COMP3411/9814: Artificial Intelligence

Course Review

Planned Topics

AI, Agents & Prolog

- What is AI?
- Agents, Agent Types
- Agents Tasks
- Prolog Programming

Machine Learning

- Supervised Machine Learning
- Perceptrons & Neural Networks
- Learning to Act - Reinforcement Learning

Solving Problems by Search

- Path Search – Basic search Algorithms
- Informed - Heuristic Path Search
- Constraint Satisfaction Problems

Knowledge and Reasoning

- Planning
- Propositions and Inference
- Reasoning with Uncertainty
- Knowledge Representation

Assessment

Assessable components of the course:

Assignment 1	20%
Assignment 2	20%
Written Exam	60%

Exam Template is available on on the course Web page

Assessment

Assessment will consist of:

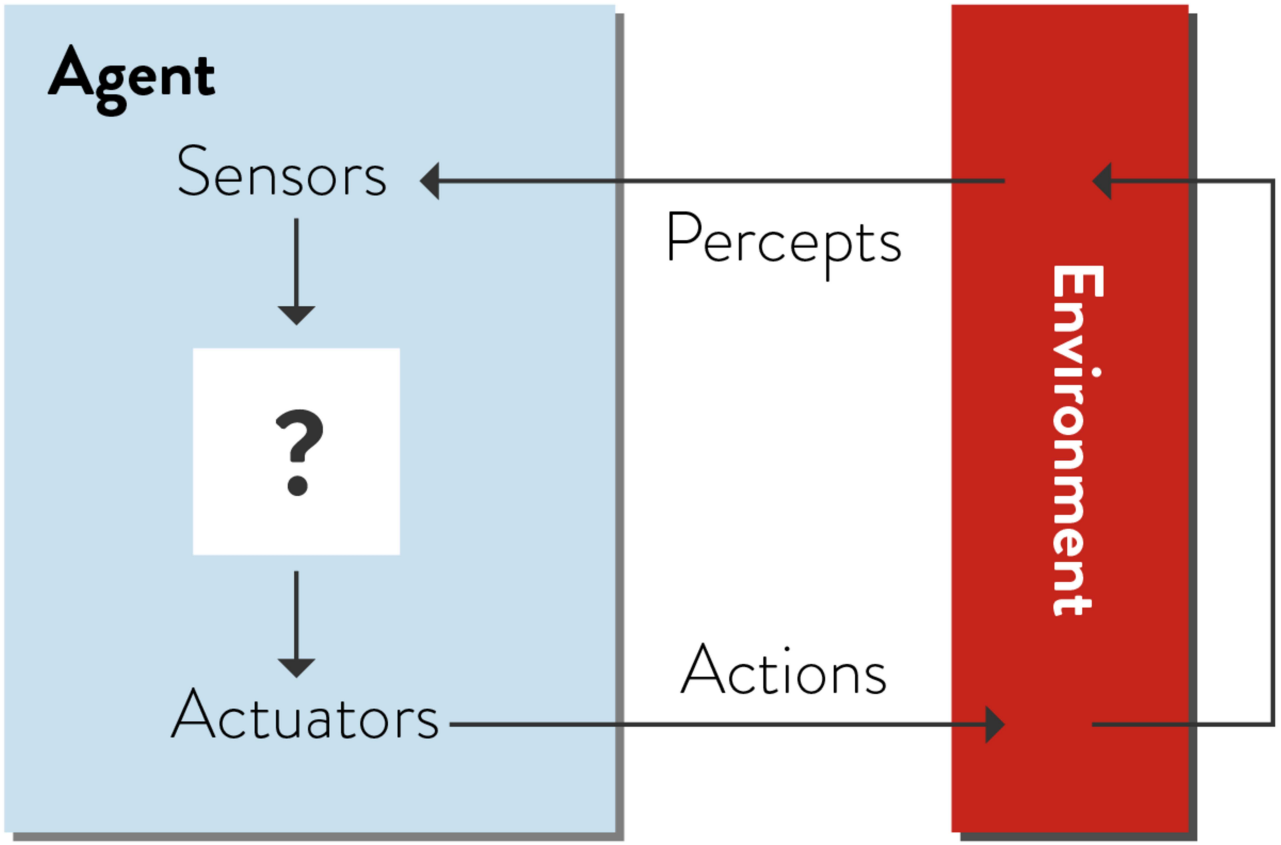
Assignments 40%

Written Exam 60%

In order to pass the course, you must score

- at least 16/40 for the assignments
- at least 24/60 for the exam
- a combined mark of at least 50/100

Agent Model



Classifying Tasks and Environment Types

Simulated vs. Situated or Embodied

Static vs. Dynamic

Discrete vs. Continuous

Fully Observable vs. Partially Observable

Deterministic vs. Stochastic

Episodic vs. Sequential

Known vs. Unknown

Single-Agent vs. Multi-Agent

Environment Types

Simulated: a separate program is used to simulate an environment, feed percepts to agents, evaluate performance, etc.

Static: environment doesn't change while the agent is deliberating

Discrete: finite (or countable) number of possible percepts/actions

Fully Observable: percept contains all relevant information about the world

Deterministic: current state of world uniquely determines the next

Episodic: every action by the agent is evaluated independently

Known: the rules of the game, or physics/dynamics of the environment are known to the agent

Single-Agent: only one agent acting in the environment

Specifying Agents

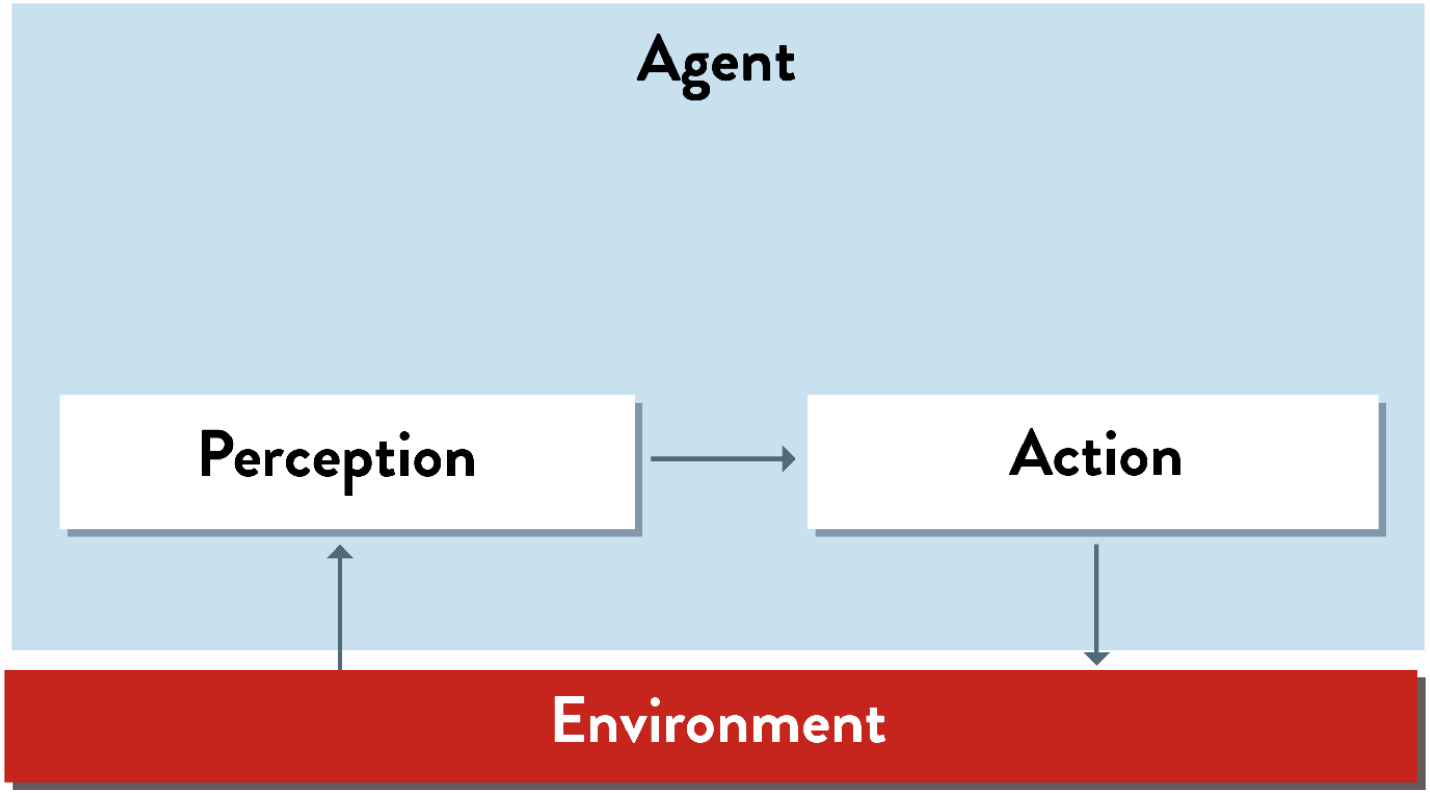
- percepts: inputs to the agent via sensors
- actions: outputs available to the agent via effectors
- goals: objectives or performance measure of the agent
- environment: world in which the agent operates

Most generally, a function from percept sequences to actions

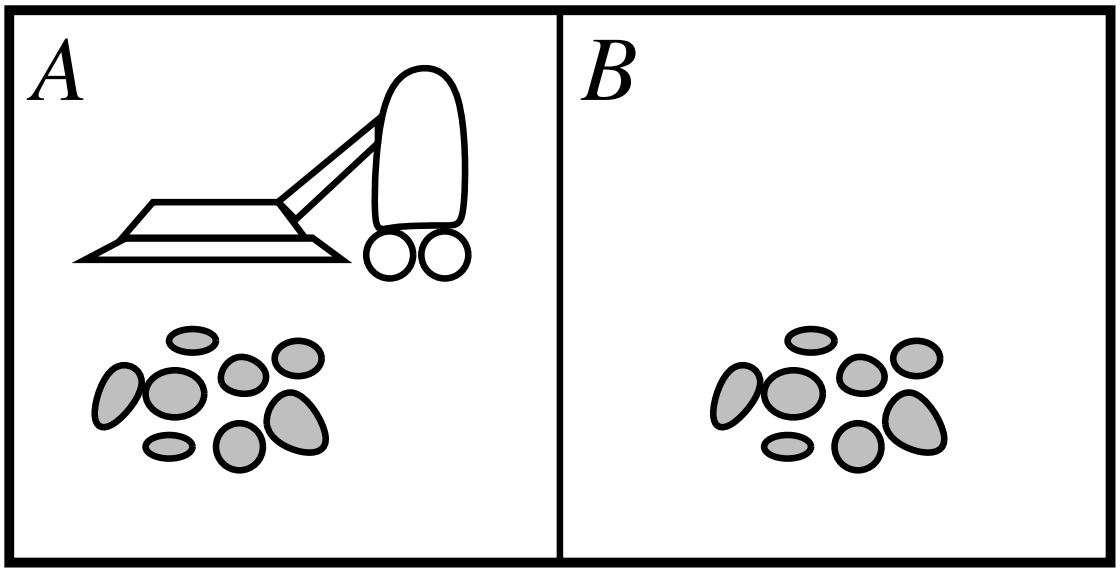
Ideally rational agent does whatever action is expected to maximize some performance measure – the agent may not know the performance measure (Russell and Norvig 2010)

Resource bounded agent must make "good enough" decisions based on its perceptual, computational and memory limitations (design tradeoffs)

Reactive Agent



Vacuum-cleaner world



Percepts: location and contents, e.g., [*A, Dirty*]

Actions: *Left, Right, Suck, NoOp*

A vacuum-cleaner -simple reflex agent

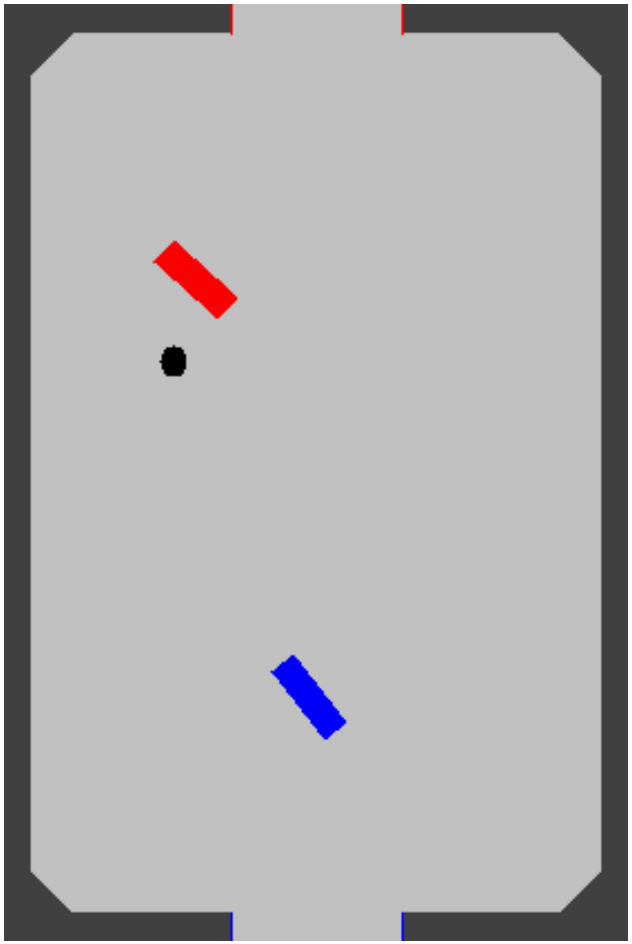
Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean]$, $[A, Clean]$	$Right$
$[A, Clean]$, $[A, Dirty]$	$Suck$
⋮	⋮

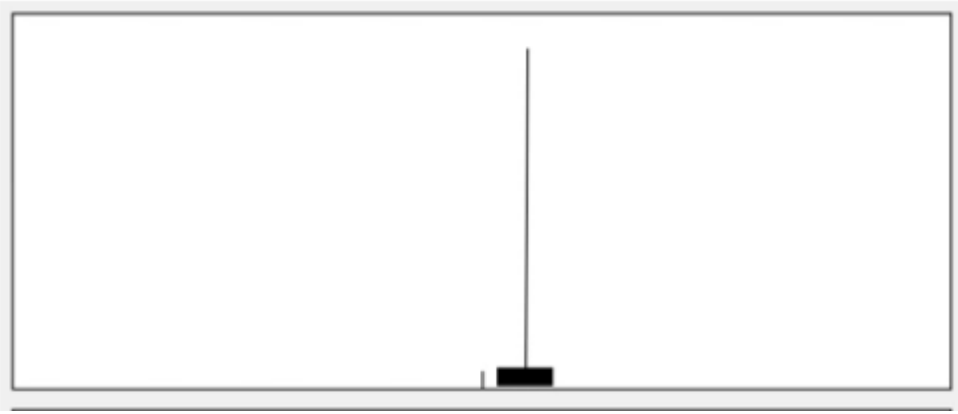
function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if $status = Dirty$ **then return** $Suck$
else if $location = A$ **then return** $Right$
else if $location = B$ **then return** $Left$

