (is-wumpus ?victim) (at ?victim ?where-victim) (adj ?where ?where-victim))
:effect (and (dead ?victim) (not (wumpus-in ?where-victim)) (not (have ?who ?with-what)))))
```

PROBLEM FILE:

```
(define (problem wumpus-c-1)
    (:domain wumpus-c)
    (:objects s-1-1 s-1-2 s-1-3 s-2-1 s-2-2 s-2-3 gold-1 arrow-1 agent
    wumpus)

    (:init (adj s-1-1 s-1-2) (adj s-1-2 s-1-1) (adj s-1-2 s-1-3) (adj s-1-3 s-
    1-2) (adj s-2-1 s-2-2) (adj s-2-2 s-2-1) (adj s-2-2 s-2-3) (adj s-2-3 s-
    2-2) (adj s-1-1 s-2-1) (adj s-2-1 s-1-1) (adj s-1-2 s-2-2) (adj s-2-2 s-
    1-2) (adj s-1-3 s-2-3) (adj s-2-3 s-1-3) (pit s-1-2) (is-gold gold-1) (at
    gold-1 s-1-3) (is-agent agent) (at agent s-1-1) (is-arrow arrow-1)
    (have agent arrow-1) (is-wumpus wumpus) (at wumpus s-2-3)
    (wumpus-in s-2-3))

    (:goal (and (have agent gold-1) (at agent s-1-1)))
)
```

PARTIAL-ORDER PLANNING (POP)

LEAST COMMITMENT: partial ordering and not fully instantiated plan. In plan space node = partial plan = transformation from initial state to final state. Different from state space where node = state in the world

An openPrecondition is a precondition of an action not yet causally linked.

A causal link is a link from one action to precondition of an other

In partial-order planning we maintains a partial ordering between actions and commits ordering only when forced: eg. actions: leftShoeON and RightShoeON can be done in any order (who cares?)

Refinements:

- add link from existing action to an open condition
- add an action to fullfill an open condition
- add order to avoid conflicts.

We gradually complete the plan refining the starting plan with only start and finish action (ordered)

clobber: an action that destroy the condition achieved by a causal link

conflict: causal link $A \xrightarrow{p} B$ and action C have a conflict between them if C has effect $\neg p$

PROMOTION and DEMOTION mean put an action after or before an other.

General heuristics for POP can be: number of open condition, most constrained variable (open precondition that are satisfied in fewest way) or planning graph (special data structure)

Cons: infinite search space, no representation of states, planning is complex and diffiult to devise heuristics.

HIERARCHICAL TASK NETWORK PLANNING (HTN)

In some problem, planning at primitive actions level can be inefficient. → hierarchical decomposition

Operators of classic planning preserved + high level actions (HLA) that cant be executed by agent but have one or more refinements into a sequence of actions (high/low level actions) i.e an action that contains more action of different levels.

HLA containing only primitive actions is called implementation. HL plan reaches goal if at least one of its implementations does.

- Searching for primitive solutions: from initial abstract specification repeatedly choose an HLA in current plan and replace it with one of its refinements until goal reached, goal check performed when plan is refined to a sequence of primitives
  More efficient if have few but long refinements.
- Searching for abstract plans: choose refinements only if reach goal before expanding them first. → define precondition-effect descriptions for HLA such as primitive actions.
  ↳ conservative approach: include ONLY positive effects that are achieved for every implementation and every negative effects of any impl.

REACH(s,h) of HLA from states is the set of states reachable by any of HLA's implementations; or

let HLA to control the value of a variable:

\+ add symbol − delete symbol ± add or delete ~ possibly symbol

eg. Action (Bus (Home, Airport),
Prec: at(Home),
Eff: at(Airport) ∧ state(Tired)

Action (Taxi (Home, Airport),
Prec: at(Home),
Eff: at(Airport) ∧ -have(money)

⇒

Action (go (Home, Airport),
Prec: at(Home),
Eff: at(Airport) ∧ -~have(money) +~state(Tired)

We need some approximation due to infinitely refinements:

Optimistic description overstate the reachable set REACH+(s,h) → Posible effect as definite

Pessimistic description understate the reachable set REACH-(s,h) → Posible effects do not happen

If the optimistic reachable set does not intersect goal then the plan doesnt work

If the pessimistic reachable set intersect the goal then the plan works

If the optimistic reachable set does intersect goal then the plan might work

If the pessimistic reachable set doesnt intersect the goal then the plan might work

Ex Environment of N×N grid. The agent can move in any of the 8 adjacent cells at cost = 1 per move. Starts from lower leftmost cell and must reach upper rightmost one. Can't go on cells with obstacles.

Formalize the problem of finding a sequence of agent moves to reach the goal.

DOMAIN

```
(define (domain basic-grid)
    (:requirements :strips)
• (:predicates (at ?x) (adj ?x ?y) (obs ?x))
• (:action move-agent
      :parameters (?from ?to)
      :preconditions (and (at ?from) (adj ?from ?to) (not (obs (?to))))
      :effect (and (at ?to) (not (at ?from))))
```

PROBLEM

