# Intelligence

Intelligence is a process capable of coordinating a large set of cognitive tools in order to take action in an environment:

Sensing/Actuating, Remembering, Reasoning, Planning, Learning, Predicting, Inferring intent (of others), Ascribing meaning, Judging, Imagining, Creating, …

So for our purposes, a cognitive process that exhibits behaviour in an environment.

## Intelligent behaviour through computational means

Need for an integration paradigm to support AI applications.

Automates (a subset of) the cognitive tools to create programs that we might refer to as intelligent agents. Agents can perceive the application environment, think (learn) and decide to take action in response to what they’re sensing and their objectives.

Concerns

security/privacy of data, biases, lack of understanding, black-box solutions, and replacing whole roles.

In the future AI must be responsible, safe and beneficial. Prioritise augmenting instead of replacing people, Treat safety of people as paramount, Ensure people’s data rights, Be able to take actions that are explainable and transparent, Conform to an ethical governance framework.

# Solving problems by search

Problem solution can be abstracted as path from some (start) node to some (goal) node in a suitably defined directed graph. Search algorithms to find (optimal) paths.

## Types

Deterministic, fully observable ⇒ single-state problem

AI program knows exactly which state it will be in; solution is a sequence.

Non-observable ⇒ conformant problem

AI program may have no idea where it is; solution (if any) is a sequence.

Nondeterministic and/or partially observable ⇒ contingency problem

input from sensors provide new information about current state. Solution is a contingent plan or a policy. Often interleave search, execution.

Unknown state space ⇒ exploration problem (“online”).

Classical search

The environment is observable, discrete (finitely many next states), deterministic (each action has exactly one outcome), known (possible actions and next states for each state).

#### Deterministic problem formulation

A problem description is defined by six items:

set of states, initial state, a set of actions, Transition model (possibly via a successor function, e.g., RESULT (At(Arad ), DriveTo(Zerid )) = At(Zerid ) ), goal test (check to see if made it to goal), path cost.

A solution is a path (seq. of actions) from the initial state to a goal state. Optimal solution is solutions with the lowest path cost.

Uninformed search

A state is a (representation of) a physical configuration.

A node is a data structure constituting part of a search tree includes parent, children, depth, path cost.

i.e., a node represents a state during search

Failure to detect repeated states can turn a linear problem into an exponential one!

#### Search strategies

A strategy is defined by picking the order of node expansion. Strategies are evaluated along the following dimensions:

* completeness—does it always find a solution if one exists?
* time complexity—number of nodes generated/expanded;
* space complexity—maximum number of nodes in memory;
* optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of

* b – maximum branching factor of the search tree;
* d – depth of the least-cost solution;
* m – maximum depth of the state space (may be ∞).

Breadth-first search – choose shallowest unexpanded node.

Uniform-cost search – choose unexpanded node with lowest path cost.

Depth-first search – choose deepest unexpanded node.

Depth-limited search – depth-first search with given depth limit l (nodes of depth l have no children).

Iterative deepening search – depth-limited search of increasing l.

##### Breadth-first search

A diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generatedfrontier (or fringe) is a FIFO queue, i.e., new successors go at end.

A diagram of a diagram

Description automatically generatedA diagram of a triangle with letters and numbers

Description automatically generatedcomplete if b is finite. Space complexity is bd – large issue. In general, not optimal unless all costs are equal.

##### Uniform-cost search

Choose unexpanded node with lowest path cost

fringe/frontier = priority queue ordered by path cost, lowest first

A close-up of a text

Description automatically generatedA diagram of a number of different types of objects

Description automatically generated with medium confidenceEquivalent to breadth-first if step costs are all equal.

##### Depth-first search

Expand deepest unexpanded node.

A green lines with black text

Description automatically generated with medium confidenceA diagram of a triangle with a triangle and a triangle with a triangle and a triangle with a triangle and a triangle with a triangle and a triangle with a triangle and a triangle with a triangle with

Description automatically generatedfrontier = LIFO queue, i.e., put successors at front.

A black and white image of a triangle with black dots and lines

Description automatically generatedA black and grey diagram

Description automatically generated with medium confidenceA diagram of a diagram

Description automatically generated   
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
A diagram of a diagram

Description automatically generatedNot complete; fails in infinite depth spaces & those with loops

##### Depth-limited search

To avoid the infinite depth problem of Depth-first search, only search until depth l

##### Iterative deepening search

If solution is deeper than l, from depth-limited search, increase l iteratively.

Combines Depth-first and Breadth-first searching

A diagram of a diagram

Description automatically generated with medium confidenceA group of images of a diagram

Description automatically generated with medium confidence   
  
  
  
  
  
  
  
  
A math equations on a white background

Description automatically generated

Iterative deepening is the preferred uninformed search when the search space is large and the depth of the solution is not known. Inherits the memory advantage of Depth-first search; has the completeness property of Breadth-first search.

#### Variants of uninformed search

##### **Backtracking search**

variant of depth-first search

Only one (successor) node generated (on the frontier) when expanded.

Each (partially expanded) successor remembers which node to generate next.

##### Bi-directional search

A group of branches on a white background

Description automatically generatedTwo simultaneous searches: from start to goal & from goal to start. Succeeds when (if) the frontiers of the two searches intersect.

### Informed Search

#### Best-first search

Use evaluation function to estimate cost of optimal path to goal. Choose node with the least-estimated path cost.

This can use a heuristic function; provides information on top of the problem description, so it informs the search. h(n) > 0, if n is the goal, then h(n) = 0. Estimates can be straight line distances, etc. (Euclidean, Manhattan, etc.)

frontier is a queue sorted in decreasing order of desirability.

##### Special cases

###### Greedy search

expands the node that appears to be closest to goal

Just uses the heuristic function; evaluation function = heuristic function

A diagram of a network

Description automatically generatedA diagram of a network

Description automatically generatedA black circle with red arrow and black text

Description automatically generatedA diagram of a number

Description automatically generated with medium confidenceCan get stuck in loops

###### A black circle with black text Description automatically generatedA\* search

A diagram of a number

Description automatically generatedachieve optimality by not expanding paths that are already expensive  
evaluation function: f(n) = g(n) + h(n)

g(n) = g(current node) + g(node before)

A diagram of a network

Description automatically generated

A diagram of a network

Description automatically generated

A diagram of a network

Description automatically generated

A diagram of a network

Description automatically generated

##### Conditions of optimality

###### Admissibility

h(n) must be an admissible heuristic. An admissible heuristic is one that never overestimates the actual cost to reach the goal.

Since g (n) is the actual cost, if h(n) is admissible then f (n) never overestimates the true cost.

If h(n) is an overestimate then if the algorithm reaches the goal node by some other path with distance < f (n) then it will never look at node n, because h(n) is then seemingly worse than an already established path; even if n is optimal.

If h(n) is an underestimate then if the algorithm reaches the goal node by some path with distance d, and the actual distance via n is less than d, then it is guaranteed that f (n) will also be less than d, so the algorithm will continue to look at n (and perhaps find a path shorter than n).

###### Consistency / monotonicity

A heuristic is consistent if, for every node, and every successor generated by an action, the estimated cost of reaching the goal is not greater than (the step cost of getting to the successor + heuristic of successor)

A white background with black text

Description automatically generated

Adversarial search

where the goals of the programs that do the searching are in conflict

#### Game search

Focus on games with perfect information that are deterministic.

A game is formally defined as a search problem with the following elements:

S0: The initial state.

PLAYER(s): defines which player has the move in a state.

ACTIONS(s): returns the set of legal moves in a state.

RESULT (s, a): transition model that returns the result of a move.

TERMINAL − TEST(s): is true if the game has terminated and false otherwise.

UTILITY (s, p): function returning 1 (win), 0 (lose), 1/2 (draw).

A zero-sum game is a game where the total payoff to all players is the same for every instance of the game. Chess is zero-sum.

##### Minimax

Perfect play for deterministic, perfect-information games

choose move to position with highest minimax value

MAX and MIN are opponents and numbers are utilities for each other. On a players turn, want to give opponent the least utility.

###### α–β pruning

minimax is exponential in the depth of the tree, so compute correct decision without looking at every node.

A white background with black text

Description automatically generated

Here z is not evaluated since we have already found that the min is < 3. We are finding the max of mins, so z is not needed as 3 is already larger than the first value found.  
  
