Grundlagen der KI V01 : Introduction

- What is AI? → 4 Schools of thought.

	Human-like behavior	Rational behavior
Reasoning	Sys. that think humanly	... think rationally
Acting	... act humanly	... act rationally

- How to test acting humanly?

  Touring-Test: Are answers to written questions from a computer or human?

  → Computer needs these capabilities:
  * Natural Language Processing (NLP) → to communicate
  * Knowledge representation → to store knowledge
  * Automated Reasoning → use stored info
  * Machine Learning → adapt & detect + explore patterns

- Total Touring Test: Additionally requires the subject to interact with and see the other person.

  → Computer needs:
  * Computer Vision
  * Robotics

- How to test ~~thinking humanly~~ whether machines think like humans?

  * Understand human mind via → Introspection
    → psychological experiments
    → brain imaging

  ⇒ Thinking humanly approach:
  Write a computer program of the mind. If the Input-Output-Pattern looks human, the program might correspond to human thinking.

- What is rationality? → If a system does the "right thing" (it has an ideal performance)

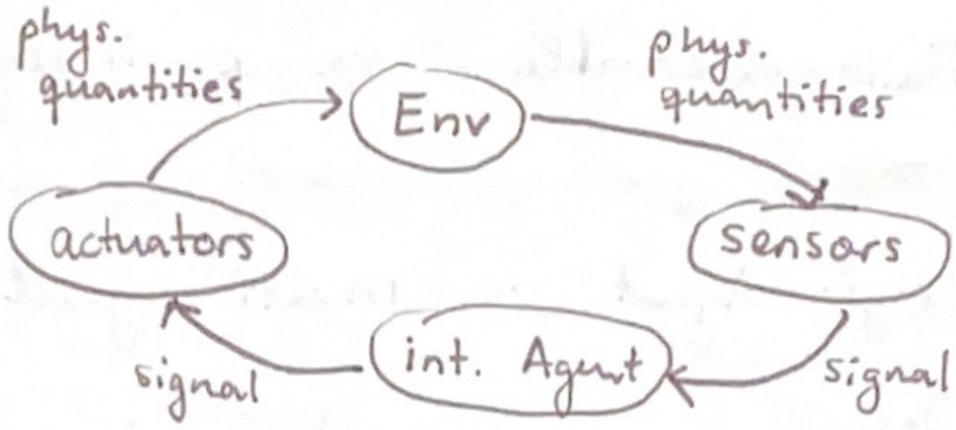
  ⇒ Usage of Logics → Needs formalized knowledge
  → Computational complexity

- Agent : Something that acts.
- Rational agent : An agent that acts so as to achieve the best outcome.
- What are enabling technologies ?
  - computers
  - embedded sys.
  - sensors & actuators
  - user interaction
  - robots
  - internet
  - computation grid

- Technological Singularity :
  → AI will exceed human intellectual capacity and control, possibly ending civilisation

- Definition: Intelligent Agent

→ Anything that
- perceives the environment through sensors
- acts in the environment through actuators



- Percept sequence: The complete history of its perception.
- Agent function: Maps a percept sequence to an action.
  - ↳ describe agent behavior
  - → longer percept sequences allow for smarter actions
  - * Tabular agent functions
  - * Agent programs (more practical)

- What is rational at any given time?

Maximizing the expected performance, by selecting the best actions given
- some performance measure
- prior percept sequence
- built-in knowledge
- possible actions

- What is an omniscient agent?

→ knows the actual outcomes of its actions

- What is learning?

Rational agents learn from perception (improve their knowledge of the env.)

- What is autonomy in AI?

If the agent relies less on prior knowledge and uses newly learned abilities instead.

- How is the task environment described?

→ Performance measures
Environment
Actuators
Sensors

- Properties of task environments:

* Fully-observable vs. partially observable
  ↳ the agent can detect the complete state of the env.
* Single-Agent vs. multi-agent
* Deterministic vs. stochastic
  ↳ if the next state is fully determined by its current state and the action

det. & fully obs. ⇓ rational agent always wins or draws

* Episodic vs. sequential
  ↳ if the actions taken in one episode do not affect later episodes
* Discrete vs. continuous (in space & time)
* Static vs. dynamic
  ↳ if the env. only changes based on the agent's actions
* Known vs. unknown
  ↳ if the outcome (probabilities) of an action are known (else they must be learned)

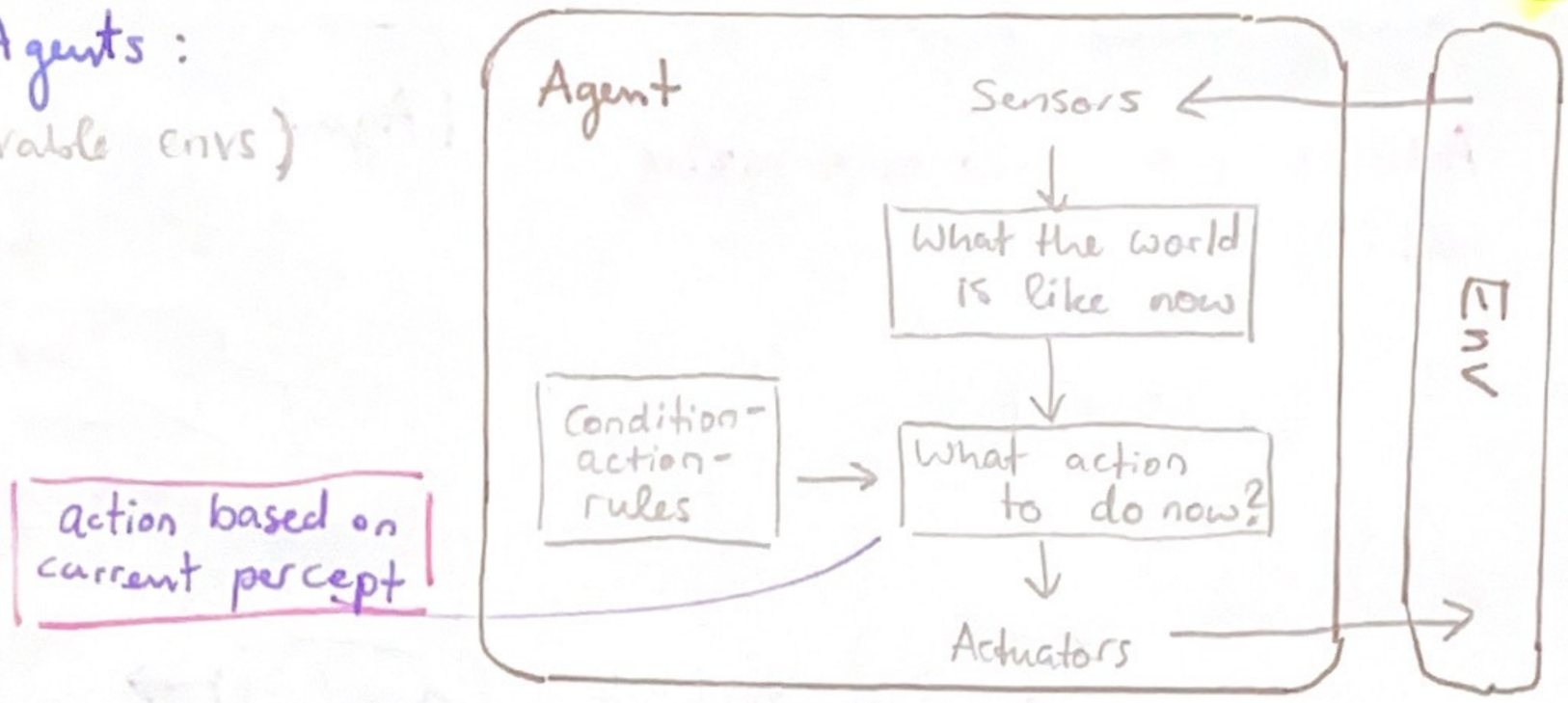
- Agent types:

- Simple reflex agent
- Reflex agents with state
- Goal-based agents
- Utility-based agents

↓ Generality

- Simple Reflex Agents:

(in fully-observable envs)



- Model-based Reflex Agents:

Handles partial observability using a world model.

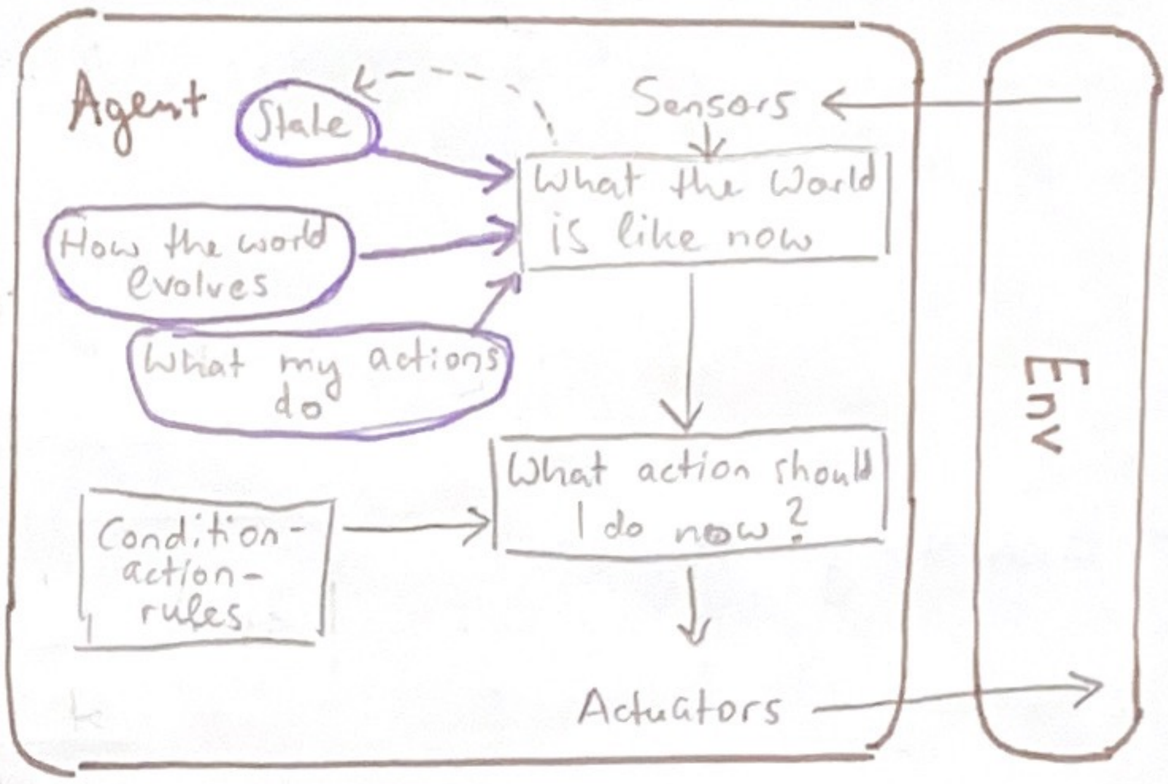
* keeps an internal state of the previous situation

* How the world evolves

(Example of "disappearing" pedestrian)

* What my actions do:

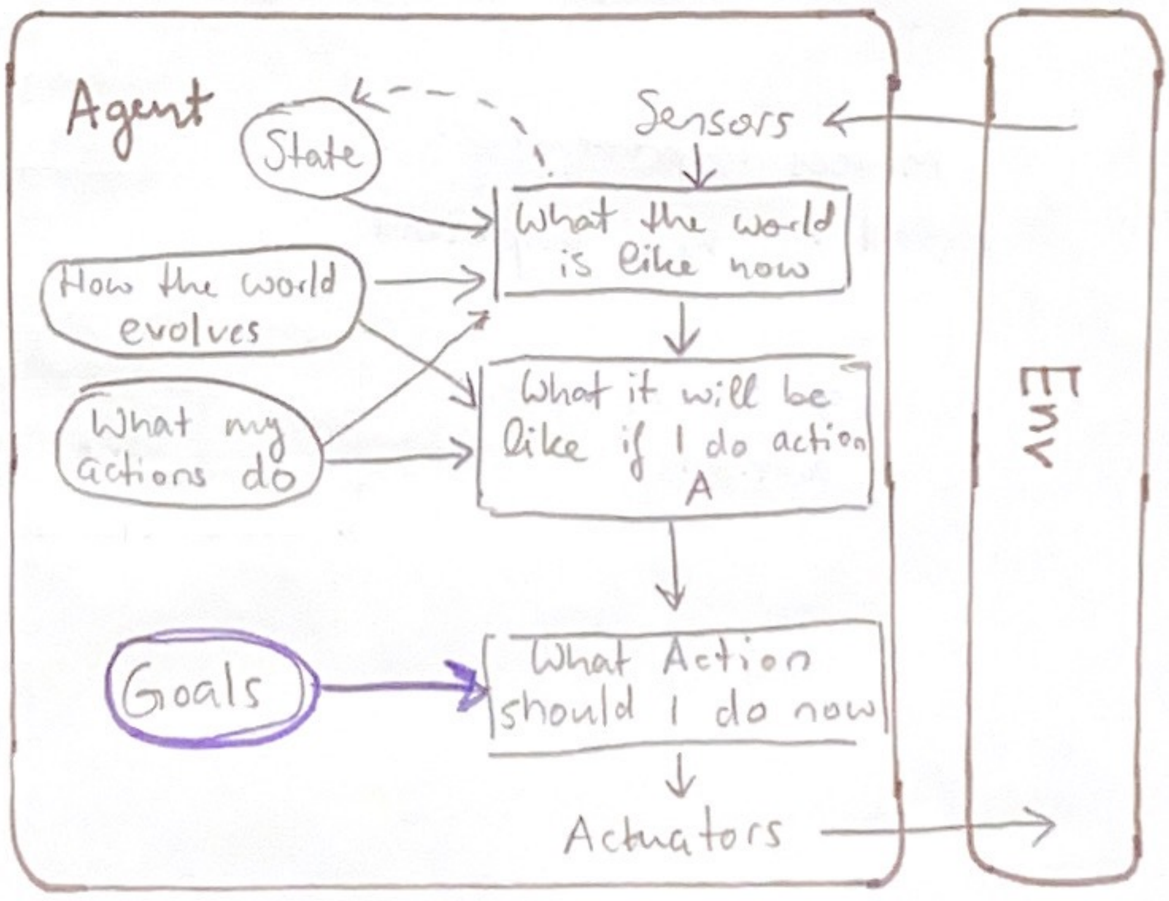
e.g.: accelerate → incr. speed



- Goal-based Agents:

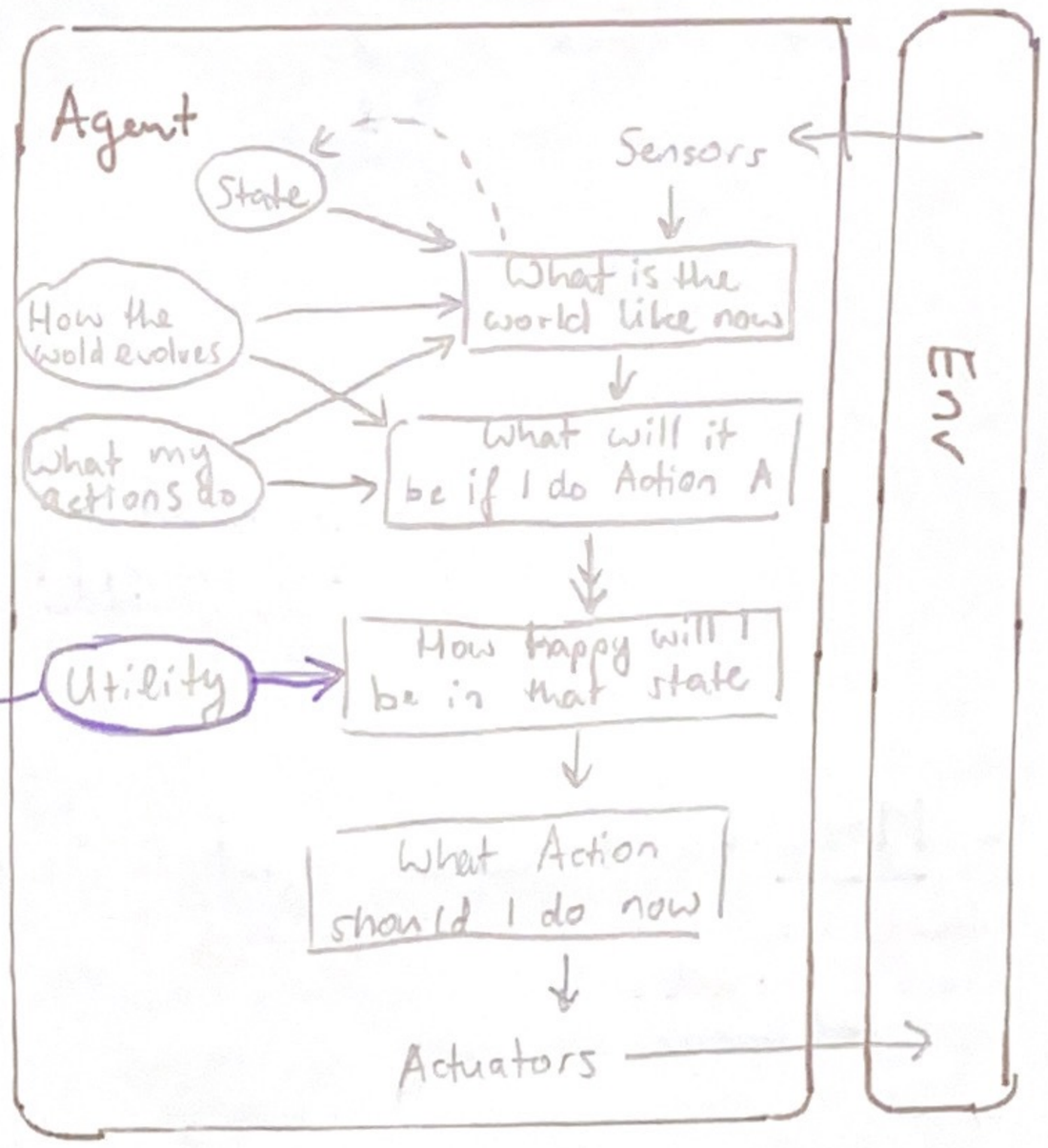
Additionally keep track of a goal

* Goal is reached by searching & planning



- Utility-based Agents:

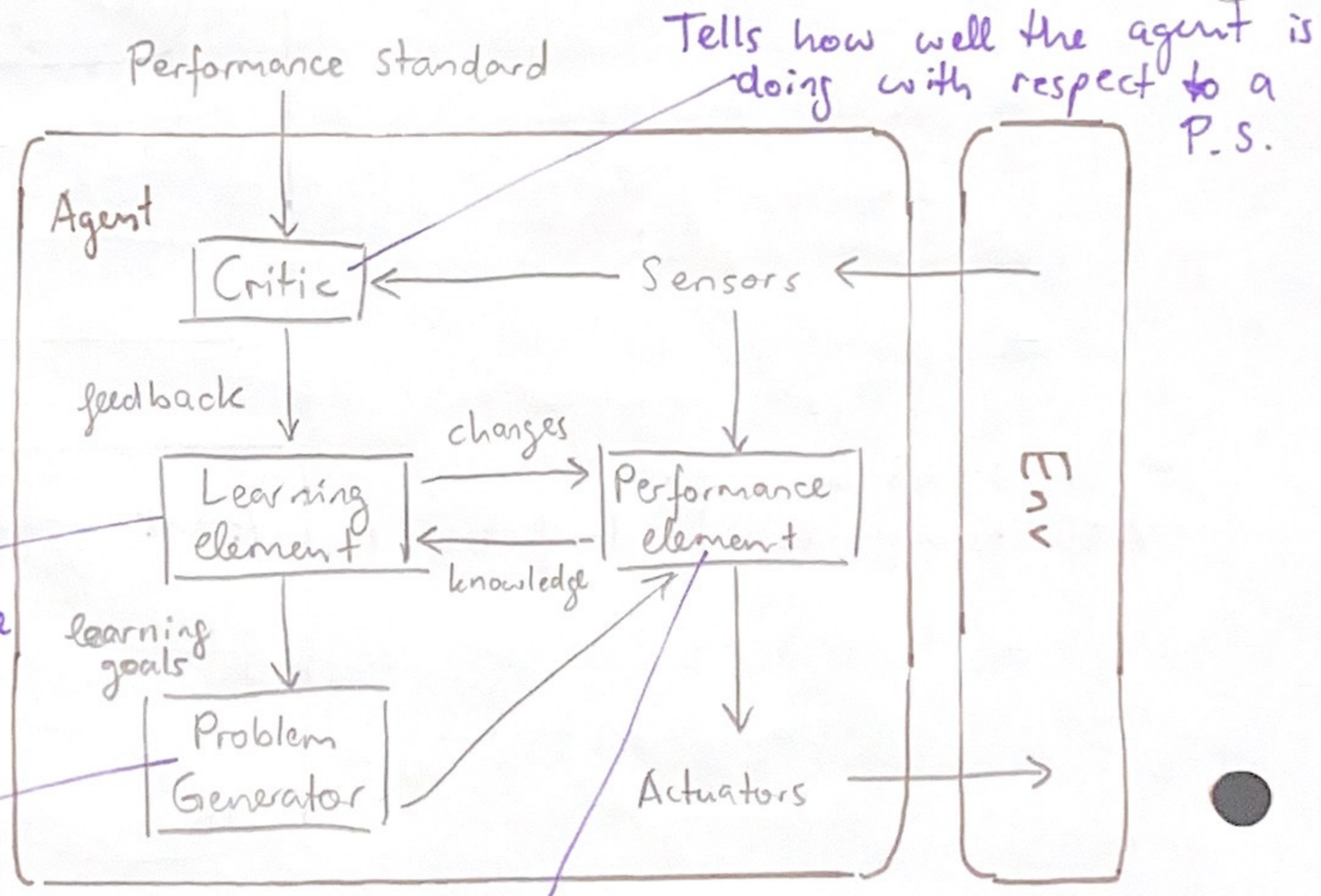
Achieve goal while maximizing utility



Allows for trade-off between different goals

- Learning Agents

* Any agent can be extended into a learning agent



makes improvements based on new experience

Suggest explorative actions

the "entire agent"
→ selects the action

Grundlagen der KI 3: Uninformed Search

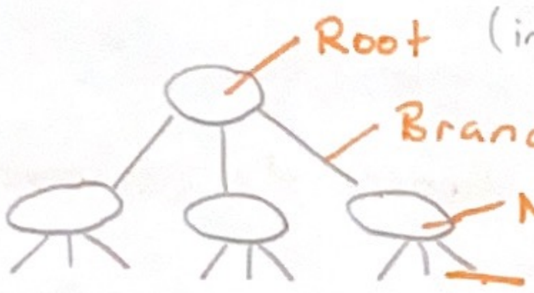
- What's a well-defined problem?

→ 5 Components:
1) Initial state of the agent
2) Description of possible actions
3) Transition model (what each action does)
4) Goal test (whether a state is the goal)
5) Path cost function

Can be stored as a directed graph

→ Examples:
* Route-Finding problem
* Automatic assembly sequencing
* Layout of digital circuits.

- What's a search tree? "Possible actions from the initial state"



- How does the tree search alg. work? (w/o green lines)

function Tree-Search (problem) returns solution or failure

initialize the frontier (set of all leaves available for expansion) using the initial state of problem
initialize explored set to be empty
loop do
  if frontier is empty return failure
  choose leaf node and remove it from frontier
  if the node is the goal state return the solution
  add the node to the explored set
  expand the chosen node, add children to the frontier only if not in frontier or explored set

* Does not always find a solution if it exists
→ since the search tree of a finite graph is not always finite
* good space complexity

- How does the graph search algorithm work? ↑ (w/ green lines)

⇒ Basically tree search, but does not expand nodes that were visited before (has an explored-set) set of nodes that were expanded

- How do we measure performance?
* Completeness (if a solution is guaranteed if it exists)
* Optimality (Does it find a solution w/ min. cost)
* Time complexity
* Space Complexity (memory)

Uninformed Search vs. informed search:

- informed search:
  - → Strategies know whether a state is ~~better~~
  - → uses measures/heuristics to indicate distance to goal
- Uninformed search:
  - → No additional information besides problem statement
  - → Can only produce next states and check if its a goal state

- How does Breadth-First-Search work?

```
function BFS (problem) returns solution or failure
node ← node with state = problem.Initial-State, Path-Cost = 0
if problem.Goal-Test(node.State) return Solution(node)
frontier ← FIFO queue with element node
explored ← empty set
loop do
    if Empty(frontier) return failure
    node ← Pop(frontier)   // chooses the shallowest node in frontier
    add node.State to explored
    for each action in problem.Actions(node.State) do
        child ← Child-Node(problem, node, action)
        if child.State is neither in explored nor frontier
            if problem.Goal-Test(child.State) return Solution(child)   // Goal test performed after selection of child
            frontier ← Insert(child, frontier)
```

(shallowest goal) — depth

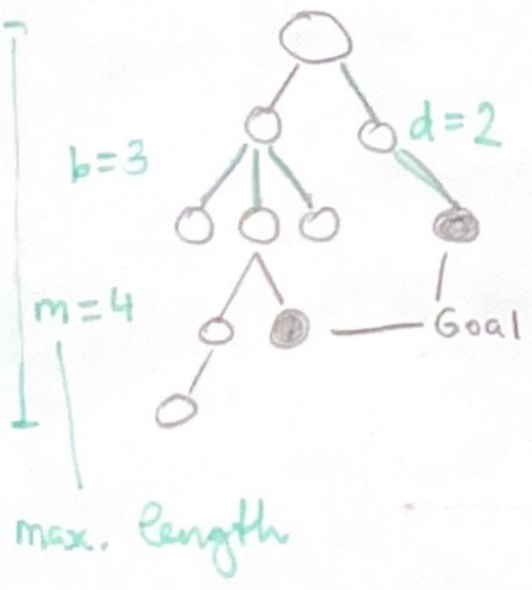
Completeness:
Yes, if d and b are finite.

b / branching factor (max. # of successors of any node)

Optimality:
Yes, if cost equal per step

Time Complexity:
worst case
$b + b^2 + \ldots + b^d = O(b^d)$

Space Complexity:
explored: $O(b^{d-1})$
frontier: $O(b^d)$



→ BFS always expands the shallowest nodes
=> is optimal for equal step costs.