```
(define (problem basic-grid)
  (:domain basic-grid)
  (:objects cell11, cell12 ...)
  (:init (adj cell11 cell12) (adj cell11 cell21) ...
         (obs cellxy) (obs cellyx) ...   position of obstacles
         (at cellN1))   starting position of agent
  (:goal (at cell1N))   final position
```

NON CLASSICAL PLANNING

In real problems agent have to deal with incomplete and incorrect information.

Bounded indeterminacy: agent can deal with all possible cases, otherwise Unbounded: agent has to be ready to revise the plan

Information can be incomplete or incorrect

Due to non determinism and/or partial observability → Search in space of belief states = set of possible actual states → Conformant / Sensorless planning: devise a plan that works regardless of state (not always possible)

e.g Replace tire even if we can't guarantee that its flat.

Conditional / contingency planning: Plan to obtain information (observation actions) and subplan for each contingency (Expensive because it plans for many unlikely cases.)

Monitoring / Replanning: check progress during execution, replan if necessary (Unanticipated outcomes may lead to failure)

Continous planning: the goal can change during execution, at each step adjust plan on the updated plan.

BELIEF STATES

Represent all possible world states (set of ordinary states)

Actions move the agent from a belief state to an other

Action in belief sets can be deterministic or non deterministic, in the second case the environment can be fully or partially observable: actions can be seen as disjunctive effects and may need percepts to determine the value of a property. → Each non deterministic action is followed by a sensing action which eliminates the uncertainty caused by previous execution.

LANGUAGE OF LOGIC

Symbols can be used to represent structures in computers, symbolic systems where symbols are created modified, copied etc.

A good representation is made by a set of symbols, stored in computer with algorithms that let us use them to solve problems

Syntax: how to write language

Semantics: the meaning of the program. e.g. how symbols connect to the world

Reasoning: given a knowledge be we can make inference on it (query)

KNOWLEDGE BASE

Set of sentences in formal language

An agent can be built with declarative approach: TELL it what it need to know and ASK it what to do (from knowledge base). It can be viewed at KNOWLEDGE LEVEL (what it know) or IMPLEMENTATION LEVEL (which data structure in KB and algorithms manimulate it)

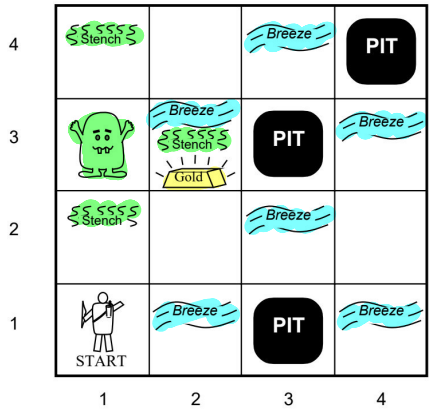
Example Wumpus

Some old wumpus environment + squares near wumpus are smelly + squares near pits are breezy

Performance measure: gold +1000, death -1000, -1 per step, -10 for using arrow

Sensors: stench, breeze etc

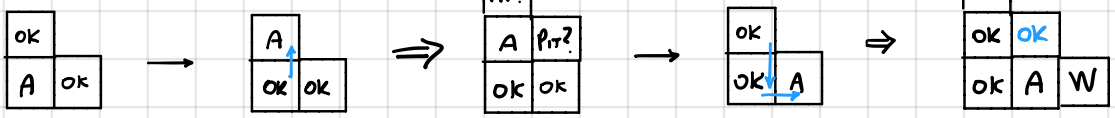
Actuators: Turn left, turn right, forward, grab, release, shoot



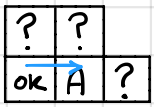
Not Observable cause have only local perception, deterministic, Not Episodic cause it follow the actions, static cause wumpus and pit don't move, Discrete

Single-Agent

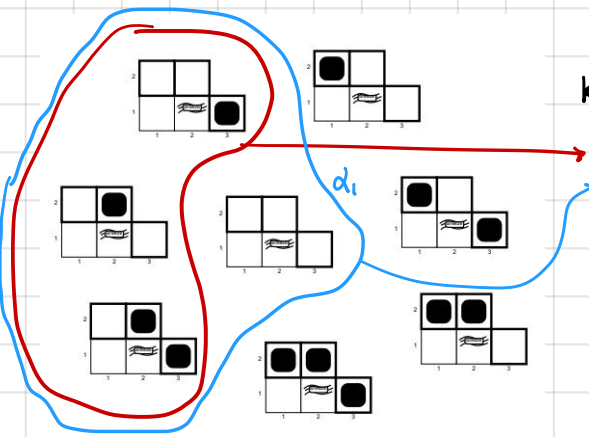
Eg.



Sensors [//] | Sensors [Breeze] | Pit "caused" by breeze | Sensors [Stench] | Wumpus can be only there and due to the fact we don't have breeze we can set cell above as safe

Consider the case: 

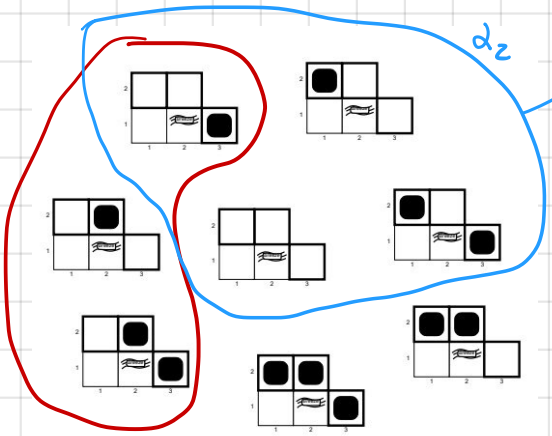
agent feels breeze → 8 possible cases (pit positions)



KB = wumpus-world rules + observation B[1,1] and B[1,2]

MODELS = possible worlds that are "coherent with KB"

Inference by model checking: KB and α_1 = "[1,2] safe" → KB $\models \alpha_1$



ENTAILMENT: one thing follow from other e.g. $KB \models \alpha$. KB entails α iff α is true in all world where KB is true.

Eg. KB with "Giants won" and "Reds won" entails "Either Giants won or Reds won"

MODELS: we say m is a model of α if α is true in m, $M(\alpha)$ = set of all models of α

→ $KB \models \alpha$ iff $M(KB) \subseteq M(\alpha)$ ← Model CHECKING

DEDUCTION: $KB \vdash_i \alpha$ → α can be derived from KB by procedure i

TRUE is true, FALSE is false → TRUE and FALSE are symbols with value true and false

Deduction works on formulae by applying inference rules

E.g. Propositional logic = concatenation of sentences

$S_1 \Rightarrow S_2$ is true iff S_1 is false or S_2 is true i.e. is false if S_1 is true and S_2 is false. ~ a bit tricky

TRUTH TABLE FOR CONNECTIVES

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Eg. wumpus world

"pit causes breezes in adjacent squares" → $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Logical Equivalence: logically equivalent iff true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta \wedge \beta \models \alpha$

Validity: valid if true in all interpretations

e.g. True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow)) \Rightarrow B$

Deduction Theorem: $KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is valid

Satisfiability: satisfiable if true in some interpretation → has a model, unsatisfiable otherwise

- $KB \models \alpha$ iff $(KB \wedge \neg \alpha)$ insatisfiable

We have two decision problems:

1) tautology checking (TAUT) → co-NP-complete
2) satisfiability checking (SAT) → NP-complete

Reasoning methods:

① Model checking → truth table enumeration → heuristics (DPLL) or local search in model (GSAT)

② Deduction → generate new sentences through application of inference rules.

CONJUNTIVE NORMAL FORM

In CNF KB is represented as set of clauses = disjuntion of literals $L_1 \vee L_2 \vee \dots L_n$

eg. $\{\{A, \neg B \neg C\}, \{\neg A, B\}\} \Leftrightarrow$

Search for model (model checking)

- Propositional symbol = boolean variable
- Model = boolean assignment
- Formula = constraints to satisfy

↳ DPLL:

① early check of satisfied or unsatisfied formulae
② pure symbols euristics (only positive or negative)
③ unitary formulae

↳ GSAT

Start from random assignment, compute best successor and continue. If fails restarts from new random.
Efficient but incomplete.

Deduction:

$KB \vdash \alpha$. Can be characterized as search: init state is the KB, operators are inference rules, final state is the formula to be proven

Inference rules

$\frac{A_1 \dots A_n}{A}$ = premises / conclusions

Γ = set of formulae. A is derived from Γ ($\Gamma \vdash A$) if exists a sequence of formulae $A_1 \dots A_n$ s.t.

- A is A_n
- $\forall i$ in $[1, n]$ either $A_i \in \Gamma$ or A_i is a direct derivation of the formulae in previous sequence.

$\langle A_1 \dots A_n \rangle$ is a proof of A from Γ. Elements of Γ are called premises / hypotheses / assumptions of A.

A deduction method R is sound if $\forall$ formula A : $\vdash_R A$ implies $\models A$
Is complete wrt a set of formulae Γ if $\forall A \in \Gamma$: $\models A$ implies $\vdash_R A$
$\Rightarrow \vdash = \models$

MODUS PONENS $\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$ and ELIMINATION $\frac{\alpha_1 \wedge \alpha_2 \dots \wedge \alpha_n}{\alpha_i}$

e.g. $\frac{\text{man} \Rightarrow \text{mortal},\ \text{man}}{\text{mortal}}$

$\Gamma = \{\text{feline} \Rightarrow \text{animal},\ \text{cat} \Rightarrow \text{feline},\ \text{cat}\}$
$\Gamma \vdash_{MP}$ animal

HORN FORM

KB = conjunction of Horn Clauses
Horn Clause = propositional symbol or conjunction of symbols
Definite clause = exactly one positive literal

FORWARD & BACKWARD CHAINING

Modus ponens for Horn form: complete for Horn KB $\frac{\alpha_1 \dots \alpha_n,\ \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$
Both algorithms run in linear time.
Forward chaining: fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found
Backward chaining: from query q, to prove q, checks if q is known already or prove all premises of some rule concluding q
