

Artificial Intelligence



- | | | | |
|---|------------------|---------------------|-----------|
| <input type="checkbox"/> Subject Code: | 18CS5AI01 | Total Hours: | 45 |
| <input type="checkbox"/> Credits: | 03 | Hours/week: | 03 |

Course objectives:

- To understand the basic concepts of Artificial Intelligence, terminologies related to AI in present business world.
 - To learn the application of artificial neural networks in Artificial Intelligence.
 - To gain the knowledge of Deep Learning in Artificial Intelligence.
 - To acquire the reasoning in Artificial Intelligence.
 - To use the different applications and techniques involved in AI for real time scenarios.

What is an AI?

- The field of **artificial intelligence**, or AI, attempts to understand intelligent entities.
- Thus, one reason to study it is to learn more about ourselves. But unlike philosophy and psychology, which are also concerned with intelligence, AI strives to *build intelligent entities* as well as *understand* them.
- Another reason to study AI is that these constructed intelligent entities are interesting and useful in their own right.
- AI has produced many significant and impressive products even at this early stage in its development.
- Although no one can predict the future in detail, it is clear that computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.

What is Artificial Intelligence Exactly?



Why?

- The field of **artificial intelligence**, or AI, attempts to understand intelligent entities.
- Thus, one reason to study it is to learn more about ourselves. But unlike philosophy and psychology, which are also concerned with intelligence, AI strives to build *intelligent entities* as well as understand them.
- Another reason to study AI is that these constructed intelligent entities are interesting and useful in their own right.
- AI has produced many significant and impressive products even at this early stage in its development. AI adds intelligence to products used by humans.
- Although no one can predict the future in detail, it is clear that computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.
- Automate the day-to-day routine work.
- Most AI examples that you hear about today – from chess-playing computers to self-driving cars – rely heavily on deep learning and natural language

Why AI ?



AI real life examples



- Face Recognition – open your iPhone automatically
- Social Media page – facebook, twitter, histogram – based on history.
- Security – fill out certain bits of info – filter fake news detection – identify cyber bullying effect on facebook- use machine learning algorithm.
- Natural language processing – spell checker, grammarly tools.
- Spam filters – send and received emails filters
- Google search – page ranking algorithm – search in fractions of seconds.
- Voice recognition – Siri (Apple), Alexa, Google Assistant - Return of answers.
- Smart homes – heat and cool up home, smart fridges, smart TV, fans etc
- Live traffic condition – drive assist functionality in car.
- Autonomous Vehicle – Google self driving car
- Product recommendation – Amazon (item recommend), netflix (movie recommend) – purchase / watch history. (past history)

Top 10 AI Applications beyond 2020



Outcome based Learning Objectives



By the end of this course, students will be able to

- Define and outline the basics, importance and application of Artificial Intelligence.
- Identify the suitable artificial agents in a problem solving environment.
- Apply the various searching techniques in AI domain.
- Outline the application of knowledge representation and reasoning in Artificial Intelligence.
- Apply the planning techniques to understand the importance of Artificial Intelligence in real time environment.
- Examine the technique of Artificial Neural Network in finding out the importance of predictable power in AI.

Course Contents

Sr #	Major and Detailed Coverage Area	Hrs
1	<p>Introduction to AI</p> <p>What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.</p>	9
2	<p>Problem-solving</p> <p>Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.</p>	9

Course Contents continue...



Sr #	Major and Detailed Coverage Area	Hrs
3	Knowledge and reasoning A Knowledge-Based Agent, Representation, Reasoning, and Logic, Prepositional Logic, An Agent for the Wumpus World, Problems with the propositional agent, First-Order Logic, Syntax and Semantics, Extensions and Notational Variations, Using First-Order Logic, A Simple Reflex Agent, Deducing Hidden Properties of the World, Toward a Goal-Based Agent, Building a Knowledge Base, Knowledge Engineering, Inference Rules Involving Quantifiers, Generalized Modus Ponens, Forward and Backward Chaining, Completeness, Resolution: A Complete Inference Procedure, Completeness of resolution.	9
4	Acting logically A Simple Planning Agent, From Problem Solving to Planning, Planning in Situation Calculus, Basic Representations for Planning, A Partial-Order Planning Algorithm, Planning with Partially Instantiated Operators, Knowledge Engineering for Planning, Practical Planners, Hierarchical Decomposition, Analysis of Hierarchical Decomposition, More Expressive Operator Descriptions, Resource Constraints, Planning and Acting, Conditional Planning, A Simple Re-planning Agent, Fully Integrated Planning and Execution	9

Course Contents continue...



Sr #	Major and Detailed Coverage Area	Hrs
5	Generalized Models A General Model of Learning Agents, Components of the performance element, Representation of the components, Inductive Learning, Learning Decision Trees, Using Information Theory, Learning General Logical Descriptions, Computational Learning Theory, Learning in Neural and Belief Networks, Neural Networks, Perceptrons, Multilayer Feed-Forward Networks, Bayesian Methods for Learning Belief Networks, Reinforcement Learning, Passive Learning in a Known Environment, Passive Learning in an Unknown Environment, Generalization in Reinforcement Learning.	9
	Assessments - Two Open Ended Tests; Every Week One Assignment / One Activity	
	End Semester	

Recommended Books

Textbook

- ***Artificial Intelligence, A Modern Approach***, 3rd Edition by Stuart J. Russell and Peter Norvig, Prentice-Hall, Inc., 1995, ISBN- 0-13-103805-2.

Reference Books

- Artificial Intelligence: Foundations Of Computational Agents, 3rd Edition by D. Poole, Cambridge University Press, 2010, ISBN.
- Artificial Intelligence and Intelligent Systems, 4th Edition by Padhy N.P, Oxford University Press, 2007, ISBN

Evaluation and Prerequisites

Evaluation/Grades will be based on:

- i) Assignments/class test/quiz = 5 marks
- ii) Attendance = 5 marks
- Internal Assessment - 30 marks (including i & ii)
- Semester End Exam - 70 marks
- Pass mark – 40

Prerequisites

Basic knowledge on Mathematics

Scheme of Evaluation (100 marks)



- **Continuous Internal Evaluation [CIE] (maximum 30 marks)**
 - Three internal tests each for a maximum of 50 marks
 - All three tests are compulsory.
 - Two best performances shall be considered.
 - The final average marks shall be scaled, for a maximum of 20 marks.
 - Quiz 5 marks and Attendance 5 marks.

- **Semester End Examination [SEE] (maximum 70 marks)**
 - Question paper consists of 8 questions (Offline)/ 70 MCQ (online)/university policy will be defined time to time based on the current situation of COVID-19.
 - Each question shall carry 14 marks with two to three subdivisions.(offline)
 - First two questions are compulsory. The students are required to answer any one questions selecting from 3 or 4, 5 or 6, 7 or 8. (offline)

The top 5 in-demand job skills for the future beyond 2020





Thank You

UNIT - 1

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

- What is an AI?
- Perspectives / Approaches to define an AI.
- Goals of AI

What is AI?

- The field of **artificial intelligence, or AI, attempts to understand intelligent entities.**
- **Thus, one reason to study it is to learn more about ourselves.** But unlike philosophy and psychology, which are also concerned with intelligence, AI strives to *build intelligent entities as well as understand them.*
- Another reason to study AI is that these constructed intelligent entities are interesting and useful in their own right.
- AI has produced many significant and impressive products even at this early stage in its development.
- Although no one can predict the future in detail, it is clear that computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.
- <https://www.youtube.com/watch?v=kWmX3pd1f10>

Defining AI

- Definitions of artificial intelligence according to eight recent textbooks are shown in Figure 1.1. These definitions vary along two main dimensions.
- The ones on **top** are concerned with *thought processes and reasoning*, whereas the ones on the **bottom** address *behavior*.
- Also, the definitions on the **left** measure success in terms of *human performance*, whereas the ones on the **right** measure against an *ideal concept of intelligence, which we will call rationality*.
- A system is rational if it does the right thing. This gives us four possible goals to pursue in artificial intelligence, as seen in the caption of Figure 1.1.
- Historically, all **four approaches** have been followed. As one might expect, a tension exists between approaches centered around humans and approaches centered around rationality.
- <https://www.youtube.com/watch?v=XuE0GqYHPqQ>

Defining AI

<p>"The exciting new effort to make computers think . . . <i>machines with minds</i>, in the full and literal sense" (Haugeland, 1985)</p> <p>"[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)</p>	<p>"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985)</p> <p>"The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)</p>
<p>"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)</p> <p>"The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)</p>	<p>"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990)</p> <p>"The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)</p>
Figure 1.1 Some definitions of AI. They are organized into four categories:	
<p>Systems that think like humans.</p> <p>Systems that act like humans.</p>	<p>Systems that think rationally.</p> <p>Systems that act rationally.</p>

Acting humanly: The Turing Test approach

- The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.
- Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator.
- Roughly speaking, the test he proposed is that the computer should be interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end.
- Programming a computer to pass the test provides plenty to work on. The computer would need to possess the following capabilities:
- **natural language processing** to enable it to communicate successfully in English (or some other human language);

Acting humanly: The Turing Test approach

- **knowledge representation** to store information provided before or during the interrogation;
- **automated reasoning** to use the stored information to answer questions and to draw new conclusions;
- **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.
- **computer vision** to perceive objects, and
- **robotics** to move them about.

Thinking humanly: The cognitive modelling approach

- If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get *inside the actual workings of human minds*.
- There are two ways to do this: through **introspection**—trying to catch our own thoughts as they go by—or through **psychological experiments**.
- Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program.
- If the program's input/output and timing behavior matches human behavior, that is evidence that some of the program's mechanisms may also be operating in humans.
- For example, Newell and Simon, who developed GPS, the "General Problem Solver" (Newell and Simon, 1961), were not content to
- have their program correctly solve problems.