→ The non-recursive implementation of DFS would be the same, except that it uses a LIFO queue.

→ BFS finds the shallowest solution

- Uniform-Cost-Search (Dijkstra's algorithm)

↳ expands the nodes with the lowest path-cost
{ ⇒ The frontier is a priority queue ordered by the cost g

```
function UCS (problem) returns solution or failure
node ← node with State = problem.Initial-State, Path-Cost = 0
frontier ← priority queue by Path-Cost with the node element
explored ← empty set
loop do
    if Empty(frontier) return failure
    node ← Pop(frontier)   // chooses the lowest-cost node in frontier
    if problem.Goal-Test(node.State) return Solution(node)   // Goal test when selecting a node for expansion
    add node.State to explored
    for each action in problem.Actions(node.State) do
        child ← Child-Node(problem, node, action)
        if child.State is neither in explored nor frontier
            frontier ← Insert(child, frontier)
        else if child.State is in frontier with higher Path-Cost   (extra check)
            replace that frontier node with child
```

CHANGES COMPARED TO BFS

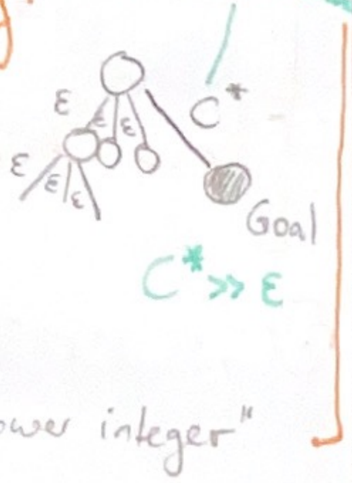
Goal-Test is applied when node is selected for expansion ⇒ optimal path

In case a better path to a frontier node is found

[Completeness: Yes, if costs are greater than 0]

[Optimality: Y, if cost ≥ ε > 0]
ε, minimum step cost

[Time Complexity:
Worst Case:
$(b-1)+(b-1)b+(b-1)b^2+\dots+(b-1)b^{\lfloor C^*/\varepsilon \rfloor} = O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
"next lower integer"



]

[Space Complexity: Same as Time C.]

- What's the performance of Depth-First-Search?

[Completeness:
No, if recursively implemented.
Yes, if repeated states are avoided and the state space is finite]

[Optimality:
No, since costs are not considered]

[Time Complexity:
Goal path is tested last: $O(b^m)$]

[Space Complexity: $O(bm)$]

→ Shortcoming:
Does not terminate in infinite space.

(3) Depth-Limited-Search: (recursive form) (exercise uses loop form with frontier + explored set)

main {

```
function DLS(problem, limit) returns solution or failure/cutoff.
return Recursive_DLS(Make-Node(problem.Initial-State), problem, limit)
```

}

[if limit is ∞, this is Depth-First-Search.]

```
function Recursive-DLS(node, problem, limit) returns solution or failure/cutoff
if problem.Goal-Test(node.State) return Solution(node)
else if limit = 0 return cutoff
else
    cutoff-occured ← false
    for each action in problem.Actions(node.State)
        child ← Child-Node(problem, node, action)
        result ← Recursive-DLS(child, problem, limit-1)
        if result = cutoff then cutoff ← true
        else if result ≠ failure return result
    if cutoff-occured return cutoff else return failure
```

[Completeness: No, if $l < d$.] ← limit

[Optimality: No, if $l > d$.]

[Time Complexity: $O(b^l)$]

[Space Complexity: $O(b \cdot l)$]

↳ One does not know the depth of the goal state.

- Iterative Deepening Search: Use DLS and iteratively increase l.

```
function IDS(problem) returns solution or failure
for depth = 0 to ∞ do
    result ← DLS(problem, depth)
    if result ≠ cutoff return result
```

[Completeness: Yes, if d is finite]

[Optimality: Yes, if equal costs]

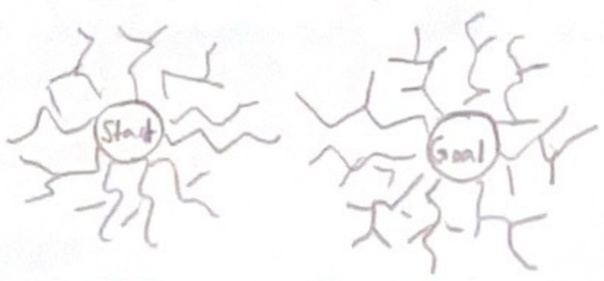
[Time Complexity: $O(b^d)$]

[Space Complexity: $O(b \cdot d)$]

(-) What's Bidirectional Search?

→ Run separate Searches from initial and goal state.

Motivation: $b^{d/2} + b^{d/2} < b^d$



Only works when all actions are reversible and the goal state is known.

Grundlagen der KI V04: Informed Search

↳ Uses indications how good a state is to reach the goal
→ More efficient than uninformed search

* Require a heuristic function $h(n)$ that estimates the cost from node n to the goal

⇒ presented informed search alg. are identical to Uniform-Cost-Search, but the evaluation function $f(n)$ is used in the priority queue
↳ chooses next node

What's greedy best-first search? (GBFS)

→ Expands the node that is closest to the goal by just using the heuristic function ⇒ $f(n) = h(n)$

Heuristic function (problem-specific with $h(n) \geq 0$ and $h(\hat{n}) = 0$ when $\hat{n}$ is the goal node)

[Completeness: Yes, if graph search is used.]

[Optimality: No.]

[Time Complexity: Worst case with misleading heuristic $O(b^m)$]

[Space Complexity: $O(b^m)$]

What's A* Search? (Combines ideas from UCS & GBFS)

→ Combines path cost and estimated cost to the goal:

$$f(n) = g(n) + h(n)$$

$h(n)$ must be admissible / lower than actual cost: $h(n) \leq g(n, goal)$
↳ underapprox. guarantees Optim.

→ Application to graph search requires consistency: $h(n) \leq c(n, a, n') + h(n')$
transition cost

→ Only paths with $f(n) \leq C^*$ are expanded
C^*: cost of opt. path

↳ Nodes with $f(n) > C^*$ are never expanded
their subtrees are pruned automatically

[Completeness: Yes, if costs > 0]

[Optimality: Yes, if costs > 0. Heuristic must be admissible in tree search and consistent in graph s.]

[Time Complexity: $O(b^{\varepsilon d})$ with relative error $\varepsilon = \frac{h^* - h}{h^*}$ — estimated / actual } costs to goal]

[Space Complexity: $O(b^{\varepsilon d})$]

What are Alternatives of A* Search? → Aim to eliminate the huge space consumption

* Iterative-Deepening A*: f-cost (g+h) is used for cutoff (limit)
* Recursive best-first search: (similar to recursive depth-first-search)
  → keeps track of f-value of best alternative path
* Memory-bounded A*: Works like A* until memory is full, then drop less promising paths.

- How is the Quality of a Heuristic characterized?

→ Effective branching factor b^* = The average number of nodes that needs to be expanded at each step to reach a solution

# of nodes generated from A* search — uniform tree with depth d

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

↳ found iteratively

- When is a heuristic better than another one?

→ If for every node $h_2(n) \geq h_1(n)$, then h_2 dominates h_1

Since only nodes with $f(n) \leq C^*$ are expanded

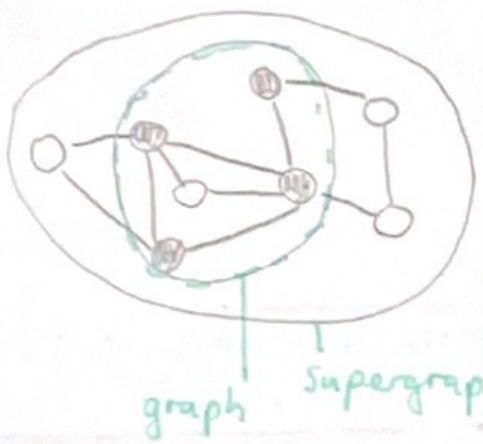
$$\Rightarrow h(n) \leq C^* - g(n)$$

($g(n)$: fixed constant)

$$\Rightarrow h_1(n) < h_2(n) \leq C^* - g(n)$$

h_2 expands fewer nodes than using h_1

- How is a heuristic derived from a relaxed problem?



a problem with more freedom
(the graph is a supergraph of the original one / it has additional edges)

→ The optimal solution of the relaxed problem is an underapproximative solution of the original problem

- How can multiple heuristics be combined?

$$h(n) = \max\{h_1(n), h_2(n), \dots\}$$

- How are heuristics derived from pattern databases?

→ Solve a subproblem ⇒ solution costs are underapproximations

* The number of subproblems must be small so it can be stored

Grundlagen der KI 05: Constraint Satisfaction problems

- What's a Constraint Satisfaction Problem? (CSP)

  → Is tuple (X, D, C)

  - $C = \{C_1, \dots, C_m\}$: Constraints — $\langle$scope, rel$\rangle$ — Each is tuple ("tuple of participating variables", "relation that defines possible vals")
  - $D = \{D_1, \dots, D_n\}$: Set of domains — allowed values for each variable X
  - $X = \{X_1, \dots, X_n\}$: Variables

  * Uses factored representation of each state

  Benefit: General-purpose algorithms can be used

  → more power than standard search

- What are possible constraints?

  → Goal: Assign a value to each variable such that all constraints are satisfied

	# of variables involved	
Unary	1	i.e. A ≠ green
Binary	2	i.e. A ≠ B
Higher-Order	≥ 3	i.e. A ≠ B ≠ C

or as Preferences (cost for each variable assignment)

- What's the standard (naive) search formulation?

  * Path is irrelevant
  * Solutions appear at depth n with n vars.

  ⇒ Use DFS

  → Initial state: $\emptyset$

  → Successor function: Assign a value to an unassigned variable according to constraints

  → Goal test: If the assignment is complete

  → # of searched Nodes: $n! d^n$ in worst case $[b = (n - \ell) d]$

  (n: # of vars, d: # of values, b: # of splits, ℓ: depth)

- What's Backtracking search?

  → Only consider assignments to a single variable for the children at each node. → $[b = d]$ with d^n leaves

  (b: branching factor, d: domain size)

  → Uses DFS

- Backtracking search algorithm:

```
function BS(csp) returns solution/failure
return RB({}, csp)
```

```
function RB(assignment, csp) returns sol/failure
if assignment is complete return assignment
var ← Select-Unassigned-Variable(csp)        ← Which variable to select next?
for each value in Order-Domain-Values(var, assignment, csp)        ← In what order to try values?
    if value is consistent with assignment given Constraints[csp]
        add {var = value} to assignment
        • inferences ← Inference(csp, var, value)
        • if inferences ≠ failure:
            • add inferences to assignment
            result ← RB(assignment, csp)
            if result ≠ failure return result
        • remove {var = value} and inferences from assignment
return failure
```

Can we detect inevitable failure early?

* Variable Selection
  - → MRV Heuristic: Minimum # of remaining values
  - ⇒ Degree Heuristic: Var. with most constraints with unassigned neighbors
* Value Selection:
  - → least-constraining-heuristic: Val. that constraints neighboring vars least

- What's inference?

  Act or process of deriving logical conclusions from known premises.