FC. is data driven and may do lot of works that is irrelevant to the goal
B.C. is goal driven, appropriate for problem solving. Complexity can be much less than linear in size of KB

Ex. Does the following formula hold? $\underbrace{(\alpha \wedge \beta) \Rightarrow \gamma}_{\phi_1} \models \underbrace{(\alpha \Rightarrow \gamma) \vee (\beta \Rightarrow \gamma)}_{\phi_2}$

α	β	γ	ϕ_1	ϕ_2	$\models$
0	0	0	1	1	✓
0	0	1	1	1	✓
0	1	0	1	1	✓
0	1	1	1	1	✓
1	0	0	1	1	✓
1	0	1	1	1	✓
1	1	0	0	0	✓
1	1	1	1	1	✓

$\Rightarrow$ it holds

$P \Rightarrow Q$ false if P true and q false only
true if both false too!

Ex If John study and his father work then his grandfather is happy.

$S \wedge W \Rightarrow H$ ✓
$S \wedge W \wedge H$ ×
$\neg S \vee \neg W \vee H$ ✓ } congruent
$S \vee W \Rightarrow H$ ×

RESOLUTION

From now on we will call literals atoms.

Add literals: $L \cup \{L_1 \vee .. L_n\} = \{L \vee L_1 \vee .. L_n\}$ → Clause is a disjuntion of literals

A formula is in negation normal form (NNF) if negations appear only in front of atoms. It is in conjunctive normal form (CNF) or clausal form or is called set of clauses if is in NNF it is $C_1 \wedge C_2 \dots C_n$ where C_i is a clause

Set of clauses: $\{\{L_1 \vee .. L_n\}\{L_1' \vee .. L_n'\}\} \equiv (L_1 \vee .. L_n) \wedge (L_1' \vee .. L_n')$

Let the following clause $A_1 \vee A_2 \vee A_3 \dots \wedge A_n \wedge \neg B_1 \wedge \neg B_2 \dots \wedge \neg B_m$

if $n \leq 1$ (positive atoms ≤ 1) the clause is Horn e.g $P \vee \neg Q \vee \neg R \rightarrow Q \wedge R \Rightarrow P$

A CNF is horn if all its clauses are horn. If $n=1$ its called definite (Horn) clause

α formulae: a conjunctive formula β formulae: a disjunctive formula

	α_1	α_2
$A \wedge B$	A	B
$\neg(A \wedge B)$	$\neg A$	$\neg B$
$\neg(A \Rightarrow B)$	A	$\neg B$

	β_1	β_2
$A \vee B$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$
$A \Rightarrow B$	$\neg A$	B

Trasformation in clausal form:

Let F be a propositional formula:

① $\{F\}$ = initial set

(n+1) result of step n $\{D_1 \dots D_n\}$ where D_i is a disjunction $\{A_1^i \dots A_k^i\}$ → if A_j^i not literal we dont have a CNF yet.

↳ choose D_i which contains non literal X:

ⓐ if X is $\neg\top$ replace it with $\bot$;
ⓑ if X is $\neg\bot$ replace it with $\top$;
ⓒ if X $\neg\neg A$ replace it with A;
ⓓ if X is a β formula replace it with β_1, β_2
ⓔ if X is an α formula replace D_i with two clauses D_i' and D_i^2 where D_i' is D_i with α replaced by α_1, and D_i^2 is D_i with α replaced by α_2

→ corrisponding expansion rules: ① $\frac{\neg\neg A}{A}$ ② $\frac{\neg\top}{\bot}$ ③ $\frac{\neg\bot}{\top}$ ④ $\frac{\beta}{\beta_1 \beta_2}$ ⑤ $\frac{\alpha}{\alpha_1 | \alpha_2}$

Note Let D be a disjunction $\{A_1 \dots A_k\}$. If D' is obtained from D by applying rules a-e then $D \equiv D'$

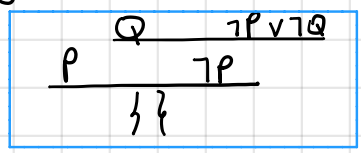
Let $\{L\} \cup D_1$ and $\{\neg L\} \cup D_2$ be two clauses. $D_1 \cup D_2$ is obtained from $\{L\} \cup D_1$ and $\{\neg L\} \cup D_2$ by resolution step:

$$\frac{\{L\} \cup D_1 \quad \{\neg L\} \cup D_2}{D_1 \cup D_2}$$

A resolution tree is a binary tree whose nodes are labelled by clauses. Let C_1 and C_2 be two brother nodes, whose father is C, then $C_1 = D_1 \cup \{L\}$, $C_2 = D_2 \cup \{\neg L\}$ and $C = D_1 \cup D_2$. The label of father is the result of resolution step done on its sons.

Given Γ finite set of clauses with C in it, C can be derived by resolution from Γ iff. exists a resolution tree with root = C and all leaves are clauses in Γ → $\boxed{\Gamma \vdash_R C}$

e.g. $\Gamma = \{\{P\}, \{Q\}, \{\neg P, \neg Q\}\}$, $\Gamma \vdash_R \{\}$:



Given Γ set of clauses, $\{L\} \cup D_1 \in \Gamma$ and $\{\neg L\} \cup D_2 \in \Gamma$. If Γ is satisfiable then $\Gamma \cup \{D_1 \cup D_2\}$ is satisfiable too. If $\Gamma \vdash_R \{\}$ then Γ is unsatisfiable.

$RC(\Gamma)$ = resolution closure = finite set of clauses that can be derived by Γ using resolution rule

→ Γ unsatisfiable iff $\{\} \in RC(\Gamma)$

Resolution is sound and complete for propositional logic

Propositional logic is:

Declarative:

Compositional: $B_{11} \wedge P_{12}$ is derived from meaning of B_{11} and P_{12}. Meaning is context-independent

State = Location + Arrow + Wumpus → Fluents: L^0_{11}, $HaveArrow^0$, $WumpusAlive^0$, $FacingEast^0$ 0-timestamp

$L^t_{xy} \Rightarrow (Breeze^t \Leftrightarrow B_{xy})$

$L^0_{11} \wedge FacingEast^0 \wedge Forward^0 \rightarrow L^1_{21}$

$Forward^t \Rightarrow (HaveArrow^t \rightarrow HaveArrow^{t+1})$

Complex! → First Order Logic

FIRST ORDER LOGIC (FOL)

Assumes world contains Objects, Relations, Functions while propositional logic assumes only "facts"
Some language + quantifiers $\forall, \exists$
TERM set of terms, is defined as follow:
1) Every constant and variable symbol is a term
2) if $t_1 \dots t_n$ are terms and f is a n-ary function symbol $\rightarrow f(t_1 \dots t_n)$ is a term (functional term)
ATOM set of atomic formulae $\longrightarrow$ True or False
FORMULAE: $P(x)$, $\exists x\, Q(x,c)$, $\forall z$ ....
Give precedence as following: $\forall, \exists, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Atom/term is ground if does not have variables. Occurrence of variable in a formula is free if not in the scope of quantifier, bound otherwise.

In general $\exists$ existential classifier works with conjunctions ($\wedge$) and $\forall$ universal classifier with implication ($\Rightarrow$)
Note $\forall x \exists y \neq \forall y \exists x$!

A structure of the language is a pair $\mathcal{U} = (D, I)$ with:
D = domain of $\mathcal{U}$ = set of individual objects
I = function called interpretation that maps
- every constant symbol c into $c^I \in D$
- every n-ary function symbol f into $f^I: D^n \rightarrow D$
- every n-ary predicate symbol P into $P^I \subseteq D^n$

A Model is a structure where a formula is true.
Eg. $\forall x \exists y\, P(x,y)$
D = set of human beings
P^I = set of pairs (A,B) s.t. B is father of A $\Big\}$ $\Longrightarrow$ All human beings have a father (all depends on P)

$\mathcal{U} \models \phi$ $\rightarrow$ truth of closed formula ϕ in structure $\mathcal{U}$ = $\mathcal{U}$ satisfies ϕ iff ϕ is true in $\mathcal{U}$
A formula is valid iff it is true in every structure, $\models A$
A set of formulae Γ is satisfiable iff there is a structure $\mathcal{U}$ s.t. $\mathcal{U} \models A$ for every $A \in \Gamma$

Formulae are semantically equivalent if they differ in:
- the name of variables in the scope of quantifier: $\forall x P(x) \equiv \forall y P(y)$
- the order of quantifier of same kind: $\forall x \forall y\, P(x,y) \equiv \forall x \forall y\, P(x,y) \equiv \forall x,y\, P(x,y)$
- the elimination of quantifiers whose variables does not occurr in their scope: $\forall x P(y) \equiv P(y)$

LOGICAL PROGRAMMING (and PROLOG)

Summarizes the idea that systems should be constructed by expressing knowledge in formal language and problems must be solved using inference on it → Declarative programming

PROLOG = programming language for logical programming

How it works? Define problem through the assertion of facts and rules then query the system which infers the answer given known facts and rules → like syllogism.

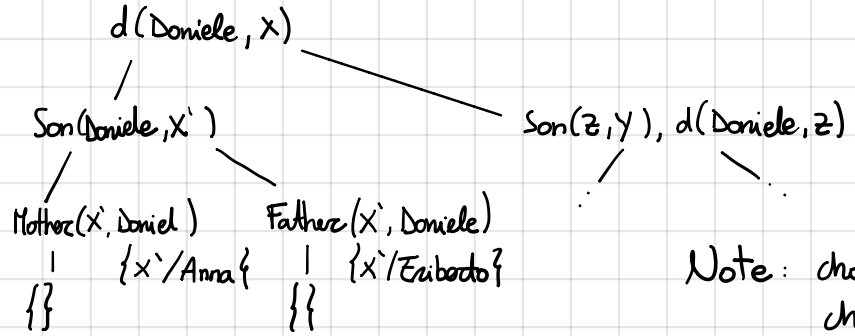
e.g. `mortal(x) :- man(x).` (= all men are mortal)
`man(Socrates).` (Socrate is a man)
→ Inference started by `?mortal(Socrates)`

`man(Socrate)` = unconditional assertion = fact
`mortal(x) :- man(x)` = conditional assertion = rule (`mortal(x)` = conclusion, `man(x)` = premise, both atoms)

`son(X,Y) :- father(Y,X)` } son is a procedure
`son(X,Y) :- mother(Y,X)`

Eg Recursive rules