Thinking humanly: The cognitive modelling approach

- They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems.
- This is in contrast to other researchers of the same time (such as Wang (1960)), who were concerned with getting the right answers regardless of how humans might do it.
- The interdisciplinary field of **cognitive science brings together computer models from AI and experimental techniques from psychology** to try to construct precise and testable theories of the workings of the human mind.
- <https://www.youtube.com/watch?v=XVP9sMyxprQ>

Thinking rationally: The laws of thought approach

- The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes.
- His famous **syllogisms provided patterns for argument structures** that always gave correct conclusions given correct premises.
- For example, "Socrates is a man; all men are mortal; therefore Socrates is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of **logic**.
- By 1965, programs existed that could, given enough time and memory, take a description of a problem in logical notation and find the solution to the problem, if one exists. (If there is no solution, the program might never stop looking for it.)
- The so-called **logicist tradition within artificial** intelligence hopes to build on such programs to create intelligent systems.

Acting rationally: The rational agent approach

- Acting rationally means acting so as to achieve one's goals, given one's beliefs. An **agent** is just something that perceives and acts. (This may be an unusual use of the word, but you will get used to it.) In this approach, AI is viewed as the study and construction of rational agents.
- In the "**laws of thought**" approach to AI, the whole emphasis was on **correct inferences**.
- Making correct inferences is sometimes *part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals, and then to act on that conclusion.*

Acting rationally: The rational agent approach

- On the other hand, correct inference is **not all of rationality**, because there are often situations where there is no provably correct thing to do, yet something must still be done.
- There are also ways of acting rationally that cannot be reasonably said to involve inference.
- For example, pulling one's hand off of a hot stove is a reflex action that is more successful than a slower action taken after careful deliberation.
- All the "cognitive skills" needed for the Turing Test are there to allow rational actions. Thus, we need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations.

Advantages of Rational Agent

- The study of AI as rational agent design therefore has two advantages.
- First, it is more general than the "laws of thought" approach, because correct inference is only a useful mechanism for achieving rationality, and not a necessary one.
- Second, it is more amenable to scientific development than approaches based on human behavior or human thought, because the standard of rationality is clearly defined and completely general.
- Human behavior, on the other hand, is well-adapted for one specific environment and is the product, in part, of a complicated and largely unknown evolutionary process that still may be far from achieving perfection.

Goals of AI

- “Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on.” (Barr & Feigenbaum, 1981)
- **Scientific Goal** To determine which ideas about knowledge representation, learning, rule systems, search, and so on, explain various sorts of real intelligence.
- **Engineering Goal** To solve real world problems using AI techniques such as knowledge representation, learning, rule systems, search, and so on.

Goals of AI

- Traditionally, computer scientists and engineers have been more interested in the engineering goal, while psychologists, philosophers and cognitive scientists have been more interested in the scientific goal.
- It makes good sense to be interested in both, as there are common techniques and the two approaches can feed off each other.
- In this module we shall **attempt to keep both goals in mind.**

Video Reference links

S. NO	Description	Video link
1	This video describes the what AI exactly is?	https://www.youtube.com/watch?v=kWmX3pd1f10
2	This video explains the four approaches to define AI	https://www.youtube.com/watch?v=XuE0GqYHPqQ
3	This video gives the insight knowledge of Thinking humanly: The cognitive modeling approach	https://www.youtube.com/watch?v=XVP9sMyxprQ

Thank you

UNIT - 1

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Foundations of Artificial Intelligence in different disciplines

- Philosophy
- Mathematics
- Economics
- Neuroscience
- Psychology/ Cognitive Science
- Computer engineering
- Control theory
- Linguistics

THE FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

- Artificial Intelligence has identifiable roots in a number of older disciplines, particularly: Philosophy Logic/Mathematics Computation Psychology/Cognitive Science Biology/Neuroscience Evolution.
- There is inevitably much overlap, e.g. between philosophy and logic, or between mathematics and computation.
- By looking at each of these in turn, we can gain a better understanding of their role in AI, and how these underlying disciplines have developed to play that role

Philosophy

- ~400 BC Socrates asks for an algorithm to distinguish piety from non-piety.
- ~350 BC Aristotle formulated different styles of **deductive reasoning**, which could mechanically generate conclusions from initial premises,
- e.g. Modus Ponens If $A \Rightarrow B$ and A then B If A implies B and A is true then B is true
- when it's raining you get wet and it's raining then you get wet
- 1596 – 1650 Rene Descartes idea of mind-body **dualism** – part of the mind is exempt from physical laws. Otherwise how do we have free will?
- 1646 – 1716 Wilhelm Leibnitz was one of the first to take the **materialist** position which holds that the mind operates by ordinary physical processes – this has the implication that mental processes can potentially be carried out by machines

Logic/Mathematics

- 1777 Earl Stanhope's **Logic Demonstrator** was a machine that was able to solve syllogisms, numerical problems in a logical form, and elementary questions of probability.
- 1815 – 1864 George Boole introduced his **formal language** for making logical inference in 1847 – Boolean algebra.
- 1848 – 1925 Gottlob Frege produced a logic that is essentially the **first-order logic** that today forms the most basic knowledge representation system.
- 1906 – 1978 Kurt Gödel showed in 1931 that there are **limits to what logic can do**. His **Incompleteness Theorem** showed that in any formal logic powerful enough to describe the properties of natural numbers, there are true statements whose truth cannot be established by any algorithm.
- 1995 Roger Penrose tries to prove the **human mind** has **non-computable capabilities**.

Computation

- 1869 William Jevon's **Logic Machine** could **handle Boolean Algebra** and **Venn Diagrams**, and was able to solve logical problems faster than human beings.
- 1912 – 1954 Alan Turing tried to characterise exactly which **functions are capable of being computed**. Unfortunately it is difficult to give the notion of computation a formal definition. However, the Church-Turing thesis, which states that a **Turing machine** is capable of computing any computable function, is generally accepted as providing a sufficient definition. Turing also showed that there were some functions which no Turing machine can compute (e.g. Halting Problem).
- 1903 – 1957 John von Neumann proposed the **von Neuman architecture** which allows a description of computation that is independent of the particular realisation of the computer.
- ~1960s Two important concepts emerged: **Intractability** (when solution time grows at least exponentially) and **Reduction** (to 'easier' problems).

Psychology / Cognitive Science

- Modern Psychology / Cognitive Psychology / Cognitive Science is the science which studies how the **mind operates**, how we **behave**, and how our **brains process** information.
- Language is an important part of human intelligence. Much of the early work on **knowledge representation** was tied to language and informed by research into linguistics.
- It is natural for us to try to use our understanding of how human (and other animal) brains lead to intelligent behaviour in our quest to build artificial intelligent systems. Conversely, it makes sense to **explore the properties of artificial systems** (computer models/simulations) to test our hypotheses concerning human systems.
- Many sub-fields of AI are simultaneously building models of how the human system operates, and artificial systems for solving real world problems, and are allowing useful ideas to transfer between them.

Biology / Neuroscience

- Our brains (which give rise to our intelligence) are made up of tens of billions of neurons, each connected to hundreds or thousands of other neurons. Each neuron is a simple processing device (e.g. just firing or not firing depending on the total amount of activity feeding into it). However, **large networks of neurons** are extremely powerful computational devices that can **learn** how best to operate.
- The field of **Connectionism or Neural Networks** attempts to build artificial systems based on simplified networks of simplified artificial neurons. The aim is to build powerful AI systems, as well as models of various human abilities.
- **Neural networks** work at a **sub-symbolic level**, whereas much of **conscious** human reasoning appears to operate at a **symbolic level**.
- Artificial neural networks perform well at many simple tasks, and provide good models of many human abilities. However, there are many tasks that they are not so good at, and other approaches seem more promising in those areas.

Foundations of AI

- Philosophy Logic, methods of reasoning, mind as physical system, foundations of learning, language, rationality.
- Mathematics Formal representation and proof, algorithms, computation, (un)decidability, (in)tractability, probability.
- Economics utility, decision theory, rational economic agents
- Neuroscience neurons as information processing units.
- Psychology/ Cognitive Science how do people behave, perceive, process information, represent knowledge.
- Computer engineering building fast computers
- Control theory design systems that maximize an objective function over time
- Linguistics knowledge representation, grammar

Video Reference links

S. NO	Description	Video link
1	This video describes the Foundations of AI	https://www.youtube.com/watch?v=yTvwSPKvjSE

Thank you

UNIT - 1

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Students will be able to

- Get the basics of History /Gestation of AI
- Gain the Power of Knowledge Base systems in AI

The gestation of artificial intelligence (1943-1956)

- The first work that is generally recognized as AI was done by **Warren McCulloch and Walter Pitts** (1943).
- They drew on **three sources**: **knowledge** of the basic physiology and function of neurons in the brain; the of **propositional logic** due to Russell and Whitehead; and Turing's **theory of computation**.
- They proposed a model of artificial neurons in which each neuron is characterized as being "on" or "off," with a switch to "on" occurring in response to stimulation by a sufficient number of neighboring neurons.
- The state of a neuron was conceived of as "factually equivalent to a proposition which proposed its adequate stimulus."
- They showed, for example, that any computable function could be computed by some network of connected neurons, and that all the logical connectives could be implemented by simple net structures.

The gestation of artificial intelligence (1943-1956)

- **Donald Hebb** (1949) demonstrated a simple **updating rule** for modifying the connection strengths between neurons, such that **learning could take place**.
- In the early 1950s, **Claude Shannon** (1950) and **Alan Turing** (1953) were writing chess programs for von Neumann-style conventional computers.
- At the same time, two graduate students in the Princeton mathematics department, **Marvin Minsky** and **Dean Edmonds**, built the **first neural network computer (SNARC) in 1951**.
- The **SNARC**, as it was called, used 3000 vacuum tubes and a surplus automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons.
- Ironically, Minsky was later to **prove theorems** that contributed to the demise of **much of neural network research** during the 1970s.