  Inference techniques: (Prune the search tree)
  - → Forward-Checking (inconsistent vals of neighbor vars are removed)
  - → Arc Consistency algorithm (inconsistent values of all vars are removed)
    - ↳ faster than w/ forward checking

- What's Arc consistency?

  of a variable: X_i is arc-cons. with Y_j if for every value in D_i there is a value in D_j satisfying the binary constraint of $arc(X_i, Y_j)$

  of a CSP: A constraint graph is arc-consistent if every variable is arc-consistent with every other variable.

- Arc Consistency algorithm:

Procedure:
→ Use table

queue	removed?
(V_4, V_3)	true
(V_2, V_3)	:
(V_4, V_3)	:
(V_2, V_1)	
:	

check if values need to be removed from dom(V_1) to satisfy arc(V_1, V_3) constraint

Then add neighbors of V_1 w/o V_3 2

```
function AC-3(csp, queue) returns failure or the reduced csp
while queue is not empty                    ← the queue is a set, such that elements are not added again
    (X_i, X_j) ← Remove-First(queue)
    if Remove-Inconsistent-Values(X_i, X_j)     → Check if X_i is arc-cons. with X_j and remove values if needed
        if size of Domain(X_i) = 0              → if all values were removed
            return failure
        for each X_k in Neighbors[X_i] \ {X_j}      "without"
            add (X_k, X_i) to queue
                                    ← Neighbors of X_i
return csp
```

→ Arc-Cons. Alg. can be applied as preprocessing step before BS
=D The BS-Alg. does not backtrack (faster than BS with Arc-cons. as inference)

```
function Remove-Inconsistent-Values(X_i, X_j) returns true if succeeds
removed ← false
for each x in Domain[X_i]
    if no value y in Domain[X_j] allows (x,y) to satisfy constraint of [X_i, X_j]
    then delete x from Domain[X_i]; removed ← true
return removed
```

- How is the complexity reduced through independent problems?

Suppose each subproblem has c variables out of n total.

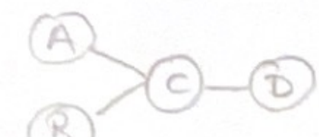
subprobs have no connections in the graph

⇒ Cost: $\frac{n}{c} d^c < d^n$

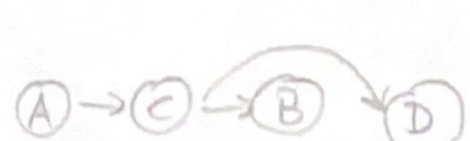
with subprobs. full problem

- How to solve tree-structured CSPs?

→ Constraint graphs without loops, e.g.



1) Choose start node & Apply topological sort: $\mathcal{O}(n)$



2) Check arc-consistency along the sorted graph: $\mathcal{O}(nd^2)$ (for each node the values of two variables are compared)

3) March down the list of variables in directed arc-consistent graph and choose remaining variable.

How to obtain and use Nearly tree-structured CSPs?

→ Conditioning: instantiate a variable and prune its neighbors domains

→ Problem: a variable needs to be chosen

→ Construct a supertree

Grundlagen der KI 06: Logical Agents (VO 6+7)

- **Knowledge:** "Int. agents need knowledge about the world to reach good solutions. Otherwise you only use search."

  * Knowledge base: set of sentences in formal language

  → Gain new knowledge via: Inference / Declarative approach (from outside) / Perception

  → Agents are viewed at: knowledge level (what they know) / implementation level (data structures & algorithms)

(-) Basics of Logic:

→ Syntax: Specifies how correct sentences are formed.

Semantics: defines the meaning of sentences, i.e., when they're true.

Model: Instances which **evaluate sentences to true or false.**

Satisfaction: If sentence α is true in model m, then m satisfies α.

Entailment: When the truth of one sentence requires the truth of the other sentence:

$$\alpha \models \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

"if the models that eval α are contained in the models which eval β."

Does not entail: $\not\models$

(-) Propositional Logic ("Aussagenlogik"):

→ assumes the world contains facts

"α entails β"

If S_1 and S_2 are sentences

* Negation:	$\neg S$
* Conjunction:	$S_1 \wedge S_2$
* Disjunction:	$S_1 \vee S_2$
* Implication:	$S_1 \Rightarrow S_2$
* Biconditional:	$S_1 \Leftrightarrow S_2$

Backus-Naur-Form

$S ::= AP$
$\neg S$
$S_1 \wedge S_2$
$S_1 \vee S_2$
$S_1 \Rightarrow S_2$
$S_1 \Leftrightarrow S_2$
(S)

P	false	false	true	true
Q	false	true	false	true
$P \Rightarrow Q$	true	true	false	true
$P \Leftrightarrow Q$	true	false	false	true

- Whats a simple technique for checking $KB \models \alpha$? "Knowledge base entails α"

→ Enumerate all models and check whether α is true in every model in which KB is true.

⊖ Theorem Proving:

→ Instead of enumeration, apply rules of inference

Logical equivalence: Two sentences are logically equivalent if they are true in the same set of models

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$
(entailment on both sides)

Connections:

α is valid if $\neg\alpha$ is unsatisfiable

Validity: A sentence is valid if it's true in all models
(valid sentence = tautology) [e.g. $(\neg P \vee P)$]

Satisfiability: If a sentence is true in some model

Apply in this order to show validity, satisfiability, unsatisfiability

Modus Ponens: $\dfrac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$ (If α implies β and α are true, then β is true)

And-Elimination: $\dfrac{\alpha \wedge \beta}{\alpha}$

Standard logical equivalences:

(also for "∨" instead of "∧")

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity

$\neg(\neg\alpha) \equiv \alpha$

(1) Implication elimination: $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$

(2) Biconditional elimination: $(\alpha \Leftrightarrow \beta) \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

De Morgan: $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$

(3) Distributivity of ∧ over ∨: $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$

- How to automate theorem proving?

→ Use Search:
* Initial state: Knowledge base
* Actions: Inference rules
* Result: ...
* Goal: State with the sentence to prove

* Incomplete if important theorems are missing

Full Resolution Rule:

example:

$$\frac{A \vee B \quad , \quad \neg B \vee C}{A \vee C}$$

$$\frac{\ell_1 \vee \dots \vee \ell_k \quad , \quad m_1 \vee \dots \vee m_n}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_{j-1} \vee \dots \vee m_{j+1} \vee \dots \vee m_n}$$

where ℓ_i and m_j are complmentary literals ($\ell_i = \neg m_j$)

Conjunctive Normal Form:

ex.:

- $(A \vee B \vee C) \wedge (\neg A \vee B \vee C)$
- $A \wedge B \wedge C \wedge (\neg A \vee B \vee C)$
- $(\neg A \vee B) \wedge (\neg C) \wedge (\neg A \vee B \vee C)$

$\bigwedge_i \bigvee_j (\neg) x_{ij}$

"or" within clauses

"and" between clauses

- How does the resolution algorithm work?

→ Proof by contradiction: Show that $KB \models \alpha$, by showing that $KB \wedge \neg \alpha$ is unsatisfiable.

1. Convert $KB \wedge \neg \alpha$ into CNF
2. Apply resolution rule to the resulting clauses
3. until:
   - no new clauses
     $KB \not\models \alpha$
   - empty clause:
     $KB \models \alpha$

```
function PL-Resolution(KB, α) returns true or false
clauses ← set of clauses in CNF representation of KB ∧ ¬α
new ← {}
loop do
    for each pair of clauses Ci, Cj in clauses do:
        resolvents ← PL-Resolve(Ci, Cj)
        if resolvents contains the empty clause return true
        new ← new ∪ resolvents
    if new ⊆ clauses return false
    clauses ← clauses ∪ new
```

→ Resolution closure RC(S) = set of all clauses derivable (for clauses S) by repeated application of the resolution rule

→ Ground Resolution theorem:
If a set of clauses S is unsatisfiable, then RC(S) contains the empty clause.

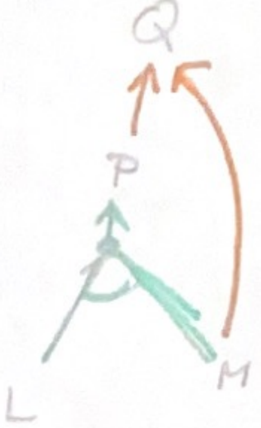
- What's a horn clause?

- proposition symbol; or (e.g. $L_{1,1}$)
- (conjunction of symbols) $\Rightarrow$ symbol (e.g. $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$)

→ Only requires Modus Ponens as inference method

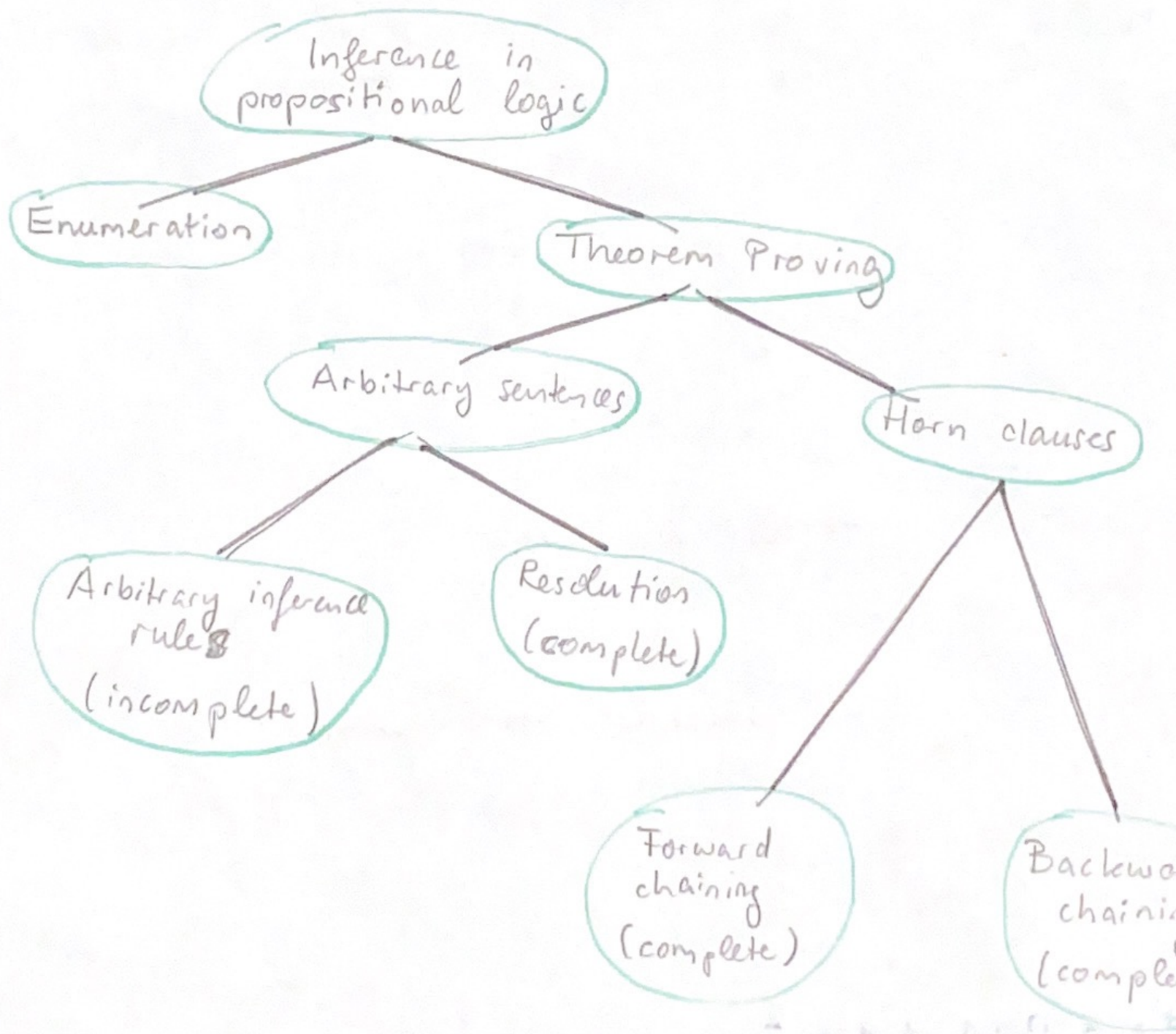
- AND-OR Graph:

  - Conjunction: Links joint by an arc
  - Disjunction: Links joint without an arc



lin. time complexity
- → Forward chaining: Inference (starting) from KB
  - ↳ Data-driven → object recognition (does lots of work irrelevant to the goal)
- → Backward chaining: 1) See which rules must be proven to infer the query
  - 2) Then prove only the required ones
  - ↳ Goal-driven → problem solving (comp. effort can be much less than linear)

- Overview of inference methods:



Grundlagen der KI 07: First-Order-Logic

- What are advantages & disadvantages of propositional logic?