α is the value of the highest-value (best) choice we have found so far. m is the best value found so far off the current path. If n is worse than m, player will avoid it ⇒ prune that branch.

β is the value of the lowest-value choice we have found so far. If n is worse than m, player will avoid it

Pruning does not affect final result

#### Resource limits

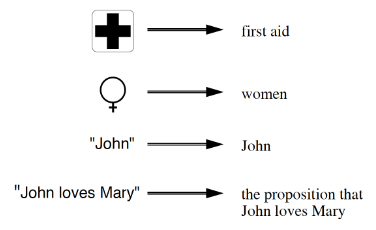
Use Cutoff-Test instead of Terminal-Test

depth limit

Use Eval instead of Utility

evaluation function that estimates desirability of position

# Logic for Knowledge Representation

What is knowledge? No, how do we talk about knowledge? We say “John knows that ...” and fill the blank with a proposition.

Other forms of knowledge:

* know how, who, what, when, ...
* sensorimotor: typing, riding a bicycle
* affective: deep understanding

For us, there is no distinction between knowledge & belief

Knowledge representation: symbolic encoding of propositions believed by some AI program

Manipulation of symbols encoding propositions to produce representations of new propositions. e.g. “John is Mary’s father ⇒ John is an adult male”

## Knowledge-based agents

Knowledge-based agents are agents that know about the world in which they are situated and reason about which courses of action to take

Components:

* Knowledge base (KB)

A component that contains a set of sentences about what is known.

Sentences taken as given (not derived from existing ones) are axioms.

* Inference engine

Allows reasoning about new sentences added to the KB and querying the KB about what is known.

Inference involves inferring new sentences from old/existing ones

The knowledge base is domain specific-content whereas the Inference engine has domain-independent algorithms

The agent is built with a Tell and Ask interface:

* Tell adds new sentences to the KB (what needs to be known). Like updates in databases.
* Ask queries what is known (answers should follow from the KB). Like queries in databases.

Knowledge-based agents can be described at three levels:

* The knowledge level: states what the agent knows about the world and what its goals are.

e.g., The automated taxi driver knows that Golden Gate Bridge links San Francisco and Marin County.

The logical level: contains the knowledge encoded into sentences of some logical language.

e.g., KB of automated taxi driver contains the FOL sentence Links(GGBridge,SF,Marin)

The implementation level: implements the sentences that should run on the agent’s platform/architecture.

e.g., The sentence Links(GGBridge,SF,Marin) is implemented by a C structure (or a Prolog fact).

Note: Declarative vs procedural way of building a system.

Autonomous agent = Knowledge-based agent + Learning mechanism

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

## Logic

Logic is a formal language for representing information so that conclusions can be drawn

Syntax defines the sentences in the language, Semantics define the “meaning” of sentence

ontological commitment – what the language assumes about the nature of reality (what exists in the world)

epistemological commitment – the possibles states of knowledge that the language allows with respect to each fact (What an agent believes about facts)

Entailment

Entailment means that one thing follows from another: KB |= α

Knowledge base KB entails sentence α

A diagram of sentence and contours

Description automatically generated with medium confidenceif and only if α is true in all worlds where KB is true.

Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

#### Inference

Inference is the process of mechanically deriving sentences entailed by a knowledge-base.

KB |- i α = sentence α can be derived from KB by procedure i.

Soundness: i is sound if it derives only entailed sentences.

I.e., whenever KB |- i α, it is also true that KB |= α

Completeness: i is complete if it derives all entailed sentences.

I.e., whenever KB |= α, it is also true that KB |- i α

While inference operates on “syntax”, the process corresponds to a real-world relationship

How do we know that KB is true in the real world?

Simple answer: agent’s sensors create the connection – when the sentence is in the KB, then it is true in the real world.

How about the rest of the agent’s knowledge e.g. its belief that wumpuses make adjacent squares smelly?

General rules are produced via learning.

But learning is fallible (what if wumpuses cause smell except Feb 29 in leap years, which when they take baths). Need of good learning is important.

Propositional Logic

#### Boolean connectives

∧ (or sometimes &) ‘and’

∨ ‘or’

¬ (or sometimes ∼) ‘not’

→ (or sometimes ⊃) ‘if ... then’, ‘implies’, ‘only if’

↔ ‘if and only if’, ‘iff’

T, ⊥ ‘true’ and ‘false’, respectively

Punctuation: brackets are used to eliminate ambiguities and aid readability.

To represent a problem logically, we need to fix a collection of symbols to stand for statements that can meaningfully be given a truth value; a propositional atom, starts with capital letters.

¬ is the strongest, ∧ and ∨ are of equal strength and are next strongest, → and ↔ are the weakest and are also of equal strength

A formula of the form:

* T, ⊥, or P (for P a propositional symbol or ‘atom’) is called atomic.
* ¬A is called a negated formula.
* A or ¬A, where A is atomic, is called a literal.
* A ∧ B is called a conjunction (A and B are its conjuncts).
* A ∨ B is called a disjunction (A and B are its disjuncts).
* A → B is called an implication (A is called the antecedent and B is called consequent).
* A ↔ B is called biconditional.
* A clause is a disjunction of one or more literals
* A definite clause is a clause with exactly 1 positive literal
* A fact, C, is where you can write true → C (C has exactly 0 negative literals)
* A goal clause is a clause with 0 positive literals and at least 1 negative literal

##### Entailment

A |= A

A ∧ B |= A

AA ∧ B

A, A → B |= B – ‘modus ponens’

A → B, ¬B |= ¬A – ‘modus tollens’

A → B, B  A

##### Normal Forms

Every formula is logically equivalent to one in CNF and one (usually a different one) in DNF.

###### Conjunctive Normal Form (CNF)

A conjunction of disjunctions of literals, i.e., a conjunction of clauses E.g., (A ∨ ¬B) ∧ (B ∨ ¬C ∨ ¬D)

###### Disjunctive Normal Form (DNF)

A disjunction of conjunctions of literals E.g., (A ∧ B) ∨ (A ∧ ¬C ) ∨ (¬A ∧ ¬D)

##### Validity

A sentence is valid if it is true in all models. E.g., True, A ∨ ¬A, A → A

Deduction Theorem: KB |= α if and only if (KB∗ → α) is valid.

KB∗ is the formula obtained by conjoining all sentences in KB into one conjunction.

##### Satisfiability

A sentence is satisfiable if it is true in some model. E.g., A ∨ B

A sentence is unsatisfiable if it is true in no models. E.g., A ∧ ¬A

KB |= α if and only if (KB∗ ∧ ¬α) is unsatisfiable,

i.e., prove α by proving (KB∗ ∧ ¬α) has no models (reductio ad absurdum).

#### Proof methods

##### Syntactic Methods

The application of inference rules. Legitimate (sound) generation of new sentences from old.

Proof = a sequence of inference rule applications used as operators in a standard search algorithm.

E.g.,

A common (sound and complete) inference system is natural deduction.

Another (sound and complete) inference system is resolution, particularly suitable for automation.

You can also use logical equivalences to re-write/simplify. Typically require translation of sentences into a normal form.

###### Resolution

Interested in KB = conjunction of Horn clauses.

A Horn clause is a clause with at most one positive literal. Any such clause belongs to one of three categories:

A rule: A definite clause with at least 1 negative literal.

A fact: A definite clause with 0 negative literals.

A goal: A goal clause.

Modus Ponens for Horn KBs can be used with forward chaining or backward chaining algorithms.

Forward chaining

fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found.

E.g., If Q is the query

P → Q

L ∧ M → P

B ∧ L → M

A ∧ P → L

A ∧ B → L

A

B

Start with B, it is true. A is next to be fired off.

These make A ∧ B true which in turn makes L true

This makes B ∧ L true which in turn makes M true

This makes L ∧ M true which in turn makes P true

We have found our query now through P → Q

Q is true.

Forward Chaining is data-driven, cf. automatic, unconscious processing, e.g., object recognition, routine decisions. May do lots of work that is irrelevant to the goal.

Backward chaining

Working backwards from the query; check if q is known already, or prove by BC all premises of some rule concluding q.

Avoid loops – check if new subgoal is already on the goal stack.

Avoid repeated work – check if new subgoal has already been proved true, or has already failed.