(1943-1956)

- Princeton was home to another influential figure in AI, John McCarthy. After graduation, McCarthy moved to Dartmouth College, which was to become the official birthplace of the field.
- McCarthy convinced Minsky, Claude Shannon, and Nathaniel Rochester to help him bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence.
- They organized a two-month workshop at Dartmouth in the summer of 1956. All together there were ten attendees, including Trenchard More from Princeton, Arthur Samuel from IBM, and Ray Solomonoff and Oliver Selfridge from MIT. Two researchers from Carnegie Tech, Alien Newell and Herbert Simon.
- Newell and Simon already had a reasoning program, the Logic Theorist (LT), about which Simon claimed, "We have invented a computer program capable of thinking non-numerically, and thereby solved the venerable mind-body problem.

The gestation of artificial intelligence (1943-1956)

- Soon after the workshop, the program was able to prove most of the theorems of Russell and Whitehead's *Principia Mathematica*.
- *Russell was reportedly delighted when Simon showed him that the program had come up with a proof for one theorem that was shorter than the one in Principia.*
- *The editors of the Journal of Symbolic Logic were less impressed; they rejected a paper coauthored by Newell, Simon, and Logic Theorist.*
- Perhaps the most lasting thing to come out of the workshop was an agreement to adopt McCarthy's new name for the field: **artificial intelligence**.

Knowledge-based systems: The key to power AI? (1969-1979)

- During the first decade of AI research was of a general-purpose search mechanism trying to string together elementary reasoning steps to find complete solutions. Such approaches have been called **weak methods, because they use weak information about the domain.**
- The **DENDRAL program** (Buchanan *et al.*, 1969) *was an early example of this approach.* It was developed at Stanford, where Ed Feigenbaum (a former student of Herbert Simon), Bruce Buchanan (a philosopher turned computer scientist), and Joshua Lederberg (a Nobel laureate geneticist) teamed up to **solve the problem of inferring molecular structure from the information provided by a mass spectrometer.**

Knowledge-based systems:

- The input to the program consists of the elementary formula of the molecule (e.g., C₆H₁₃NO₂), and the mass spectrum giving the masses of the various fragments of the molecule generated when it is bombarded by an electron beam.
- For example, the mass spectrum might contain a peak at *in- 15 corresponding to the mass of a methyl (CH₃) fragment.*
- The naive version of the program **generated all possible structures consistent with the formula**, and then **predicted** what mass spectrum would be observed for each, comparing this with the actual spectrum.
- The significance of **DENDRAL** was that it was arguably the **first successful knowledge-intensive system**: its expertise derived from large numbers of special-purpose rules.
- With this lesson in mind, Feigenbaum and others at Stanford began the **Heuristic Programming Project (HPP)**, to investigate the extent to which the new methodology of **expert systems** could be applied to other areas of human expertise.

Knowledge-based systems:

- The next major effort was in the area of **medical diagnosis**.
- Feigenbaum, Buchanan, and Dr. Edward Shortliffe developed **MYCIN** to **diagnose blood infections**.
- With about 450 rules, MYCIN was able to perform as well as some experts, and considerably better than junior doctors.
- It also contained two major differences from DENDRAL. **First**, unlike the DENDRAL rules, **no general** theoretical model existed from which the MYCIN rules could be deduced. They had to be acquired from extensive interviewing of experts, who in turn acquired them from direct experience of cases.
- **Second**, the rules had **to reflect the uncertainty** associated with medical knowledge.
- MYCIN incorporated a calculus of uncertainty called **certainty factors, which seemed (at the time) to fit well with how doctors assessed the impact of evidence on the diagnosis**.

Knowledge-based systems:

- At Rutgers University, Saul Amarel's *Computers in Biomedicine project* began an ambitious attempt to diagnose diseases based on explicit knowledge of the causal mechanisms of the disease process.
- **Probabilistic reasoning system, PROSPECTOR** (Duda *et al.*, 1979), generated enormous publicity by recommending exploratory drilling at a geological site that proved to contain a large molybdenum deposit.
- **Molybdenum** is an essential mineral found in high concentrations in legumes, grains and organ meats. It activates enzymes that help break down harmful sulfites and prevent toxins from building up in the **body**.



Knowledge-based systems:

- Winograd's **SHRDLU system** for understanding natural language had engendered a good deal of excitement, its dependence on syntactic analysis caused some of the same problems as occurred in the early machine translation work.
- It was able to overcome ambiguity and understand pronoun references, but this was mainly because it was designed specifically for one area—the blocks world.
- William Woods (1973) built the **LUNAR system**, which allowed geologists to ask questions in English about the rock samples brought back by the Apollo moon mission.
- LUNAR was the first natural language program that was used by people other than the system's author to get real work done.
- Since then, many natural language programs have been used as interfaces to databases.

Video Reference links

S. NO	Description	Video link
1	This video describes the McCulloch and Pitts Model based on biological neurons.	https://www.youtube.com/watch?v=ZB0Q8oW1gQ
2	This video describes the origin of neural network in 1950s. First NN computer by Marvin Minisky.	https://www.youtube.com/watch?v=Suevq-kZdlw
3	The Evolution of AI from 1956 to Present Day	https://www.youtube.com/watch?v=G2KD6KmHuZ0
4	The video gives the insight knowledge of AI from 1950s to 2011.	https://www.youtube.com/watch?v=UqJJkMFfVbk
5	This video describes briefly History of Artificial Intelligence.	https://www.youtube.com/watch?v=SP-w-lBgUql https://www.youtube.com/watch?v=7wSU-nOgTIw

Thank you

UNIT – 1

L5

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Students will be able to

- Review the Power of AI
- Grasp how AI becomes an industry

AI becomes an industry (1980-1988)

- The first successful commercial expert system, **RI**, began operation at Digital Equipment Corporation (McDermott, 1982).
- The program helped configure orders for new computer systems, and by 1986, it was saving the company an estimated \$40 million a year.
- By 1988, DEC's AI group had 40 deployed expert systems, with more on the way.
- Du Pont had 100 in use and 500 in development, saving an estimated \$10 million a year.
- Nearly every major U.S. corporation had its own AI group and was either using or investigating expert system technology.

AI becomes an industry (1980-1988)

- In 1981, the **Japanese** announced the "**Fifth Generation**" project, a 10-year plan to build intelligent computers running Prolog in much the same way that ordinary computers run machine code.
- The idea was that with the ability to make millions of inferences per second, computers would be able to take advantage of vast stores of rules.
- The project proposed to achieve full-scale natural language understanding, among other ambitious goals.
- The Fifth Generation project fueled interest in AI, and by taking advantage of fears of Japanese domination, researchers and corporations were able to generate support for a similar investment in the United States.
- The **Microelectronics and Computer Technology Corporation (MCC)** was formed as a **research consortium** to counter the Japanese project

AI becomes an industry (1980-1988)

- In Britain, the Alvey report reinstated the funding that was cut by the Lighthill report.
- In both cases, AI was part of a broad effort, including chip design and human-interface research.
- The booming AI industry also included companies such as Carnegie Group, Inference, Intellicorp, and Teknowledge that offered the **software tools** to build expert systems, and **hardware companies** such as Lisp Machines Inc., Texas Instruments, Symbolics, and Xerox that; were **building workstations optimized** for the development of Lisp programs.
- Over a hundred companies built industrial **robotic vision systems**.
- Overall, the industry went from a few **million in sales** in 1980 to **\$2 billion** in 1988.

The State of the Art – Applications of AI

- Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997.
- Proved a mathematical conjecture (Robbins conjecture) unsolved for decades.
- No hands across America (**driving autonomously** 98% of the time from Pittsburgh to San Diego).
- During the 1991 Gulf War, US forces deployed an **AI logistics planning and scheduling program** that involved up to 50,000 vehicles, cargo, and people.
- NASA's on-board **autonomous** planning program controlled the scheduling of operations for a spacecraft.
- **Proverb** solves crossword puzzles better than most humans
- Stanford vehicle in **Darpa** challenge completed autonomously a 132 mile desert track in 6 hours 32 minutes.
- These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics

Example videos industrial application of AI

- https://www.youtube.com/watch?v=dTjuC_nLq8U AI in food industry.
- <https://www.youtube.com/watch?v=p9qmQxamjtg> Artificial Intelligence (AI) Impact on Industries (Health)

Some AI videos

- <http://www.youtube.com/watch?v=1JJsbFiXGl0&feature=related>
- <http://www.youtube.com/watch?v=lCgL1OWsn58&feature=related>
- <http://www.cs.utexas.edu/~kdresner/aim/video/fcfs-insanity.mov>
- http://www.youtube.com/watch?v=HacG_FWWPOw&feature=related
- http://videolectures.net/aaai07_littman_ai/
- http://www.ai.sri.com/~nysmith/videos/SRI_AR-PA_AAAI08.avi
- <http://www.youtube.com/watch?v=ScXX2bndGJc>

Thank you

UNIT – 1

L6

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Students will be able to

- Gain the knowledge of structure of Intelligent Agents
- Judge how Agents Should Act to improve their performance

INTELLIGENT AGENTS

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.
- A **human agent** has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors.
- A **robotic agent** substitutes cameras and infrared range finders for the sensors and various motors for the effectors.
- A **software agent** has encoded bit strings as its percepts and actions.
- A generic agent is diagrammed in Figure 2.1.

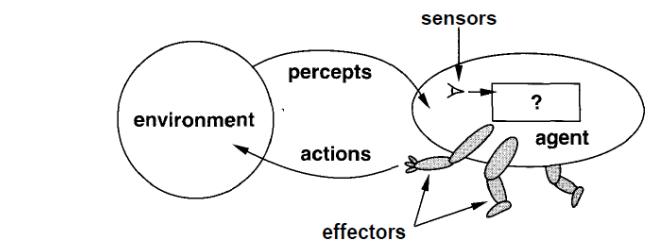


Figure 2.1 Agents interact with environments through sensors and effectors.