```
descendant(X,Y) :- son(X,Y)
descendant(X,Y) :- son(Z,Y), descendant(X,Z)
son(X,Y) :- father(Y,X)
son(X,Y) :- mother(Y,X)
```



⇒ Descendant(Daniele, X) = Anna
⇒ Descendant(Daniele, X) = Eriberto
⇒ ...

Note: changing order of conjunctions the execution will change too (not the meaning)
changing the order of formulas instead may break everything

SUBSTITUTION

FOL requires substitutions of variables with terms → map from VAR to TERM

$\sigma = \{x/t\}$ or $\sigma = \{t/x\}$ ⇒ SUBST $(\{x/t\}, \alpha)$ with α = premise

$\frac{\forall v \ \alpha}{\text{SUBST}(\{x/t\}, \alpha)}$ ↓

$\forall x \ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields to: $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$, $\text{King}(\text{Rich.}) \wedge \text{Greedy}(\text{Rich}) \Rightarrow \text{Evil}(\text{R.})$ and so on → UNIVERSAL INSTANTIATION (UI)

$\exists v \ \alpha$

$\text{SUBST}(\{v/k\}, \alpha) \rightarrow \exists x \ \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields to $\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$ → EXISTENTIAL INST.

EI can be applied once to replace the existential sentence while UI many times to add new sentences → PROPOSITIONALIZATION i.e. expand the KB

Herbrand Th. If sentence α is entailed by FOL KB, it is entailed by a finite subset of propositional KB

Propositionalization generate lot of useless sentences

Unification: find a substitution that makes two expressions identical: $\{x/b\}$ is unifier of $p(a,x)$ and $p(a,b)$

Substitution affects only variables, not constant symbols.

An expression s is more general than t if t is an instance of s but not viceversa. ie it places fewer restrictions on the value of variables → pick the most general unifier (MGU) because we may have multiple unifier for a single unification.

Horn Data Bases (HDB): no function symbols and domain restricted to known individuals (finite)

Eg. The law says that is a crime for an American to sell weapons to hostile nations. Nono is enemy of America and has some missiles sold by Colonel West, an American. Prove that west is a criminal.

1) selling weapons to hostile nations is a crime for American: $\text{Amer}(x) \wedge \text{Weapon}(y) \wedge \text{Sell}(x,y,z) \wedge \text{Host}(z) \Rightarrow \text{Criminal}(x)$

2-3) Nono has missiles i.e. $\exists x \ \text{Own}(\text{Nono}, x) \wedge \text{missil}(x)$: $\text{Own}(\text{Nono}, M_1)$ and $\text{missil}(M_1)$

4) all of it missiles were sold to it by West: $\text{missil}(x) \wedge \text{Own}(\text{Nono}, x) \Rightarrow \text{Sell}(\text{West}, x, \text{Nono})$

5) Missiles are weapon: $\text{missil}(x) \Rightarrow \text{Weapon}(x)$

6) An enemy of America count as "Host": $\text{Enemy}(x, \text{America}) \Rightarrow \text{Host}(x)$

7) West is American: $\text{American}(\text{West})$

8) Nono is enemy of America: $\text{Enemy}(\text{Nono}, \text{America})$

Generalized Modus ponens (GMP): $\frac{p_1', p_2', \dots p_n', \ (p_1 \wedge p_2 \dots p_n \Rightarrow q)}{q\theta}$ θ = unifier of all pair $p_i' = p_i\theta \ \forall i$

p_1' is $\text{missil}(M_1)$ p_1 is $\text{missil}(x)$ θ is $\{x/M_1\}$ q is $\text{Weapon}(x)$ $q\theta$ is $\text{weapon}(M_1)$ in our case.

From ③ and ⑤ → $\text{Weapon}(M_1)$ ⑨

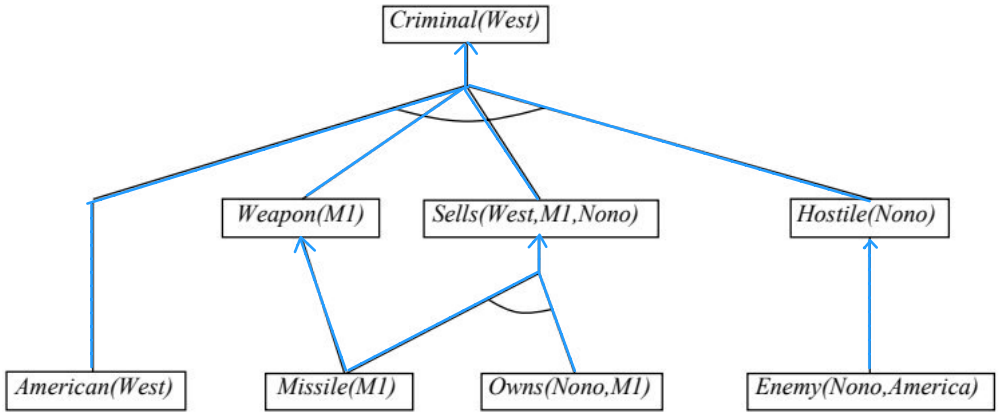
From ⑧ and ⑥ → $\text{Host}(\text{Nono})$ ⑩

From ②, ③ and ④ → $\text{Sell}(\text{West}, \text{Nono}, M_1)$ ⑪

From ⑦, ⑧, ⑩, ⑪ and ① → $\text{Criminal}(\text{West})$

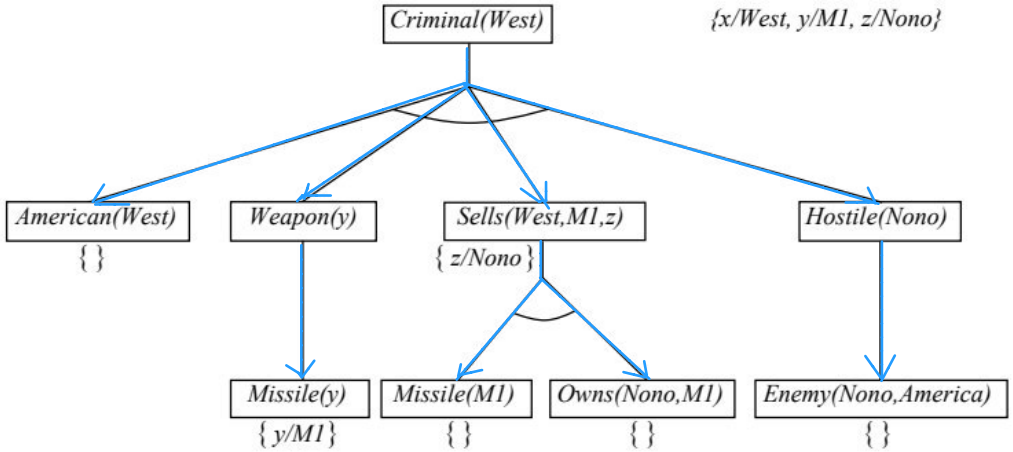
To prove that $\text{Criminal}(\text{West}) \in$ KB GMP is hard and quantifier elimination may be expensive.

FORWARD CHAINING



FC is sound and complete for F.O. definite clauses. With no function and finite domain terminates in poly time $d \times n$, n = max number of premises in rules and d = size of HDB

BACKWARD CHAINING



Ex 1 : An elephant is happy if all its children can fly

① $\forall x\,(elephant(x) \Rightarrow ((\forall y\ child(y,x) \Rightarrow fly(y)) \Rightarrow happy(x)))$ ✓

② $\forall x((elephant(x) \wedge happy(x)) \Rightarrow \forall y\,(child(y,x) \Rightarrow fly(y)))$ ✗ implies only children of happy elephant can fly

③ $\forall x\,(elephant(x) \Rightarrow ((\neg\exists y\,(child(y,x) \wedge \neg fly(y))) \Rightarrow happy(x)))$ ✓ same as first but written differently (negation)

④ $\forall x\,(elephant(x) \Rightarrow \forall y\,((child(y,x) \wedge fly(y)) \Rightarrow happy(x)))$ ✗ it states that elephant is happy if all element in domain are its children and can fly

Ex 2. Choose constants and predicates for following sentences:

① Steve likes easy classes: $\forall x\ EC(x) \Rightarrow likes(x, steve)$ ✓ $\forall x\ likes(EC(x), steve)$ ✗

② Violin classes are not easy: $\forall x\ VC(x) \Rightarrow \neg EC(x)$

③ Every class of percussion is easy: $\forall x\ PC(x) \Rightarrow EC(x)$

④ The class of Afro drums is a class of percussion: $PC(AD)$

⑤ "Violin for pre-school kids" is a violin class: $VC(VK)$

Can we infer that Steve likes AD with:

- Modus Ponens

using $\sigma = \{x/AD\}$ and ③ and ④ $\rightarrow$ $PC(AD) \rightarrow EC(AD)$ so $EC(AD)$

$EC(AD) \rightarrow like(AD, Steve)$ so $like(AD, Steve)$

- Resolution

transform KB and negated thesis into clauses:

$\{\neg EC(x) \vee like(x, Steve)\}_1$, $\{\neg VC(x) \vee \neg EC(x)\}_2$, $\{\neg PC(x) \vee EC(x)\}_3$, $\{PC(AD)\}_4$, $\{VC(VK)\}_5$, $\{\neg like(AD, Steve)\}_6$

from ③ and ④ with $\sigma = \{x/AD\}$ $\rightarrow$ $\{EC(AD)\}_7$

from ① and ⑦ with $\sigma = \{x/AD\}$ $\rightarrow$ $\{likes(AD, Steve)\}_8$

from ⑥ and ⑧ $\rightarrow$ $\{\}$

SKOLEMIZATION

$\forall x\,(Person(x) \Rightarrow \exists y\, Heart(y) \wedge Has(x,y)$ if we substitute y with a constant we would say that every person has the same heart → wrong!
So we replace the constant with a function $F(x)$ that doesn't appear in KB. F is called skolem function

A formula Ψ is in prenex form if:
1) doesn't contain quantifier (no variables or is open and variables are free); or
2) is in form $Q_1 x_1 Q_2 x_2 \dots Q_n x_n A$ where A is a quantifier free formula, x_i are variables on $Q_i \in [\forall, \exists]$.

Th. $\forall \phi$ there exist a ψ s.t. $\phi = \psi$ and $var(\phi) = var(\psi)$
Construction: ϕ formula of L
1) build ϕ^L where negations are pushed inward and double negations are negated s.t. $\vdash \phi \leftrightarrow \phi^L$
2) Rename bound variables so that each quantifier uses a different variable
3) Build the prenex formula moving all quantifier to the left
Validity preserving

Apply skolemization ϕ^{SKO} to ϕ in prenex normal form: $(\exists x \phi)^{SKO} = \phi[F(x_1 \dots x_n)/x]$ where $var(\phi) = \{x_1 \dots x_n\}$ and F is a new n-ary function symbol. Given ϕ^P in prenex form:
4) Build ϕ^{SKO} s.t. ϕ^{SKO} is satisfiable iff. ϕ^P is. ϕ^{SKO} is of the form $\overline{\forall x}\,\psi^-$ where ψ^- doesn't contain quantifiers.