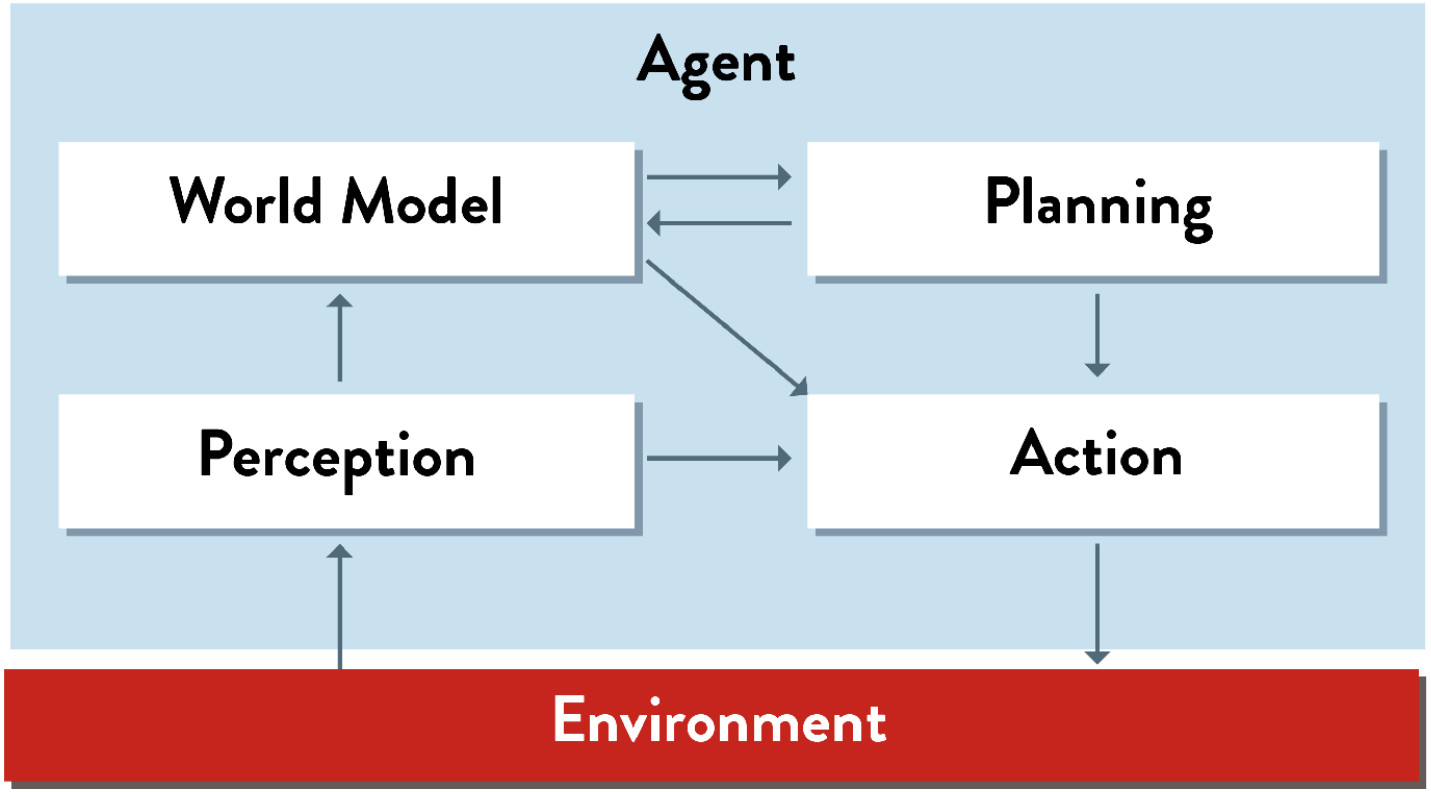
condition-action rules

Reactive Agent



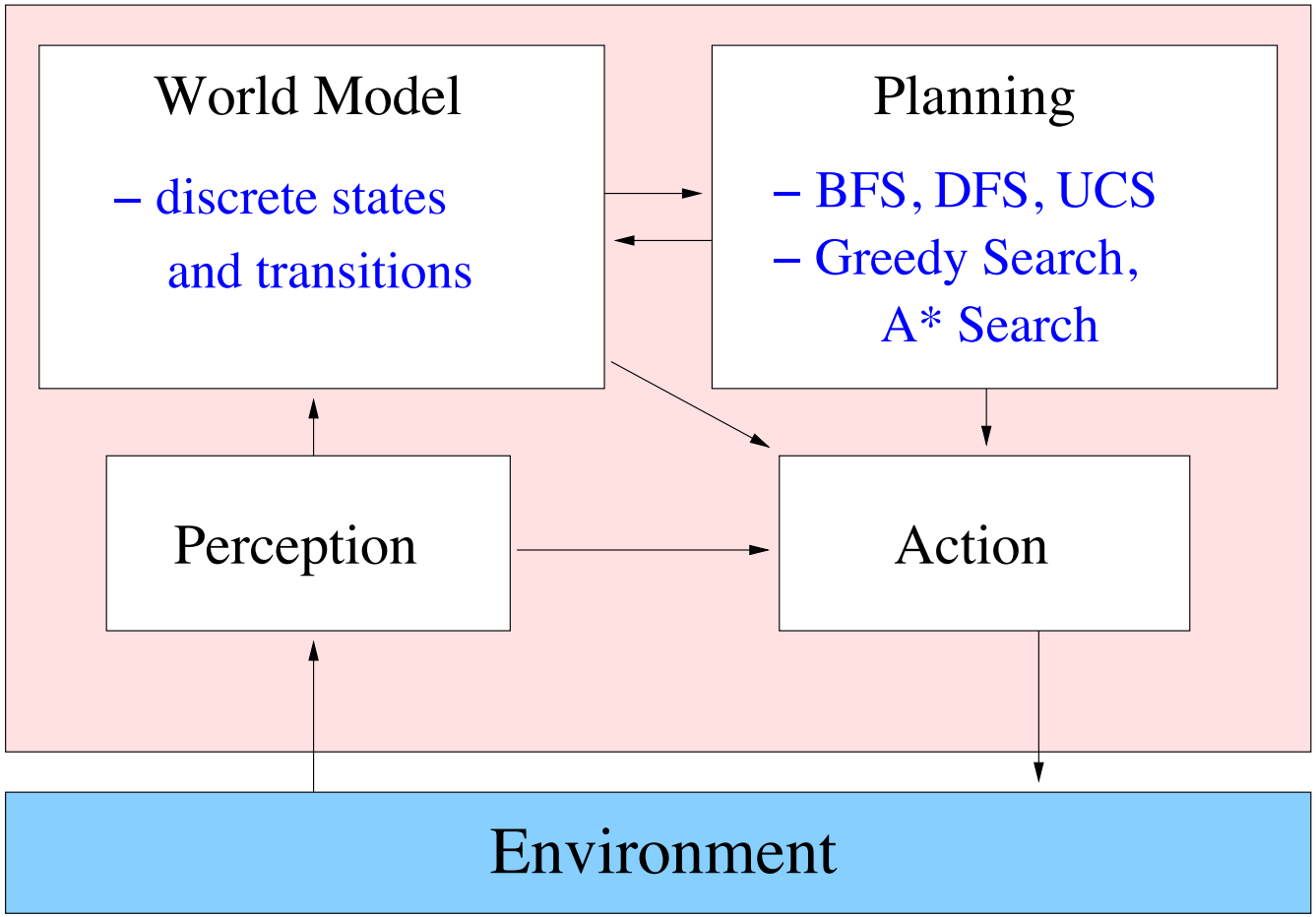


Planning Agent



Goal-Based Agent

Path Search Agent



Single-state Task Specification

A task is specified by states and actions: :

- state space e.g. other rooms
- initial state e.g., ” room 103”
- Transition - actions or operators (or successor function *S(x)*) = set of action–state pairs reachable from state x by performing a single action
- goal test, check if a state is goal state
  In this case, there is only one goal specified (“inside of room 123. ”)
- path cost e.g. sum of distances, number of actions etc.
- A solution is a sequence of actions leading from the initial state to a goal state

Path Search Algorithms - Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - "expand" it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

Search Strategies

- BFS and DFS treat all new nodes the same way:
  - BFS add all new nodes to the back of the queue
  - DFS add all new nodes to the front of the queue

- Best First Search uses an evaluation function f () to order the nodes in the queue;
  - Similar to UCS $f(n)$ = cost $g(n)$ of path from root to node n

- Informed or Heuristic search strategies incorporate into f () an estimate of distance to goal
  - Greedy Search $f(n)$ = estimate $h(n)$ of cost from node n to goal
  - A∗ Search $f(n) = g(n) + h(n)$

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 ∞)

Complexity Results for Uninformed Search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$	$O(b^k)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bk)$	$O(bd)$
Complete?	Yes[1]	Yes[2]	No	No	Yes[1]
Optimal ?	Yes[3]	Yes	No	No	Yes[3]

b = branching factor, d = depth of the shallowest solution,
m = maximum depth of the search tree, k = depth limit.

1 = complete if b is finite.

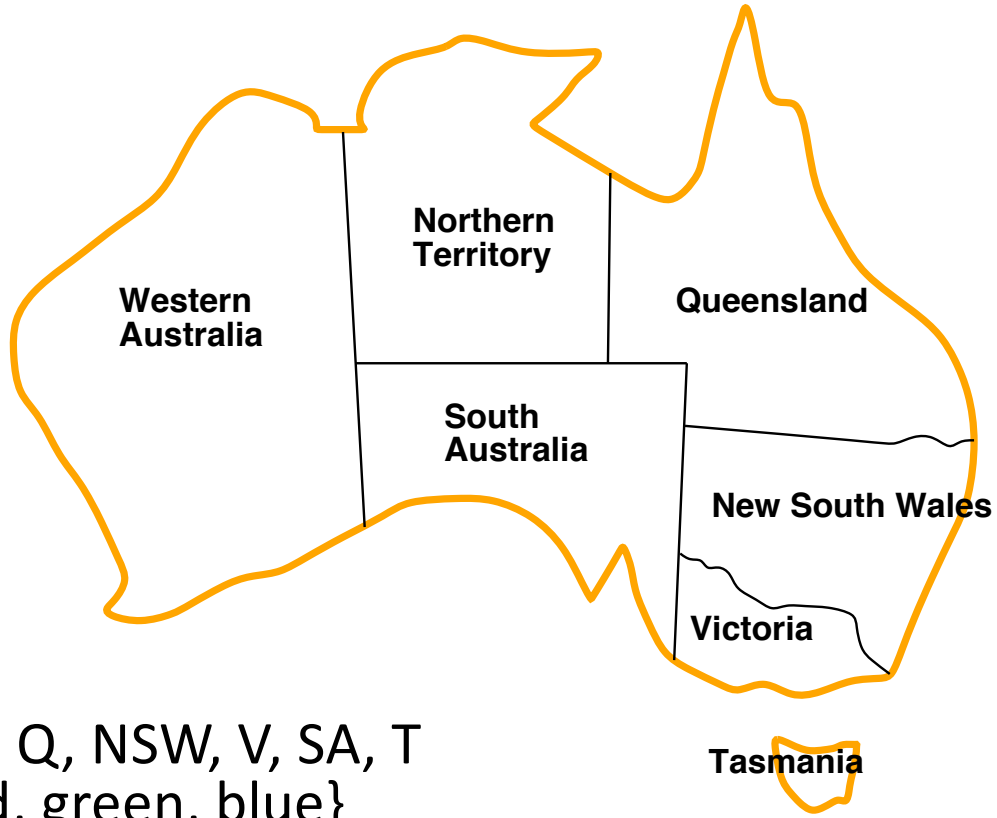
2 = complete if b is finite and step costs $\geq \varepsilon$ with $\varepsilon > 0$.

3 = optimal if actions all have the same cost.

Constraint Satisfaction Problems (CSPs)

- Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C.
- The aim is to find an assignment of the variables X_i from the domains D_i in such a way that none of the constraints C are violated.

Example: Map-Coloring

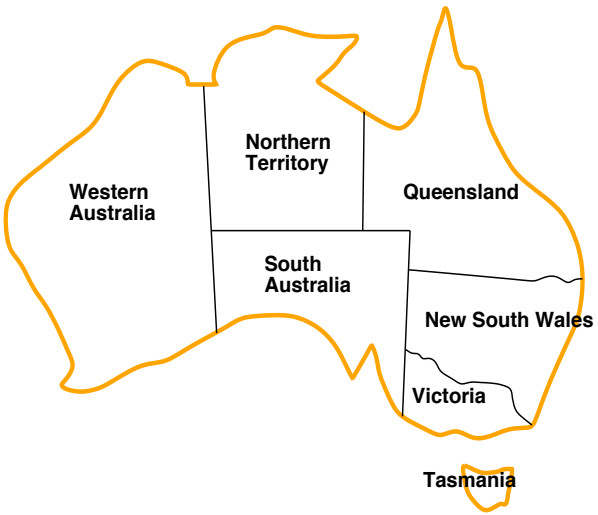


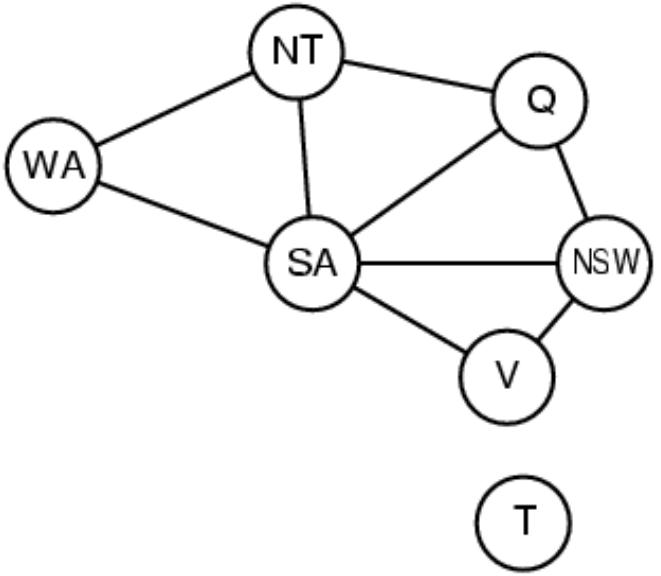
Variables WA, NT, Q, NSW, V, SA, T
Domains *Di* = {red, green, blue}
Constraints: adjacent regions must have different colors

e.g. WA≠NT, etc.