Agents and environments

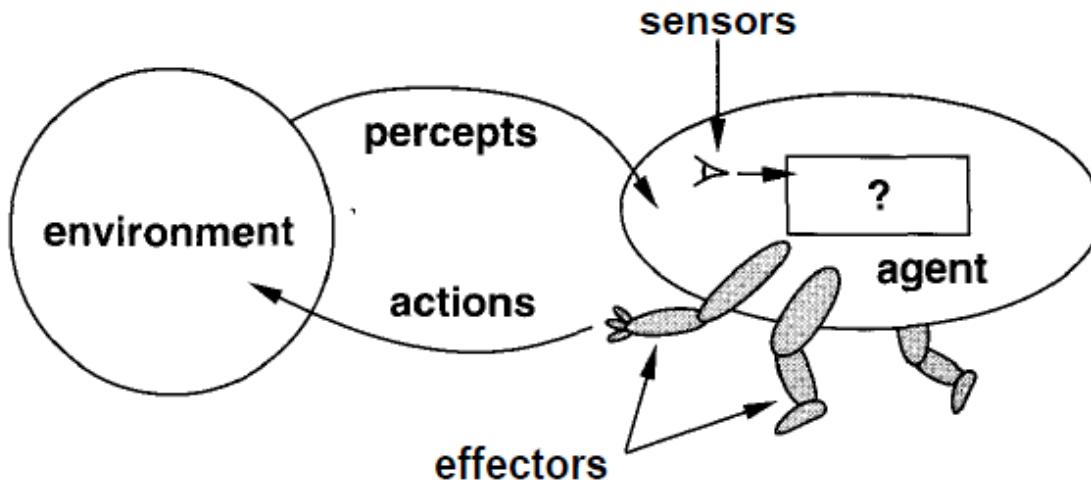


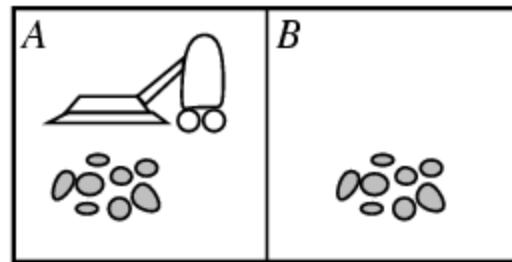
Figure 2.1 Agents interact with environments through sensors and effectors.

- The **agent function** maps from percept histories to actions:
 $[f: \mathcal{P}^* \rightarrow \mathcal{A}]$
- The **agent program** runs on the physical **architecture** to produce f
- **agent** = architecture + program

RATIONAL AGENTS

- A rational agent is one that does the right thing. Obviously, this is better than doing the wrong thing, but what does it mean?
- As a first approximation, we will say that the right action is the one that will cause the agent to be most successful. That leaves us with the problem of deciding *how and when to evaluate the agent's success.*
- *Doing right thing – will reach to the goal.*
- *Doing wrong thing – deviates from goal.*
- Are doing right thing? how to measure it?
- As an example, consider the case of an agent that is supposed to vacuum a dirty floor.

Vacuum-cleaner world



- **Percepts:** location and state of the environment, e.g., [A,Dirty], [B,Clean]
- **Actions:** *Left, Right, Suck, NoOp*

Performance Measure (How)

- The criteria that determine how successful an agent is.
- Obviously, there is not one fixed measure suitable for all agents.
- We could ask the agent for a subjective opinion of how happy it is with its own performance, but some agents would be unable to answer, and others would delude themselves.
- So, **outside observers** establish a standard of what it means to be successful in an environment and use it to measure the performance of agents.
- A performance measure would be the amount of dirt cleaned up in a single eight-hour shift.
- A more sophisticated performance measure would factor in the amount of electricity consumed and the amount of noise generated as well.
- A third performance measure might give highest marks to an agent that not only cleans the floor quietly and efficiently, but also finds time to go windsurfing at the weekend.

Performance Measure (when)

- *when of evaluating performance is also important.*
- *If we measured how much dirt the agent had cleaned up in the first hour of the day, we would be rewarding those agents that start fast (even if they do little or no work later on), and punishing those that work consistently.*
- Thus, we want to measure performance over the long run, be it an eight-hour shift or a lifetime.
- An **omniscient agent** knows the *actual outcome of its actions, and can act accordingly; but omniscience is ; impossible in reality.* Consider the following example: I am walking along the Champs Elysees one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street.
- Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner, and before I make it to the other side of the street I am flattened. Was I irrational to cross the street?

Rational agent

- **Rationality** is concerned with *expected success given what has been perceived*.
- *Crossing the street was rational because most of the time the crossing would be successful*, and there was no way I could have foreseen the falling door.
- Note that **another agent** that was equipped with radar for detecting falling doors or a steel cage strong enough to repel them would be more successful, but it would **not be any more rational**.
- we **cannot blame an agent** for failing to take into account something it could not perceive, or for failing to take an action (such as repelling the cargo door) that it is incapable of taking.
- An **intelligent agent** should always do what is ***actually the right thing***, it will be impossible to design an agent to fulfill this specification—unless we improve the performance of agents.

Rational agent

Rational at any given time depends on four things:

- The **performance measure** that defines degree of success.
- Everything that the agent has perceived so far. The complete perceptual history is called as the **percept sequence**.
- What the agent **knows** about the environment.
- The **actions** that the agent can perform.
- An **ideal rational agent**: *For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*
- *Example consider an agent A and B crossing the road street.*
- *Agent A cross road without looking both sides of road. it is rational.*
- *Agent B looks both sides then crosses road. It is ideal rational agent.*
- *Rational agent (No action), ideal rational agent (looking action).*
- **Doing actions in order to obtain useful information is an important part of rationality.**

Examples

- Ask students to read and identify the example is rational and ideal rational agent.
- Consider a clock. It can be thought of as just an **inanimate object**, or it can be thought of as a **simple agent**.
- As an agent, most clocks always do the right action: moving their hands (or displaying digits) in the proper fashion.
- Clocks are a kind of degenerate agent in that their percept sequence is empty; no matter what happens outside, the clock's action should be unaffected.
- Well, this is not quite true. If the clock and its owner take a trip from California to Australia, the right thing for the clock to do would be to turn itself back six hours.
- We do not get upset at our clocks for failing to do this because we realize that they are acting rationally, given their lack of perceptual equipment.

The ideal mapping from percept sequences to actions

- An agent's behavior depends only on its percept sequence to date, then we can describe any particular agent by making a table of the action it takes in response to each possible percept sequence. Such a list is called a **mapping from percept sequences to actions**.
- And if mappings describe agents, then **ideal mappings describe ideal agents**.
- *Specifying which action an agent ought to take in response to any given percept sequence provides a design for an ideal agent.*
- This **does not mean**, of course, that we have to create an explicit table with an entry for every possible percept sequence. It is possible to define a specification of the mapping **without exhaustively enumerating** it.
- Consider a very simple agent: To find the square root of the given number.

The square-root function on a calculator

- The **percept sequence** for this agent is a sequence of keystrokes representing a number, and the **action** is to display a number on the display screen.
- The ideal mapping is that when the percept is a positive number x , *the right action is to display a positive number z such that $z^2 \approx x$, accurate to, say, 15 decimal places.*
- *This specification of the ideal mapping does not* require the designer to actually **construct a table of square roots**. Nor does the square-root function have to use a table to behave correctly:
- Figure 2.2 shows part of the ideal mapping and a simple program that implements the mapping using Newton's method.
- The agent is a nice, **compact program**. It turns out that it is possible to design nice, compact agents that implement the ideal mapping for much more general situations.

The square-root function on a calculator

Percept x	Action z
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.140175425099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1.378404875209022
:	:

```
function SQRT(x)
    z ← 1.0          /* initial guess */
    repeat until |z2 - x| < 10-15
        z ← z - (z2 - x)/(2z)
    end
    return z
```

Figure 2.2 Part of the ideal mapping for the square-root problem (accurate to 15 digits), and a corresponding program that implements the ideal mapping.

Ideal rational agent - Autonomy

- If the agent's actions are based completely on built-in knowledge, such that its AUTONOMY need pay **no attention** to its percepts, then we say that the agent lacks **autonomy**.
- **For example**, if the clock manufacturer was prescient enough to know that the clock's owner would be going to Australia at some particular date, then a mechanism could be built in to adjust the hands automatically by six hours at just the right time. This would certainly be successful behavior, but the **intelligence seems to belong to the clock's designer** rather than to the clock itself.
- An agent's behavior can be based on both its own experience and the built-in knowledge used *in constructing the agent for the particular environment in which it operates*.
- *A system is autonomous to the extent that its behavior is determined by its own experience.*

Autonomy

- when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance.
- So, just as **evolution provides animals** with enough built-in reflexes so that they can survive long enough to learn for themselves.
- it would be reasonable to provide an **artificial intelligent agent** with some **initial knowledge as well as an ability to learn**.

An Artificial Intelligent Agent Program

=

- Perception (initial knowledge of facts) + Autonomy (ability to adapt to environment by learning)

Video Reference links

S. NO	Description	Video link
1	Structure of Artificial intelligent Agents	https://www.youtube.com/watch?v=XO6SV0Mup1E
2	This video describes the structure of intelligent agent and mention the types of agents in the environment.	https://www.youtube.com/watch?v=1D6yubBQjuo
3	This video link provide the Introduction to Intelligent Agents and their types with Example in Artificial Intelligence.	https://www.youtube.com/watch?v=BkedAnQfJ_U

Thank you

UNIT – 1

L7

Intelligent Agents

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Students will be able to

- Understand the structure of Intelligent Agents
- Gain the knowledge of PEAS components of an agent.
- Design an intelligent system using PEAS /PAGE Components of an Agent

STRUCTURE OF INTELLIGENT AGENTS

- ***behavior***—the action that is performed after any given sequence of percepts.
- Now, we will talk about how AGENTPROGRAM the insides work. The job of AI is to design the **agent program**: a function that **implements** the agent mapping from percepts to actions.
- Agent program will run on some sort of ARCHITECTURE computing device, which we will call the **architecture**.
- The architecture might be a **plain computer**, or it might include **special-purpose hardware** for certain tasks, such as processing camera images or filtering audio input. It might also include **software** that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level.