  - ⊕ Is **declarative** (pieces of syntax correspond to facts)
  - ⊕ Allows partial / disjunctive / negated info (e.g. "Pit in [2,2] or [3,1])
  - ⊕ Is **compositional** ($B_{1,1} \wedge P_{1,2}$ is derived from $B_{1,1}$ and $P_{1,2}$)
  - ⊕ Meaning is **context independent** (unlike nat. language)
  - ⊖ Limited **expressive** power (cannot express general rules like nat. lang.)
    → e.g. in Wumpus-world needs rules for every cell

⊖ First-Order-Logic: (FOL)

→ Assumes the world contains:

* Objects ("nouns")

* Relations ("verbs & adjectives") → unary: red, round
→ N-ary: brother of, >

* **Functions** (relations where each input is related to one output)
:= Mother | LeftLeg | ... (returns object)

→ Other **basic elements**:

* **Variables** := a | x | s | ...

* Connectives ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$)

* Equality (=) (signifies that two terms refer to the same object)

* **Quantifiers** := $\forall$ | $\exists$ — Universal quantification ("for all")
→ Typically, $\Rightarrow$ is the main connective with $\forall$
— Existential quantification. ("there exists")
→ Typically, $\wedge$ is the main connective with $\exists$

Operator precedence:
$=, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$ →

→ Syntax: (Backus-Naur-Form)

Sentence := AtomicSentence | ComplexSentence

AtomicSentence := Predicate | Predicate(Term, ...) | Term = Term

ComplexSentence := (Sentence) | [Sentence] | ¬Sentence | Sentence ∧ Sentence |
Sentence ∨ Sentence | Sentence ⇒ Sentence |
Sentence ⇔ Sentence | Quantifier Variable ... Sentence

Term := Function(Term, ...) | Constant | Variable
(logical expression that refers to an object)

Constant := A | X_1 | John | ...

Predicate := True | False | After | Loves | ...
(func that only returns true or false (representation of relations))

Input to a predicate must be an object!

⇒ Example: You can fool all of the people some of the time.

$\forall x, \exists t \quad person(x) \Rightarrow time(t) \wedge can\text{-}fool(x,t)$

(complex sentence)

Quantifiers : - The scope is marked with parenthesis

- A variable is bound by the innermost quantifier

↳ e.g. $\underbrace{\forall x (\ldots \wedge (\underbrace{\exists x \ \ldots}_{1}))}_{2}$ better to use diff. names

- $\forall x \forall y$ is the same as $\forall y \forall x$ and $\exists x \exists y$ is the same as $\exists y \exists x$

- Duality: $\forall x \ \neg P \equiv \neg \exists x \ P$

$\neg \forall x \ P \equiv \exists x \ \neg P$

$\forall x \ P \equiv \neg \exists x \ \neg P$

$\exists x \ P \equiv \neg \forall x \ \neg P$

- How to add sentences to a knowledge base and ask questions ?

→ Add sentences = Assertions

ex.: Tell (KB , King(John)) "John is a king"

→ Ask questions = Queries

ex: Ask (KB , King (John)) "returns true"

↳ With variable value

ex.: AskVars (KB , $\exists x$ King(x)) "returns $\{x/John\}$"

Grundlagen der KI 08: Inference in First-Order-Logic

- How to remove the Universal ~~Instantiation~~ Quantification Qualifier through inference?

Universal Instantiation

Infer any sentence from substituting a ground term (term w/o variables) → constant g

$$\frac{\forall v \ \alpha}{Subst(\{v/g\}, \alpha)}$$

Apply substitution $\{v/g\}$ to sentence α

→ Can be applied multiple times with diff. constants to get more sentences

* The new KB is logically equivalent when → performing all possible subst + del. the quantified sentence
  → ... & keeping it

— " — the ~~universal~~ existential Quantifier — " — ?

Existential Instantiation

→ Infer in the same way: $\frac{\exists v \ \alpha}{Subst(\{v/k\}, \alpha)}$

→ Can be applied once

* The new KB is satifiable iff the old KB was.
  (is not logically equivalent)

- Generalized Modus Ponens:

Problem → Propositionalization generates lots of irrelevant sentences (when applied for all possible constants)

Solution: For given facts p_i, p_i' and q, if there is a subst. θ for all i, such that $Subst(\theta, p_i) = Subst(\theta, p_i')$

Infer the Substitution

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{Subst(\theta, q)}$$

Finding the θ that satisfies this is called Unification → $Unify(p, q) = \theta$

e.g. $\forall x \ \underbrace{King(x)}_{p_1} \wedge \underbrace{Greedy(x)}_{p_2} \Rightarrow \underbrace{Evil(x)}_{q}$

$\underbrace{King(John)}_{p_1'}$

$\forall y \ \underbrace{Greedy(y)}_{p_2'}$ $\Rightarrow$ θ is $\{x/John, y/John\}$

→ Often multiple unifiers exist (θ that satisfy $Unify(p, q) = \theta$)

→ There is always a most general unifier

- Forward-Chaining-Alg.: → starts from known facts & adds conclusions

function FOL-FC-Ask (KB, α) **returns** a subsitution or *false*

repeat until new is empty
  new ← ∅
  for each sentence r in KB **do**
    $(p_1 \wedge \dots \wedge p_n \Rightarrow q)$ ← Standardize-Apart(r)
    for each θ such that
      Subst(θ, $p_1 \wedge \dots \wedge p_n$) = Subst($\theta$, $p_1' \wedge \dots \wedge p_n'$) for some $p_1', \dots, p_n'$ in KB
      q' ← Subst(θ, q)
      if q' is not a renaming of a sentence already in KB or new **then**
        add q' to new
        ϕ ← Unify(q', α)
        if ϕ is not fail **then return** ϕ
  add new to KB
return false

Datalog KB
= FO definite clauses + no functions
→ entailment is decidable

→ Complete for first-order definite clauses
→ Does not terminate in general, if function symbols are involved

Problem 6.2
- Backward-Chaining-Alg.: → starts from goal, searches rules to find new facts that support the proof

→ Uses DF-recursive proof search

function FOL-BC-Ask (KB, goals, θ) **returns** a set of substitutions

answers ← {$\emptyset$}
if goals is empty **return** {θ}
q' ← Subst(θ, First(goals)) (Subst. the first goal w/ all previous latest substitutions)
for each sentence r **in** KB where $(p_1 \wedge \dots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow$ Unify(q, q') succeeds (r is a conjunction of sentences that infers q; finding a subst. θ' such that Subst(θ', q) = Subst(θ', q'))
  Standardize-Apart(r) =
  recursively show new goals [which infer the current goal q']:
    new-goals ← [$p_1, \dots, p_n$ | Rest(goals)]
    answers ← FOL-BC-Ask(KB, new-goals, Compose(θ', θ)) ∪ answers ("combine")
return answers

* If the sentence has no "⇒", it does not have assumptions
↳ no new goals
$p_1 \dots p_n$ (are empty set)

⇒ To show KB ⊨ goal

- Logic Programming:

→ Analogy to ordinary programming

	Logic Programming	Ordinary programming
1	Identify problem	Identify problem
2	Assemble info	Assemble info
3	→ Use inference machine	Figure out solution
4	Encode info in KB	Program solution
5	Encode problem instance as facts	Encode problem instance as data
6	Ask queries	Apply program to data
7	Find false facts	Debug procedural errors

- Conjunctive Normal Form for FOL:

CNF → Identical to the one in propositional logic

Procedure for conversion → Every sentence in FOL can be converted into an inferentially equivalent CNF.

Example: "Everyone who loves all animals is loved by someone"

$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

① Eliminate implications ($A \Rightarrow B$ is $\neg A \vee B$)

$\Rightarrow \forall x\ [\neg \forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(x,y)]$

② Move $\neg$ inwards ($\neg \forall x\ p$ is $\exists x\ \neg p$; $\neg \exists x\ p$ is $\forall x\ \neg p$)

$\Rightarrow \forall x\ [\exists y\ \neg(\neg Animal(y)) \vee Loves(x,y)] \vee [\exists y\ Loves(x,y)]$

De Morgan

$\Rightarrow \forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(x,y)]$

③ Standardize Variables (to avoid confusion)

$\Rightarrow \forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(x,z)]$

④ Skolemization (process of removing $\exists$)

$\Rightarrow \forall x\ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$

$F(x)$ is just a func of all vars in the beginning

Skolem functions always return the correct y belonging to a given x

⑤ Drop universal quantifiers

$\Rightarrow [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), y)$

⑥ Distribute $\vee$ over $\wedge$

$\Rightarrow [Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$

} is in CNF

→ Then resolution

- Completeness of Resolution in FOL:

→ Refutation-Complete

↳ Attempting to prove a satisfiable FOL formula as unsatisfiable may result in a non-terminating computation

Grundlagen der KI 09: Bayesian Networks

- Overview of probabilistic models:

	Static env	Dynamic Env
W/o actions	Bayesian Netw.	Hidden Markov Models
w/ actions	Decision Netw.	Markov Decision Processes

→ Use probabilities to express the degree of belief

- Basics of probability theory:

  - **Sample Space**: Set Ω of all possible outcomes ω. $\omega \in \Omega$.
  - **Event space**: Set $\mathcal{F}$ of all possible combinations of outcomes.
  - **Probability space**: consists of
    - sample space Ω
    - event space $\mathcal{F}$
    - a function P that assigns a probability to each event $e_i \in \mathcal{F}$, with
      1. $P(e_i) > 0$
      2. $P(e_1 \cup e_2 \cup \dots) = \sum_i P(e_i)$ when the events are mutually exclusive
      3. $\sum_{\omega \in \Omega} P(\omega) = 1$

  - **Random variable**: A function that maps the sample space to some other set D: $X: \Omega \to D$

  - **Expectation**: $E(X) = \sum_{x \in D_x} x P(X = x)$ = "avg. outcome over infinitely many experiments"
    (D_x: domain of X)

  - **Multidim. random variable**:
    → Joint probability: $P((X = x), (Y = y))$
    → Marginalization: $P(X = x) = \sum_{y \in D_y} P((X = x), (Y = y))$ (2nd axiom of probability spaces)

⊙ Conditional probability:

$$P((X=x)|(Y=y)) = \frac{P((X=x),(Y=y))}{P(Y=y)}$$

Probability that $X=x$, if $Y=y$ is known

⊙ Bayes' Rule:

$$\underbrace{P((Y=y)|(X=x))}_{\text{posterior}} = \frac{P((X=x)|(Y=y))\,P(Y=y)}{P(X=x)}$$

likelihood · prior · normalization factor / evidence

with $P(X=x) = \sum_{y \in D_y} p((X=x)|(Y=y))\,p(Y=y)$

⊙ Independence of random variables:

$$P((X=x)|(Y=y)) = P(X=x)$$

and $$P((X=x),(Y=y)) = P(X=x)\,P(Y=y)$$

⊙ Conditional Independence:

$$P((X=x)|(Y=y),(Z=z)) = P((X=x)|(Y=y))$$

X is conditionally independent of Z given Y.