Given ϕ^{SKO} in prenex form (without existential quantifier):
5) Get ψ^C conjunctive normal form s.t. $\psi^C \equiv \psi^- \equiv \psi_1^- \wedge \dots \wedge \psi_m^-$ and every ψ_i^- is an open clause.
6) Remove the universal quantifiers and transform ψ^C into set of clauses $S(\phi) = \{\psi_1^- \dots \psi_m^-\}$
$S(\phi)$ is called skolem normal form of ϕ

M model of ϕ^{SKO} $\Rightarrow$ M model of ϕ but not viceversa
ϕ^{SKO} valid $\Rightarrow$ ϕ valid but not viceversa
ϕ satisfiable iff ϕ^{SKO} satisfiable
ϕ unsatisfiable iff ϕ^{SKO} unsatisfiable
$\neg\phi$ valid iff $\neg(\phi^{SKO})$ valid

UNIFICATION

$f(g(x,w))$ $f(g(z,h(y,w)))$ can be unified? → $SUBST(\sigma,t_1) = SUBST(\sigma,t_2)$
Substitution σ is an unifier of t_1 and t_2 if $t_1\sigma = t_2\sigma$ for the terms in the two formulae.
Ex
$s_1 = f(a, g(x,b), h(y))$
$t_1 = f(y, z, w)$
$C = \{\langle s_1, t_1\rangle\}$
$\theta = \{\}$

$C = \{\langle a,y\rangle, \langle g(x,b), z\rangle, \langle h(y), w\rangle\}$

Take $\langle a,y\rangle$ → $\theta = \{\} \vee \{y/a\}$ $C = \{\langle a,y\rangle, \langle g(x,b), z\rangle, \langle h(y), w\rangle\}$
Take $\langle g(x,b), z\rangle$ → $\theta = \{\} \vee \{y/a\} \vee \{z/g(x,b)\}$ $C = \{\langle a,y\rangle, \langle g(x,b), z\rangle, \langle h(y), w\rangle\}$
Take $\langle h(y), w\rangle$ → $\theta = \{\} \vee \{y/a\} \vee \{z/g(x,b)\} \vee \{w/h(y)\}$ $C = \{\langle a,y\rangle, \langle g(x,b), z\rangle, \langle h(y), w\rangle\}$

Ex Unification

$g(a,x)$ $g(a,b)$ $\rightarrow \langle a \not\rightarrow a \rangle$ and $\langle x,b \rangle \rightarrow \{x/b\}$

$g(y,y)$ $g(f(a),f(z)) \rightarrow \langle y, f(a) \rangle$ and $\langle y, f(z) \rangle \rightarrow \{y/f(a)\} \rightarrow \langle f(a), f(z) \rangle \rightarrow \{a/z\}$

$\Rightarrow \{x/b, y/f(a), a/z\}$

Ex Translate in FOL and say if it is Horn.

1) Textbooks of class CA are easy: $\forall x\ text(x, CA) \Rightarrow easy(x)$
2) Textbooks of class CB are difficult: $\forall x\ text(x, CB) \Rightarrow \neg easy(x)$
3) Mary studie all and only easy books: $\forall x\ easy(x) \Rightarrow study(Mary, x)$
4) Mary passes the exam of a class if she studies at least a textbook of it: $\forall x [\exists y\ text(y,x) \wedge study(Mary, y)] \Rightarrow pass(Mary, x)$
5) Russel & Norvig is a textbook of class CA: $text(RN, CA)$
6) Tenenbaum is a textbook of class CB: $text(TB, CB)$

All Horn because at most one positive atom.

Ex Prove using Resolution that "mary passes on exam" by adding the needed knowledge

Knowledge base. (CNF)

$\{\neg text(CA, x) \vee easy(x)\}_1$
$\{\neg text(CB, x) \vee \neg easy(x)\}_2$
$\{\neg study(Mary, x) \vee easy(x)\}_{3.1}$
$\{study(Mary, x) \vee \neg easy(x)\}_{3.2}$
$\{\neg text(x,y) \vee \neg study(Mary, y) \vee pass(Mary, x)\}_4$
$\{text(CA, RN)\}_5$
$\{\neg pass(Mary, z)\}_7$
$\{text(CB, TN)\}_6$

From ④ and ⑦ with $\sigma = \{z/x\}$
$\{\neg text(z,y) \vee \neg study(Mary, y)\}_8$
From ⑤ and ⑧ with $\sigma = \{z/CA, y/RN\}$:
$\{\neg study(Mary, RN)\}_9$
From ③.② and ⑨ with $\sigma = \{x/RN\}$
$\{\neg easy(RN)\}_{10}$
From ① and ⑤ with $\sigma = \{x/RN\}$
$\{easy(RN)\}_{11}$
From ⑩ and ⑪ $\rightarrow \{\}$

RESOLUTION FOR DUMMIES

We will use the following example: $\forall x\,[\forall y\ Animal(x) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

① Eliminate implications: $\forall x\,[\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$
Everything before "$\Rightarrow$" get negation "$\neg$" and the implication symbol becomes "$\vee$"

② Move $\neg$ inwards:
$\neg\forall x\ p \rightarrow \exists x\ \neg p$ and $\neg\exists x\ p \rightarrow \forall x\ \neg p$ so we will have:
$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]$
$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$ ($\neg\vee \rightarrow \wedge$ and $\neg\wedge \rightarrow \vee$ de Morgan)

③ Change variables to avoid confusion in case such as "$(\exists x\ P(x)) \vee (\exists x\ Q(x))$"
$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$

④ Remove existential quantifier with skolemization: $\forall x\,[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$
Cancel all existential quantifiers and replace their variables with some skolem function (in our case F and G). Their arguments are all the universally quantified variables (in our case x) on the left of existential quantifier.
If an existential quantifier doesn't have an universal one on the left then replace its variable with a constant: $\exists y \forall x \exists z\ (P(x,y) \wedge P(z,x) \wedge P(y,z)) \rightarrow P(x,a) \wedge P(g(x),x) \wedge P(a,g(x))$

⑤ Drop all universal quantifiers: $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

⑥ Distribute $\vee$ over $\wedge$: $[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$
I.e. do the same thing you do in a multiplication $(a+b)*c = a*c + b*c$

LISTS in Prolog

[a,b,c,d] = 4 element list (constant)
[a|X] = first constant and the rest is denoted by variable X. = cons(a, X)
[Y|X] = first element denoted by var Y and rest by X.

eg. cons(d,[a,b,c]) = [d,a,b,c]
↳ cons(d, cons(a, cons(b, cons(c, nil))))
[c]

Member1(X,L) true if X in L:
```
member1(X,[X|Ys]).
member1(X,[Y|Ys]) :- member1(X,Ys)
```

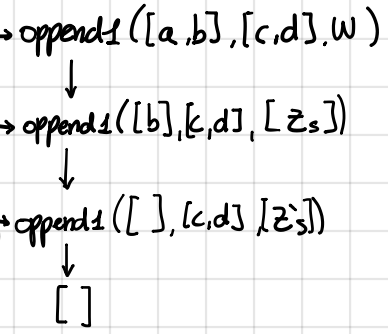
Append1(X,Y,Z) is true if Z is the concatenation of X and Y
```
append1([],Ys,Ys).
append1([X|Xs],Ys,[X|Zs]) :- append1(Xs,Ys,Zs)
```

Eg. ?append1([a,b],[c,d],W)

Note: [a,b] ≡ [a|[b|[]]]

{X/a, Xs/[b], Ys/[c,d], W/[a|Zs]}
↳ because X/a

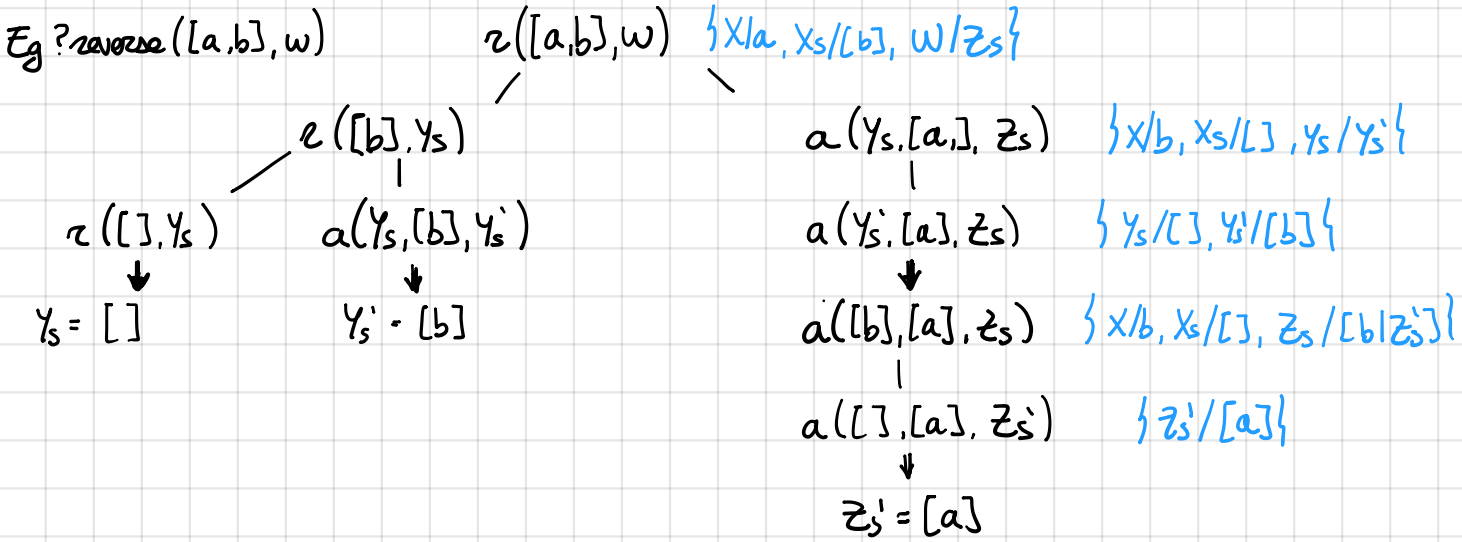
{X/b, Xs/[], Ys/[c,d], Zs/[b,Zs']
{Ys/[c,d], Zs'/Ys}



Prefix(L1,L) is true if L1 is a prefix of L
```
Prefix([],_Ys)
Prefix([X|Xs],[X,Ys]) :- prefix(Xs,Ys)
```

Reverse(L1,L2) is true if L2 is reverse of L1
```
reverse1([],[]).
reverse1([X|Xs],Zs) :- reverse1(Xs,Ys), append1(Ys,[X],Zs)
```

Eg ?reverse([a,b],W)



Ex Write prolog program for suffix, subset and intersection using list to represent sets

Suffix(X, Y) X suffix of Y
- suffix(X, X) base case
- suffix(X, [Y|Ys]) :- suffix(X, Ys) doesn't decompose first element, not empty

Subset(X, Y) X subset of Y
- subset([], _X) ← singleton = variable used only one time
- subset([X|Xs], Y) :- member(X, Y), subset(Xs, Y)

Intersection(X, Y, Z) Z is intersection of X and Y
- intersect([], Y, [])
- intersect([X|Xs], Y, [X|Z]) :- member(X, Y), intersect(Xs, Y, Z)
- intersect([X|Xs], Y, Z) :- intersect(Xs, Y, Z)

IS predicate: (A is B) true when evaluation of B returns a value that is assigned to variable A. Predicates defined by IS are not invertible.

Eg X is 3+4
5 is X+Y

Eg. factorial(0, 1).
factorial(Y, X) :- Y1 is Y-1, factorial(Y1, X1), X is Y * X1

LISTS OF LISTS

memberlist(X, L) is true if one of the following:
- X is first element of L
- X is not the first element of L, the first element of L is a list and X is a member of first element of L.
- X is member of the rest of L