Agent Architecture

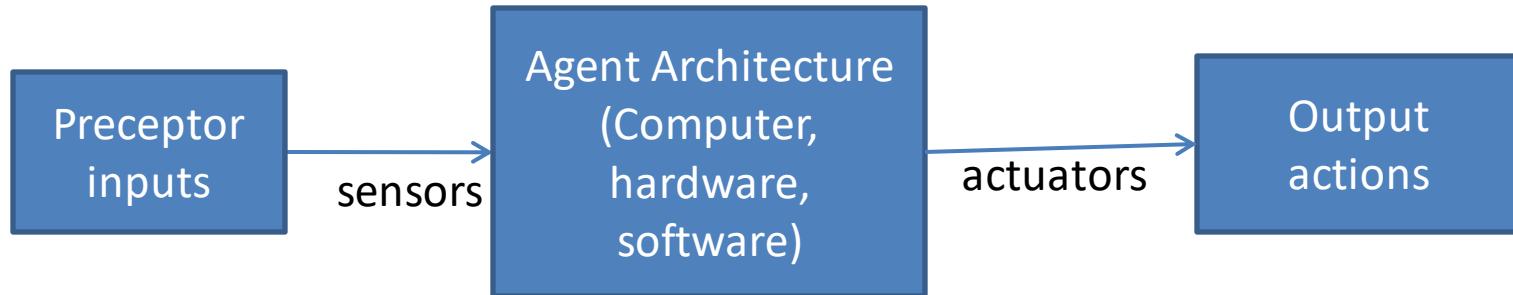


Fig A : Agent Program

- In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated.
- The relationship among agents, architectures, and programs can be summed up as follows:
- $agent = architecture + program$

Task Environment

- Before we design an intelligent agent, we must specify its “task environment”:

PEAS:

Performance measure/Goals
Environment
Actuators/Actions
Sensors/Percepts

PAGE:

Percepts / Sensors
Actions / actuators
Goals /Performance
Environment

Examples of Agents - PAGE

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Figure 2.3 Examples of agent types and their PAGE descriptions.

Agent Program

- Accepting percepts from an environment and generating actions.
- Each will use some internal data structures that will be updated as new percepts arrive.
- These data structures are operated on by the agent's decision-making procedures to generate an action choice, which is then passed to the architecture to be executed.

```
function SKELETON-AGENT(percept) returns action
  static: memory, the agent's memory of the world
    memory ← UPDATE-MEMORY(memory, percept)
    action ← CHOOSE-BEST-ACTION(memory)
    memory ← UPDATE-MEMORY(memory, action)
  return action
```

Figure 2.4 A skeleton agent. On each invocation, the agent's memory is updated to reflect the new percept, the best action is chosen, and the fact that the action was taken is also stored in memory. The memory persists from one invocation to the next.

Agent Program

- There are **two things** to note about this skeleton program.
- First, even though we defined the agent mapping as a function from percept *sequences to actions*, *the agent program receives only a single percept as its input*.
- It is up to the agent to build up the ***percept sequence (more than one percept)*** in memory, if it so desires.
- In some environments, it is possible to be quite successful without storing the percept sequence, and in complex domains, it is infeasible to store the complete sequence.
- **Second**, the goal or performance measure is ***not part of the skeleton program***.
- *This is because* the performance measure is applied externally to judge the behavior of the agent, and it is often possible to achieve high performance without explicit knowledge of the performance measure (see, e.g., the square-root agent).

Agent Program with a lookup table.

- It operates by keeping in memory its entire percept sequence, and using it to index into *table*, which contains the appropriate action for all possible percept sequences.

It is instructive to consider why this proposal is **doomed to failure**:

- The table needed for something as simple as an agent that can only play chess would be about 3510° entries.
- It would take quite a long time for the designer to build the table.
- The agent has no autonomy at all, because the calculation of best actions is entirely built-in. So if the environment changed in some unexpected way, the agent would be lost.
- Even if we gave the agent a learning mechanism as well, so that it could have a degree of autonomy, it would take **forever** to learn the right value for all the table entries.

Table-driven Agent Program

```
function TABLE-DRIVEN-AGENT(percept) returns action
  static: percepts, a sequence, initially empty
         table, a table, indexed by percept sequences, initially fully specified
  append percept to the end of percepts
  action — LOOKUP(percepts, table)
  return action
```

Figure 2.5 An agent based on a prespecified lookup table. It keeps track of the percept sequence and just looks up the best action.

PEAS

- Example: Agent = taxi driver
 - Performance measure / Goals: Safe, fast, legal, comfortable trip, maximize profits
 - Environment: Roads, other traffic, pedestrians, customers
 - Actuators / Actuators: Steering wheel, accelerator, brake, signal, horn
 - Sensors / Percepts: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

Taxi driver Agent

- **Environment:** The taxi will need to know where it is, what else is on the road, and how fast it is going.
- This information can be obtained from the **percepts provided by one or more controllable TV cameras, the speedometer, and odometer.**
- To **control the vehicle** properly, especially on curves, it should have an **accelerometer**; it will also need to know the mechanical state of the vehicle, so it will need the usual array of engine and electrical system sensors.
- It might have instruments that are not available to the average human driver: a **satellite global positioning system (GPS)** to give it accurate position information with respect to an electronic map; or **infrared or sonar sensors** to detect distances to other cars and obstacles.
- Finally, it will need a **microphone or keyboard** for the passengers to tell it their destination.

Taxi driver Agent

- **The actions available to a taxi driver will be more or less the same ones available to a human**
- driver: **control over the engine** through the gas pedal and control over steering and braking.
- In addition, it will need output to a screen or **voice synthesizer** to talk back to the passengers, and perhaps some way to communicate with other vehicles.
- What **performance measure would we like our automated driver to aspire to?**
- **Desirable** qualities include getting to the correct destination;
- minimizing fuel consumption and wear and tear; minimizing the trip time and/or cost;
- minimizing violations of traffic laws and disturbances to other drivers;

Taxi driver Agent

- maximizing safety and passenger comfort; maximizing profits.
- Finally, were this a real project, we would need to decide what kind of driving **environment** the taxi will face.
- Should it operate on local roads, or also on freeways?
- Will it be in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not?
- Will it always be driving on the right, or might we want it to be flexible enough to drive on the left in case we want to operate taxis in Britain or Japan?
- Obviously, the more restricted the environment, the easier the design problem.

PEAS

- Example: Agent = Medical diagnosis system

Performance measure: Healthy patient, minimize costs, lawsuits

Environment: Patient, hospital, staff

Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)

Sensors: Keyboard (entry of symptoms, findings, patient's answers)

PEAS

- Example: Agent = Part-picking robot
- Performance measure: Percentage of parts in correct bins
- Environment: Conveyor belt with parts, bins
- Actuators: Jointed arm and hand
- Sensors: Camera, joint angle sensors

Video Reference links

S. NO	Description	Video link
1	Structure of Artificial intelligent Agents	https://www.youtube.com/watch?v=XO6SV0Mup1E
2	This video describes the structure of intelligent agent and mention the types of agents in the environment.	https://www.youtube.com/watch?v=1D6yubBQjuo
3	Design intelligent agents using PEAS Components of an agent	https://www.youtube.com/watch?v=_7hj-eYKAuM

Thank you

UNIT – 1

L8

INTRODUCTION TO AI

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-1

- **Introduction to AI** **(8 Hours)**
- What is AI? , Thinking humanly, Acting rationally, The Foundations of Artificial Intelligence, The History of Artificial Intelligence, The gestation of artificial intelligence, AI becomes an industry, Knowledge-based systems, The return of neural networks, The State of the Art, Intelligent Agents, How Agents Should Act, Structure of Intelligent Agents, Simple reflex agents, Goal-based agents, Utility-based agents , Environments, Environment programs.
- Chap 1, Chap 2

Objectives

Students will be able to

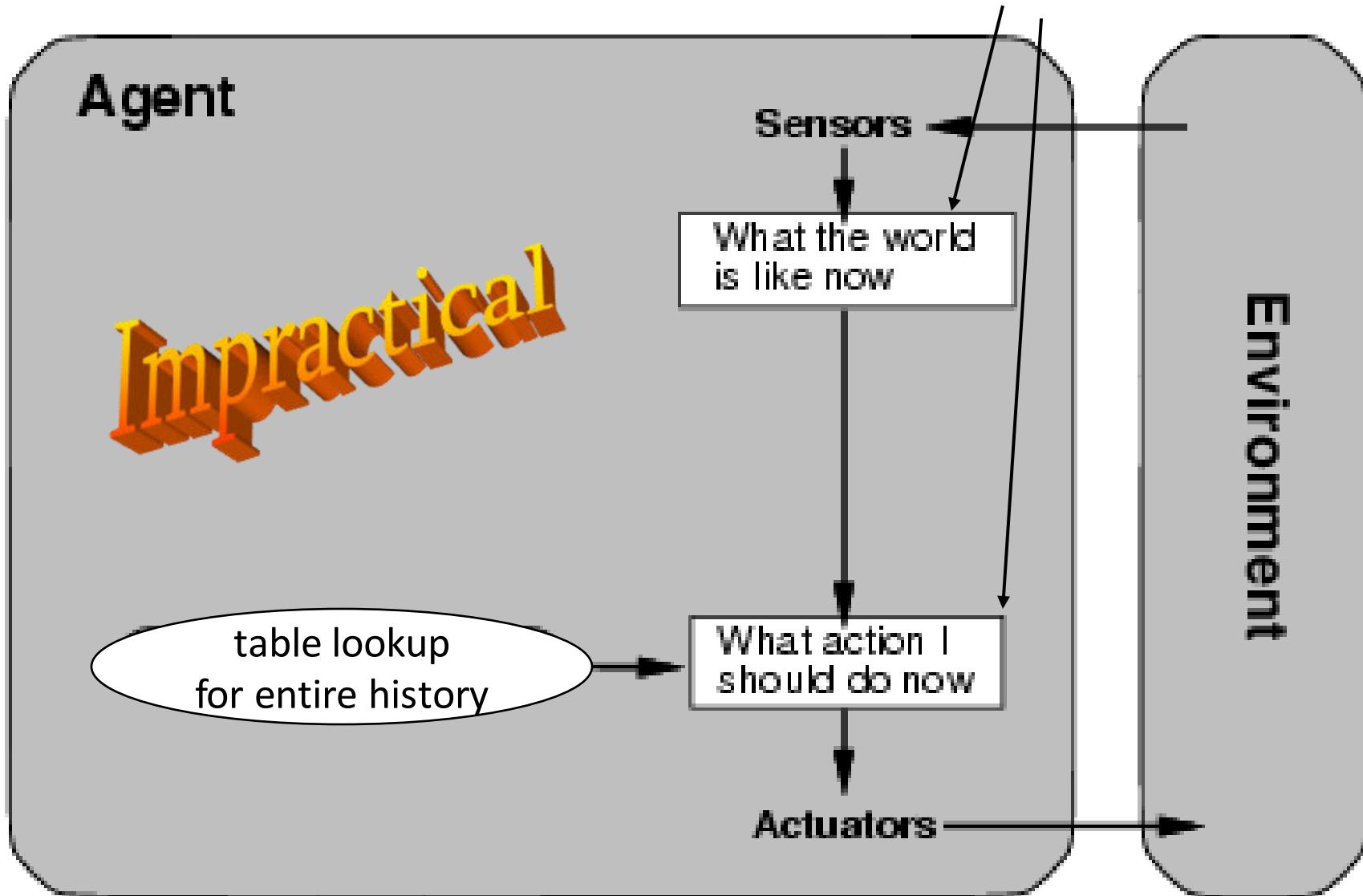
- Gain the knowledge of different types of agents.
- Identify the applications where the types of agents can be applied.
- Grasp the agent environments and their types.