Notation:

$P(\text{Weather}=\text{sunny}) = P(\text{sunny})$

only this RV uses "sunny"

for boolean RV:

$P(\text{Cavity}=\text{true} \mid \text{Toothache}=\text{false}, \text{Team}=\text{true})$

$= P(\text{cavity} \mid \neg\text{toothache}, \text{team})$

for a defined ordering of domain of Weather:

$P(\text{Weather}) = [0.6, 0.1, 0.29, 0.01]$

sunny, rain, cloudy, snow

• Normalization:

→ To compute conditional probabilities of Boolean RV

Normalization constant

e.g. $P(\text{Cavity} \mid \text{Toothache}) = \alpha \cdot P(\text{Cavity}, \text{toothache})$

fixing evidence variables

summing over hidden variables

$= \alpha\,[P(\text{Cavity}, \text{toothache}, \text{catch}) + P(\text{Cavity}, \text{toothache}, \neg\text{catch})]$

(-) Bayesian networks: "used to represent dependencies (conditional independence) among variables"

→ is a directed acyclic graph

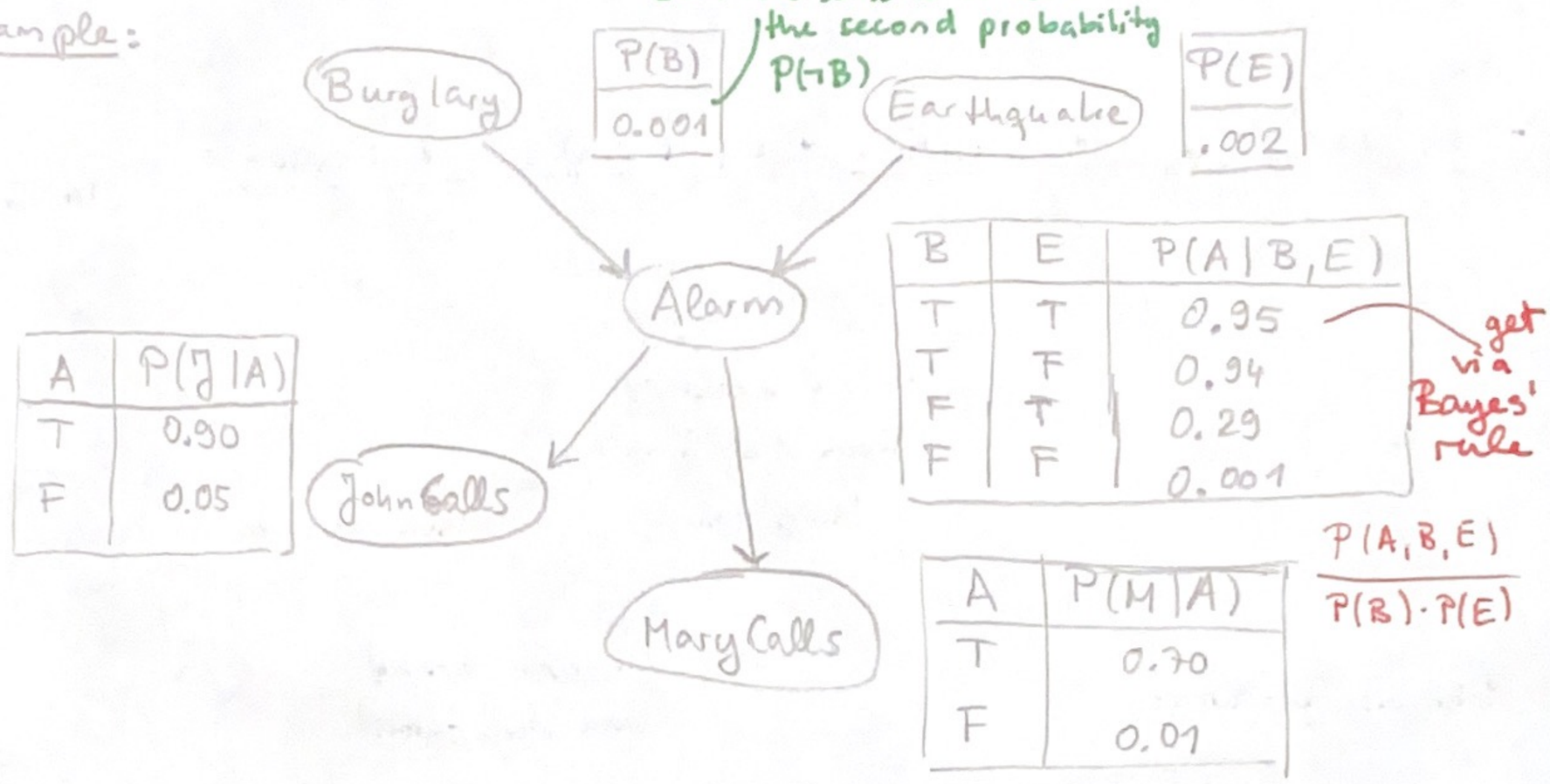
- each node corresponds to a random variable
- arrows between nodes start at parents
- each node has a conditional probability distribution $P(X_i | Parents(X_i))$

"smaller than storing the joint probability distribution in a table"

↓

$\mathcal{O}(2^n)$ values

Burglary example:



→ Each conditional probability table for boolean X_i with k boolean parents has 2^k rows

⇒ Complete network requires $\mathcal{O}(n \cdot 2^k)$ values

(-) Determine {conditional} independence:

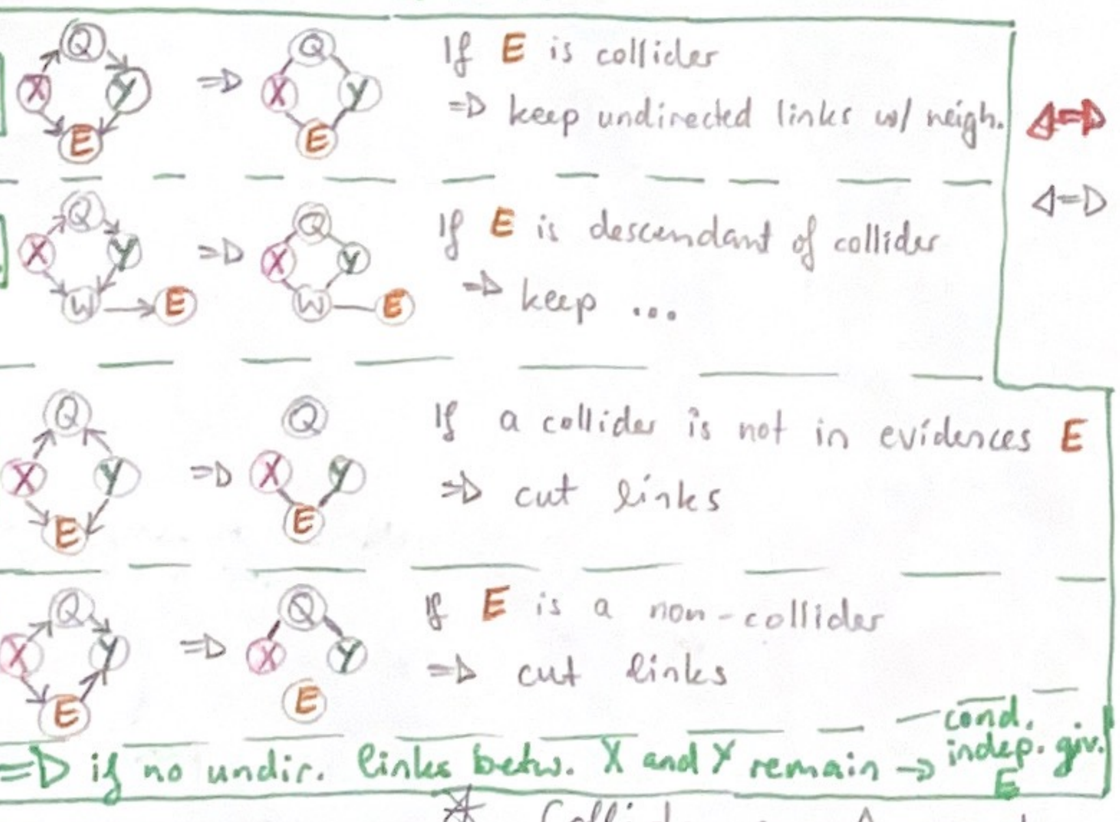
Markov Blanket → independence: X and Y are independent ⇔ $P(X,Y) = P(X)P(Y)$ or $P(X|Y) = P(X)$

A node is cond. ind. of all other nodes given its parents, children & children's parents.

⇔ no common ancestry in Bayesian network

→ conditional independence: X and Y are conditionally independent given a set of evidences E

GRAPHICAL SOLUTION:

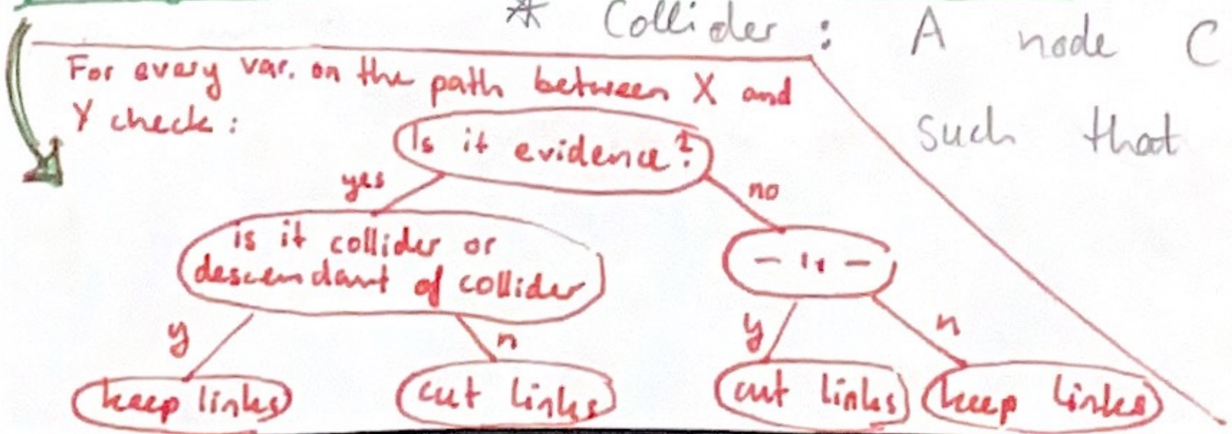


⇔ $P(X | Y, E) = P(X|E)$ or $P(Y|X,E) = P(Y|E)$

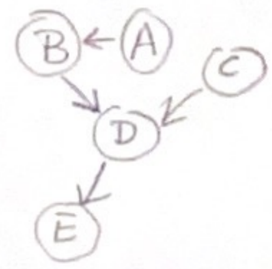
⇔ every path U from X to Y is blocked / there is a node W on U such that

- W is a collider and neither W nor any of its descendants are in E, or
- W is not a collider on U and W is in E

* Collider: A node C on Path U w/ neighbors A and B such that $A \to C \leftarrow B$.



Example:



D is collider along path A-B-D-C

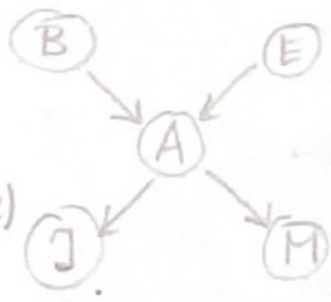
- Joint distribution from local conditional distributions:

lower-case x_i is value of RV, capital X_i is RV

$$\rightarrow P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i))$$

example: $P(j, m, a, \neg b, \neg e)$

$= P(j|a)\, P(m|a)\, P(a|\neg b, \neg e)\, P(\neg b)\, P(\neg e)$



Ex. 8.2

- Inference by enumeration + by variable elimination:

Using the example: $P(B \mid j, m) = \frac{P(B, j, m)}{P(j, m)} \overset{\text{Normalization}}{=} \alpha P(B, j, m)$

"Prob. of burglary given that John and Mary call"

$$= \alpha \sum_e \sum_a P(B, j, m, e, a)$$

enumerate hidden variables; joint distr.

$$= \alpha P(B) \sum_e P(e) \sum_a P(a \mid B, e) P(j, a) P(m \mid a)$$

eliminate variables and store factors: $f_1(B)$, $f_2(E)$, $f_3(A, B, E)$, $f_4(A)$, $f_5(A)$

3-dim. matrix (A is first dim); 1-dim matrix; j and m are given

$$= \alpha f_1(B) \times \sum_e f_2(E) \times \sum_a f_3(A, B, C) \times f_4(A) \times f_5(A)$$

pointwise product

sum from right to left

multiply each B,C-submatrix with the corresponding value in $f_4(A)$ ⇒ output is 3-dim

sum over all A ⇒ output is 2-dim

$$= \alpha f_1(B) \times f_7(B)$$

Pointwise product:

If two factors have variables $Y_1, \dots, Y_k$ in common:

$$f(X_1, \dots, X_j, Y_1, \dots, Y_k, Z_1, \dots, Z_l)$$

$$= f_1(X_1, \dots, X_j, Y_1, \dots, Y_k) \times f_2(Y_1, \dots, Y_k, Z_1, \dots, Z_l)$$

e.g.

$$f_3(\neg a, b, c) = f_1(\neg a, b) \times f_2(b, c)$$

Heuristic for variable ordering / choosing which variable to eliminate:

Eliminate the variable which minimizes the # dependency-variables of the factor to be constructed.

- Monte Carlo simulation:

① Create samples according to given probability distr.
② Deterministic simulation
③ Aggregation of indiv. simulations to get expected values of prob. distr.

probability that a specific event is generated by sampling from a bayesian network

$$S_{PS}(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i)) = P(x_1, \dots, x_n)$$

→ Rejection sampling: "reject all samples that do not match evidence e"

$$\hat{P}(X \mid e) = \alpha N_{PS}(X, e) = \frac{N_{PS}(X, e)}{N_{PS}(e)} \approx P(X \mid e)$$

Number of samples gen. for event; prob. distr.; for many samples

→ Problem: Very expensive if $P(e)$ is small

⇒ Likelihood Weighting: Only generate samples that include e

(Fix the values of the evidence variables + sample from the non-evidence variables + weight each event with the likelihood of its occurence

KI 10: Hidden Markov Models

- Time Varying RVs:

  * Copy state & evidence variables at each time step (the set of vars does not change ove time)

    → E_t (observable)

    → X_t (unobservable)

- Definitions:

  * Stochastic process: A sequence of RVs X_1, X_2, X_3, etc.