Constraint graph

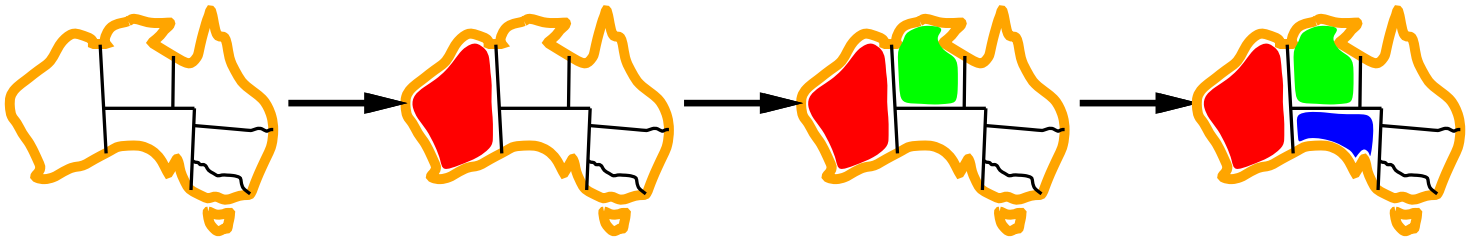
Constraint graph: nodes are variables, arcs are constraints





Binary CSP: each constraint relates two variables

Constraint Satisfaction Problems

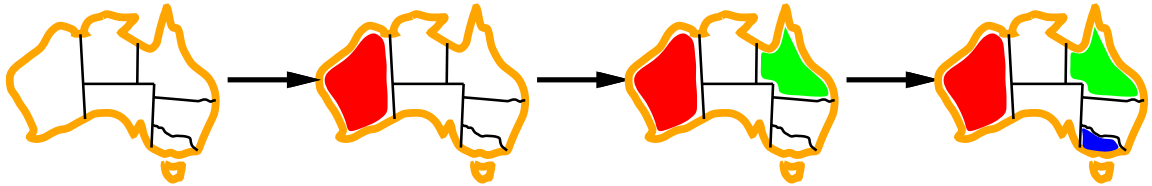


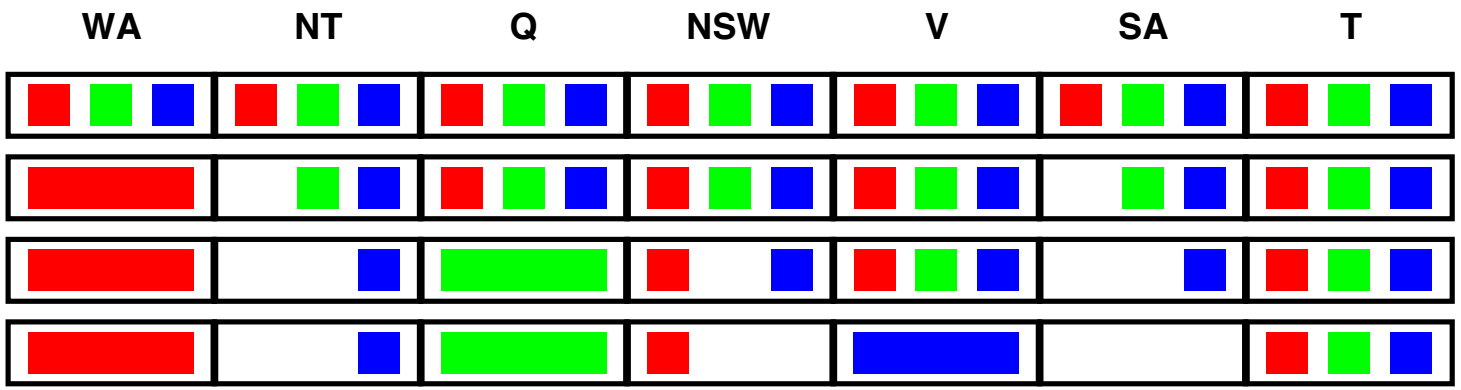
- Backtracking search
- Enhancements to backtracking search
- Local search
  - hill climbing
  - simulated annealing

Forward checking

Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



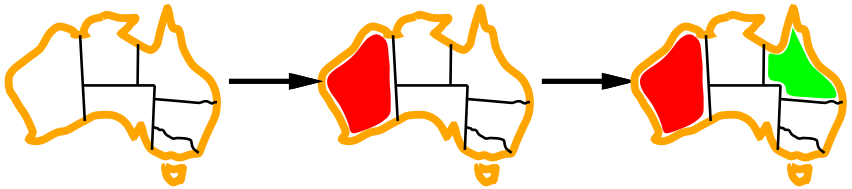


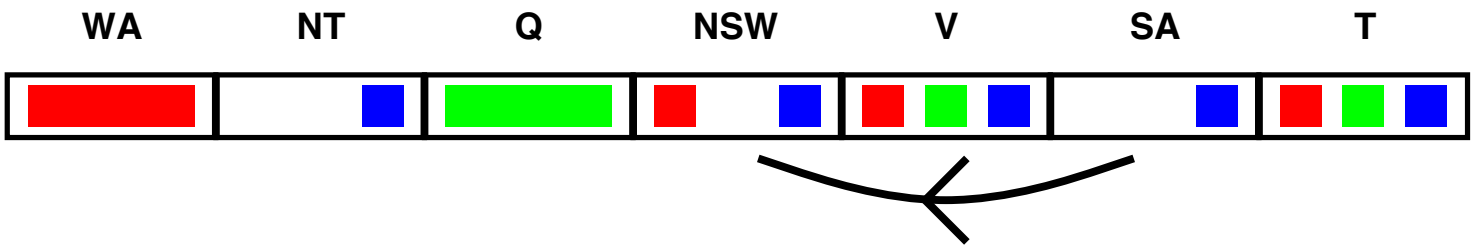
Arc consistency

Simplest form of propagation makes each arc consistent

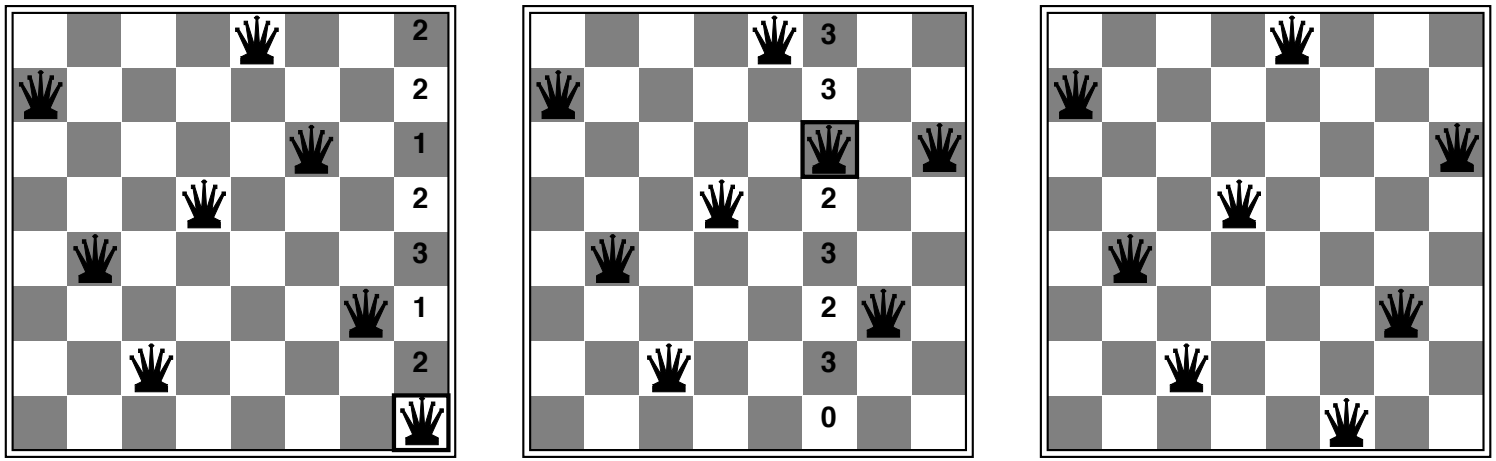
$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y



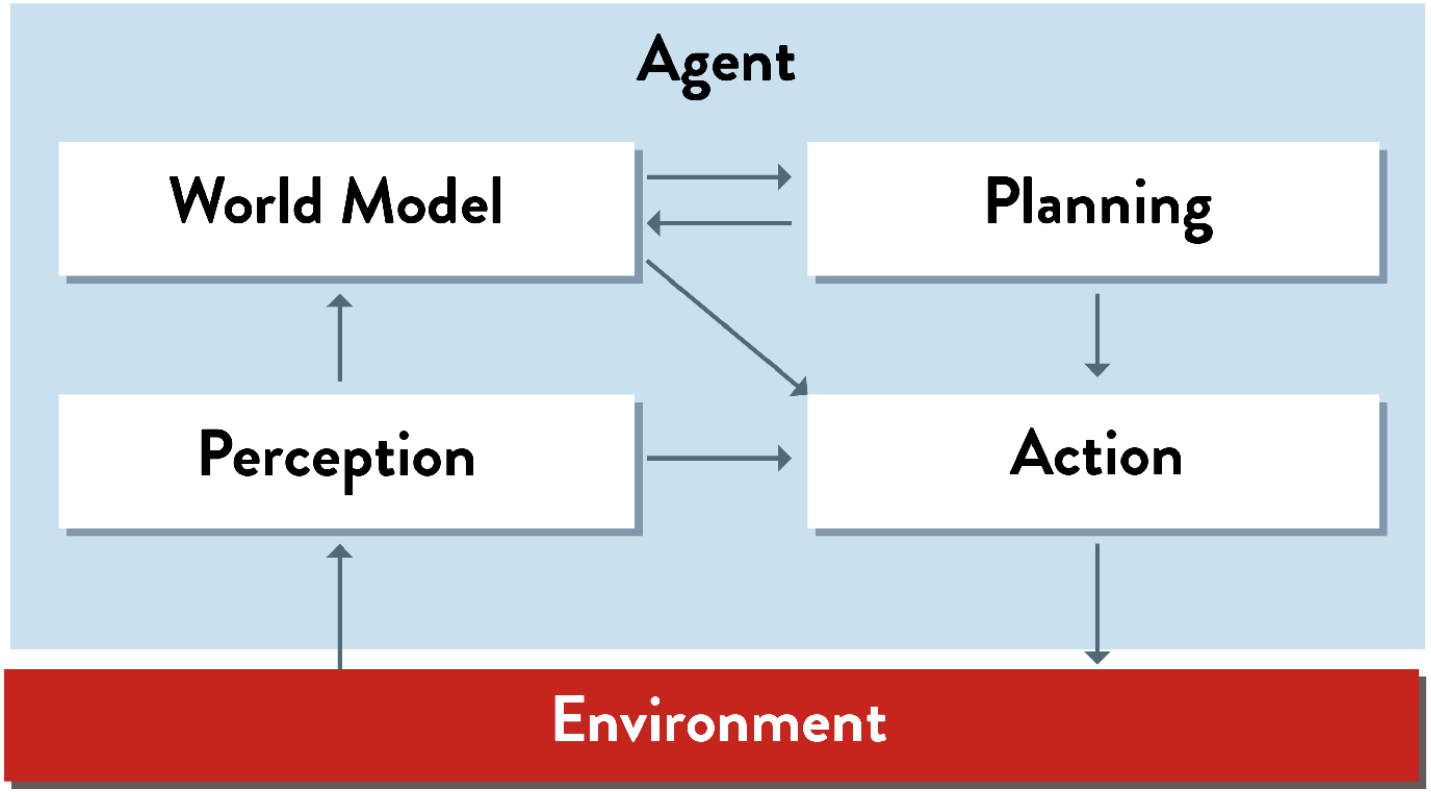


Hill-climbing by min-conflicts



- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

Planning Agent



Goal-Based Agent

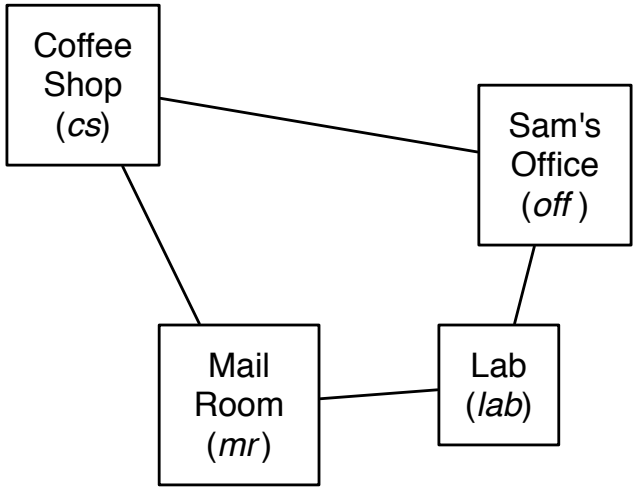
Representation

- How to represent a classical planning problem?
- Representing with States, Actions, and Goals

Actions

- A deterministic action is a partial function from states to states.
- The preconditions of an action specify when the action can be carried out.
- The effect of an action specifies the resulting state.

Delivery Robot Example



Features:
RLoc – Rob's location
RHC – Rob has coffee
SWC – Sam wants coffee
MW – Mail is waiting
RHM – Rob has mail

Features to describe states

Actions:
mc – move clockwise
mcc – move counterclockwise
puc – pickup coffee
dc – deliver coffee
pum – pickup mail
dm – deliver mail

Robot actions

STRIPS Representation

- Divide the features into:
  - primitive features
  - derived features. There are rules specifying how derived can be derived from primitive features.
- For each action:
  - precondition that specifies when the action can be carried out.
  - effect a set of assignments of values to primitive features that are made true by this action.

STRIPS assumption: every primitive feature not mentioned in the effects is unaffected by the action.

Example STRIPS representation

Pick-up coffee (puc):

- precondition: [cs,¬rhc]
- effect: [rhc]

Deliver coffee (dc):

- precondition: [off,rhc]
- effect: [¬rhc,¬swc]

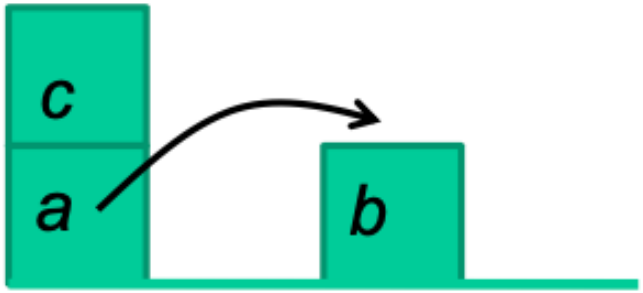
Feature-based representation of actions

- For each action:
  - precondition is a proposition that specifies when the action can be carried out.
- For each feature:
  - causal rules that specify when the feature gets a new value and
  - frame rules that specify when the feature keeps its value.