E.g., If Q is the query

Start from Q. Q has not been proven to be true so check concluding rules.

P → Q so check what concludes P

L ∧ M → P so check what concludes L and M

B ∧ L → M so need B as well

Avoid A ∧ P → L since this leads to a loop

Could also use A ∧ B → L, but this will need A to be true, as well as B

A & B are true so all premises are true

Q is true

P → Q

L ∧ M → P

B ∧ L → M

A ∧ P → L

A ∧ B → L

A

B

Backward Chaining is goal-driven, appropriate for problem-solving,

e.g., Where are my keys? How do I get into a PhD program?

Complexity of Backward Chaining can be much less than linear in size of KB.

A black and white text

Description automatically generated

Resolution inference rule

P → Q ≡ ¬P ∨ Q

For CNF, α and ¬α resolve, thus disjunction is reduced as the computation proceeds in the next step.

A close-up of a number of letters

Description automatically generatedResolution is sound and complete for propositional logic!

A screenshot of a computer

Description automatically generated

##### Semantic Methods

Model checking

Truth table enumeration (always exponential in n).

Boolean satisfiability solvers (SAT solvers) - many.

Improved backtracking, e.g., Davis–Putnam and many developments.

Heuristic search in model space (sound but incomplete). e.g., GSAT algorithm.

Beyond the scope of this course.

First-Order Logic

Propositional logic

is declarative: pieces of syntax correspond to facts;

allows partial/disjunctive/negated information (unlike most data structures and databases).

is compositional:

meaning of P1,2 ∧ B1,1 is derived from the meaning of P1,2 and B1,1

But ...

unlike natural language, it has limited expressive power e.g. to say pits cause breezes in adjacent squares. One needs writing a sentence for each square.

Whereas propositional logic represents the world in terms of facts – statements which can be true or false – first-order logic represents the world in terms of:

Objects: people, houses, numbers, theories, Russel, Norvig, courses, games, wars, . . .

Properties and Relations: red, round, happy, lazy, prime, smelly . . ., brother of, bigger than, inside, …

Functions: father of, price of, age of, one more than, end of, . . .

First-order logic also provides quantifiers:

∀ – ‘for all’ – conjunction of instantiations of P

∃ – ‘there exists’ - disjunction of instantiations of P

#### Syntax

* Constants John, 2, Richard, CS2910 . . .
* Predicates Brother , >, . . .
* Functions Sqrt, LeftLeg , Length . . .
* Variables x, y , a, b, . . .
* Connectives ∧ ∨ ¬ → ↔
* Quantifiers ∀ ∃

Some versions of FOL also have:

* Equality =

A predicate of arity n means the predicate takes n arguments (n ≥ 0).

The specified set of constants, functions, and predicates of a language is called its signature.

A term is a constant or variable, or an expression f(term1, ..., termn) where each of term1, ..., termn is a term, and f is a function of arity n.

An atomic sentence (or just atom) is an expression of the form P(term1, ..., termn) where each of term1, ..., termn is a term, and P is a predicate of arity n, or of the form term1 = term2

The predicate > (Length(LeftLeg (Richard)), Length(LeftLeg (John))) can be written in infix notation as Length(LeftLeg (Richard)) > Length(LeftLeg (John))

Complex sentences are made from atomic sentences using connectives.

T and ⊥ are sentences.

If S, S1, and S2 are sentences, then so are: ¬S, S1 ∧ S2, S1 ∨ S2, S1 → S2, S1 ↔ S2

If S is a sentence, then so are ∀xS – ‘for all x it is the case that S’, ∃xS – ‘there exists an x such that S’

A model is a pair M = (D, I ), where D is a domain and I is an interpretation.

D contains ≥ 1 objects (domain elements) and relations among them.

I specifies referents for

constant symbols → objects in the domain;

predicate symbols → relations over objects in the domain;

function symbols → functional relations over objects in the domain.

Recall that mathematically, a relation is a set of ordered n-tuples.

An atomic sentence, predicate(term1, . . . , termn), is true in M iff the objects referred to by term1, . . . , termn are in the relation referred to by ‘predicate’.

A diagram of two people

Description automatically generated

Previous I is what logicians would call intended interpretation.

Other interpretations exist e.g.:

one that maps Richard to the crown;

one that maps John to King John’s left leg;

... 25 possible I s just for constants Richard and John.

Note:

Not all objects need to have a name e.g. no name for left legs or crown.

It is possible for an object to have many names e.g. there is an I in which both Richard and John map to the crown. If you find this confusing, in propositional logic you can have both Cloudy and Sunny true in the same model.

Knowledge base should rule out models inconsistent with our knowledge.

Entailment in propositional logic can be computed by enumerating models.

We can enumerate the FOL models for a given KB vocabulary:

For each number of domain elements n from 1 to ∞

For each k-ary predicate Pk in the vocabulary

For each possible k-ary relation on n objects

For each constant symbol C in the vocabulary

For each choice of referent for C from n objects . . .

Computing entailment by enumerating FOL models is not easy!

Quantifier duality: each can be expressed using the other

∀x Likes(x, IceCream) ≡ ¬∃x ¬Likes(x, IceCream)

∃x Likes(x, Broccoli) ≡ ¬∀x ¬Likes(x, Broccoli)

Squares are breezy near a pit.

Diagnostic rule – infer cause from effect ∀y Breezy (y) → ∃x Pit(x) ∧ Adjacent(x, y )

Causal rule – infer effect from cause ∀x, y Pit(x) ∧ Adjacent(x, y) → Breezy (y)

Neither of these is complete – e.g., the causal rule doesn’t say whether squares far away from pits can be breezy.

Definition for the Breezy predicate:

∀y Breezy (y ) ↔ [∃x Pit(x) ∧ Adjacent(x, y )]

Substitution: a set of variable bindings.

Consider a substitution σ that assigns x = John.

We will write σ = {x/John}.

Given a sentence S and a substitution σ,

Sσ (postfix notation) is the result of applying σ to S.

S = King (x)

σ = {x/John}

Sσ = King (John)

ASK(KB, S) returns some/all σ such that KB |= Sσ

#### Inference

Every instantiation of a universally quantified sentence is entailed by it.

A ground term is a term that contains no variables.

##### A close up of a letter Description automatically generatedUniversal instantiation

For any variable υ and ground term g, if θ is the substitution {υ/g}, then

A white background with black text

Description automatically generatedi.e., if for all u, α, then g (a u) is also a part of α

A white background with black text

Description automatically generatedInstantiations need not be ground

##### Existential instantiation

A close up of a number

Description automatically generatedFor any sentence α, variable υ, and constant k that doesn’t appear elsewhere in the KB, if θ is the substitution {υ/k}, then i.e., if there exists an u where α, then k (a u) is also a part of α

A group of black and red text

Description automatically generated

C1 is called a Skolem constant.

For a small set of sentences S, one way to proceed:

replace all sentences in S by their ground instantiations;

now just use inference methods for propositional logic.

But ...

with p predicates of arity k and n constants, there are p.nk instantiations!

If there are function symbols in S, there are infinitely many instantiations, e.g.,

##### Unification

A close up of a name

Description automatically generatedA substitution θ unifies atomic sentences p and q if p.θ = q.θ

A group of text on a white background

Description automatically generatedIdea: Unify rule premises with known facts, apply unifier to conclusion.

###### Most general unifier

Usually we don’t want just any unifier, but a most general unifier (m.g.u).

θ is a most general unifier of formulas α and β if

* θ is a unifier of formulas α and β, i.e. α.θ = β.θ, and
* if σ is any other unifier of α and β (α.σ = β.σ), then α.σ is an instance of α.θ,

i.e., α.σ = (α.θ)σ′ for some substitution σ′.

A close-up of a computer code

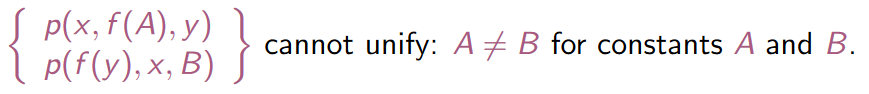
Description automatically generated

Given any two formulas α and β, there is a (very efficient) unification algorithm

UNIFY(α, β) = θ if αθ = βθ.