Agent types

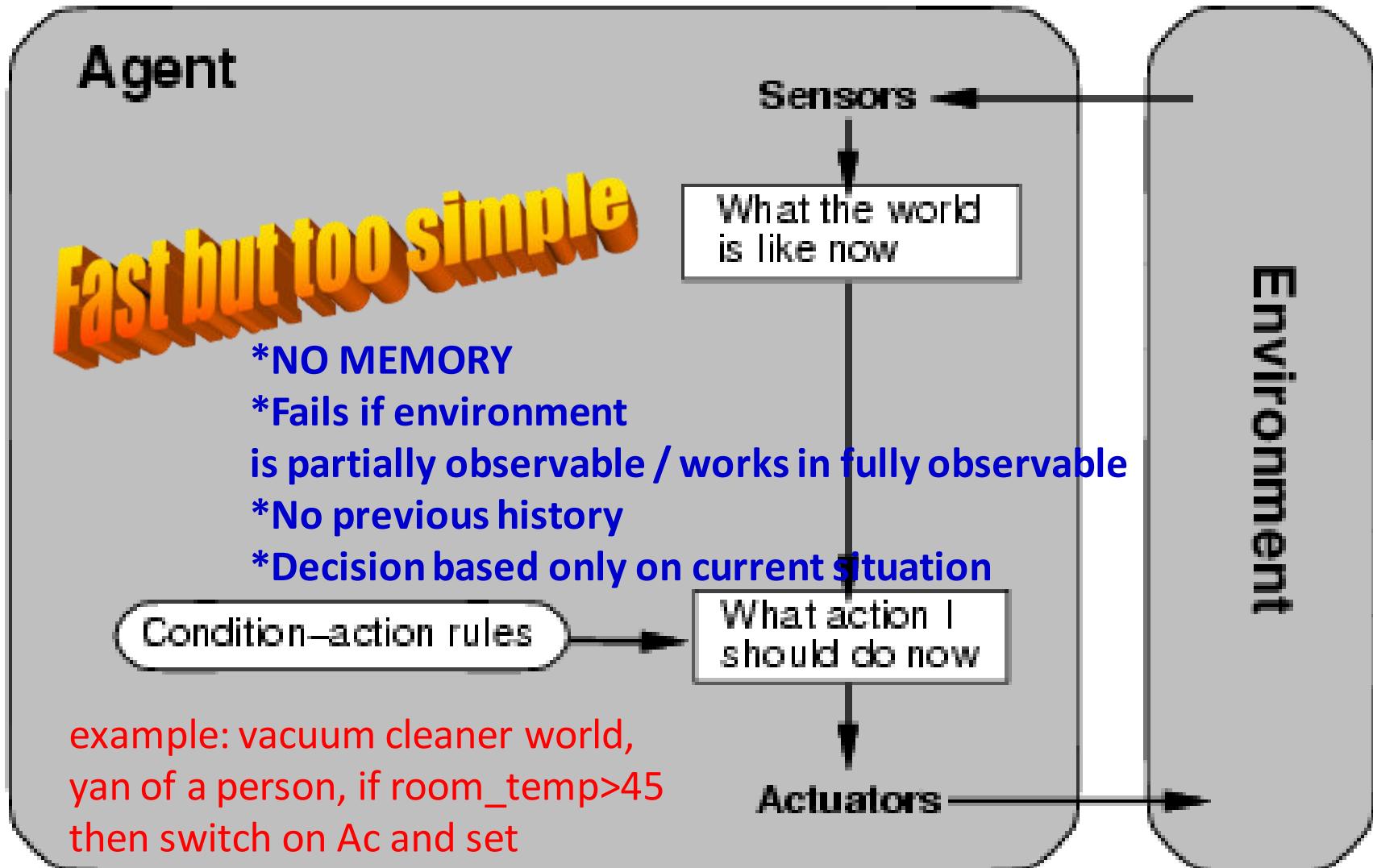
- Five basic types in order of increasing generality:
- Table Driven agent
- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Table Driven Agent.