Defining Goals and possible Actions

- Example of goals:
  on(a,b), on(b,c)

- Example of action:
  move(a, 1, b)
  (Move block a from 1 to b)



- Action preconditions:
  clear(a), on(a,1), clear(b)

- Action effects:
  on(a,b), clear(1), ~on(a,1), ~clear(b)

"add" (true after action)

"delete" (no longer true after action)

Planning

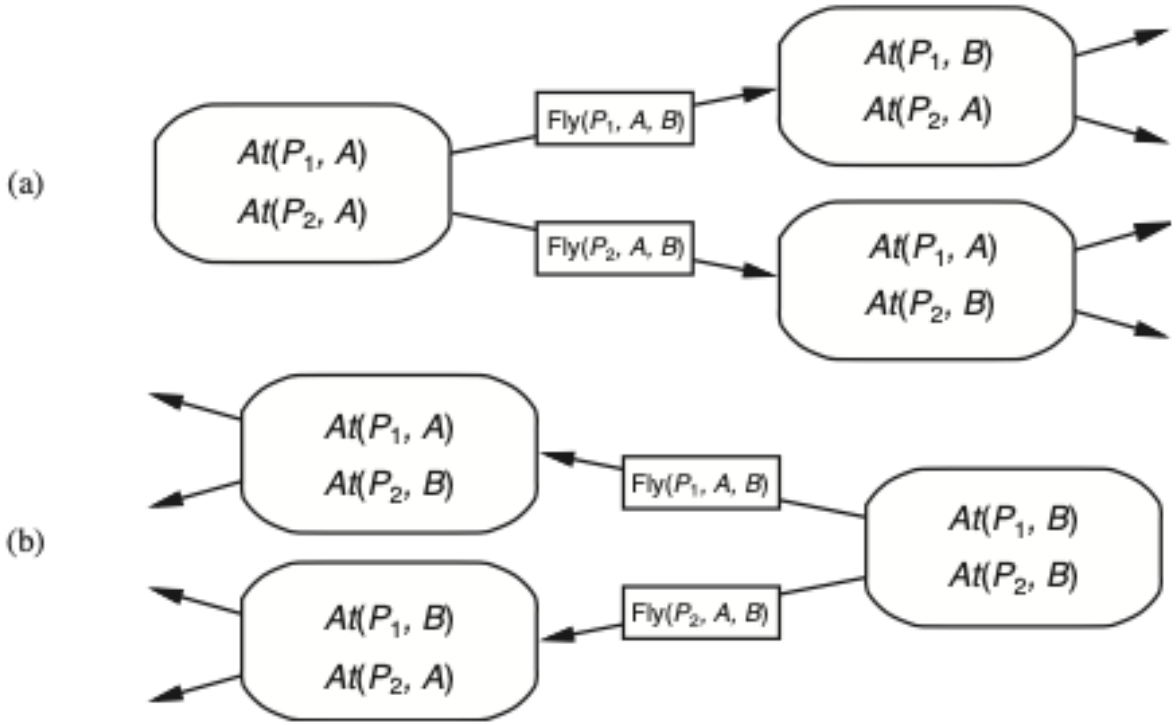
- Plan — sequence (or ordered set) of actions to achieve some goal
- Planner — problem solver that produces plans
- Goal – typically a conjunction of literals
- Initial State – typically a conjunction of literals

Blocks World Example for goal $on(B, C) \wedge on(C, Table)$

- $move(C, A, Table), move(B, Table, C)$

Simple Planning Algorithms

- Forward search and goal regression



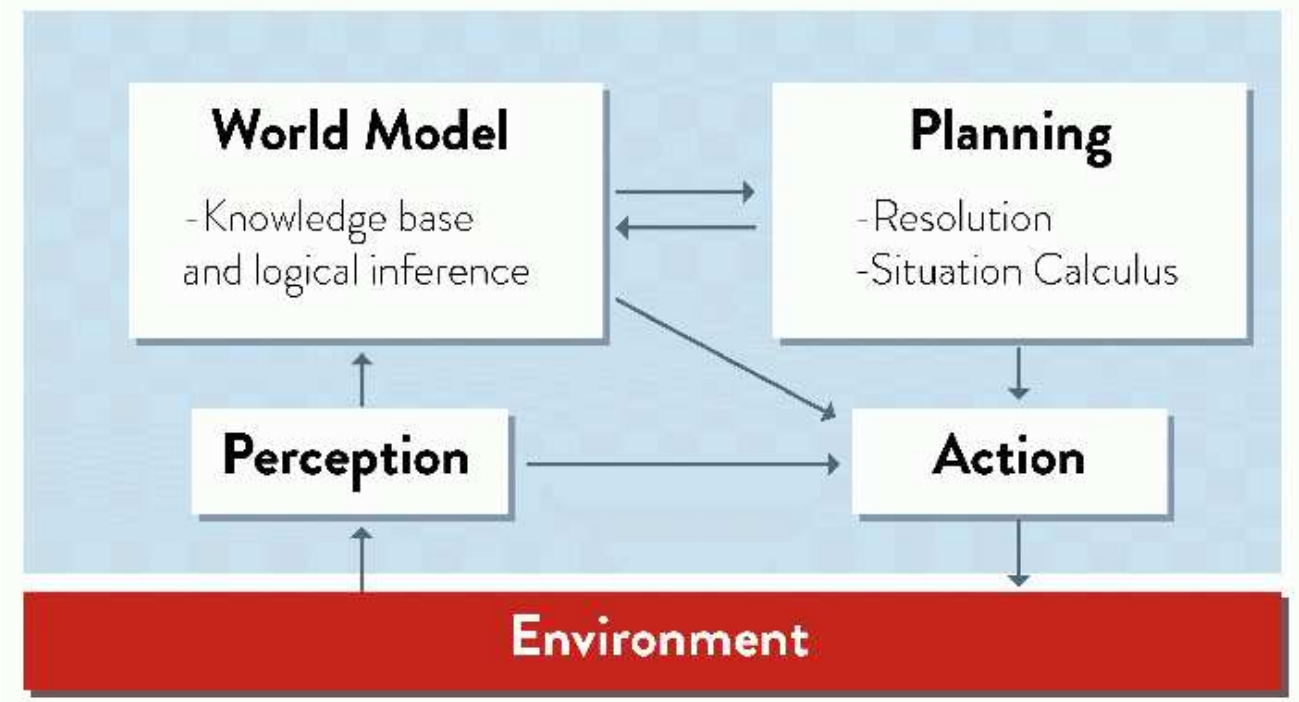
Problem with forward search is state space can be very large
Problem with regression is that it is hard and doesn't always work

Planning

- Planning is the process of choosing a sequence of actions to achieve a goal.
- An action is a partial function from a state to a state.
- Two representations for actions that exploit structure in states are
  - the STRIPS representation, which is an action-centric representation,
  - the feature-based representation of actions, which is a feature-centric representation.
  - The feature-based representation is more powerful than the STRIPS representation; it can represent anything representable in STRIPS, but can also represent conditional effects.

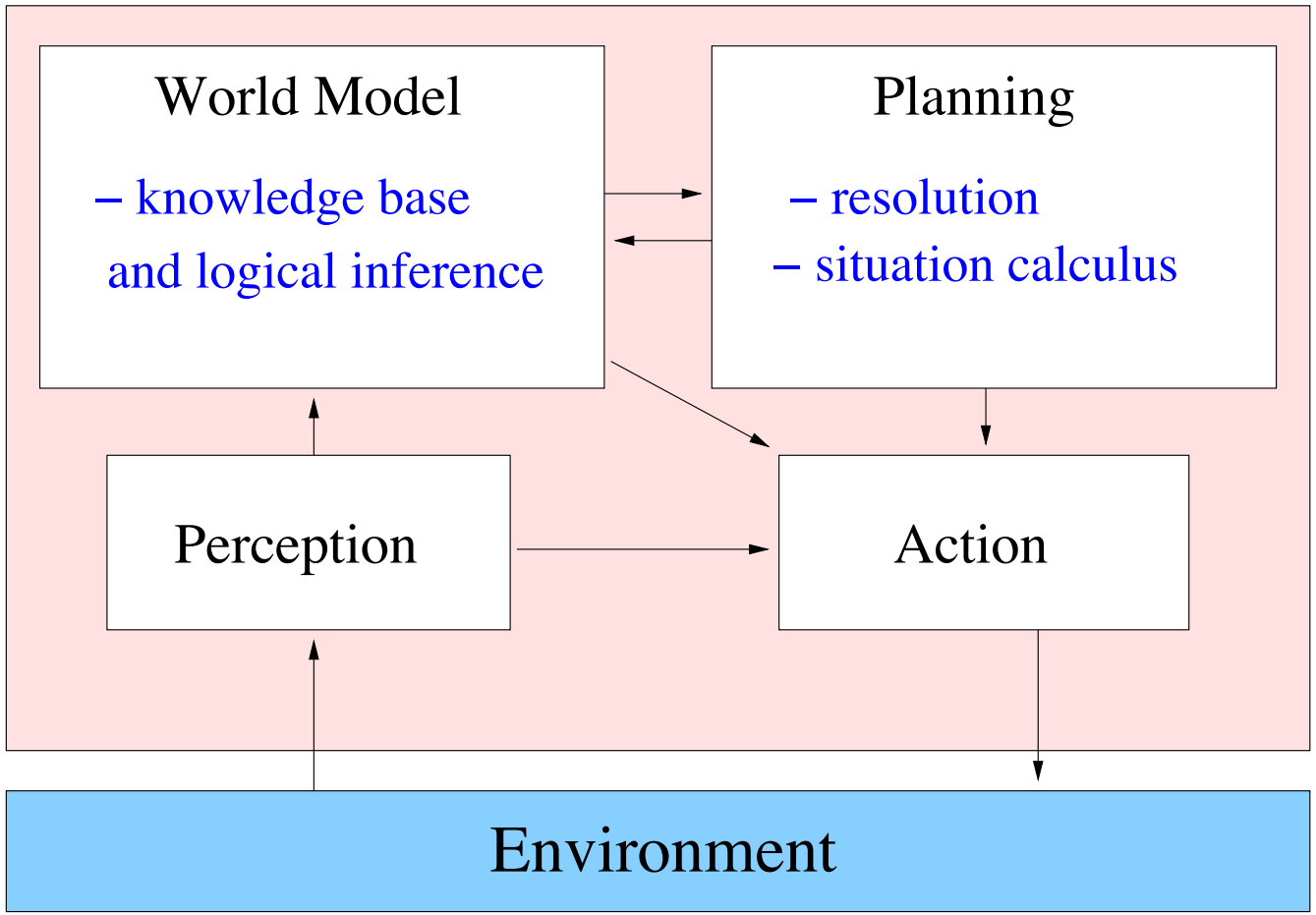
Propositions and Inference

Models and Planning



Some environments instead require a Knowledge Base of facts and a set of Logical Inference Rules to reason about those facts.

Logical Agent



Knowledge Based Agent

The agent must be able to:

- represent states, actions, etc.
- incorporate new percepts
- update internal representations of the world
- deduce hidden properties of the world
- determine appropriate actions

Propositional Logic: Syntax

Propositional logic is the simplest logic

- illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences.

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional Logic - Validity and Satisfiability

A sentence is valid if it is true in all models,

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

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model

e.g. $(A \vee B) \wedge C$

A sentence is unsatisfiable if it is true in no models

e.g. $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

i.e. prove α by *reductio ad absurdum*

Truth Tables

P	Q	$\neg$ P	P $\wedge$ Q	P $\vee$ Q	P $\Rightarrow$ Q
F	F	T	F	F	T
F	T	T	F	T	T
T	F	F	F	T	F
T	T	F	T	T	T

Semantics

- The semantics of the connectives can be given by truth tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
True	True	False	True	True	True	True
True	False	False	False	True	False	False
False	True	True	False	True	True	False
False	False	True	False	False	True	True

- One row for each possible assignment of True/False to variables
- **Important**: *P* and *Q* are any sentences, including complex sentences

Conjunctive Normal Form

In order to apply Resolution, we must first convert the KB into Conjunctive Normal Form (CNF).

This means that the KB is a conjunction of clauses, and each clause is a disjunction of (possibly negated) literals.

e.g. $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Resolution

Conjunctive Normal Form (CNF – universal)

conjunction of disjunctions of literals

clauses

e.g. $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

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

where ℓ_i and m_j are complementary literals. e.g.

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

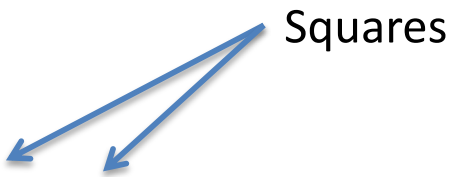
Resolution is sound and complete for propositional logic.

Syntax of First Order Logic

- Objects: people, houses, numbers, theories, colors, football games, wars, centuries ...
- Predicates: red, round, bogus, prime, multistoried, ... brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
- Functions: father of, best friend, third inning of, one more than,...