A close-up of a computer code

Description automatically generatedwhich checks whether α and β can be unified, and produces a most general unifier if they can



A close up of a text

Description automatically generatedOne important point of detail: a variable x can never be unified with a term containing x. This is called the occurs check.

##### Resolution inference rule

The resolution inference rule combines two clauses to make a new one.

A math equations on a white background

Description automatically generatedInference stops when empty clause is derived (contradiction).

###### A screenshot of a computer Description automatically generatedGeneralized Modus Ponens (GMP)

##### Forward chaining (bottom-up computation)

When a new fact p is added to a set of sentences S,

for each rule such that p unifies with a premise

if the other premises are known

then add the conclusion to S and continue chaining.

We split S into a set of facts E and a set of rules (definite clauses) P.

Then apply the rules in P to the facts in E to derive (using GMP) a new set of implied facts E ′.

Repeat until no new facts are generated.

Sound and complete for first-order definite clauses.

May not terminate in general if α is not entailed.

This is unavoidable: entailment with definite clauses is semi-decidable

(i.e., equivalent to the halting problem).

Can guarantee termination if restrictions are satisfied, e.g., first-order definite clauses + no functions

Simple observation: no need to match a rule on iteration k if a premise wasn’t added on iteration k – 1 ⇒ match each rule whose premise contains a newly added literal.

Matching itself can be expensive

Database indexing allows O(1) retrieval of known facts

e.g., query Missile(x) retrieves Missile(M1).

But matching conjunctive premises against known facts is NP-hard.

Partial fix: store partial matches in data structures such as rete networks.

Forward chaining is widely used in deductive databases and expert systems

##### Backward chaining (top-down computation)

Given a set of definite clauses S and a goal clause G, to solve G:

if there is a matching fact G′ in S, return unifier θ where G .θ = G ′.θ

A white background with black text

Description automatically generatedfor each rule G′ ← G1, ..., Gn in S whose head G′ matches G, solve the set of new goals G1.θ, ..., Gn.θ where G.θ = G′.θ.

repeat until there is nothing left to prove

The computation of a goal (query) G is a series of derivation steps:

The θi are the unifiers (m.g.u.’s) produced by each derivation step. The

answer computed θ is the composition of these unifiers: θ = θ1 ◦ ... ◦ θn.

The sub-goal B selected for matching can be any one of the sub-goals in

the current goal. The answers computed are the same, whichever

sub-goal is selected! Prolog always chooses the leftmost sub-goal.

#### Logic Programming

Computation as inference on a logical knowledge base (KB). – definite clauses + many extensions.

Bottom-up + top-down computations + many optimizations.

Prolog: the most widely used logic programming language. – a programming language!

Compilation techniques ⇒ 10 million LIPS (Logical Inferences per Second).

Program = set of clauses:

head :- literal1,...,literaln.

Logic programming (through Prolog) has:

* Efficient unification.
* Efficient retrieval of matching clauses by indexing techniques.
* Depth-first, left-to-right search to find alternative solutions

(using backtracking - which we only touched upon).

* Built-in predicates for arithmetic etc., e.g., X is Y\*Z+3.

## Internal Representations & Frames

Representation in general is an idealised world description (not necessarily symbolic);

For an internal symbolic representation of knowledge we need a common symbol language, in which an agent can express and manipulate propositions about the world (its knowledge/beliefs).

Good choice for symbolic representations are languages of logic, however, some preparations have to be made...

Referential Uniqueness: symbolic representations must explicitly define relations for entity references.

E.g., The chair was placed on the table. It was broken. = The chair (c1) was placed on the table (t1). It (c1) was broken.

All individuals get a unique name (one individual per name); Such unique names are denoted as instances.

E.g., Dave would not be sufficient since multiple people can be named Dave, but Dave (p3) is since there is only one p3.

Semantic Uniqueness: All symbols of an internal representation must be unique (“unambiguous”)!.

E.g.,

Jack caught the ball. [catch-object];

John caught a cold. [catch-disease];

Different symbols imply different semantics (even if their linguistic roots might be the same).

E.g., The one who caught a cold bought lemsip

A computer code with blue text

Description automatically generated with medium confidenceFunctional Uniqueness: Internal representations must uniquely express the functional roles!.

E.g.,

Jack catches the ball.

The ball Jack catches.

The ball is caught by Jack.

Who is the catcher? What is being caught?

Assertions are logic sentences which we

take as given facts (elements of

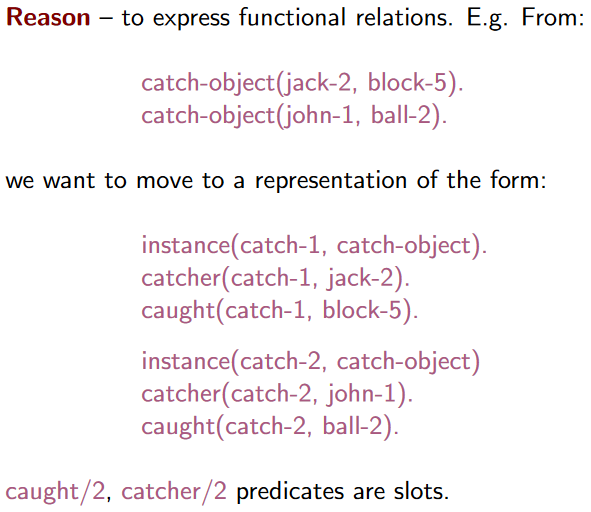
an actual internal representation).

E.g.,

instance(block-1, block) assertion

block-1 ∈ blocks = block(block-1) but is

more flexible

Slot-Assertion Notation

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generatedSlot-And-Filler Notation (→ Frames)

A white background with black text

Description automatically generated

#### Frame-based Systems

A frame consists of a selection of slots which can be filled by values, or

A screenshot of a computer

Description automatically generatedprocedures for calculating values, or pointers to other frames. E.g.,

A complete frame representation will be a whole hierarchy or

network of frames connected together by appropriate links/pointers.

Frames originally proposed for scripts, stereotyped situations, e.g.

going into a hotel room, where each frame contains info about:

* how to use the frame;
* what one can expect to happen next, or what one should

do next;

* what to do if expectations are not confirmed;
* when one encounters a new situation, one can select

from memory an appropriate frame and this can be

adapted to fit reality by changing particular details as necessary.

Frames become more powerful when slots contain procedures for computing things from information in other slots or other frames. Frames have default values and use inheritance - also available in

a semantic network.

## Semantic Networks

We can define a Semantic Network by specifying its key components:

Lexical part

nodes – denoting objects;

links – denoting relations between objects;

labels – denoting particular objects and relations.

Structural part

the links and nodes form directed graphs;

the labels are placed on the links and nodes.

Semantic part

meanings are associated with the link and node labels

(the details will depend on the application domain).

Procedural part

constructors allow creation of new links and nodes;

destructors allow the deletion of links and nodes;

writers allow the creation and alteration of labels;

readers can extract answers to questions.

Semantic Networks;

* They allow us to structure the knowledge to reflect the structure of that part of the world which is being represented.
* The semantics, i.e. real world meanings, are clearly identifiable.
* There are powerful representational possibilities as a result of is a and is a part of inheritance hierarchies.
* They can accommodate a hierarchy of default values (for example, we can assume the height of an adult male to be 178cm, but if we know he is a baseball player we should take it to be 195cm).
* They can be used to represent events and natural language sentences.

Clearly, notion of a semantic network is extremely general. However, that can be a problem, unless we are clear about syntax & semantics in each case.

A diagram of a network

Description automatically generatedAND/OR Trees

IS-A Hierarchy

A diagram of a tree

Description automatically generated

A diagram of a diagram

Description automatically generatedIS-PART (PART-OF) Hierarchy

A diagram of a function

Description automatically generatedRepresenting Events and Language

A diagram of a baseball player

Description automatically generated

Intersection Search

Early uses of semantic networks were to find relationships between objects by spreading activation from each of two nodes and seeing where the activations met. This process is called intersection search.

A diagram of a baseball player

Description automatically generated

Inheritance and Defaults

A diagram of a baseball player

Description automatically generatedTwo features of semantic networks are the ideas of default (or typical) values and inheritance. E.g.:

#### Multiple Inheritance

A diagram of a pacifist

Description automatically generatedWith simple trees, inheritance is straight-forward. However, when multiple inheritance is allowed, problems can occur. E.g.:

Conflicts like this are common is the real world.