```
memberlist(X, [X|_Xs]).
memberlist(X, [Y|_Ys]) :- memberlist(X, Y).
memberlist(X, [_Y|Ys]) :- memberlist(X, Ys).
```

NON-MONOTONIC REASONING

Monotonic: adding knowledge can only increases the number of logical consequences.
Non monotonicity can invalidate some of previous conclusions.
- all birds fly → OK
- then we discover penguin = bird penguins dont fly
- all birds fly → NOT TRUE ANYMORE

ASSUMPTIONS: if we dont have specific knowledge of given instances we can generalize the knowledge.
Universal assertions: properties that holed for all instances
General assertions: properties that hold in general case
eg: a violin has 4 cords → if it has 3 it is still a fucking violin but broken. (Non-monotonicity)

CLOSED WORLD REASONING

If something is true and relevant put it in KB → not in KB assumed false

CLOSED WORLD ASSUMPTION

closure(KB) = $KB \cup \{\neg L \text{ s.t. } L = \text{positive literal} \wedge KB \not\models L\} = KB^+$
$KB^+ \models \alpha$ or $KB^+ \models \neg\alpha$ $\forall\alpha$. If $KB^+ \models A \vee B$ then $KB^+ \models A$ or $KB^+ \models B$ so the answer to a query α can be obtained verifying if literals in α are implied by KB^+.
Prob: if $KB \models A \vee B$ but $KB \not\models A$ and $KB \not\models B$ (incomplete knowledge) then KB^+ will contain both $\neg A$ and $\neg B$ → inconsistency.
⇒ $KB^* = KB \cup \{\neg L$ s.t. L - positive literal $\wedge \nexists C$ positive clause s.t. $KB \models L \vee C$ and $KB \not\models C\}$ = generalized CWA

NEGATION AS FAILURE

if from P (a KB) we dont prove A then we deduce $\neg A$.

Operational models are extended to treat negation as failure (stable model) and according to it computes the answer

MINIMAL MODEL

Find a canonical / preferred model where to verify truth. Horn clauses: minimal model = unique
eg. P(1) Q(2) $P(x) \Rightarrow Q(x)$ minimal model: $M = \{P(1), Q(2), Q(1)\}$ other model $M' = \{P(1), P(2), Q(1), Q(2)\}$
Reduction: given a set M of atoms of Π logic program, the reduct Π_M is obtained by eliminating:
1) all clauses that have negated atom of M in the body
2) all negative literals not in M
A reduct is a program that has only Horn clauses and unique minimal model, if it coincides with M it is called stable set. If a program has only one stable model then it is the preferred one.

1) guess a model M
2) build the reduct
3) compare M with M' = model of reduct

eg. P ← not Q
1) $M = \{P\}$ (Q = false)
2) $\Pi_R = P$
3) $M' = \{P\}$ OK → if not ↑

1) $M = \{Q\}$ repeat with other minimal model (Q = true)
2) $\Pi_R = \{\}$
3) $M' = \{\}$

eg. $\begin{cases} P(1,2) \\ P(x,y), \text{ not } q(y) \Rightarrow q(x) \end{cases}$

$p(1,2)$

$q(1) :- p(1,1), \text{not}(q(1))$

$q(1) :- p(1,2), \text{not}(q(2))$

$q(2) :- p(2,1), \text{not}(q(1))$

$q(2) :- p(2,1), \text{not}(q(2))$

$M = \{p(1,2), q(1)\}$

$M' = \{p(1,2), q(1)\}$

DEFAULT LOGIC

Prec : Just / concl → if PREcondition is provable and the hypotheses in just can be consistently assumed, then we can conclude CONCL.

Default Theory : pair (W, D) with W = axiomatible theory and D = set of default rules $\frac{\alpha : \beta}{\gamma}$ with $\alpha\ \beta\ \gamma$ formulae.

$\mathcal{E}$ is an extension of (W, D) iff $\forall$ sentence π : $\pi \in \mathcal{E}$ iff $W \cup \{\gamma \text{ s.t. } \frac{\alpha : \beta}{\gamma} \in D, \alpha \in \mathcal{E}, \neg\beta \notin \mathcal{E}\} \vdash \pi$

A default rule is applicable if the default assumption (just) remains valid after the extension has been built.

eg. (W, D) with $W = \{q\}$ $D = \{\frac{q : p}{r}\}$ $S = \{q, r\}$ is an extension

If just = concl → default is called normal. A theory with all defaults normal has always an extension

eg Quackers (usually) are pacifist, republicans (usually) not.

$\frac{Q(x) : P(x)}{P(x)}$ $\frac{R(x) : \neg P(x)}{\neg P(x)}$

Nixon is a republican quacker $W = \{Q(\text{nixon}), R(\text{nixon})\}$ Can we conclude that Nixon is pacifist?

$S = Cn(\{Q(\text{nixon}), R(\text{nixon}), P(\text{nixon})\})$. S is an extension

$S' = Cn(\{Q(\text{nixon}), R(\text{nixon}), \neg P(\text{nixon})\})$. S' is an extension too.

A conclusion is cautious if belong to all extensions, brave otherwise. → Nixon is pacifist is a brave conclusion

TAXONOMIC REPRESENTATION AND REASONING

Semantic Networks & Frames

Connections between classes/objects in which ares between them are the relations.
Networks are the "set" of these connections with wich we can do inference.
eg. instance-of , is-a etc.
A Frame is an abstract description of a prototype object of a class → Connecte frames form semantic network with richer description of nodes.
Frames relationship may be complex: numerical restriction, logical operators etc.
TAXONOMIC KNOWLEDGE: organize knowledge in classes of objects. It is epistemologically and computationally efficient (thanks to classes).

DESCRIPTION LOGIC

Language for frames and networks. Classes or concepts are unary predicates, relations are binary.
Specification of individuals that belong to a concept. eg $C \sqcap D$ = individuals who belong to C and D.
Role restriction:
- Universal, $\forall R.C$: all individuals who are in relation R belong to C
- Existential, $\exists R.C$: require the existence of an individual in relation R that belong to C.

We may have number restriction too. C is called role filler. And ofcourse role can be intersected.

Intersection: $C \sqcap D$ Union: $C \sqcup D$ complement: $\neg C$
eg. "individuals with 2 or 3 children → $(\leq 3\ hasChild) \sqcap (\geq 2\ hasChild)$

TERMINOLOGIES

TBOX: also called terminology, includes concept definitions
e.g. Woma ≡ Person ⊓ Female
Definitions are interpreted as logical equivalences.
ABOX: includes the extensional knowledge
es. Female ⊓ Person (Anna), hasChild (Anna, Jacopo) ...

Prolog exercises

① Define Brother (siblings)

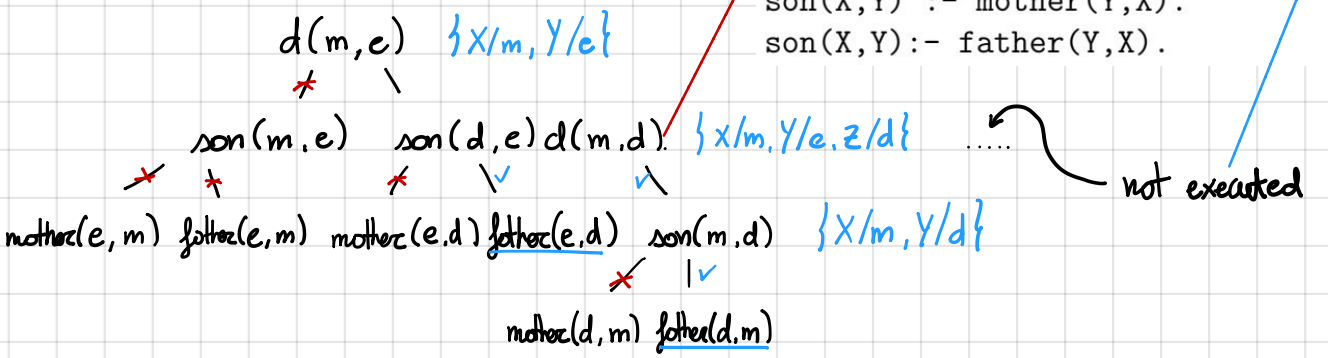
```
brother(B,S):- mother(M,S), mother(M,B), father(F,B), father(F,S), B\=S
```

Define Cousin.