Syntax of First Order Logic

In the Wampus world:



Constants	$Gold, Wumpus, [1,2], [3,1]$, etc.
Predicates	$Adjacent(), Smell(), Breeze(), At()$
Functions	$Result()$
Variables	$x,\ y,\ a,\ t, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow\ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

Syntax is same as in propositional logics plus the Quantifies

Fun with Sentences

Brothers are siblings

$$\forall x, y \; Brother(x, y) \Rightarrow Sibling(x, y)$$

"Sibling" is symmetric

$$\forall x, y \; Sibling(x, y) \Leftrightarrow Sibling(y, x)$$

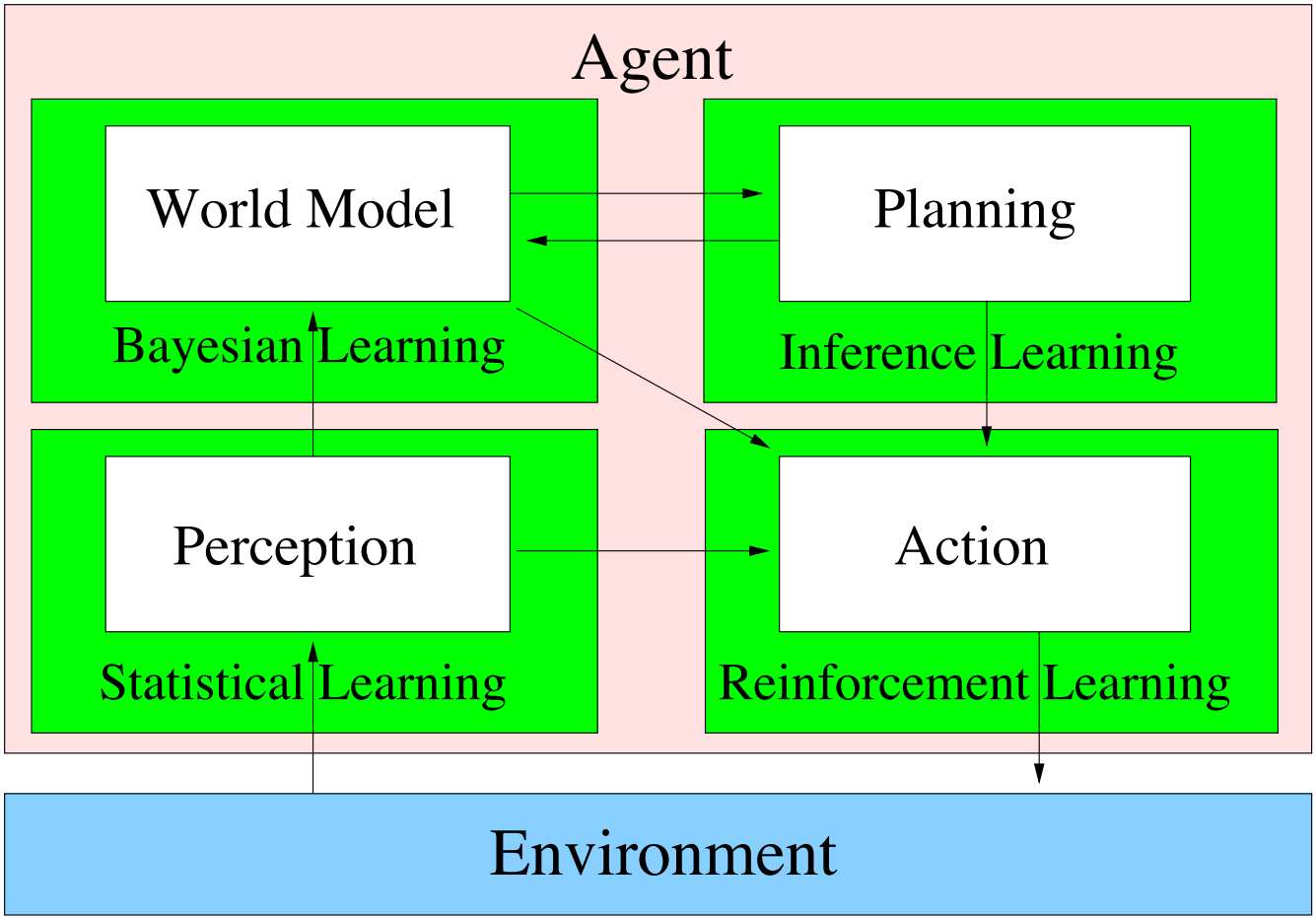
One's mother is one's female parent

$$\forall x, y \; Mother(x, y) \Leftrightarrow (Female(x) \wedge Parent(x, y))$$

A first cousin is a child of a parent's sibling

$$\forall x, yFirstCousin(x, y) \Leftrightarrow \exists p, ps \; Parent(p, x) \wedge Sibling(p, q) \wedge Parent(q, y)$$

Learning Agent



The Aim is to improve its performance on future tasks after making observations about the world.

Types of Learning

- Supervised Learning
  - Agent is presented with examples of inputs and their target outputs, and must learn a function from inputs to outputs that agrees with the training examples and generalizes to new examples
- Reinforcement Learning
  - Agent is not presented with target outputs for each input, but is periodically given a reward, and must learn to maximize (expected) rewards over time
- Unsupervised Learning
  - Agent is only presented with a series of inputs, and must find and aims to find structure in these inputs

Supervised Learning

- We have a training set and a test set, each consisting of a set of items; for each item, a number of input attributes and a target value are specified.
- The aim is to predict the target value, based on the input attributes.
- Agent Is presented with the input and target output for each item in the training set; it must then predict the output for each item in the test set
- Various learning paradigms are available:
  - Decision Tree
  - Neural Network
  - Support Vector Machine, etc.

Inductive learning

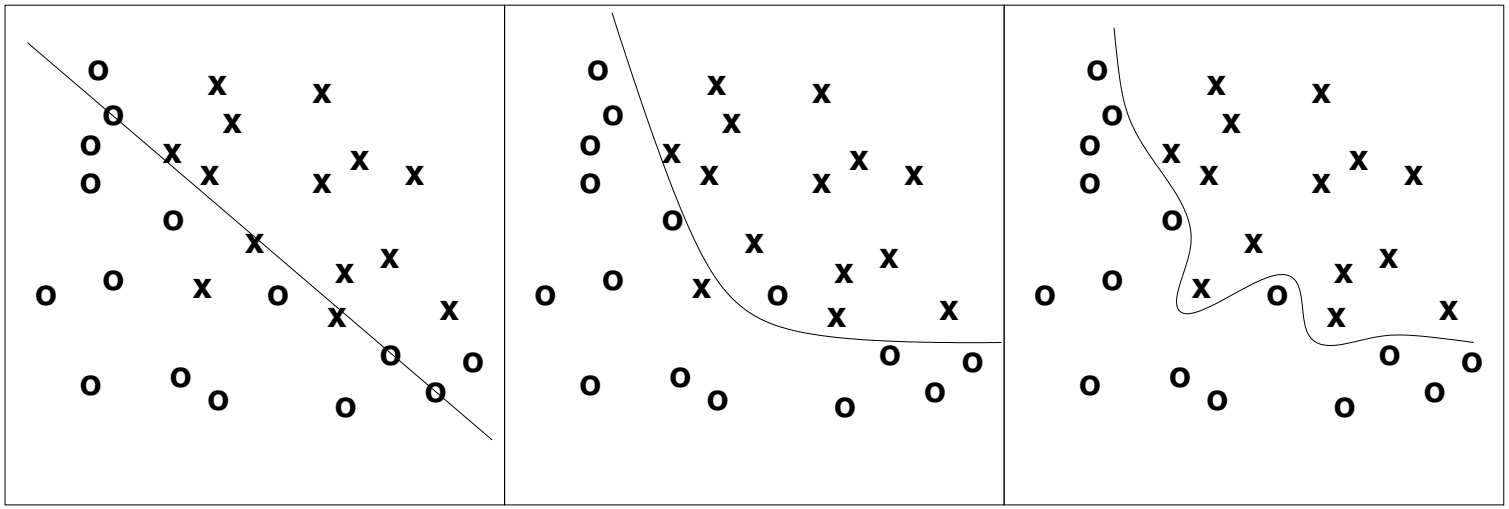
Simplest form: learn a function from examples

f is the target function
An example is a pair (*x*, *f(x)*)

Problem: find a hypothesis *h*
such that $h \approx f$
given a training set of examples

Ockham's razor

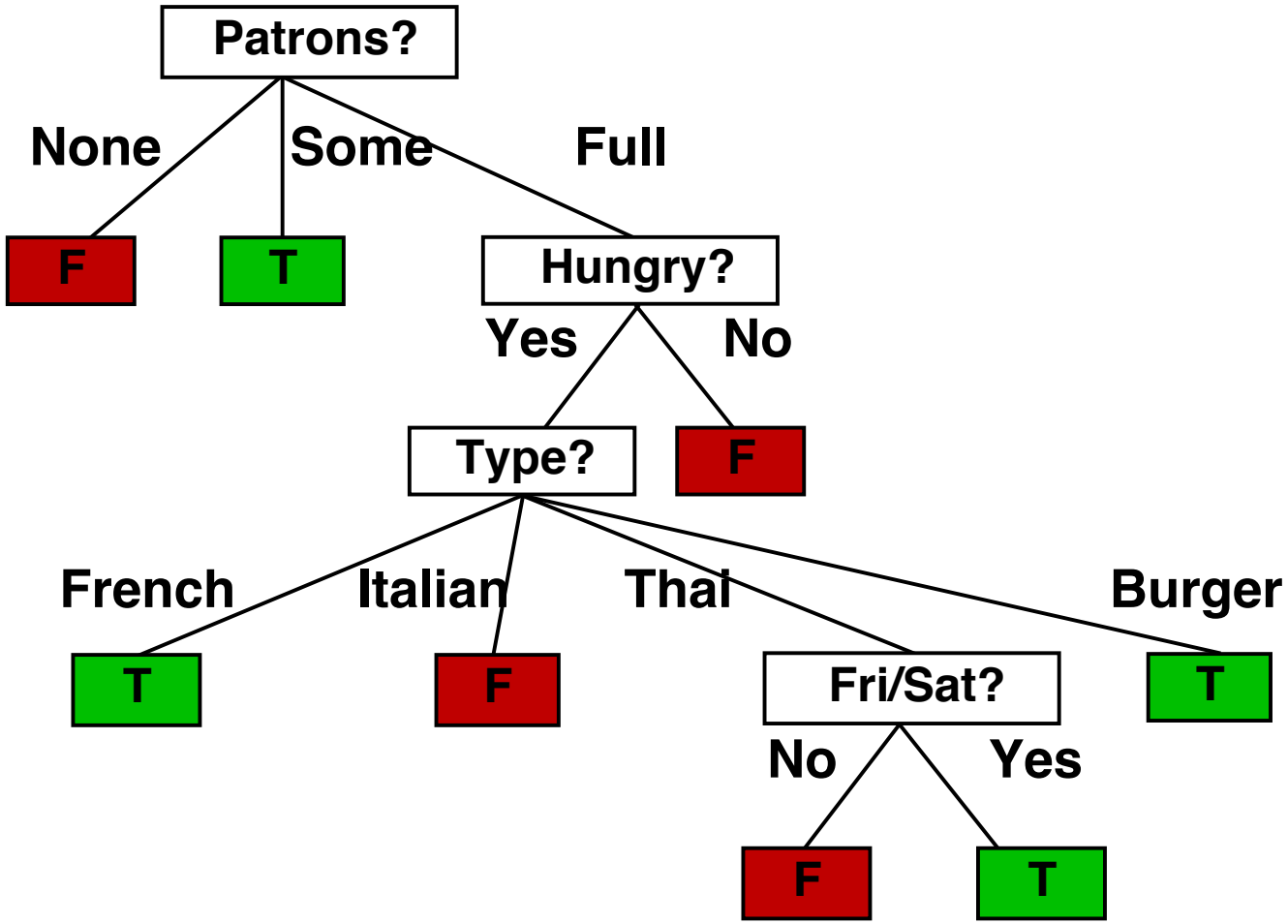
"The most likely hypothesis is the simplest one consistent with the data."



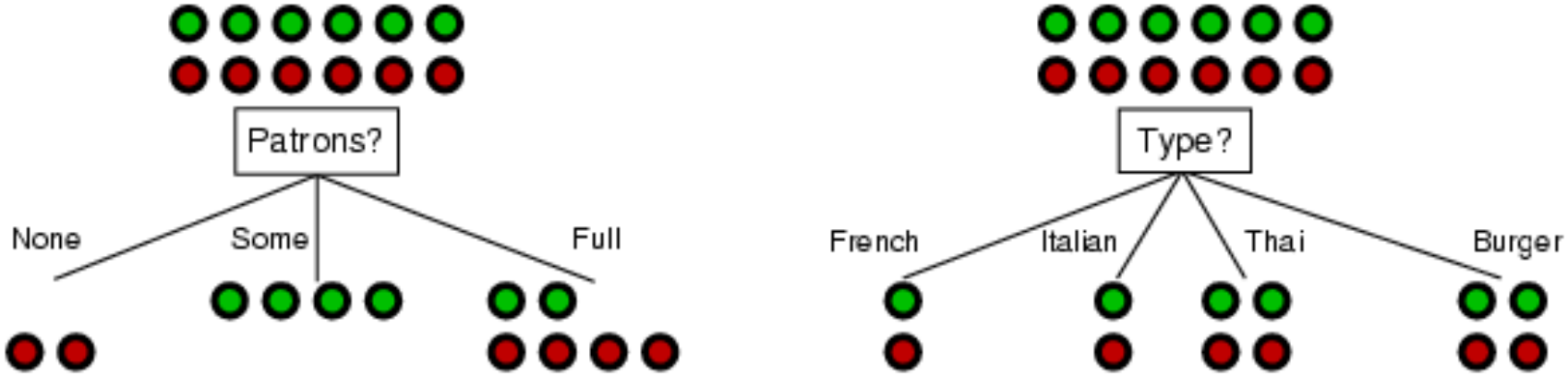
inadequate good compromise over-fitting

Since there can be noise in the measurements, in practice need to make a tradeoff between simplicity of the hypothesis and how well it fits the data.

Induced Tree



Choosing an attribute



- *Patrons?* is a better choice
- Patrons is a "more informative" attribute than Type, because it splits the examples more nearly into sets that are "all positive" or "all negative".
- This notion of "informativeness" can be quantified using the mathematical concept of "entropy".
- A parsimonious tree can be built by minimizing the entropy at each step

Entropy

- Suppose we have *p* positive and *n* negative examples at a node.
  $\rightarrow H(\langle p/(p+n), n/(p+n)\rangle)$ bits needed to classify a new example.
  - e.g. for 12 restaurant examples, *p* = *n* = 6 so we need 1 bit.

- An attribute splits the examples *E* into subsets E_i, each of which (we hope) needs less information to complete the classification.

  Let E_i have p_i positive and n_i negative examples
  $\rightarrow H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$ s needed to classify a new example
  $\rightarrow$ expected number of bits per example over all branches is

$$\sum_i \frac{p_i+n_i}{p+n} H(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\rangle)$$

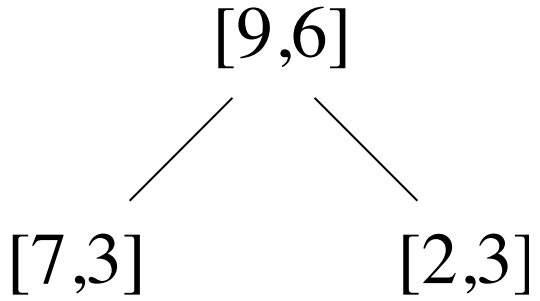
For Patrons, this is 0.459 bits, for Type this is (still) 1 bit.

Minimal Error Pruning

Should the children of this node be pruned or not?

Left child has class frequencies [7,3]

$$E = 1 - \frac{n+1}{N+k} = 1 - \frac{7+1}{10+2} = 0.333$$



Right child has $E = 0.429$

Parent node has $E = 0.412$

Average for Left and Right child is

$$E = \frac{10}{15}(0.333) + \frac{5}{15}(0.429) = 0.365$$

Since $0.365 < 0.412$, children should NOT be pruned.