Conflict detection important.

We aim to either over-ride or resolve all such conflicts.

#### Tangled Hierarchies

A diagram of a bird

Description automatically generatedHierarchies that are not simple trees are called tangled hierarchies. These allow another type of inheritance conflict. E.g.:

Relation between Semantic Networks and Frames

Semantic networks are a natural way to represent labelled connections between entities. But:

* complex problems require more structure to nodes, as well as to links;
* in many cases we need node labels that can be computed, rather than being fixed in advance;
* if we use database structures to keep track of everything, then the nodes and their relations begin to look more like frames;
* in the literature the distinction between frames and semantic networks is often blurred;
* the more structure a system has, the more likely it is to be termed a frame system rather than a semantic network.

A screenshot of a computer

Description automatically generated

## A white background with black text Description automatically generatedDefault Reasoning

The Qualification Problem

Let BIRDS be the set of sentences about

flying birds. Even if we could list all these

exceptions, classical logic would still not

allow BIRDS ∪ bird(frank) |= flies(frank)

We would also have to affirm all the

qualifications, viz.:

¬ penguin(frank)

¬ ostrich(frank)

¬ wounded(frank)

…

Classical logic is inadequate What we want is a new kind of ‘entailment’:

BIRDS ∪ bird(frank) |=∆ flies(frank)

From BIRDS and bird(frank) it follows by default – in the absence of information to the contrary – that flies(frank).

This kind of reasoning will be defeasible.

The Frame Problem

Actions change the truth value of some facts, but almost everything else remains unchanged.

Painting my house pink changes the colour of the house to pink... but does not change:

the age of my house is 90 years

the father of Brian is Bill

the capital of France is Paris

...

Here we have the qualification problems, in spades.

The ‘Closed World Assumption’

Consider the set of sentences that could be describing a KB. Does IBM have an office in NewYork?

We do not know, but we can make the Closed World Assumption (CWA): if α is not in the KB, assume ¬α.

CWA is another common form of defeasible, default reasoning.

Non-monotonic logics

Classical logic is monotonic. For a set of sentences S: if S |= α then S ∪ X |= α.

New information X always preserves old conclusions α.



Default reasoning is typically non-monotonic. We can have that:

There have been huge developments in AI over the last 30 years in non-monotonic logic for default reasoning and other applications.

Normal logic programs

A normal logic program is a set of clauses of the form: A ← L1, ..., Ln (n ≥ 0)

where A is an atom and each Li is a literal. A literal is either an atom (a ‘positive literal’) or of the form

not B where B is an atom. (not B is a ‘negative literal’).

Atom A is the head of the clause and literals L1, ..., Ln are the body of the clause.

When body is empty (n = 0 above) arrow ← is usually omitted.

not is negation by failure (NBF): not B succeeds when all attempts to prove B fail in finite time.

Negation-by-failure is non-monotonic

Abduction

A screenshot of a computer code

Description automatically generatedThe main idea behind abduction is that given a knowledge base KB and an observation O, find an explanation E such that KB ∪ E |= O. E.g.

Default reasoning makes assumptions

about what is false (e.g., bill not an

abnormal bird), abduction can also make

assumptions about what is true

(e.g., that paul is a friend of peter).

## Limitations of Knowledge Representation

A close-up of a pipe

Description automatically generatedLogical/symbolic knowledge representations closer to the way humans reason but too abstract for some tasks. E.g.:

Speech recognition;

Image/video recognition.

Amalgamating representations is the way to go

# Temporal Reasoning in Situation Calculus

## Classification of temporal reasoning problems

A standard classification of AI temporal reasoning problems:

The prediction problem (aka the projection problem) – given an initial state and causal rules describing a domain (“domain description”) we want to derive the state of the world resulting from some given sequence of actions.

The planning problem – given a description of the initial state and some causal rules describing a domain (“domain description”) we want to derive a sequence of actions (or some other structure of actions) that will lead from the initial state to a specified goal state.

The explanation problem – given some causal rules describing a domain (“domain description”) we want to discover facts about the initial state, given information about later states.

The explanation problem is usually associated with searching for possible causes. It is also possible to reason about past states without looking for causes. We might call this the: temporal interpolation problem.

Temporal projection can then be seen as a special case of the interpolation problem.

There are also references to narratives. A narrative is a set of observations about what was true in various states and a set of facts about what actions have occurred. The task is use a set of causal rules (“domain description”) to determine from the narrative what was true at any state, past, present, and future.

Outside AI there is more emphasis on the problem of specifying and proving properties of a dynamic system:

safety properties: nothing bad will happen;

liveness properties: something good will eventually happen.

This receives comparatively little attention in AI but is the dominant concern in other areas of Computer Science.

Fluents are propositions whose truth varies over time, also known as temporal propositions. Think of them as (Boolean) state variables.

One can also say things about time instants and time periods not represented by fluents. E.g. ‘1987 was a wet year’; ‘1-1-2015 to -3-2015 is a qualifying period of employment for Jim’.

Statements such as Manchester United were League Champions in 2000-01 could be regarded either way: as an instance of a non-fluent relation of which a time instant (2000-01) is an argument, or as a fluent ‘League Champions’ whose value is ‘Manchester United’ at time instant 2000-01.

## The Frame problem

The frame problem is about finding an elegant way to handle non-change.

Persistence or inertia: the assumption that facts are not affected by an action will have the same truth value after the action as they did beforehand.

The root of the problem lies in the ambition to represent and compute with arbitrarily large domains.

For a small domain with just a small number of fluents and action types, there is no problem – we can just write everything out explicitly.

There are three aspects of the frame problem:

the qualification problem: the difficulty of exhaustively specifying the preconditions of an action, and the conditions under which it will or will not affect a given fluent.

the computational frame problem: the computational effort required to make the required inferences to show that unaffected properties are indeed unchanged, and (perhaps, not everyone includes this) to determine the indirect effects of action that may be implied by general domain constraints (see ‘ramification problem’ below).

the ramification problem: the problem of exhaustively specifying the effects of an action. The ramification problem is often considered to be the problem of specifying and reasoning about the effects of actions in the presence of general domain constraints: an action may have indirect effects implied by such constraints. The focus these days seems to be mostly on the ramification problem.

## Situation Calculus

A simple and expressive framework for the representation of temporal information in first-order predicate-logic;

the most widely used approach to temporal reasoning in AI (and starting point of other developments that we will not cover).

many different versions now exist – i.e., there are many different styles of representation all using the basic ideas of the situation calculus.

Assumption: the world may be modelled as a system of interacting finite automata (or transition systems), hence the representation devices are states and transition systems.

A domain description is a (usually first order) theory defining the relationships of: fluents, actions, and situations.

A situation is a ‘complete state of the world at an instant of time’. Situations can be ‘temporally related’.

Situations are said to define states.

A state is a complete set of values for all fluents (a Boolean in most versions of the situation calculus).

We use the function do(A,S) to denote (name) the situation that results from performing action A in situation S.

A close-up of a math problem

Description automatically generated

A diagram of a diagram of a block diagram

Description automatically generated

A diagram of a diagram

Description automatically generated

The Situation Calculus relies on the following language:

* constants and variables for fluents and actions, usually sorted (i.e. ‘typed’). Ours in these notes will not be ‘typed’.
* names for situations:
* a single distinguished situation constant s0;
* the do(A, S) function denotes (names) the situation that results from performing action A in situation S;
* a binary predicate, holds, for representing what fluents are true in a given situation: holds(F , S) represents that fluent F is true in situation S.

A close-up of a graph

Description automatically generated

A diagram of a computer code

Description automatically generated

Main idea: rules for what holds ‘true’ after an action occurs, what remains the same and what does change after an action, checking an action’s preconditions, and how new facts are derived from existing ones.

Note (1): the difference between ‘situation’ and ‘state’.

A state is the set of fluents that are true.

A situation provides a history of transitions (actions) from some initial state. The same set of fluents can be true at many different situations.

Note (2): some authors prefer to include the situation parameter as an extra argument for fluents, writing for example on(b, a, s0) instead of holds(on(b,a), s0).

holds is convenient.