```
cousin(B,S):- mother(M,B), mother(M2,S), brother(M,M).
cousin(B,S):- father(F,B), father(F2,S), brother(F,F2).
cousin(B,S):- father(F,B), mother(M,S), brother(F,M).
cousin(B,S):- mother(M,B), father(F,S), brother(M,F).
```

② Search tree for descendant(michela, eriberto).

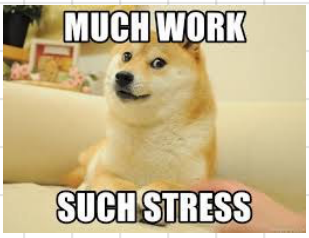
```
descendant(X,Y):- son(X,Y).
descendant(X,Y) :- son(Z,Y), descendant(X,Z).
son(X,Y) :- mother(Y,X).
son(X,Y):- father(Y,X).
```



Difference with descendant2 ?

```
descendant2(X,Y):-son(Z,Y),descendant2(X,Z).
descendant2(X,Y):-son(X,Y).
```

Descendant2 does the "not executed" branch before the "son" branch needed
→ more runs, longer tree.



③ Write a Prolog program that given a binary tree and a list of forbidden elements, verify that exists a path from root to a leaf that contain at most one occurrence of a forbidden elements

```
P(Bintree, ForbNumbers)
P(X,Y):- path(X,Z), legal(Z,Y)

path(tree(_X, void, void), [_X]).
path(tree(_X, L, _R), [_X|W]):- path(L,W).
path(tree(_X, _L, R), [_X|W]):- path(R,W).

legal(Tree, List):- legal2(Tree, List, count), count =< 1

legal2(void, List, 0)
legal2(tree(Elem, left, right), List, count):- member(Elem, List), legal2(left, List, countLeft), legal2(Right, list, countRight),
                                               Count is min(countLeft, countRight) + 1
legal2(tree(Elem, left, Right), list, count):- \+ member(Elem, list), legal2(left, list, leftCounter), legal2(Right, list, countR)
                                               count is min(countLeft, countRight)
```

not =

④ Prolog program counting list elements

```
count([],0).
count([X|Xs],N):- count(X,N1), count([Xs],N2), N is N1+N2.
count(_X,1).
```

Equal to:

```
count([],0).
count([_X|Xs],N):- count([Xs],N2), N is 1+N2.
```

⑤ An associative list is a list all of whose elements are 2-element lists. The first element of each is an atom the second one an arbitrary term. Eg. AL = [[a,f(0)],[b,f(1)],[c,f(2)]]. Use atom(x) to test if X=atom

a: Write prolog program that checks if argument is an associative list.

```
check([]).
check([[X,Y]|Xs]):- atom(X), check(Xs)
```

b: Write a prolog program FINDT that given in input an associative list and an atom A returns a term T corresponding to the second element of elements in AL (lists) whose first element is equal to A
Assume that every atom appears once only.

```
FINDT([],A,null).
FINDT([[X,Y]|Xs],X,Y).
FINDT([[X,Y]|Xs],A,T):- FINDT(Xs,A,T).
```

⑥ Prolog program to check if a matrix is square (M = [[row1],[row2]...]) and if it is return N (N×N)