Reasoning with Uncertainty

Probability and Uncertainty

Subjective or Bayesian probability:

Probabilities relate propositions to one's own state of knowledge
e.g. $P(A30 | \text{no reported accidents}) = 0.06$

These are not claims of a "probabilistic tendency" in the current situation (but might be learned from past experience of similar situations)

Probabilities of propositions change with new evidence:

e.g. $P(A30 | \text{no reported accidents, 5 a.m.}) = 0.15$

Prior probability

Prior or unconditional probabilities of propositions

e.g. $P(\texttt{Cavity = true}) = 0.1$ and $P(\texttt{Weather = sunny}) = 0.72$

correspond to belief prior to arrival of any (new) evidence.

Probability distribution gives values for all possible assignments:

$P(\texttt{Weather}) = \langle 0.72, 0.1, 0.08, 0.1 \rangle$ (normalized, i.e., sums to 1)

Joint probability

Joint probability distribution for a set of r.v.'s gives the probability of every atomic event on those r.v's (i.e., every sample point)

$P(\texttt{Weather}, \texttt{Cavity})$ is a 4×2 matrix of values:

$Weather =$	$sunny$	$rain$	$cloudy$	$snow$
$Cavity = \texttt{true}$	0.144	0.02	0.016	0.02
$Cavity = \texttt{false}$	0.576	0.08	0.064	0.08

Every question about a domain can be answered by the joint distribution because every event is a sum of sample points.

Probability and Uncertainty - Inference by Enumeration

Start with the joint distribution:

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬ *cavity*	**.016**	**.064**	**.144**	**.576**

Can also compute conditional probabilities:

$$P(\neg \texttt{cavity} \mid \texttt{toothache}) = \frac{P(\neg \texttt{cavity} \wedge \texttt{toothache})}{P(\texttt{toothache})}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Conditional Probability

If we consider two random variables *a* and *b*, with $P(b) \neq 0$, then the conditional probability of *a* given *b* is

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

Alternative formulation: $P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$

Bayes' Rule

The formula for conditional probability can be manipulated to find a relationship when the two variables are swapped:

$$P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$$

$$\rightarrow \text{Bayes' rule } P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

This is often useful for assessing the probability of an underlying cause after an effect has been observed:

$$P(\text{Cause}|\text{Effect}) = \frac{P(\text{Effect}|\text{Cause})P(\text{Cause})}{P(\text{Effect})}$$

Bayes' Rule

Product rule $P(a \wedge b) = P(a|b)P(b) = P(b|a)P(a)$

$$\rightarrow \text{Bayes' rule } P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

Useful for assessing diagnostic probability from causal probability:

$$P(\text{Cause}|\text{Effect}) = \frac{P(\text{Effect}|\text{Cause})P(\text{Cause})}{P(\text{Effect})}$$

e.g., let M be meningitis, S be stiff neck:

$$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = \frac{0.8 \times 0.0001}{0.1} = 0.0008$$

Note: posterior probability of meningitis still very small!

Bayesian Networks - Bayesian network

- A Bayesian network (also Bayesian Belief Network, probabilistic network, causal network, knowledge map) is a directed acyclic graph (DAG) where
  - Each node corresponds to a random variable
  - Directed links connect pairs of nodes – a directed link from node
  - *X* to node *Y* means that *X* has a direct influence on *Y*
  - Each node has a conditional probability table quantifying effect of parents on node
- Independence assumption of Bayesian networks
  - Each random variable is (conditionally) independent of its nondescendants given its parents

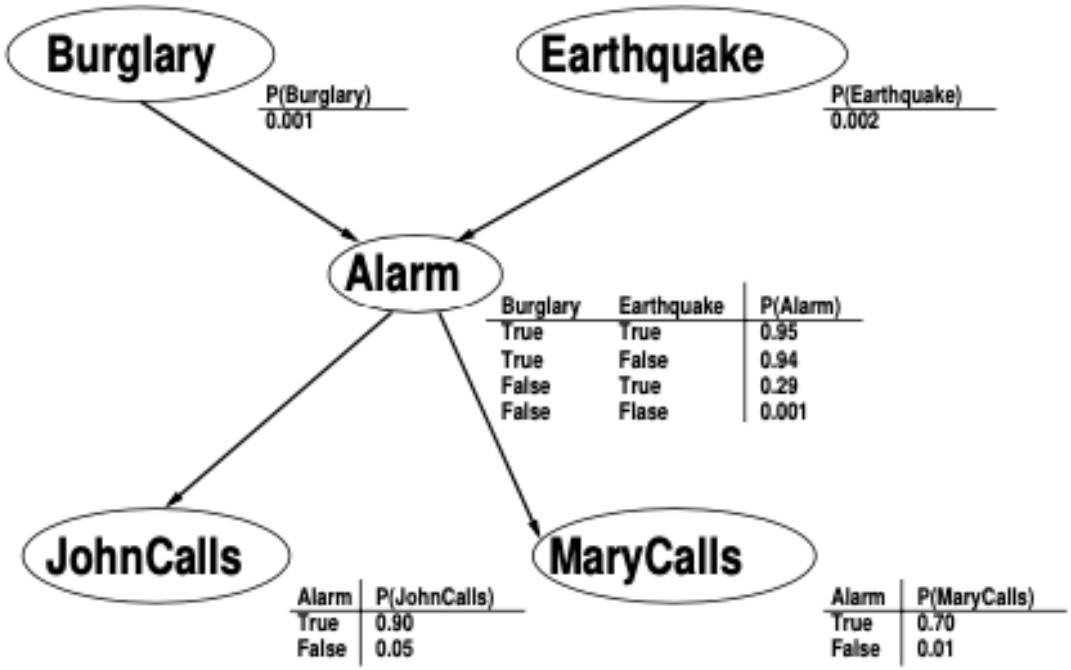
Bayesian networks - Bayesian network

- Independence and conditional independence relationships among variables can greatly reduce the number of probabilities that need to be specified in order to define the full joint distribution.
- Bayesian networks can represent essentially *any* full joint probability distribution
  - in many cases can do so very concisely.

- The probability over all of the variables, $P(X_1,X_2,\ldots,X_n)$ is the **joint probability distribution**.
- A belief network defines a **factorization** of the joint probability distribution into a product of conditional probabilities.

Bayesian Networks

- Example (Pearl, 1988)



- Probabilities summarize potentially infinite set of possible circumstances

Semantics of Bayesian Networks

- Bayesian network provides a complete description of the domain
- Joint probability distribution can be determined from the network

$$P(X_1, X_2, \cdots, X_n) = \prod_{i=1}^{n} P(X_i | Parents(X_i))$$

- For example, $P(J \wedge M \wedge A \wedge \neg B \wedge \neg E) =$
  $P(J|A).P(M|A).P(A|\neg B \wedge \neg E).P(\neg B).P(\neg E) =$
  $0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.000628$
- Bayesian network is a complete and non-redundant representation of domain (and can be far more compact than joint probability distribution)

Inference in Bayesian Networks

- Diagnostic Inference From effects to causes

$$P(Burglary|JohnCalls) = 0.016$$

- Causal Inference From causes to effects

$$P(JohnCalls|Burglary) = 0.85; P(MaryCalls|Burglary) = 0.67$$

- Intercausal Inference Explaining away

$P(Burglary|Alarm) = 0.3736$ but adding evidence, $P(Burglary|Alarm \wedge Earthquake) = 0.003$; despite the fact that burglaries and earthquakes are independent, the presence of one makes the other **much** less likely

- Mixed Inference Combinations of the patterns above

Diagnostic + Causal: $P(Alarm|JohnCalls \wedge \neg Earthquake)$

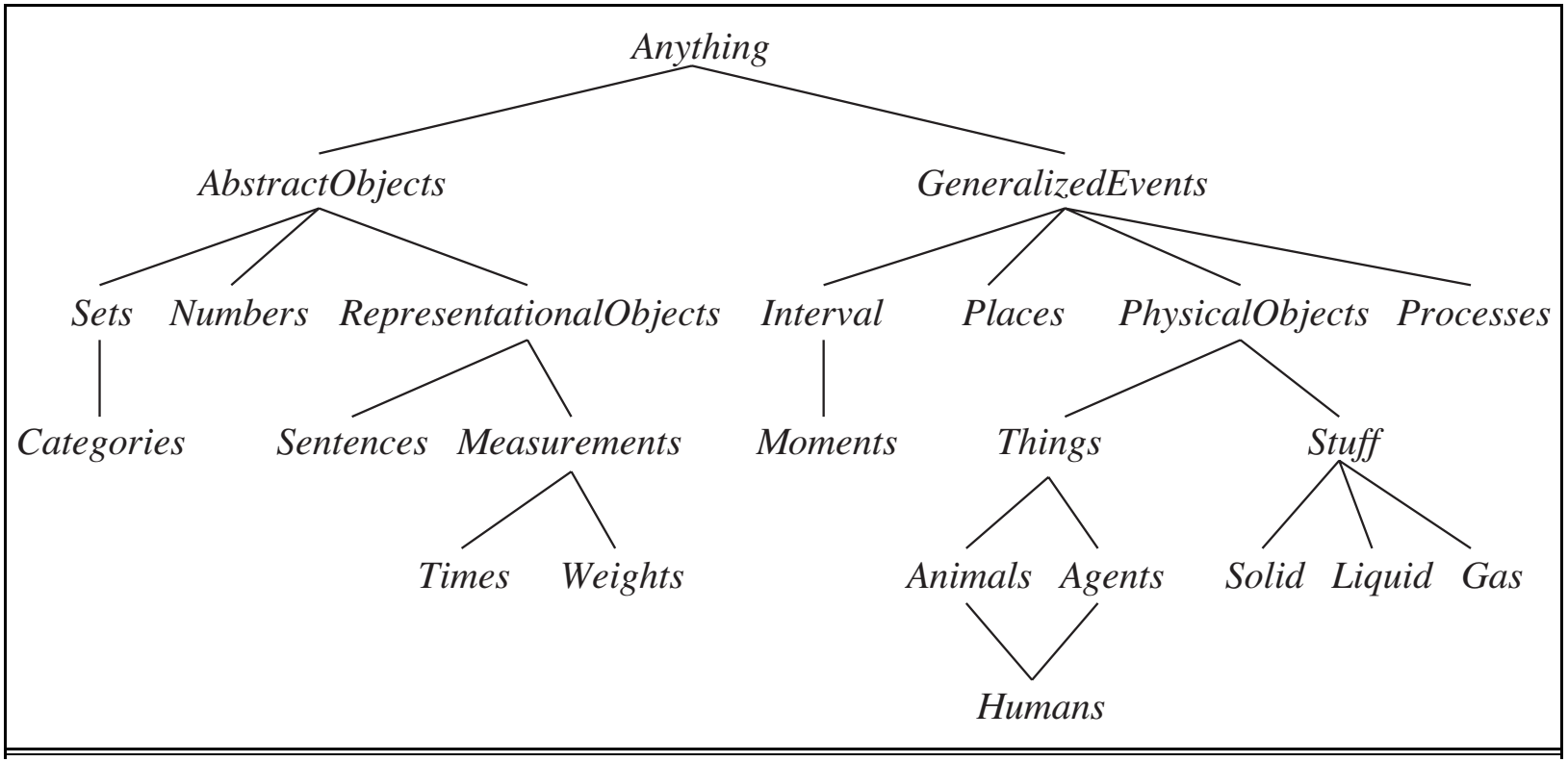
Intercausal + Diagnostic: $P(Burglary|JohnCalls \wedge \neg Earthquake)$

Knowledge Representation

Knowledge Representation and Reasoning

- A knowledge-based agent has at its core a knowledge base
- A knowledge base is an explicit set of sentences about some domain expressed in a suitable formal representation language
- Sentences express facts (true) or non-facts (false)
- Fundamental Questions
  - How do we write down knowledge about a domain/problem?
  - How do we automate reasoning to deduce new facts or ensure consistency of a knowledge base?

Ontologies - example



An ontology of the world,
Each link indicates that the lower concept is a specialization of the upper one. Specializations are not necessarily disjoint;
- a human is both an animal and an agent

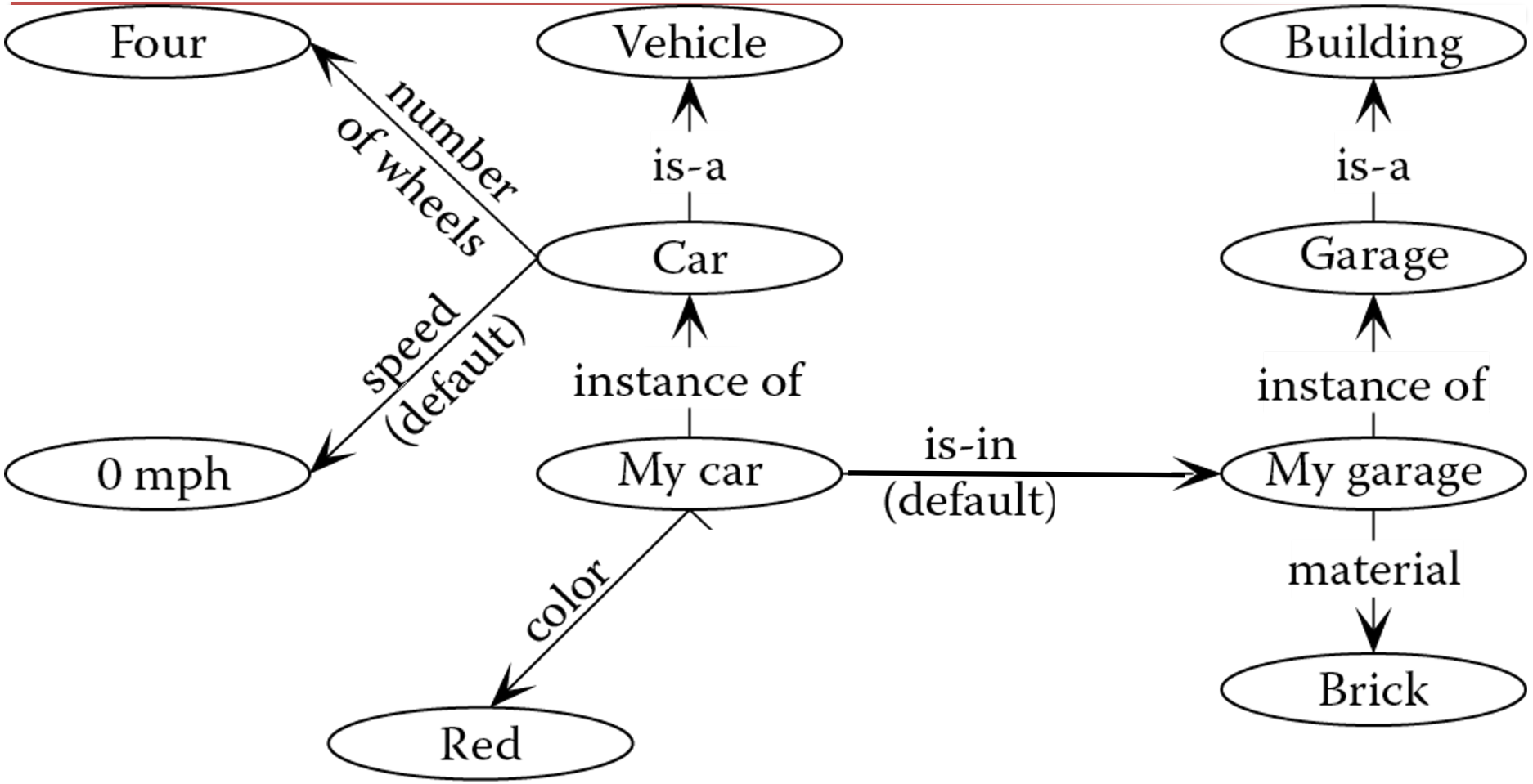
Categories and Objects

- Categories serve to organize and simplify the knowledge base through inheritance.
- If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible.
- We say that the individual apples inherit the property of edibility, in this case from their membership in the Food category.