A group of triangles with letters and numbers

Description automatically generatedThe space of all possible evolutions starting at the initial situation s0 is thus a tree rooted at s0. Here is a small fragment, for the simple case where there just two actions α and β:

Is the representation of Situation Calculus ‘adequate’?

* abstracts away (deliberately) from explicit time; a transition might be instantaneous, or take 100 years;
* no concurrency: there is no (direct) support for two or more actions α and β taking place at the same time;
* do is a function: all actions are deterministic.

A close up of a text

Description automatically generated

Axioms

Consider the process of specifying actions.

It is often convenient to specify pre-conditions separately.

#### Precondition-Effect axioms

A fluent P holds in situation do(α, S) if P and execution of α in S makes P true and the preconditions of α (if any) are true in S

#### Frame axioms

A fluent P holds in situation do(α, S) if P holds in S and execution of α in S does not stop P being true.

A close up of a text

Description automatically generated

##### A general frame axiom

A white background with black text

Description automatically generatedOne big advantage of using holds is that we can write one general frame axiom and specify exceptions separately.

A screenshot of a computer

Description automatically generated

Verifying general properties

A close up of a text

Description automatically generatedThe Situation Calculus is not as suitable for expressing and verifying general properties (Modal/Temporal Logics more successful). E.g.,

Reachability analysis

We know what holds in situation do(α3, do(α2, do(α1, s0))). But how do we know this situation exists – that is, that it is reachable via poss transitions from the initial state s0?

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# Temporal reasoning in Event Calculus

Event Calculus is a general approach to representing and reasoning about events and their effects in a logic programming framework.

Instead of starting with situations focus instead on events.

Instead of global states think in terms of local states, one for each period of time for which a fluent F holds continuously.

Events initiate and terminate periods of time for which fluents hold continuously.

A screenshot of a diagram

Description automatically generated

A diagram of a flowchart

Description automatically generated

A close up of text

Description automatically generated

Event Calculus has different formulations:

In the original Event Calculus events could say something about the past, as in the example in the previous slide. E terminates F means F holds for some time period ending at event E. Then there is some more or less complicated reasoning to infer which fluents persist past E and which do not.

In the simplified Event Calculus (SEC) events will be associated with the time in which they happen. In this context, E terminates F at T does not assert anything about the past. It simply blocks persistence of F past the event E happening at a time T.

Example;

winning the lottery initiates rich (but you might be rich already);

losing your wallet terminates rich (but you might not be rich when you lose it).

A diagram of a diagram

Description automatically generated

A black arrow pointing to a black line

Description automatically generated

A diagram of a function

Description automatically generated

Inferences are non-monotonic

A diagram of a flowchart

Description automatically generated

A diagram of a diagram

Description automatically generated with medium confidenceEvents can be assimilated in a different order:

## General formulation – simplified EC

In the simplified Event Calculus (SEC)

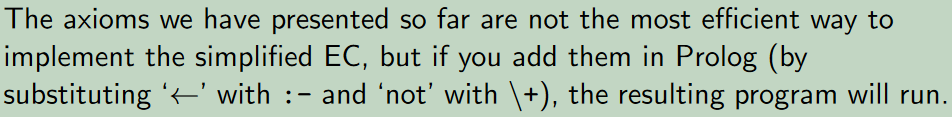
* Events are given times (when they ‘happen’).
* We will assume that all events are instantaneous. (One can easily do a refinement where events have durations).
* Many events can happen at the same time.

All examples here will have (non-negative) integer times. (But this is for convenience – SEC makes no assumption that time is discrete and/or that time points are integers).

In the original Event Calculus it is not necessary to give times to all events. All we need is a relative ordering of all events. This ordering could be partial.

Given a (consistent) narrative, in these notes we examine how to use the Event Calculus to determine what holds when – now, in the past, in the future. In this simplified version we will use:

* initially(F) – Fluent F holds at the initial time point (usually 0).
* holds at(F, T) – F is a fluent; T is a time point.
* happens(E, T) – An event/action of type E happened at time T.
* initiates(E, F, T) – An event/action of type E initiates a fluent F at a time T for a period of a time.
* terminates(E, F, T) – An event/action of type E terminates at a time T the period of time for which a fluent F holds.

Note the role of negation-by-failure, it ensures that inference is non-monotonic. Infer using forward or backward chaining

A white rectangular object with black text

Description automatically generated

You cannot load and shoot a gun at the same time.

How do you express such action precondition?

Action preconditions = Integrity constraints

incons ← happens(load, T), happens(shoot, T).

Here incons stands for inconsistency.

Every time a narrative changes, we check that a narrative is consistent.

Action preconditions in EC are more complicated than in Situation Calculus (typically). Many events can happen simultaneously in EC and some combinations are not possible. These cannot happen in situation

calculus - only one action at a time (unless you use some exotic version).

Frame problem(s):

The qualification frame problem is addressed by the built-in treatment of initiates/terminates and the use of negation-by-failure to make default inferences about how fluent persist.

The computational frame problem is addressed by having local states (‘periods’) and avoid having to reason from one global situation to the previous one; moreover, it is possible to include some simple indexing techniques to make the search for potential initiating and terminating events very much more efficient.

The Ramification frame problem is not addressed by EC. For a simple example: a person must be alive to be happy. So an indirect effect of shooting someone with a loaded gun is to terminate any period of time for which they are currently happy. In EC, one has to anticipate this and write additional terminates clause explicitly.

There are many extensions of the event calculus:

* events with durations;
* partially ordered events;
* structured events;
* events with continuous change;
* events that are assumed to have happened.

# Classical Planning

Planning research has been central to AI because of practical interest but also because of the intelligence features of human planners;

* Large logistics problems, operational planning, robotics, scheduling ...
* A number of international Conferences on Planning.
* Bi-annual Planning competition.

For us the planning problem can be defined as follows;

Given a description of an initial state of the world, a set of possible actions and a set of desired goals,

then

synthesise a plan that is guaranteed (when applied to the initial state) to generate a state which contains the desired goals (we have already seen that such a state is called a goal state).

## Automated Planning

Several dimensions of planning problems can be identified;

* Are the actions deterministic or nondeterministic? For nondeterministic actions, are the associated probabilities available?
* Are the state variables discrete or continuous? If discrete, is there a finite number of possible values?
* Can current state be observed unambiguously? (full vs partial observability)
* Do actions have a duration?
* Can several actions be taken concurrently or only one at a time?
* Is the objective to reach a goal state or maximize a reward function?
* Is there one or many agents? Are they cooperative or selfish? Do they construct own plans, or are plans constructed centrally for all agents?

The Classical Planning Problem, is simplest to solve and determined by:

* a unique known initial state,
* durationless and deterministic actions, which can be taken only one at a time,
* and a single agent.

Observability irrelevant: initial state is known and actions deterministic, so state of world after any sequence of actions can be accurately predicted.

Classical Planning = Search?

STRIPS (STanford Research Institute Problem Solver)

Special purpose, restricted language for planning.

* States: conjunctions of function-free ground atoms (positive fluents).
* Goals: conjunctions of function-free literals (positive or negative atoms - possibly with variables, implicitly existentially quantified).
* Operators: (schemas possibly with variables, implicitly universally quantified)
  + Action description = name of action
  + Precondition = conjunction of literals
  + Effects = what changes after action execution
    - Delete list = list of atoms no longer true
    - Add list = list of atoms made true
  + Every variable in effect must appear in precondition
* Actions: fully instantiated operators – different instances of the same operator are different actions.

Semantic assumptions:

* Fluents not mentioned in a state representation are (implicitly) false
* Fluents persist (do not change) unless they are explicitly changed by an action (recall frame axiom).
* Time is implicit, e.g. have time & then after action, time+1
* An action α is applicable in the current state s if the preconditions are satisfied in s

α ∈ ACTIONS(s) ↔ s |= PRECOND(a).

* The result of executing an applicable action α in state s is defined as state s′ represented by removing the fluents that appear as negative literals in action’s effects, which we call delete list; DEL(α); and adding the fluents that are positive literals in the action’s effects that we call add list; ADD(α). RESULT (s, α) = (s − DEL(α)) ∪ ADD(α)

PDDL

We will use the Planning Domain Definition Language (PDDL); an attempt to standardize Artificial Intelligence (AI) planning languages. Inspired by STRIPS among others, mainly to make the 1998/2000

International Planning Competition (IPC) possible. It has evolved with each competition.