  * Markov process: A stochastic process that has the Markov property

  * Markov property: if $P(X_n = x \mid X_{n-1} = x_j, \dots, X_0 = x_k) = P(X_n = x \mid X_{n-1} = x_j)$

  * Stationary process: A stochastic process whose joint probability distr. does not change when shifted in time

    $\forall t: P(X_n = x_i \mid X_{n-1} = x_j) = P(X_{n+t} = x_i \mid X_{n-1+t} = x_j)$

- Stationary Markov chain:

  = discrete stationery process w/ the Markov property

$$\underbrace{P(X_n = x_i)}_{(p_i)_n} = \sum_{j=1}^{N} \underbrace{P(X_n = x_i \mid X_{n-1} = x_j)}_{\text{Transition matrix } T_{i,j}} P(X_{n-1} = x_j)$$

probability distribution over the states

$$p_n = T \cdot p_{n-1}$$

T has dimension $|s| \times |s|^n$ — order of HMM; number of possible states

* Higher-Order Markov chain:

  → Likelihood of current state depends on m previous states

$$P(X_n = x_i \mid X_{n-1} = x_j, \dots, X_1 = x_0) = P(X_n = x_i \mid X_{n-1} = x_j, \dots, X_{n-m} = x_l) \quad \text{for } n > m$$

  => Convert to a normal Markov chain by introducing new states corresponding to the sequence of m previous ones

Assumption: Random sensor values E_t only depend on current state

$$P(E_t \mid X_{0:t}, E_{0:t-1}) = P(E_t \mid X_t)$$

→ else add more states

- Hidden Markov model:

prob. of the state at time n being x_i

$$p_n = T \cdot p_{n-1} \quad \text{with } (p_i)_n = P(X_n = x_i)$$

"transition model" $T_{i,j} = P(X_n = x_i \mid X_{n-1} = x_j)$

$$\hat{p}_n = H \cdot p_n \quad \text{with } (\hat{p}_i)_n = P(E_n = e_i)$$

"Sensor model" $H_{i,j} = P(E_n = e_i \mid X_n = x_j)$

each state has its own observation model

Given the initial prob. distr. at time 0 ($P(X_0)$), one can compute the ~~distri~~ state & evidence distribution later on:

$$P(X_{0:t}, E_{0:t}) = \left(\prod_{i=1}^{t} \underbrace{P(E_i|X_i)}_{H} \underbrace{P(X_i|X_{i-1})}_{T}\right) P(X_0)$$

- Inference Tasks in HMMs:

* Filtering: $P(X_t|e_{1:t})$ "What's the belief state given previous evidence?"
* Prediction: $P(X_{t+k}|e_{1:t})$ for $k>0$ "like filtering w/o new evidence"
* Smoothing: $P(X_k|e_{1:t})$ for $0 \leq k < t$ "What's the state in between?"
* Most likely explanation: $\underset{x_{1:t}}{\operatorname{argmax}} P(x_{1:t}|e_{1:t})$

"Given the evidence, what's the most likely evolution of the state?"

- Filtering:

$$\overbrace{P(X_{t+1}|e_{1:t+1})}^{\text{"a one-step prediction of the next state"}} = \alpha \underbrace{P(e_{t+1}|X_{t+1})}_{H} P(X_{t+1}|e_{1:t})$$

possible values of X_t

$$T = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \downarrow X_{t+1}$$

$$O_t = \begin{bmatrix} P(e_t|X_t=\dots) & 0 & 0 \\ 0 & P(e_t|X_t=\dots) & 0 \\ 0 & 0 & \dots \end{bmatrix} \rightarrow X_t$$

Observation model — H

prediction from previous iteration

$$= \alpha \underbrace{P(e_{t+1}|X_{t+1})}_{\text{sensor model corrects the prediction with the latest evidence}} \sum_{x_t} \underbrace{P(X_{t+1}|x_t, e_{1:t})}_{\text{Markov property}} P(x_t|e_{1:t})$$

$= P(X_{t+1}|x_t)$ — T

prediction using the transition model

in matrix notation:

$$\rightarrow f_{1:t+1} = \alpha O_{t+1} T f_{1:t}$$ with

$f_{1:t}$: vector of prob. distr. → sums to 1

$$(f_i)_{1:t} = P(X_t = x_i | e_{1:t})$$

$$(O_{ij})_t = \begin{cases} P(e_t|X_t = x_i) & \text{if } i=j \\ 0 & \text{else} \end{cases}$$

- Prediction:

$$P(X_{t+k+1}|e_{1:t}) = \sum_{x_{t+k}} P(X_{t+k+1}|x_{t+k}) P(x_{t+k}|e_{1:t})$$

in matrix form: $p_n = T^n \cdot p_0$

$\hat{p}_n = H \cdot p_n$

- Smoothing: "Process of computing the distr. over the past states given evidence up to the present"

→ $P(X_k | e_{1:t}) = \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k)$

$= \alpha \underbrace{f_{1:k}}_{\text{forward}} \times \underbrace{b_{k+1:t}}_{\text{backward}}$

with $P(e_{k+1:t} | X_k) = \sum_{x_{k+1}} P(e_{k+1} | x_{k+1}) P(e_{k+2:t} | x_{k+1}) P(x_{k+1} | X_k)$

- $P(e_{k+1} | x_{k+1})$: sensor model H
- $P(e_{k+2:t} | x_{k+1})$: from recursive execution backwards in time
- $P(x_{k+1} | X_k)$: transition probability T

The backward phase is initialized by

$b_{t+1:t} = P(e_{t+1:t} | X_t)$

$= P(\emptyset | X_t)$

$= 1$

→ $b_{k+1:t} = \text{Backwards}(b_{k+2:t}, e_{k+1})$

$f_{1:t+1} = \alpha \cdot \text{Forward}(f_{1:t}, e_{t+1})$

in matrix notation: → $b_{k+1:t} = T^T O_{k+1} b_{k+2:t}$ (observations as in filtering)

with $(b_i)_{k+1:t} = P(e_{k+1:t} | X_k = x_i)$

- Forward-Backward-Alg:

function Forward-Backward(ev, prior) **returns** a vector of probability distr.

inputs: ev, a vector of evidence values $1, \dots, t$

prior, $P(X_0)$

local vars: fv, vector of forward messages $0, \dots, t$

b, backward message representation (initially all ones)

sv, vector of smoothed estimates for steps $1, \dots, t$

```
fv[0] ← prior
for i=1 to t do:
    fv[i] ← Forward(fv[i-1], ev[i])
for i=t down to 1 do:
    sv[i] ← Normalize(fv[i] × b)
    b ← Backward(b, ev[i])
return sv
```

- Most likely explanation: "Find the most likely sequence of states, given the evidences"

* Most likely sequence ≠ sequence of most likely states

Viterbi Algorithm:

$$\max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) \max_{x_t} \left(P(X_{t+1} \mid x_t) \cdot \underbrace{\max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t \mid e_{1:t})}_{\text{time is reduced by 1}} \right)$$

with $\max_{x_t} P(X_{t+1} \mid x_t) = \left\langle \max_{x_t} P(X_{t+1} = true \mid x_t), \max_{x_t} P(X_{t+1} = false \mid x_t) \right\rangle$

Obtaining the most likely sequence:

1) start w/ the most likely final state

2) backtracking along the path which maximizes the probability of the final state

- Particle filtering

* Monte Carlo method for HMMs

1) Weight samples in states by their likelihood for the observation (according to the evidence)

2) Generate new samples by weighted random selection

3) Propagate samples through transition model

KI 11 - Rational Decisions:

decision theory = probability theory + utility theory

* Probability theory: $\underbrace{P(Result(a) = s')}_{\text{Probability of an outcome from doing action } a} = \sum_s P(s) P(s' | s, a)$

* Utility theory: $\underbrace{EU(a)}_{\text{expected utility of action } a} = \sum_{s'} P(Result(a) = s') \underbrace{U(s')}_{\text{utility of state } s' \text{ ("agent's preference")}}$

→ Maximum expected utility:

"ultimate goal of AI" { $action = \arg\max_a EU(a) = \arg\max_a \sum_{s'} P(Result(a) = s') U(s')$

- Preferences:

$(A, B, \ldots)$

* Agents choose between prizes and lotteries (situations w/ uncertain prizes, e.g. $L = [p, A; (1-p), B]$)

Rules:

$A \succ B$	"A preferred to B"
$A \sim B$	"indifference"
$A \succsim B$	"A preferred to B or indifference"

Constraints on rational agent's preferences: ("Axioms of utility theory")

Orderability: $(A \succ B) \vee (B \succ A) \vee (A \sim B)$ "the agent cannot avoid deciding"

Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

Continuity: $A \succ B \succ C \Rightarrow \exists p \; [p, A; 1-p, C] \sim B$

Substitutability: $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$ (also for $\succ$ instead of $\sim$)

Monotonicity: $A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1-p, B] \succ [q, A; 1-q, B])$

Decomposability: $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$

preferences lead to utility:

There exists a utility function U, s.t.:

$U(A) > U(B) \Leftrightarrow A \succ B$

$U(A) = U(B) \Leftrightarrow A \sim B$

Expected utility of a lottery: $U([p_1, s_1; \ldots; p_n, s_n]) = \sum_i p_i U(s_i)$

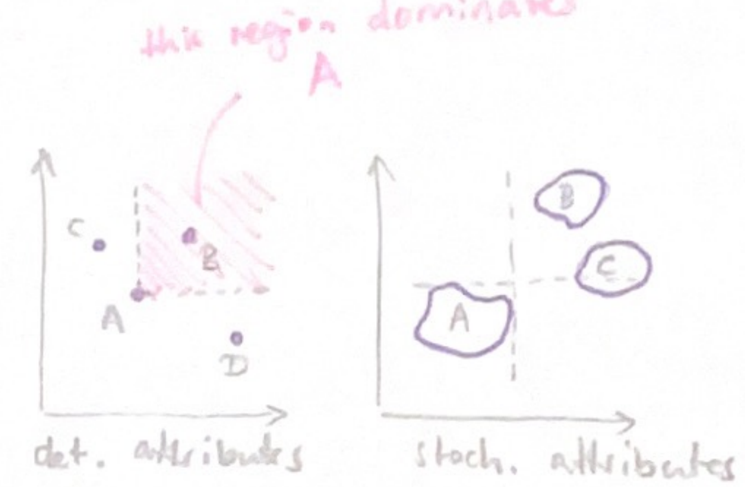
- Multiattribute utility:

→ Utility functions that depend on many attributes $X = X_1, \dots, X_n$

→ How to determine which utility is better ?

* Strict dominance:

$$\forall i \quad X_i(B) \geq X_i(A) \quad (\text{and } U(B) \geq U(A))$$

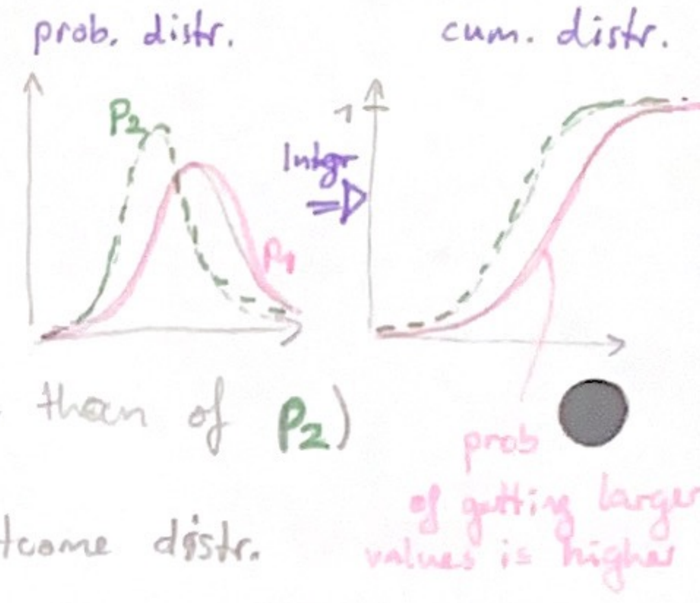


* Stochastic dominance:

* Distr. P_1 stochastically dominates P_2 iff

$$\forall t \quad \int_{-\infty}^{t} P_1(x)dx \leq \int_{-\infty}^{t} P_2(x)\,dx$$

(the cumulative distr. of P_1 is always smaller than of P_2)



* <u>If U is monotonic in x</u>, then A_1 with outcome distr. P_1 stochastically dominates A_2 w/ P_2:

$$\int_{-\infty}^{\infty} p_1(x)\,U(x)dx \geq \int_{-\infty}^{\infty} p_2(x)\,U(x)dx$$

attributes

Preference Independence: X_1 and X_2 are **preferentially independent** of X_3 iff preference between $\langle x_1, x_2, x_3 \rangle$ and $\langle x_1', x_2', x_3 \rangle$ does not depend on X_3. (fixed value)

Mutual preference independence: If every pair of attributes is **preferentially independent** of its complement, then this also applies to every subset of the attributes.