Example – facts, objects and relations

- My car is a **car** (static relationship)
- A car is a **vehicle** (static relationship)
- A car has four wheels (static attribute)
- A car's speed is 0 mph (default attribute)
- My car is red (static attribute)
- My car is in my garage (default relationship)
- My garage is a **garage** (static relationship)
- A garage is a **building** (static relationship)
- My garage is made from brick (static attribute)
- My car is in the High Street (transient relationship)
- The High Street is a **street** (static relationship)
- A street is a **road** (static relationship)

A semantic network (with a default)



Semantic Networks - reasoning

- The central reasoning mechanism in a semantic network is inheritance: a category and its instances inherit the properties of the categories that contain them.
- The advantages of the semantic network architecture are that it is easy and natural to use, its meaning can be defined precisely, and the inheritance mechanism is easy to implement and efficient.

The Knowledge Base – rule based

- The simplest type of rule is called a production rule and takes the form

```
if <condition> then <conclusion>
```

- Production rule for dealing with the payroll of ACME, Inc., might be

```
rule r1_1

if the employer of Person is acme

then the salary of Person becomes large.
```

The Knowledge Base – rule based

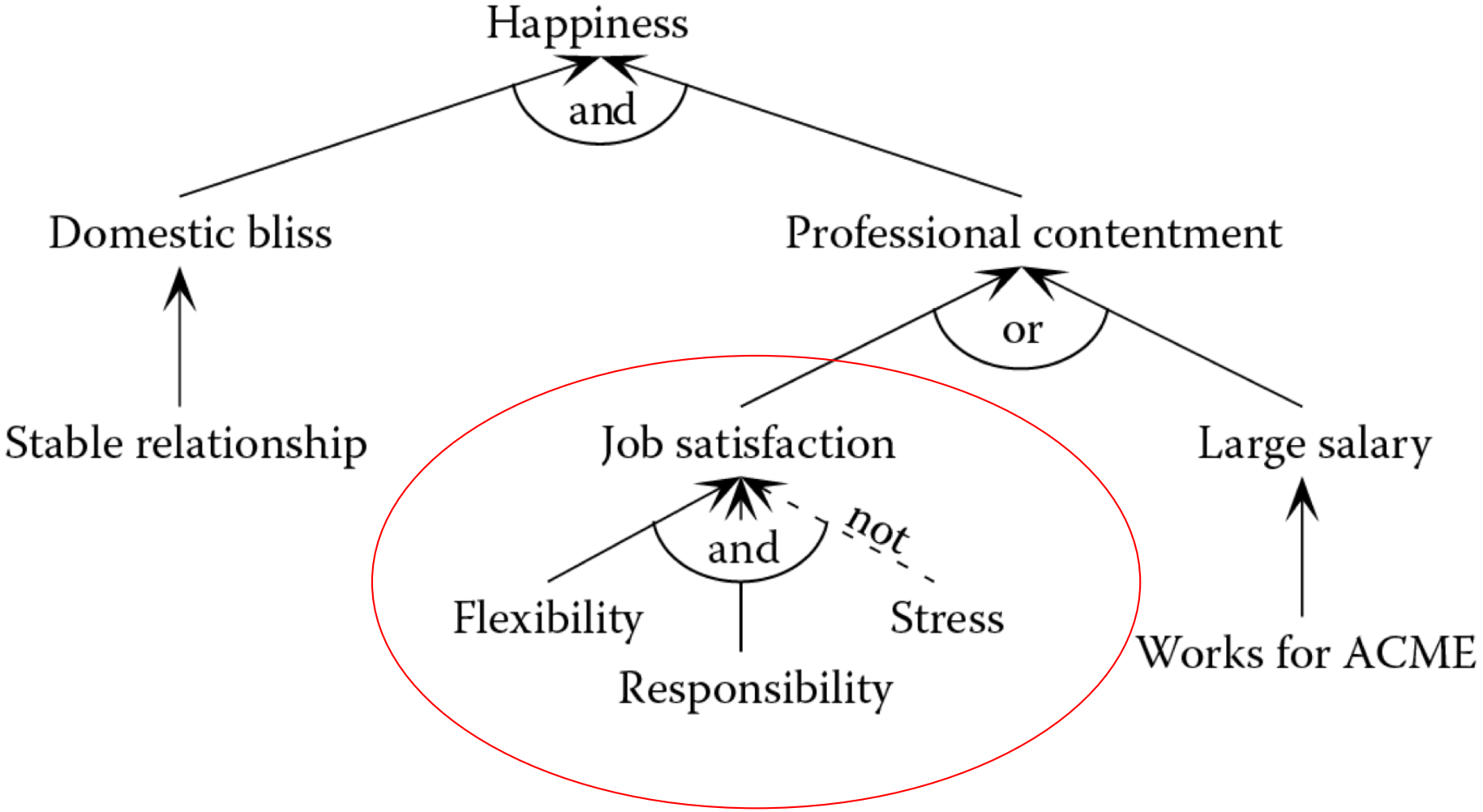
```
rule r1_1

if the employer of Person is acme

then the salary of Person becomes large.
```

```
/* fact f1_1 */

the employer of joe_bloggs is acme.
```

```
rule r1_2

if the salary of Person is large

or the job_satisfaction of Person is true

then the professional_contentment of Person becomes true.
```

```
/* derived fact f1_2 */

the salary of joe_bloggs is large.
```

An inference network



Deduction, Abduction and Induction

- The rules that make up the inference network about "happiness" and the network taken as a whole, are used to link cause and effect:

  if <cause> then <effect>

We can infer that:

if

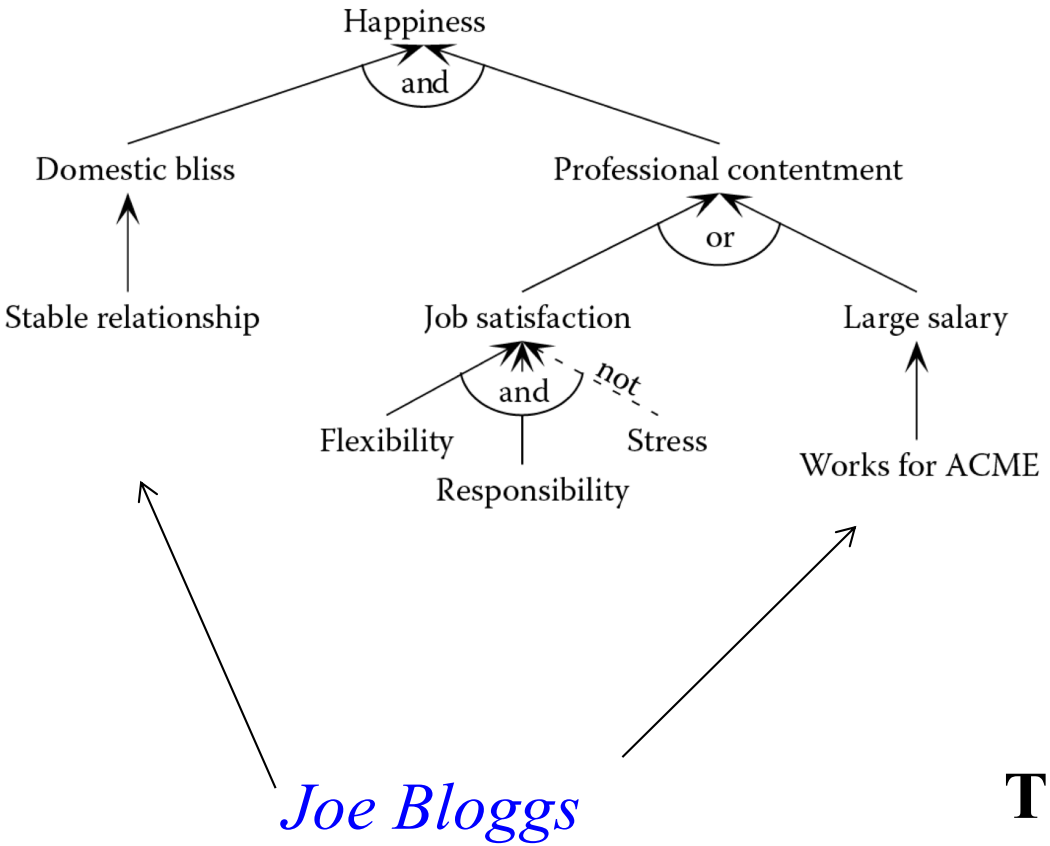
Joe Bloggs works for ACME and is in a stable relationship (the causes),

then

he is happy (the effect).

Deduction, Abduction, and Induction

if <cause> then <effect>



We can infer that:
if *Joe Bloggs* works for ACME and is in a stable relationship (the causes),
then
he is happy (the effect).

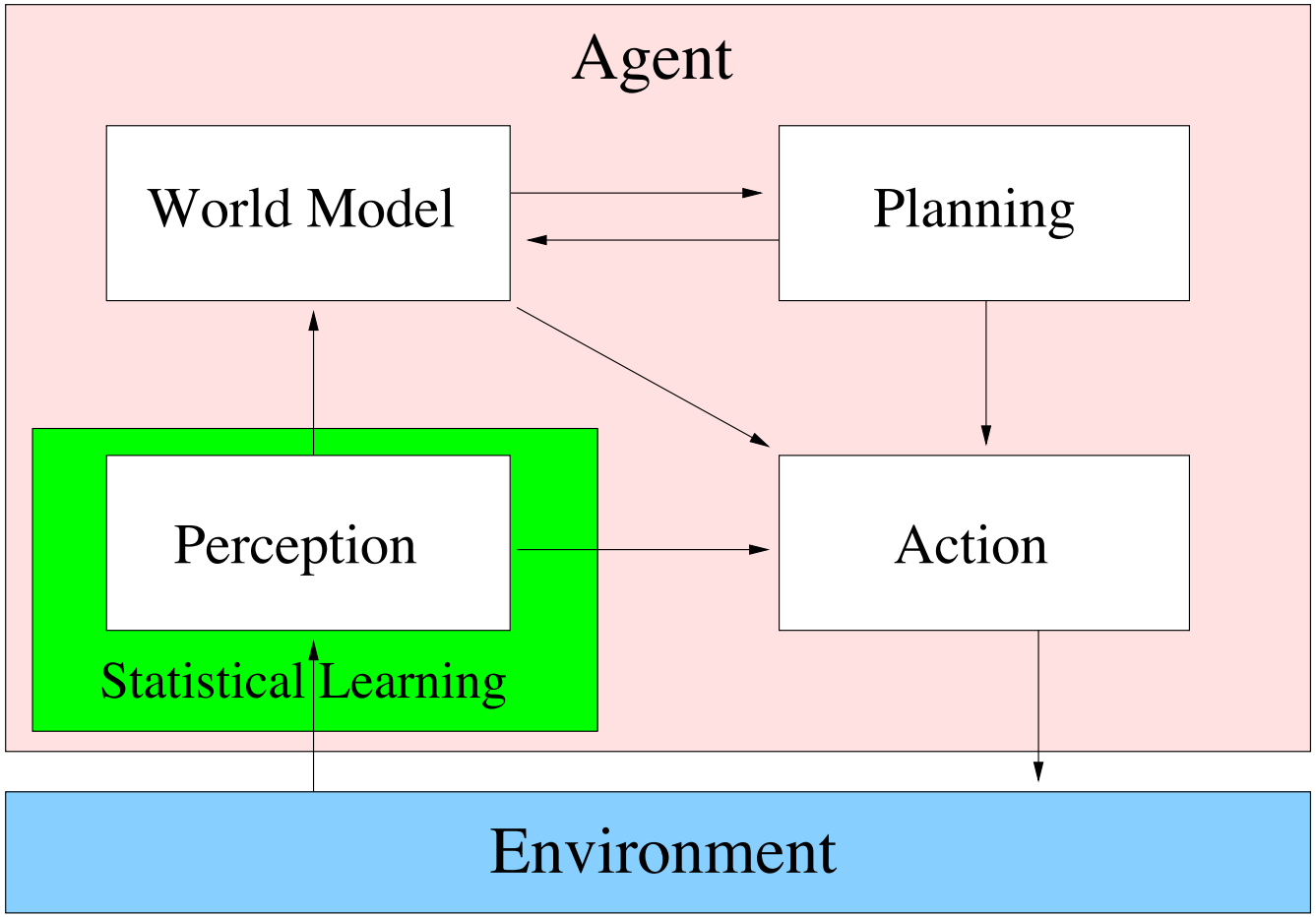
This is the process of deduction.