Does there exist a plan that achieves the goal? PlanSat.

Does there exist a solution of length at most k? Bounded PlanSat

PlanSat and Bounded PlanSat are PSPACE-complete. – i.e., difficult!

PlanSat without negative preconditions and without negative effects

is in P. – i.e., solveable!

Planning as Search

Declarative representations allow planning algorithms to search forward or backwards for the goal using the heuristics we examined for search. We can distinguish:

Forward (progression): algorithm uses state-space search to consider actions that are applicable.

Backward (regression): algorithm uses state-space search to consider actions that are relevant.

A blue rectangle with black text

Description automatically generated

A diagram of a plane

Description automatically generated

considers applicable actions at a state;

* planning problems have large space graphs (for 50 planes, 10 airports, and 200 different packages to be unloaded/loaded – we have 2000 actions per state & depth of solution ≈ 200041 nodes);
* good heuristics required for relatively small problem instances (to be discussed shortly).

A diagram of a plane

Description automatically generated

Given a goal g and a ground action α, then we can regress to a sub-goal state g′ as g′ = (g − ADD(α)) ∪ Precond(α);

Issue is how to choose relevant actions contributing to the goal (at least one of action’s effects must contribute to goal, which results to a set of states to consider);

Harder to come up with good heuristics for a set of states (hence majority of planning systems favour forward search).

Progression search is prone to exploring irrelevant actions.

Relax-problem heuristic: add more edges to find the easier path (e.g. by ignoring preconditions and effects not in goal). Then count min number of actions such that the union of their effects satisfies goal.

Counting gives rise to set cover problem, which is NP-hard. A simple greedy algorithm is available for true min covering, but loses guarantees of admissibility (as discussed in our Informed Search topic).

A screenshot of a graph

Description automatically generated

Planning Graphs

Planning heuristics suffer from inaccuracies.

Planning graphs are data structures that can be used to give better heuristic estimates.

These heuristics can be applied to any of the search techniques we have seen so far.

Alternatively, we can search for a solution over the space formed by the planning graph, using an algorithm known as GRAPHPLAN.

A white text with purple letters

Description automatically generated with medium confidence



A diagram of a cake

Description automatically generated



A math equations on a white background

Description automatically generated

A close up of words

Description automatically generated

A diagram of a algorithm

Description automatically generated

A diagram of a algorithm

Description automatically generated

A diagram of a algorithm

Description automatically generated

# Learning

A computer program (Any AI program, e.g. one representing an agent (knowledge based or otherwise)) is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

In AI applications, learning is essential

* as a system construction method, i.e., expose the agent to reality rather than trying to write it down;
* as a mechanism that modifies the agent’s decision making to improve performance;
* for environments that may not be fully known, i.e. designer lacks omniscience.

A learning agent can

* act autonomously;
* synthesize rules/patterns from large volumes of data;
* handle complex data;
* deal with (partially or fully) unknown environments.

Learning improves performance of future tasks carried out by an agent. Improvements, and associated techniques, depend on four major factors:

* The component to be improved (e.g. conditions for action, goals, plans, utility, models ...).
* Prior knowledge the agent already has (e.g. about environment, other agents, ...)
* The descriptions used for what is observed/known in the form of data (e.g. declarative, mathematical, physical...).
* Feedback available to learn from.

## Forms of Learning

Three types of feedback determine the three main types of learning:

* unsupervised learning – agent learns patterns in the input even though no explicit feedback is supplied. e.g. an autonomous car learns concept of “good traffic days” and “bad traffic days”.
* reinforcement learning – agent learns from a series of reinforcements rewards or punishments. e.g. (lack of a) tip at the end of driving a person to a place.
* supervised learning – agent observes some example input-output pairs and learns a function that maps input to output. Inputs are percepts and output is provided by a teacher e.g. in autonomous driving says “Brake!” or “Turn left.”
* Also, semi-supervised learning – agent is given a few labelled examples and must make what it can of a large collection of unlabelled examples. e.g. a system to guess a person’s age from a photo.

Inductive learning

Inductive learning is like human learning based on past experiences. The closest way of an AI agent to learn is to do so from observations, which represent some “past experiences” of an application domain.

Agent will use learning of a target function to predict the values of an attribute, e.g., approve or not-approved, and high-risk or low-risk.

Induction:

From correlated observations a1, b1, ..., an, bn, learn a → b;

Compare and contrast with deduction and abduction.

Inductive learning is a form of supervised learning

Given examples (x, G (x)), where G(x) is the correct value of the target function G for input x, we want to learn G .

Task of inductive inference:

From a collection of examples in G , return a function h approximating G .

h is a hypothesis taken from a hypothesis space H.

(Pure) inductive inference assumes no prior knowledge.

Validation: construct/adjust h using a training set, evaluate generalisation capabilities on test set.

A screenshot of a computer

Description automatically generated

Prefer the simplest hypothesis consistent with the data. I.e. prefer

Animals weighing ≥ 5.2 kg cannot fly over Animals weighing ≥ 5.2 kg and have wingspan < 25 cm cannot fly.

Why is this a reasonable policy? Intuitively:

Why choose complex hypothesis if a simple one does the job?

There exist more long (i.e. complex) hypotheses than short ones.

accidental choice of bad hypothesis consistent with data more unlikely if hypothesis is simple.

Problem: identifying what simple hypotheses are.

Trade-off: the more expressive the hypothesis space, the more examples are needed (and the more complex the learning algorithm).

## Decision Trees

Decision trees provide attribute-based classification learning:

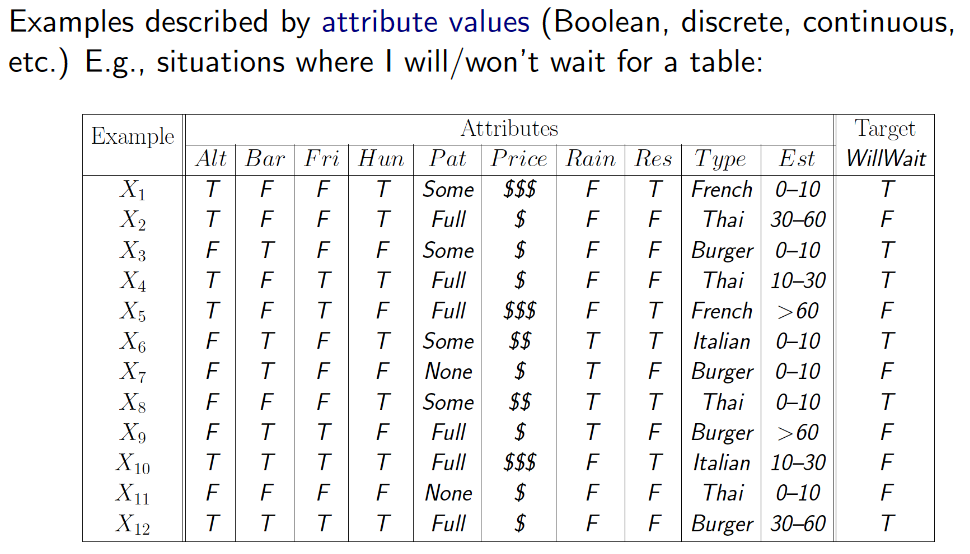
Example input x: situation/object described in terms of attribute values;

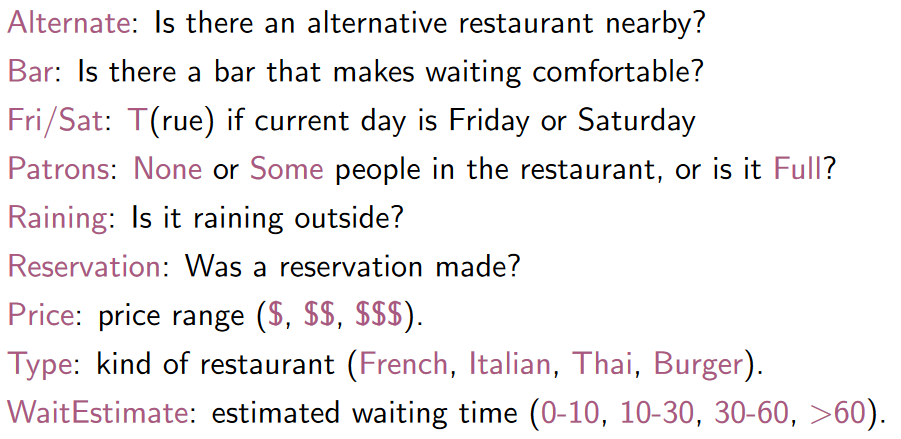
Example output G (x): a discrete-valued classification decision.