current state of decision process



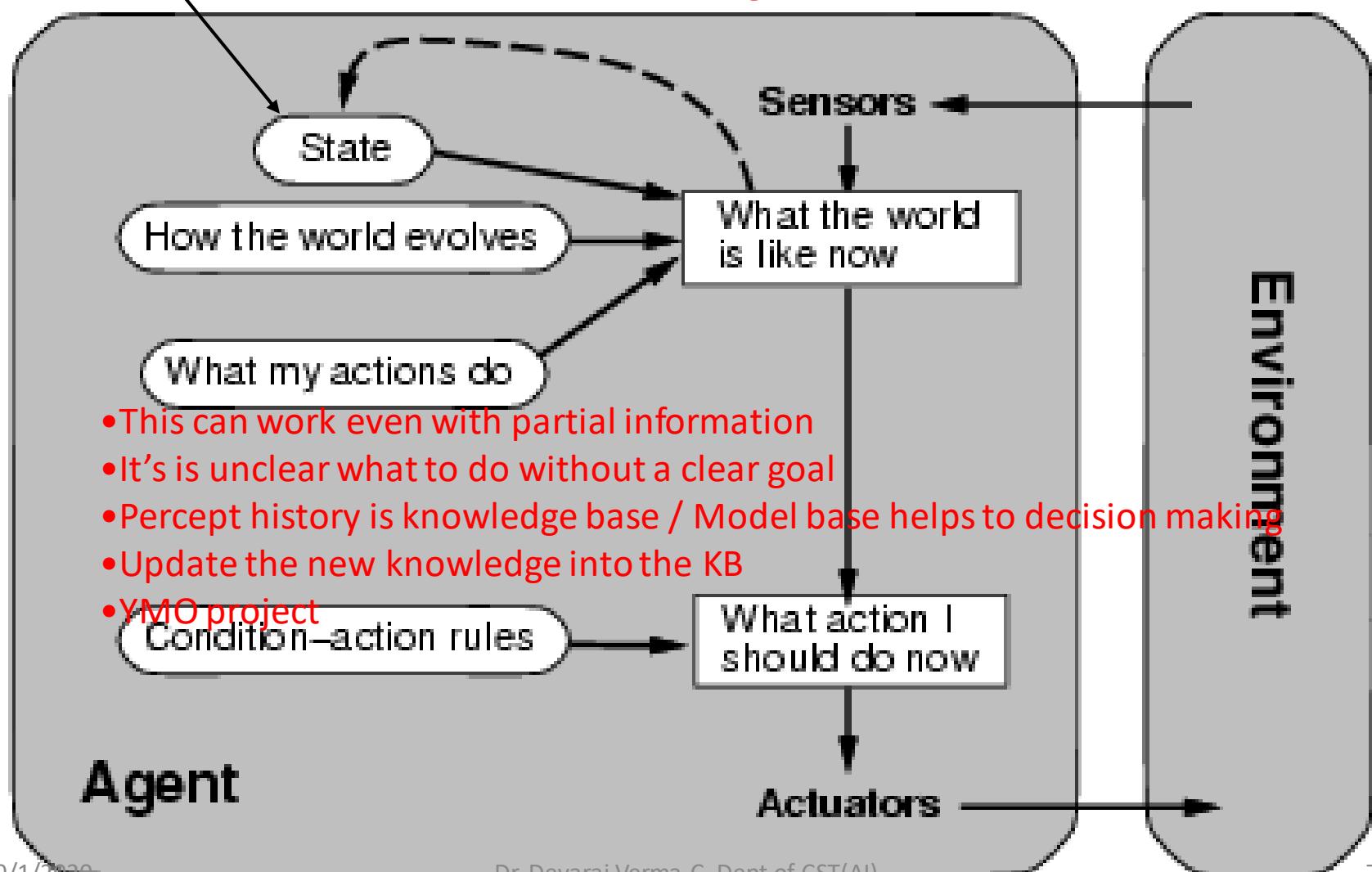
Simple reflex agents



Model-based reflex agents

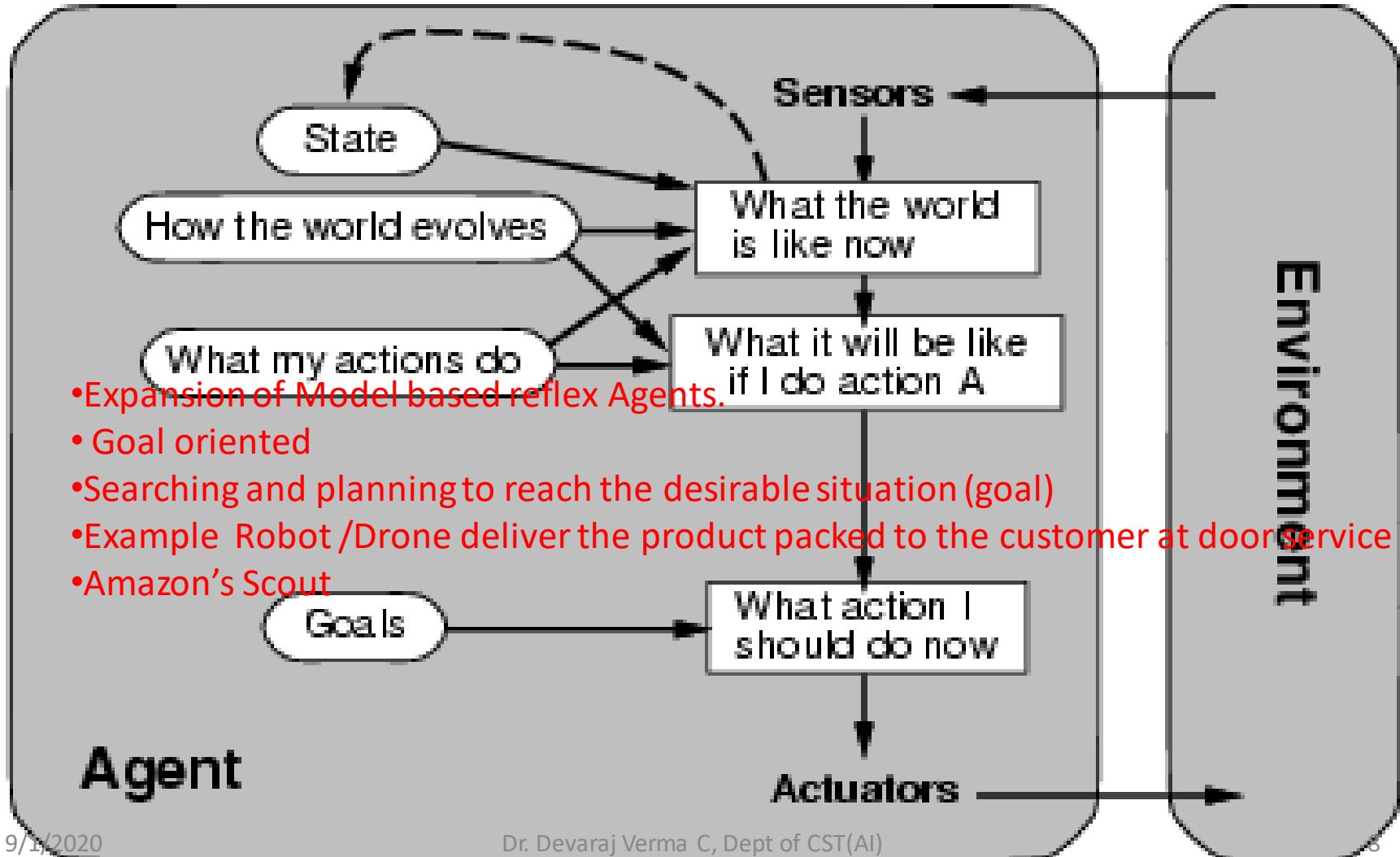
description of
current world state

Model the state of the world by:
modeling how the world changes
how its actions change the world



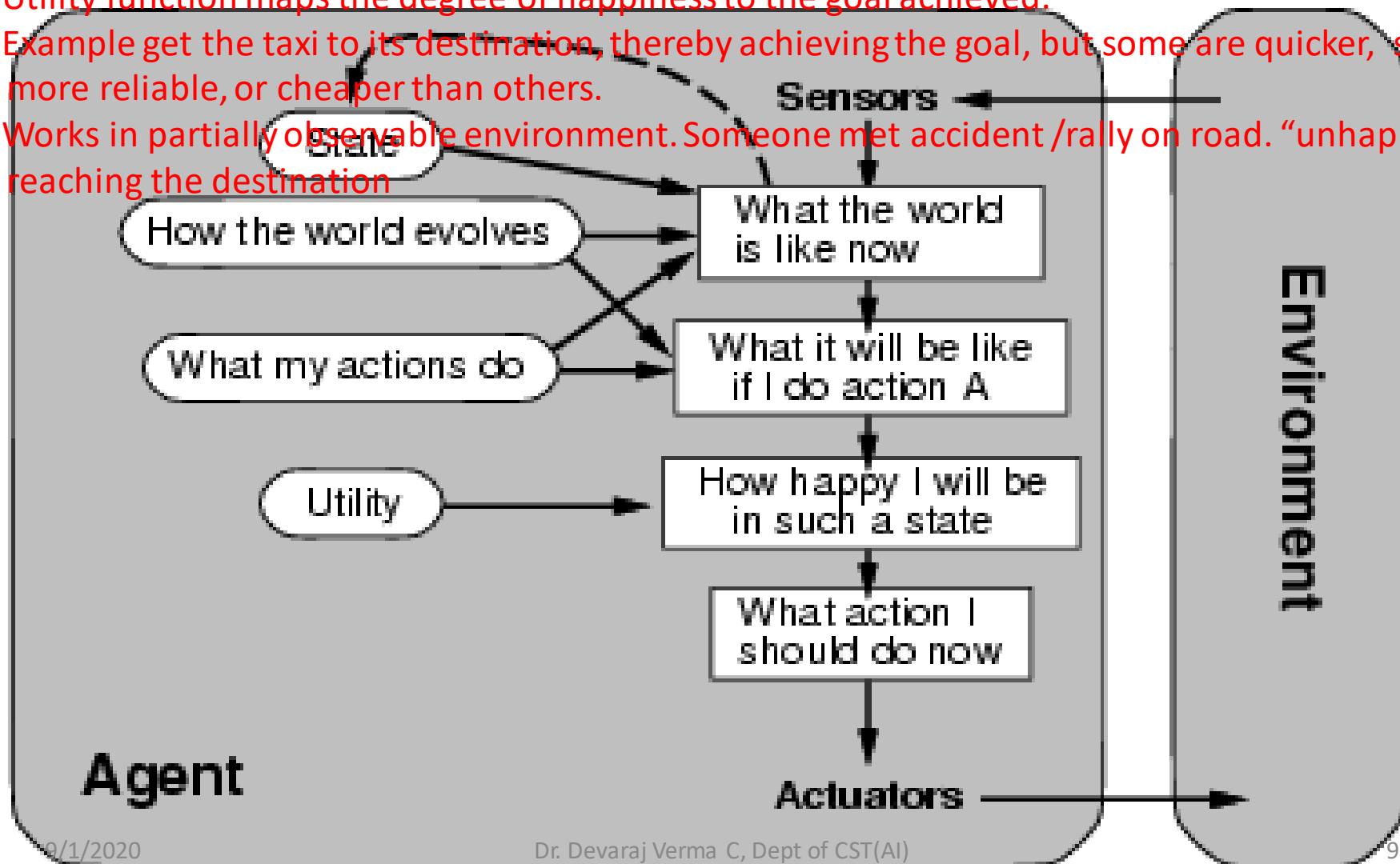
Goal-based agents

Goals provide reason to prefer one action over the other.
We need to predict the future: we need to plan & search



Utility-based agents

- Some solutions to goal states are better than others.
- Which one is best is given by a utility function. Deals with happy and unhappy state.
- Which combination of goals is preferred to make agent happy.
- Utility function maps the degree of happiness to the goal achieved.
- Example get the taxi to its destination, thereby achieving the goal, but some are quicker, safer, more reliable, or cheaper than others.
- Works in partially observable environment. Someone met accident /rally on road. “unhappy”-la reaching the destination