```
square([[]],0).
square([[X|Ys]|Zs],W):- square2([[X|Ys]|Zs],0,W).
square2([],Y,Y).
square2([[X|Ys]Zs],A,W):- count([[X|Ys]|Zs],R), count([X|Ys],C), R is C-A, plus(A,1,K),
                          square2(Zs,K,W).
```

Exercises FOL

1) Is $(A \wedge B) \vee (\neg A \wedge \neg B)$ valid?

A	B	$A \wedge B$	$\neg A \wedge \neg B$	... V ...
1	1	1	0	1
0	1	0	0	0
1	0	0	0	0
0	0	0	1	1

Not valid!

2) Consider following modus ponens: $\neg A \Rightarrow B$, $B \Rightarrow A$, $A \Rightarrow (C \wedge D)$
Prove $A \wedge C \wedge D$ using modus ponens if possible.
Modus ponens cant be applied to any pair of formulae cause we dont have premises

3) Some KB as before. Transform in CNF and say if Horn.
$\neg A \Rightarrow B \rightarrow A \vee B$ Not Horn cause >1 positive literals
$B \Rightarrow A \rightarrow \neg B \vee A$
$A \Rightarrow (C \wedge D) \rightarrow \neg A \vee (C \wedge D) \rightarrow (\neg A \vee C) \wedge (\neg A \wedge D)$

4) Derive $(A \wedge C \wedge D)$ with resolution and some KB.
CNF + negated thesis (goal): ① $A \vee B$ ② $\neg B \vee A$ ③ $\neg A \vee C$ ④ $\neg A \vee D$ ⑤ $\neg A \vee \neg C \vee \neg D$
- From ① and ② → ⑥ A
- From ⑥ and ③ → ⑦ C
- From ④ and ⑥ → ⑧ D
- From ⑤ and ⑥ → ⑨ $\neg C \wedge \neg D$
- From ⑨ and ⑦ → ⑩ $\neg D$
- From ⑩ and ⑧ → ⑪ $\{\}$

5) Using resolution the following formula can be proven? $\{A \Leftrightarrow B, A \vee B\} \vdash_R (A \wedge B)$
↳ this little shit means that $(A \wedge B)$ can be derived by resolution
↳ THIS IS THE GOAL!

CNF + negated goal: ① $\neg A \vee B$ ② $\neg B \vee A$ ③ $A \vee B$ ④ $\neg A \vee \neg B$
- From ① and ③ → ⑤ B
- From ② and ③ → ⑥ A
- From ④ and ⑤ → ⑦ $\neg A$
- From ⑦ and ⑥ → $\{\}$

6) Model checking: given $ST \wedge \neg SY \Rightarrow PS$ and $\neg LU \wedge SY \Rightarrow \neg PS$ find a model for both.
First thing to do is convert in CNF: ① $\neg ST \vee SY \vee PS$ ② $LU \vee \neg SY \vee \neg PS$
A model is something that make true both of the two CNF.
$\{LU, PS\}$ = LU = true, PS = true, the rest considered false → this is a model cause PS makes true ① and LU makes true ②
$\{ST, SY, PS\}$ is not a model cause SY and PS makes ① true but nothing make true ②.

⑦ KB = {A ⇒ C, B ⇒ C, A ∨ B} can C be derived with modus ponens?

No, we dont have premises

with resolution?

Yes: ① ¬A ∨ C ② ¬B ∨ C ③ A ∨ B ④ ¬C

- From ① and ③ → ⑤ B ∨ C
- From ⑤ and ② → ⑥ C
- From ⑥ and ④ → {}

⑧ Inconsistency:

"If it rains then it is wet"

"If it is wet then it doesnt rain"

"It rains"

Inconsistent = you can reach {} even without goal with resolution

CNF: ① R ⇒ W → ¬R ∨ W ② W ⇒ ¬R → ¬W ∨ ¬R ③ R

From ① and ③ → ④ W

From ④ and ② → ⑤ ¬R

From ⑤ and ③ → {}

⑨ You have: Gov, Univ, Lca, Bill and prof(x), stud(x), unhappy(x), cut_fund(x,y), foil_exam(x,y).

Expres in FOL:

- If government cuts funds to Universities, profesors are unhappy. cutfunds(Gov, Univ) ⇒ ∀x (prof(x) ⇒ unhappy(x))
- If profesors are unhappy all students foil exams. ∀x∀y ((prof(x) ∧ unhappy(x) ∧ stud(y)) ⇒ foil exam(x,y))
- Government cuts funds to universities cutfund(Gov, Univ)
- Lca is a prof prof(Lca)
- Bill is a student stud(Bill)

have that Bill foils Lca's exam with modus ponens:

① $\dfrac{cutfund(Gov, Univ) \Rightarrow (\dots),\ cutfund(Gov, Univ)}{\forall x\ prof(x) \Rightarrow unhappy(x)}$ ② $\dfrac{\forall x \dots,\ prof(Lca)}{unhappy(x)}$ {x/Lca}

③ $\dfrac{\forall x \forall y \dots,\ [stud(Bill), prof(Lca), unhappy(x)]}{foilexam(x,y)}$ {x/Lca, y/Bill}

With resolution:

① ¬cutfund(Gov, Univ) ∨ (¬prof(x) ∨ unhappy(x)) ② ¬prof(x) ∨ ¬unhappy(x) ∨ ¬stud(y) ∨ foilexam(x,y) ③ cutfund(Gov, Univ)

④ prof(Lca) ⑤ stud(Bill) ⑥ ¬foilexam(Lca, Bill)

From ① and ③ → ⑦ ¬prof(x) ∨ unhappy(x)

From ② and ⑦ → ⑧ ¬prof(x) ∨ ¬stud(y) ∨ foilexam(x,y)

From ④ and ⑧ with {x/Lca} → ⑨ ¬stud(y) ∨ foilexam(Lca, y)

From ⑤ and ⑨ with {y/Bill} → ⑩ foilexam(Lca, Bill)

From ⑥ and ⑩ → {}

⑩ $[(A \Rightarrow C) \vee (B \Rightarrow C)] \Rightarrow [(A \wedge B) \Rightarrow C]$ tell if satisfiable, valid, contradiction

(Φ_1 = $(A \Rightarrow C) \vee (B \Rightarrow C)$, Φ_2 = $(A \wedge B) \Rightarrow C$)

A	B	C	A⇒C	B⇒C	Φ_1	Φ_2	✓
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	0	1	1	1
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	0	0	0	0	1
1	1	1	1	1	1	1	1

✓ Satisfiable : at least one mode (✓ = 1)
✓ Valid : all true
x Contradiction : contrary of valid → all false

⑪ Express in FOL using Rose(x), Thorn(x), Has(x,y), Dangerous(x)
① There is no rose without a thorn → $\forall x (Rose(x) \Rightarrow \exists y (Thorn(y) \wedge Has(x,y)))$
② Thorns are dangerous → $\forall x (Thorn(x) \Rightarrow Dangerous(x))$
③ Whoever has something dangerous is dangerous → $\forall x ((\exists y (Dangerous(y) \wedge Has(x,y))) \Rightarrow Dangerous(x))$
Show that rose are dangerous using resolution.
④ $\forall x (Rose(x) \rightarrow Dangerous(x))$
CNF + negated goal
① $\forall x (\neg Rose(x) \vee (\exists y (Thorn(y) \wedge Has(x,y)))) \rightarrow \neg Rose \vee Thorn(F(x))$ ⑪a $\neg Rose(x) \vee Has(x, F(x))$ ⑪b
implication elimination ; ∃ elimination and ∨ distribution
② $\neg Thorn(x) \vee Dangerous(x)$
③ $\forall x (\neg (\exists y (Dangerous(y) \wedge Has(x,y))) \vee Dangerous(x)) \rightarrow \forall x \forall y ((\neg (Dangerous(y) \wedge Has(x,y))) \vee Dangerous(x))$
implication elimination ; move ¬ inward = no need F
$\rightarrow \neg Dangerous(y) \vee \neg Has(x,y) \vee Dangerous(x)$
④ $\neg (\forall x (Rose(x) \Rightarrow Dangerous(x))) \rightarrow \exists x \neg (\neg Rose(x) \vee Dangerous(x)) \rightarrow$ ⑫a $Rose(R)$ ⑫b $\neg Dangerous(R)$
constants used for ∃ elimination

From ⑪a and ② with $\{x/F(x)\}$ → ⑤ $\neg Rose(x) \vee Dangerous(F(x))$
From ⑪b and ③ with $\{y/F(x)\}$ → ⑥ $\neg Rose(x) \vee \neg Dangerous(F(x)) \vee Dangerous(x)$
From ⑤ and ⑥ → ⑦ $\neg Rose(x) \vee Dangerous(x)$
From ⑦ and ⑫a → ⑧ $Dangerous(x)$
From ⑫b and ⑧ with $\{x/R\}$ → $\{\}$

(12) Giovanni loves nature. Nature lovers preserve the environment. People who preserve the environment do not leave garbage in the woods. People who preserve the environment do not light fires in the woods.

1) Represent the above sentences in FOL.
2) Show that Giovanni does not leave garbage and does not light fires in the woods using modus ponens.
3) Transform the above sentences CNF and show that Giovanni does not leave garbage and does not light fires in the woods using resolution.

1) loves(Giovanni, nature)
$\forall x$ loves(x, nature) $\Rightarrow$ preserve(x, environment)
$\forall x$ preserve(x, environment) $\Rightarrow \neg$ leavegarbage(x)
$\forall x$ preserve(x, environment) $\Rightarrow \neg$ lightfire(x)

2) Goal: $\neg$ leavegarbage(Giovanni)
$\neg$ lightfire(Giovanni)

m.p: $\dfrac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$

$$\frac{\text{loves}(x,n) \Rightarrow \text{preserve}(x,e) \quad \text{loves}(g,n)}{\text{preserve}(g,e)} \quad \{x/\text{Giovanni}\}$$

$$\frac{\text{preserve}(x,e) \Rightarrow \neg \text{leavegarbage}(x) \quad \text{preserve}(g,e)}{\neg \text{leave}(\text{Giovanni})} \quad \{x/g\}$$

$$\frac{\text{preserve}(x,e) \Rightarrow \neg \text{lightfire}(x) \quad \text{preserve}(g,e)}{\neg \text{lightfire}(\text{Giovanni})} \quad \{x/g\}$$

3) CNF:
(a) loves(G, N)
(b) $\neg$ loves(X, N) $\vee$ preserve(X, E)
(c) $\neg$ preserve(X, E) $\vee \neg$ lightfire(X)
(d) $\neg$ preserve(X, E) $\vee \neg$ leavegarbage(X)

RESOLUTION:
negated goal $\rightarrow \neg$ goal $= \neg(\neg$ leavegarbage(G) $\wedge \neg$ lightfire(G)) $\rightarrow$ (e) leavegarbage(G) $\vee$ lightfire(G)
(a) and (b) with $\{X/G\} \rightarrow$ preserve(G, E) (1)
(1) and (c) with $\{X/G\} \rightarrow \neg$ lightfire(G) (2)
(1) and (d) with $\{X/G\} \rightarrow \neg$ leavegarbage(G) (3)
(2) and (e) $\rightarrow$ leavegarbage(G) (4)
(3) and (4) $\rightarrow \{\}$

(13) Unification

① $f(cons(car(X), cdr(Y)), Z, X)$ and $f(Z, Z, cons(car(X), cdr(a)))$

$\langle cons(car(X), cdr(Y)), Z \rangle$ $\quad \langle Z, Z \rangle$ $\quad \langle X, cons(car(X), cdr(a)) \rangle$

$\{Z / cons(car(X), cdr(Y))\}$ $\quad$ ↳ deep cause identity $\quad \{X / cons(car(X), cdr(a))\}$

error → cant unify!

② $f(g(x,a), g(b,a))$ and $f(y,y)$

$\langle g(x,a), y \rangle$ $\quad \{y / g(x,a)\}$

$\langle g(b,a), y \rangle$ $\quad \{y / g(b,a)\}$

→ $\langle g(x,a), g(b,a) \rangle$

$\langle x, b \rangle$ $\quad \langle a, a \rangle$

$\{x/b\}$ → success $\{y / g(b,a), x/b\}$

Note this!

③ Unification in Prolog $\quad P([x|[a,b]])$ and $P([a|[a|[x_s]]])$

$\langle x, a \rangle$ $\quad \{x/a\}$

$\langle [a,b], [a|[x_s]] \rangle$

⇓

$\langle [a,[b]], [a|[x_s]] \rangle$

$\langle a, a \rangle$ $\quad \langle [b], [x_s] \rangle$

⇓

$\langle b, x_s \rangle$ $\quad \{x_s/b\}$ $\quad \langle [], [] \rangle$

→ $\{x/a, x_s/b\}$

(14) Represent in FOL

a: Italian soccer players are better than french ones

b: The best italian soccer player is better than some French ones

a: $\forall x \forall y \; (ISP(x) \wedge FSP(y) \Rightarrow better(x,y))$

b: $\exists x \; ISP(x) \wedge [\forall y \; ISP(y) \Rightarrow better(x,y)] \wedge [\exists y \; FSP(y) \wedge better(y,x)]$

↑ better than other italians ↑ at least one FSP is better than him

(15) Tell if following two expressions unify

- $cons(a, cons(b, cons(X, [])))$
- $cons(Y, cons(Y, cons(b, [])))$

1: $\langle a, Y \rangle$ $\quad \{Y/a\}$

$\langle cons(b, cons(X, [])), cons(Y, cons(b, [])) \rangle$

⇓

$\langle cons(b, cons(X, [])), cons(a, cons(b, [])) \rangle$

cant unify cause cant substitute a constant (b) with an other constant (a)

(16) Unification and MGU

1: $p(X1, f(X1))$ and $p(g(X2), f(g(a)))$

$\langle X1, g(X2) \rangle$ $\quad$ $\langle f(X1), f(g(a)) \rangle$

$\{X1/g(X2)\}$ $\quad$ $\Downarrow$

$\langle f(g(X2)), f(g(a)) \rangle$

$\{X2/a\}$ $\quad \Rightarrow \{X1/g(a), X2/a\}$

2: select(X, [X|Xs]) and select(Y, [a,b,c])

$\langle X, Y \rangle$ $\quad$ $\langle [X|Xs], [a|[b,c]] \rangle$

$\Downarrow$ $\quad$ $\{X/a, Xs/[b,c]\}$

$\langle a, Y \rangle$

$\{Y/a\}$ $\quad \Rightarrow \{X/a, Xs/[b,c], Y/a\}$

(17) Unification

1: $f(g(a,X), g(X,b)) = f(g(a,b,c,d))$ not possible, different number of arguments

2: $f(g(a,X), g(Y,Y)) = f(g(a,b), g(f(a), f(z)))$

$\langle g(a,X), g(a,b) \rangle$ $\quad$ $\langle g(Y,Y), g(f(a), f(z)) \rangle$

$\{X/b\}$ $\quad$ $\{Y/f(a)\}$

$\Downarrow$

$\langle g(f(a), f(a)), g(f(a), f(z)) \rangle$

$\{z/a\}$ $\quad \Rightarrow \{X/b, Y/f(a), z/a\}$

3: $f(cons(cons(a,b))) = f(cons(cons(a, nil))$ not possible, can't substitute nil with b