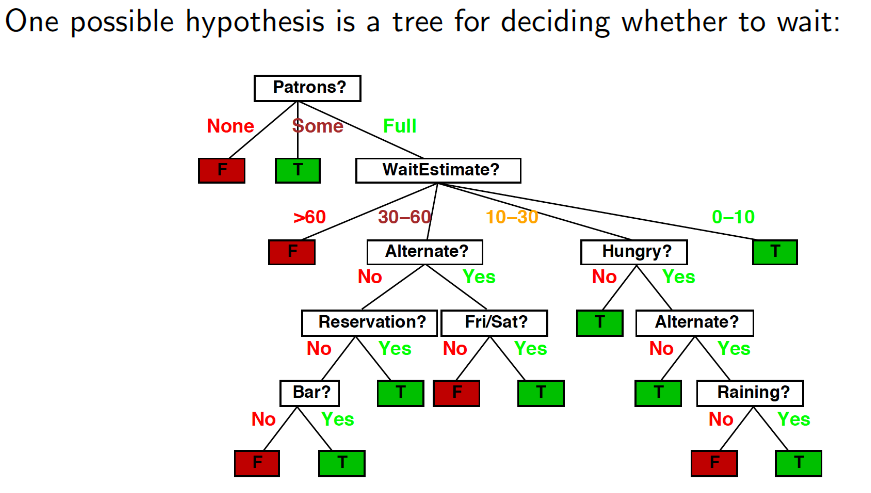
For the purposes of this course we look at Boolean classification, where each example is classified as positive (T) or negative (F).

Alternatively: G describes an unknown concept, and all values of x for which G (x) = T describe the instances of this concept.

Hypothesis = a decision tree whose nodes correspond to tests on attribute values to decide whether G (x) is true or false.







What kind of logical constraints can decision trees (DTs) express?

Consider conjunction Pi of attribute values on each path leading to Yes and disjunction G = P1 ∨ ... ∨ Pn over these conjunctions ⇒ DTs can represent any formula of propositions.

A diagram of a diagram

Description automatically generated

Easy to build tree consistent with all examples, but will it generalise? Prefer to find more compact decision trees.

A screenshot of a computer

Description automatically generated

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.

A diagram of a type of type

Description automatically generated

Information-theoretic entropy: a basic quantity associated to a random variable and interpreted as average level of “information”, “surprise”, or “uncertainty” inherent in the variable’s possible outcomes.

A close up of a text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a math problem

Description automatically generated

A math equations on a white background

Description automatically generated

A diagram of a tree

Description automatically generated

Overfitting is the problem of finding meaningless regularities and it is a general phenomenon, with all types of learners, even when the target function is not at all random.

A white text with black text

Description automatically generated

tree pruning: use significance tests to determine irrelevance of attributes.

A white background with black text

Description automatically generated

A white background with black text

Description automatically generated

Many functions not easy to represent with DTs (e.g. majority functions or mathematical functions).

Best for problems with limited number of attributes and attribute values.

Assumes examples are unambiguously and completely (no missing data) described/classified (deterministic and fully observable environment).

No use of prior knowledge learning ⇒ can be very slow.

Is DTL (1) an incremental and/or (2) an anytime algorithm?

Is this an adequate model of real learning?

A Logical formulation of Decision Trees

A screenshot of a computer

Description automatically generated

Aim of inductive learning in general is to find a hypothesis (a decision tree from all possible ones) that classifies the examples well and generalizes well to new examples.

A close up of symbols

Description automatically generatedHere we are concerned with hypotheses expressed in logic; each hypothesis hj will be represented by a logical schema of the form;

where Cj(x) is a candidate definition – some expression involving the attribute predicates.

A white paper with black text

Description automatically generated

Each hypothesis predicts that a certain set of examples – namely, those that satisfy its candidate definition – will be examples of the goal predicate. This set will be called the extension of the predicate.

The hypothesis space H is the set of all hypotheses {h1, ..., hn} that a learning algorithm is designed to consider. The learning algorithm assumes that one of the hypotheses is correct, as if

it believes the sentence h1 ∨ h2 ∨ h3 ∨ ... ∨ hn.

For decision trees we didn’t make use of prior knowledge.

Advantage of prior knowledge: narrowing down the hypothesis space.

If we let:

Descriptions denote the conjunction of all examples in training set and

Classifications denote the conjunction of all example classifications

A Hypothesis that explains the observations must satisfy: Hypothesis ∧ Descriptions |= Classification.

We call this the entailment constraint of pure inductive learning where the Hypothesis is the unknown.

A diagram of a learning process

Description automatically generated

We can generalize our previous entailment constraint as:

Background ∧ Hypothesis ∧ Descriptions |= Classifications

Background knowledge and hypothesis combine to explain the examples.

Algorithms that satisfy the above constraint are called knowledge-based inductive learning (KBIL), algorithms.

KBIL algorithms, have been studied mainly in the field of inductive logic programming, or ILP.

## Inductive Logic Programming (ILP)

A rigorous approach to knowledge-based inductive learning problem. There are methods for inducing general, first-order theories from examples. Using FOL to represent learning hypotheses is useful where attribute-based methods (e.g. decision trees) fail.

ILP allows for capturing relationships between objects rather than only their attributes.

Hypotheses generated are relatively easy for humans to understand.

A diagram of a family tree

Description automatically generated

E.g., Suppose we want to

learn family relationships

from examples.

Descriptions represent the

family tree;

A screenshot of a white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

FOIL: Top-Down Inductive Learning

Grow a hypothesis starting from a very general rule, but instead of a decision tree use Horn clauses with negation as failure.

Generate specialised clauses by adding conditions to the rule:

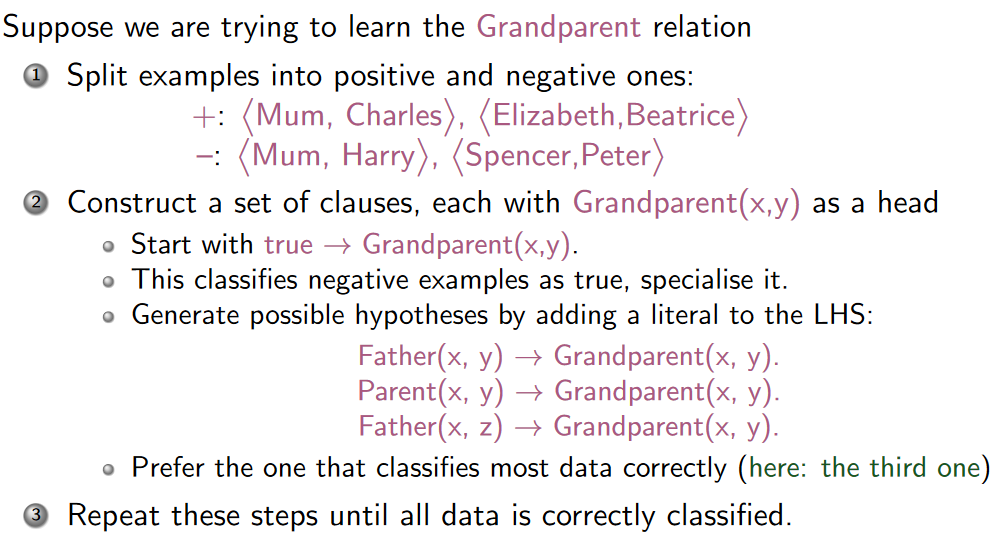
Literals can be added using predicates (including the goal predicate) with only variables as their arguments.

A literal must include at least one variable already appearing in the rule.

Conditions include equality/inequality constraints and arithmetic comparisons.

Large branching factor, but typing information can reduce it.

Heuristic for choice of literal similar to information gain, and hypotheses longer than the total length of examples are removed.



A math equation with text

Description automatically generated with medium confidenceInductive Learning with Inverse Resolution