Deduction, Abduction, and Induction

We can summarize deduction, abduction, and induction as follows:

- deduction: cause + rule ⇒ effect
- abduction: effect + rule ⇒ cause
- induction: cause + effect ⇒ rule

Statistical Learning Agent



Artificial Neural Networks

(Artificial) Neural Networks are made up of nodes which have

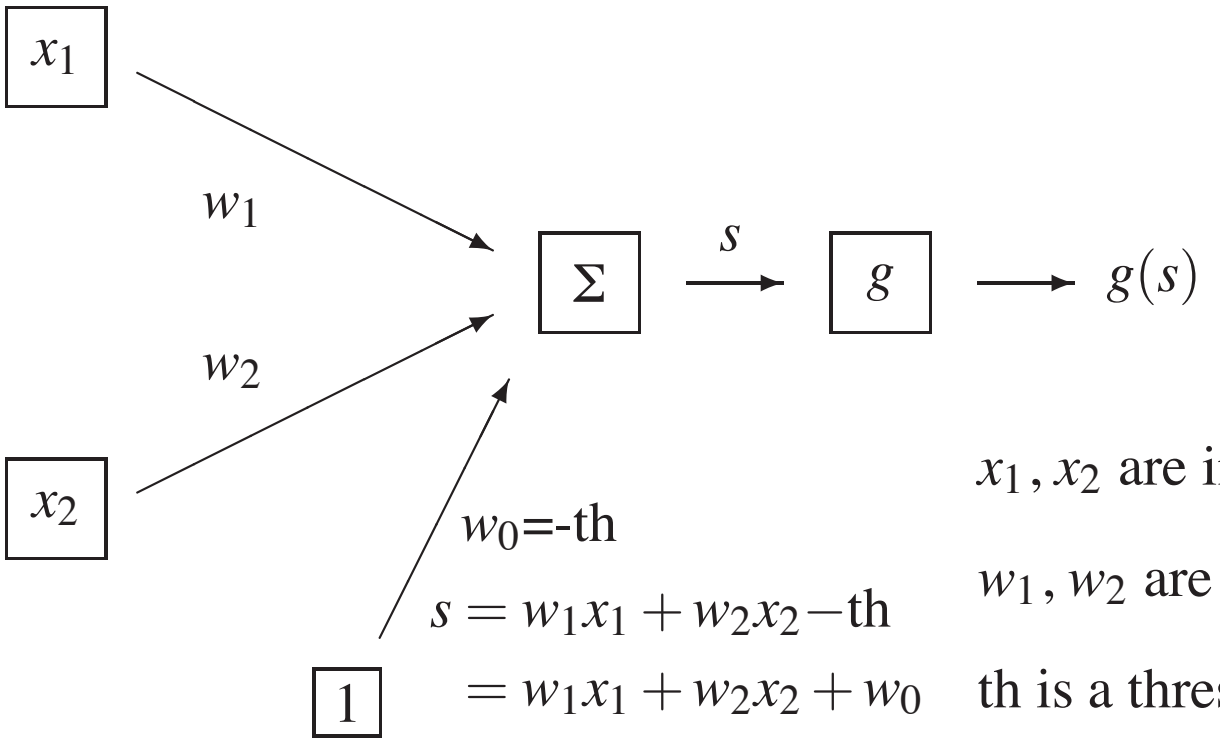
- inputs edges, each with some weight
- outputs edges (with weights)
- an activation level (a function of the inputs)

- Weights can be positive or negative and may change over time (learning).
- The input function is the weighted sum of the activation levels of inputs.
- The activation level is a non-linear transfer function *g* of this input:

$$\text{activation}_i = g(s_i) = g(\sum_j w_{ij} x_j)$$

- Some nodes are inputs (sensing), some are outputs (action)

McCulloch & Pitts Model of a Single Neuron



x_1, x_2 are inputs

w_1, w_2 are synaptic weights

th is a threshold

w_0 is a **bias** weight

g is transfer function

A graphical representation of the McCulloch-Pitts model

Linear Separability

Q: what kind of functions can a perceptron compute?

A: linearly separable functions

Examples include:

$$\begin{array}{ll} \text{AND} & w_1 = w_2 = 1.0, \quad w_0 = -1.5 \\ \text{OR} & w_1 = w_2 = 1.0, \quad w_0 = -0.5 \\ \text{NOR} & w_1 = w_2 = -1.0, \quad w_0 = 0.5 \end{array}$$

Q: How can we train it to learn a new function?

Perceptron Learning Rule

Adjust the weights as each input is presented.

recall: $s = w_1 x_1 + w_2 x_2 + w_0$

if $g(s) = 0$ but should be 1,

$$\begin{aligned} w_k &\leftarrow w_k + \eta x_k \\ w_0 &\leftarrow w_0 + \eta \\ \text{so} \quad s &\leftarrow s + \eta (1 + \sum_k x_k^2) \end{aligned}$$

if $g(s) = 1$ but should be 0,

$$\begin{aligned} w_k &\leftarrow w_k - \eta x_k \\ w_0 &\leftarrow w_0 - \eta \\ \text{so} \quad s &\leftarrow s - \eta (1 + \sum_k x_k^2) \end{aligned}$$

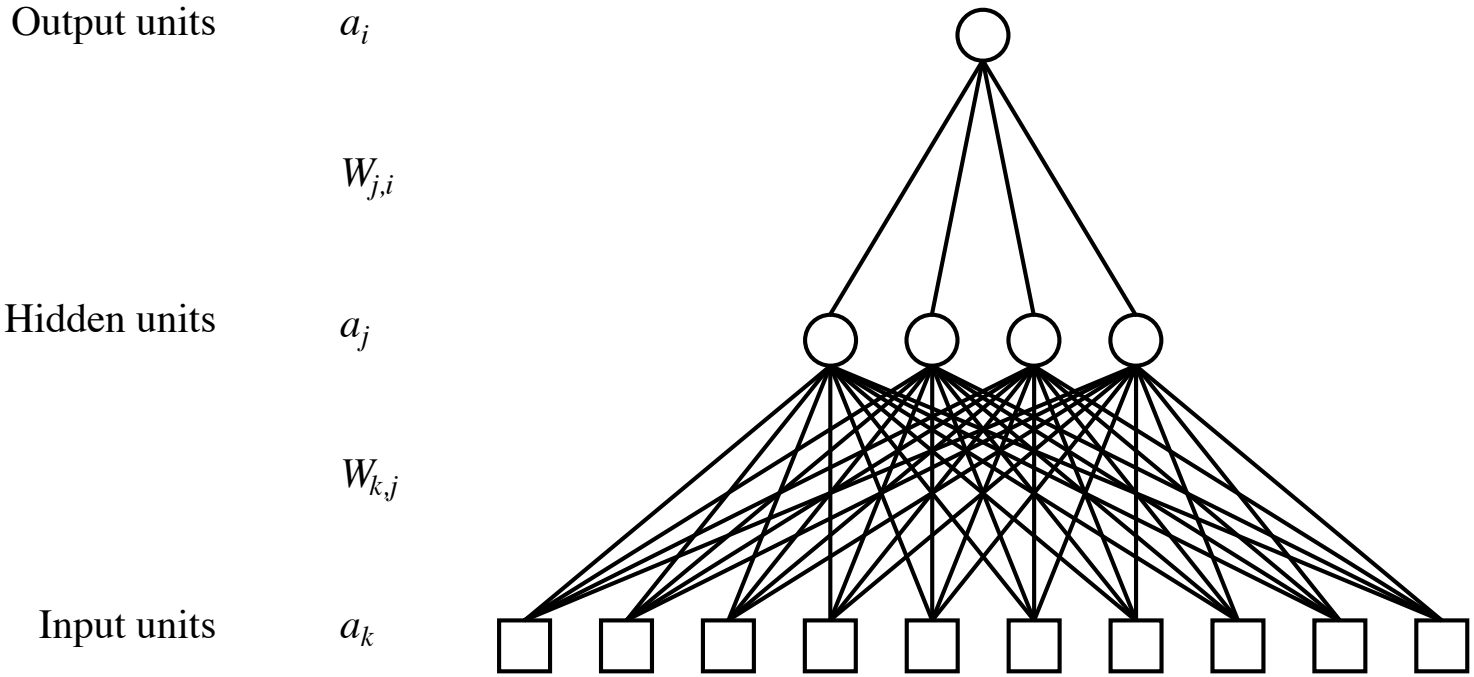
otherwise, weights are unchanged. ($\eta > 0$ is called the **learning rate**)

Theorem: This will eventually learn to classify the data correctly, as long as they are **linearly separable**.

Multilayer perceptrons

Layers are usually fully connected;

numbers of hidden units typically chosen by hand



Gradient Descent

Recall that the error function *E* is (half) the sum over all input patterns of the square of the difference between actual output and desired output

The aim is to find a set of weights for which *E* is very low.

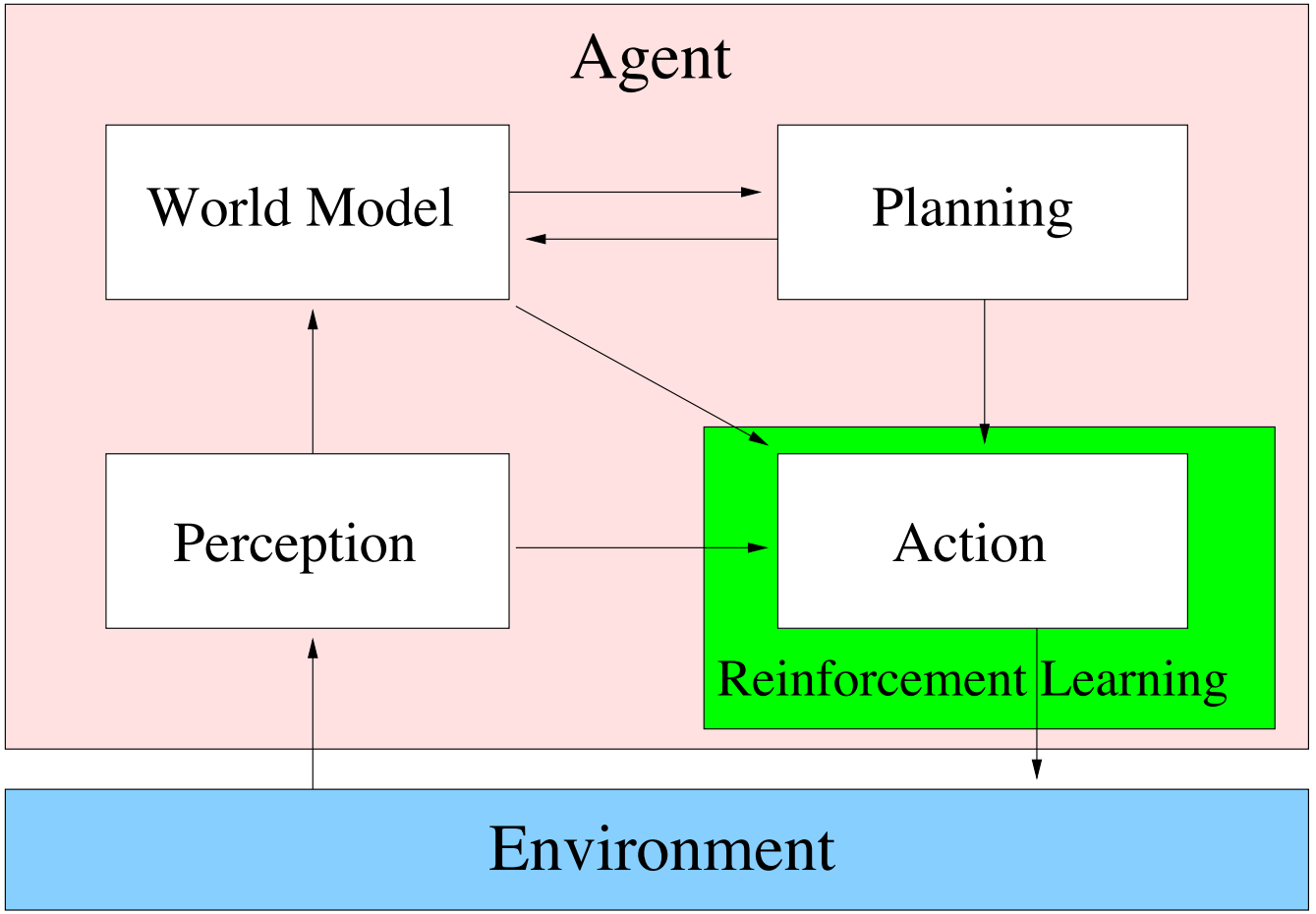
If the functions involved are smooth, we can use multi-variable calculus to adjust the weights in such a way as to take us in the steepest downhill direction.

$$E = \frac{1}{2}\sum(z-t)^2$$

Parameter η is called the learning rate.

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

Reinforcement Learning Agent



States and Actions

- Each node is a *state*
- *Actions* cause transitions from one state to another
- A *policy* is the set of transition rules
  - i.e. which action to apply in a given state
- Agent receives a *reward* after each action
- Actions may be non-deterministic
  - Same action may not always produce same state

Reinforcement Learning Framework

An agent interacts with its environment.

There is a set *S* of *states* and a set *A* of *actions*.

At each time step t, the agent is in some state s_t. It must choose an action a_t , whereupon it goes into state

$s_t + 1 = \delta(s_t, a_t)$ and receives reward $r(s_t, a_t)$.

In general, $r()$ and $\delta()$ can be multi-valued, with a random element

The aim is to find an optimal *policy* $\pi : S \rightarrow A$ which will maximize the cumulative reward.

Q-Learning – summary

For each $s \in S$, let $V*(s)$ be the maximum discounted reward obtainable from s, and let $Q(s, a)$ be the discounted reward available by first doing action a and then acting optimally.

Then the optimal policy is $\pi^*(s) = \text{argmax}_a Q(s,a)$

where $$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

then $$V^*(s) = \max_a Q(s,a),$$

so $$Q(s,a) = r(s,a) + \gamma \max_b Q(\delta(s,a),b)$$

which allows us to iteratively approximate Q by

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_b \hat{Q}(\delta(s,a),b)$$

Examination Instructions

(1) READING TIME – 10 MINUTES

(2) TIME ALLOWED – 2 HOURS

(3) TOTAL NUMBER OF PAGES –

(4) THIS EXAMINATION COUNTS FOR 60% OF THE FINAL MARK

(5) TOTAL NUMBER OF QUESTIONS - 36

(6) ANSWER **ALL** QUESTIONS

(8) CHOOSE **ONE** ANSWER PER QUESTION

(9) UNIVERSITY APPROVED CALCULATORS MAY BE USED

(10) THIS PAPER MAY **NOT** BE RETAINED BY THE CANDIDATE

Final Exam

- Similar to Tutorial Questions, but in Multiple Choice format
- Questions 1-20 - 1 mark each,
- Questions 21-36 – 2.5 marks each
- For each question, choose the ONE BEST Answer
- No marks taken off for wrong answers
- All modules covered in roughly equal proportion
- Quick questions are at the beginning; longer questions at the end

Sample 1-mark Question

Completeness of a search algorithm answers the question:

(a) Is the algorithm guaranteed to find a solution when there is one?

(b) Does the strategy find the solution that has the lowest path cost of all solutions?

(c) How long does it take to find a solution?

(d) How much memory is needed to perform the search?

Sample 2.5 -marks Question

Consider this joint probability distribution:

	short		¬ short	
	wide	¬ wide	wide	¬ wide
striped	0.07	0.05	0.08	0.12
¬ striped	0.14	0.17	0.12	0.25

Compute (to two decimal places): Prob(short ∨ ¬ wide | striped)

(a) 0.50

(b) 0.63

(c) 0.75

(d) 0.80

Beyond COMP3411/9814

- COMP9444 Neural Networks and Deep Learning
- COMP9417 Machine Learning and Data Mining
- COMP4418 Knowledge Representation and Reasoning
- COMP3431 Robotic Software Architecture
- COMP9517 Machine Vision
- 4th Year Thesis topics

UNSW myExperience Survey

Please remember to fill in the UNSW myExperience Survey.

COMP3411/9414 Artificial Intelligence

QUESTIONS?

COMP3411/9414 Artificial Intelligence

GOOD LUCK!