⇒ ∃ additive value function: $V(x_1, \dots, x_n) = \sum_i V_i(x_i)$

det. preference structures

Utility independence: Set of attributes X is **utility-independent** of Y iff. preferences over lotteries in X do not depend on attributes in Y.

Mutual utility independence: If every subset is **utility independent** of its complement ⇒ ∃ multiplicative utility function

e.g. for three attributes

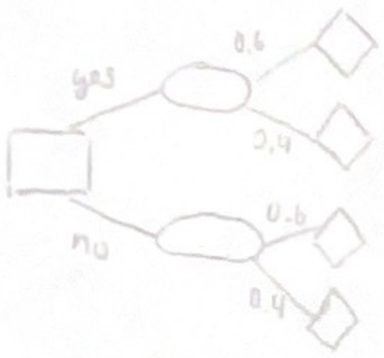
$$U = k_1U_1 + k_2U_2 + k_3U_3 + k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_1k_3U_1U_3 + k_1k_2k_3U_1U_2U_3$$

$U_i \hat{=} U_i(x_i)$

stoch. preference structures

Decision trees:

→ Components:
* Decision nodes (rectangles) → branches for each alternative dec.
* Utility nodes (diamonds) → leaf nodes that represent the utility value of each branch
* Chance nodes (ovals) → represent random variables



⇒ Expected utility of any decision: weighted summation of all branches from the decision to all reachable leaves.

Value of information:

exp. value of info = exp. val. of best action given the info at no charge
− exp. val. of best action w/o info

Value of the current best action: $MEU(\alpha|e) = \max_a \sum_{s'} P(Result(a)=s'|e)\,U(s')$
(α: best action; e: initial evidence)

Value of the new best action: $MEU(\alpha_{e_j}|e,e_j) = \max_a \sum_{s'} P(Result(a)=s'|e,e_j)\,U(s')$
(α_{e_j}: new best action; e_j: new evidence)

⇒ value of information: $VOI_e(E_j) = \left(\sum_k P(E_j=e_{jk}|e)\,MEU(\alpha_{e_{jk}}|e,E_j=e_{jk})\right) - MEU(\alpha|e)$

($\sum_k$: avg. over possible values)

max. expected utility of getting the info for free

max. exp. utility w/o this info

* Properties:

Nonnegative $\forall e, E_j \quad VOI_e(E_j) \geq 0$

Nonadditive $VOI_e(E_j,E_k) \neq VOI_e(E_j) + VOI_e(E_k)$

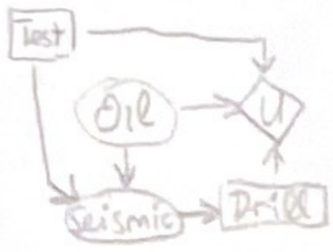
Order independent $VOI_e(E_j,E_k) = VOI_e(E_j) + VOI_{e,e_j}(E_k)$

$= VOI_e(E_k) + VOI_{e,e_k}(E_j)$

Decision networks: (aka Influence Diagrams)

↓ smaller in size than decision trees

- Links to RVs: $(Y) \to (X)$: X conditionally depends on Y
  $[D] \to (X)$: Value of X is revealed when decision D is taken.
- Links to Utility Nodes: $[D] \to (U) \leftarrow (X)$ At most one utility node per network
- Links to decision nodes (information links): $(X) \to [D]$: Value of X is known before taking decision D
  $[d] \to [D]$: Decision d is known before taking decision D



* Decision networks define a partial ordering

$$X_0 < D_1 < X_1 < D_2 < \dots < X_{n-1} < D_n < X_n$$

Obtaining partial orders:

1) Identify first decision D_1 and all variables X_0 to make that decision
2) Identify the next decision D_2 and the variables X_1 revealed between D_1 and D_2
3) Place any unrevealed variables at the end of the ordering

* Policies to evaluate decisions

$$\pi^*(e) = \underset{d_i}{\operatorname{argmax}}\, EU(d_i \mid e)$$

$$\Rightarrow \underbrace{\pi^*(x_{1:i-1}, d_{1:i-1})}_{\text{policy for decision variable } d_i} = \underset{d_i}{\operatorname{argmax}} \sum_{x_i} \underbrace{P(Result(d_i) = x_i \mid x_{1:i-1}, d_{1:i-1})}_{= P(x_i, d_i \mid x_{1:i-1}, d_{1:i-1})} U(x_{1:l}, d_{1:l})$$

AI 12: Rational Decisions over time

- Markov Decision Processes: (MDP)

↳ 5-tuple (S, A, P, R, γ)

- γ: Discount factor $\gamma \in [0,1]$
- $R(s,a,s')$ is the immediate reward
- $P_a(s,s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$
- A: Finite set of actions (or $A(s)$)
- S: Finite set of states

* Utility of sequences: 1) Additive: $U([s_0,a_0,s_1,a_1,\dots]) = R(s_0,a_0,s_1) + R(s_1,a_1,s_2) + \dots$

2) Discounted: $U([s_0,a_0,s_1,a_1,\dots]) = R(s_0,a_0,s_1) + \gamma R(s_1,a_1,s_2) + \dots$

(γ: discount factor)

* Utility of states: $U(s)$ = expected (discounted) sum of reward until termination with optimal actions

→ Optimal policy of state s and Action space $A(s)$:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma U(s')]$$

- Bellman Principle of Optimality:

→ Optimal policy has the property that whatever the first state and action are, the remaining decisions must constitute an optimal policy w/ respect to the state

⇒ Bellman equation: exp. sum of rewards = current reward + γ · exp. sum of rewards after taking the best action

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma U(s')]$$

Q-function: The expected utility when starting in a given state with a given action.

$$\underbrace{U(s)}_{\text{utility}} = \max_{a \in A(s)} \underbrace{Q(s,a)}_{\text{Q-func.}} \quad \Rightarrow \quad \pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}}\, Q(s,a)$$

$$\Rightarrow Q(s,a) = \sum_{s'} P(s' \mid s,a)\,[R(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a')]$$

→ Value-iteration algorithm:

corresponds to solving n non-lin. equations with n unknowns

1) Start w/ arbitrary utility values
2) terminal states: $U(s) = R(s)$
   other states: $U(s) \leftarrow \max_{a \in A(s)} Q(s,a)$
3) Repeat until convergence

- Policy iteration:

```
function Policy-Iteration (mdp) returns a policy
inputs: mdp, a Markov decision process with states S, actions A(s)
        and transition model P(s'|s,a)
local variables: U, a vector of utilities for states in S, initially 0
                 π, a policy vector indexed by state, initially random

repeat                                   ← evaluate the policy in each states and
    U ← Policy-Evaluation(π, U, mdp)       compute the utilities
    unchanged ← true
    for each state s in S do                                          ┐
        a* ← argmax_{a∈A(s)} Q-Value(mdp, s, a, U)                    │ policy
        if Q-Value(mdp, s, a*, U) > Q-Value(mdp, s, π(s), U) then     │ improvement
            π(s) ← a*                                                 │
            unchanged ← false                                         ┘
until unchanged
return π
```

AI 13 - Learning

Learning = an agent improves its performance on future tasks through observation

-Types of Learning:
1) Unsupervised learning → learn patterns in the input
2) Reinforcement learning → learn from series of reinforcements
3) Supervised learning → learn from input-output-pairs

-Supervised Learning:

Problem: Given data $(x_1, y_1), \dots, (x_N, y_N)$
find mapping h such that $h(x) \approx y$

→ Max. the probability that the hypothesis belongs to the data set

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(h \mid data) = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(data \mid h)\, p(h)$$

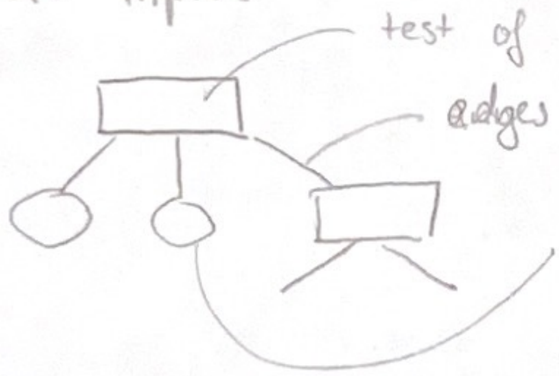
→ Use test set to check generalization

→ Decision Trees:

→ function w/ attribute values as input and a boolean decision as output

n boolean attributes
→ 2^{2^n} possible decision trees

* discrete inputs



* Information-theoretic entropy (measure of uncertainty of a RV)

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

entropy of boolean RV → $B(q)$ = truth probability

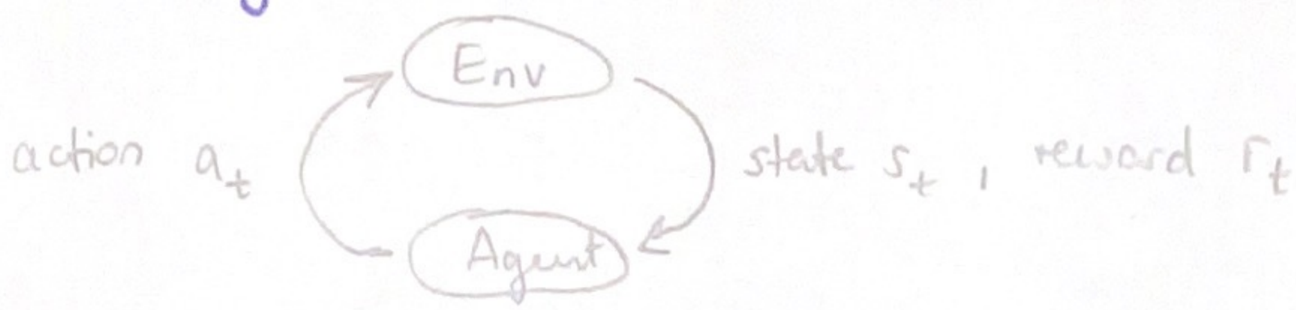
$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

→ Information gain:

$$Gain(A) = B\left(\frac{p}{p+n}\right) - Remainder(A)$$

$$\text{with } Remainder(A) = \sum_{k=1}^{d} \underbrace{\frac{p_k + n_k}{p+n}}_{\text{likelihood of going to the k'th node}} \underbrace{B\left(\frac{p_k}{p_k+n_k}\right)}_{\text{entropy of the k'th node}}$$

- Reinforcement Learning:



* Policy
  - Deterministic: $\pi : S \to A$
  - Stochastic: $\pi : S \times A \to [0,1]$ with: $\sum_{a \in A} \pi(a,s) = 1$

Exploration: Random next actions to gain new experience

Exploitation: Use existing policy to get next action

⇒ Maximize Value function / Utility:

$$\max_{\pi \in \Pi} U_\pi(s)$$

Probabilistic policy:

Value-func: $$U_\pi(s) = \sum_{a \in A(s)} \pi(a,s) \left(\sum_{s'} P(s'|s,a) [R(s,a,s') + \gamma U_\pi(s')] \right)$$

Q-func: $$Q_\pi(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a' \in A(s')} \pi(s',a') Q_\pi(s',a') \right]$$

→ Q-Learning:

$$\underbrace{Q_{i+1}(s,a)}_{\text{next value}} = \underbrace{Q_i(s,a)}_{(2)} + \overbrace{\alpha}^{\text{learning rate}} \Bigg[\underbrace{\sum_{s'} P(s'|s,a) [R(s,a,s') + \gamma \max_{a' \in A(s')} Q_i(s',a')]}_{\text{recent value } Q^*(s,a)} - \underbrace{Q_i(s,a)}_{(2)} \Bigg]$$

(2) prior value

temporal difference (TD) error

* On-policy: Only data collected w/ recent policy used for learning update
* Off-policy: Learning update uses any data
* model-based: State-transition model used ($P_a(s,s')$ known)
* model-free: Learning from sampling & simulation
  - → Q-Learning
  - → Policy-optimization