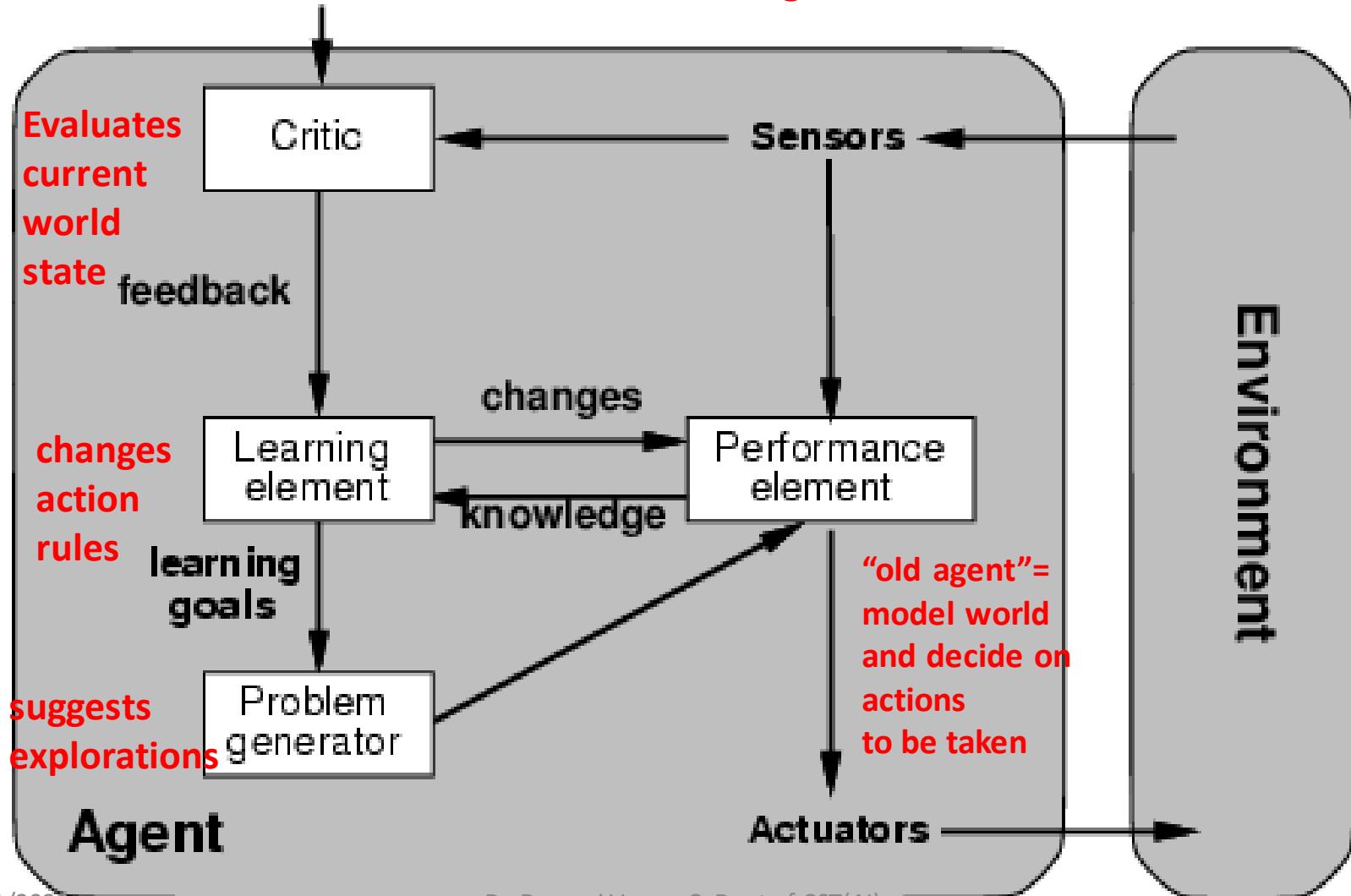


Learning agents

How does an agent improve over time?

By monitoring it's performance and suggesting

Performance standard better modeling, new action rules, etc.



Environment types

- **Fully observable** (vs. **partially observable**) or **Accessible** (Vs **inaccessible**) Environment: An agent's sensors give it access to the complete state of the environment at each point in time. An accessible environment is convenient because the agent need not maintain (or Store) any internal state to keep track of the world.
- **Deterministic** (vs. **stochastic/non deterministic**): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is **strategic/non deterministic**)
- **Episodic** (vs. **sequential**): An agent's action is divided into atomic episodes. i,e Past history/experience divided into episodes. Decisions do not depend on previous decisions/actions. Episodic environments are much simpler because the agent does not need to think ahead.

Environment types

- **Static** (vs. **dynamic**): The environment is unchanged while an agent is deliberating. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time.(The environment is **semi dynamic** if the environment itself does not change with the passage of time but the agent's performance score does)
- **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions. How do we **represent** or **abstract** or **model** the world? **Chess is discrete**—there are a fixed number of possible moves on each turn. **Taxi driving is continuous**—the speed and location of the taxi and the other vehicles sweep through a range of continuous values.
- **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment. Does the other agent interfere with my performance measure?

Video Reference links

S. NO	Description	Video link
1	This video gives the knowledge of Types of intelligent Agents	https://www.youtube.com/watch?v=QKXk9SowwT4
2	This video provides the information of Agent, PEAS and environments types	https://www.youtube.com/watch?v=ujEPyAGT7Q

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

Students will be able to

- Understand the steps in problem solving agent.
- Apply the knowledge of problem solving steps to solve the different types of problems in the real life applications

Problem and search

- An agent can act by establishing goals and considering sequences of actions that might achieve those goals.
- A goal and a set of means for achieving the goal is called a problem, and the process of exploring what the means can do is called search.

Draw backs / limitations of simple reflex agents

- They are unable to plan ahead.
- They are limited in what they can do because their actions are determined only by the current percept.
- Furthermore, they have no knowledge of what their actions do nor of what they are trying to achieve.

To overcome these drawbacks, problem solving agents are used.

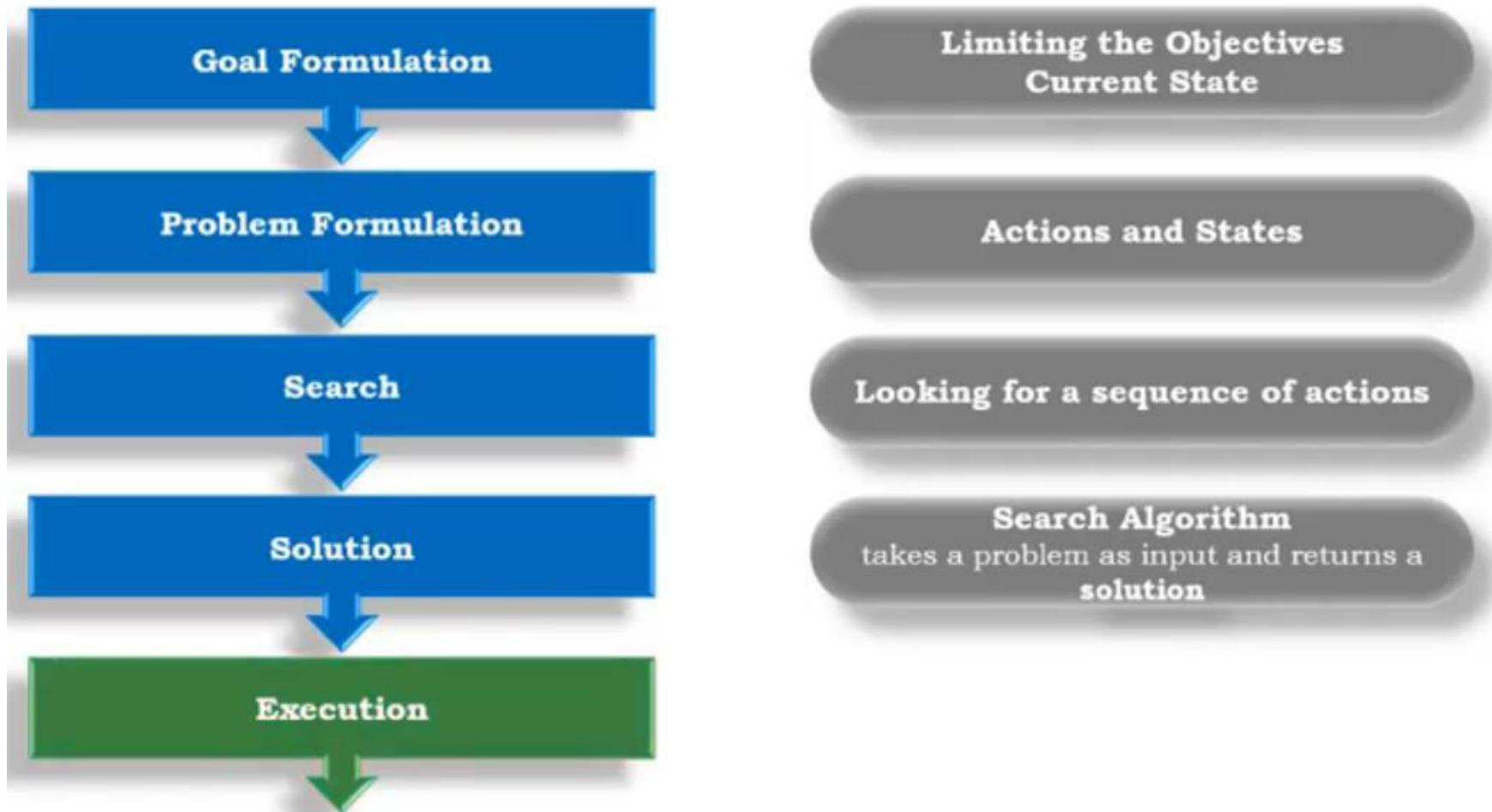
PRQBLEM-SOLVING AGENTS

- one kind of goal-based agent called a **problem-solving agent**.
- Problem-solving agents decide what to do by finding sequences of actions that lead to desirable states.
- Intelligent agents are supposed to act in such a way that the environment goes through a sequence of states that maximizes the performance measure.

Steps in Problem solving:

- Goal Formulation phase
- Problem Formulation phase
- Search phase
- Solution phase
- Execution Phase

Functionality of Problem solving Agent / Steps in Problem Solving



Steps in Problem Solving

- **Goal formulation, limiting the objectives based on the current situation,** is the first step in problem solving. As well as formulating a goal, the agent may wish to decide on some other factors that affect the desirability of different ways of achieving the goal.
- **Problem formulation** is the process of deciding what **actions and states to consider**, and follows goal formulation.
- Agent with several immediate options of unknown value can decide what to do by first examining ; different **possible sequences of actions** that lead to states of known value, and then choosing the best one. This process of looking for such a sequence is called **search**.
- A **search algorithm** takes a problem as input and returns a **solution** in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out. This is called the **execution phase**. Thus, we have a simple "formulate, search, execute" design for the agent, as shown in Figure 3.1.

Simple Problem solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
    inputs: p, a percept
    static: s, an action sequence, initially empty
            state, some description of the current world state
            g, a goal, initially null
            problem, a problem formulation

    state  $\leftarrow$  UPDATE-STATE(state, p)
    if s is empty then
        g  $\leftarrow$  FORMULATE-GOAL(state)
        problem  $\leftarrow$  FORMULATE-PROBLEM(state, g)
        s  $\leftarrow$  SEARCH(problem)
    action  $\leftarrow$  RECOMMENDATION(s, state)
    s  $\leftarrow$  REMAINDER(s, state)
    return action
```

Figure 3.1 A simple problem-solving agent.

Types of Problems

- Let the world contain just two locations. Each location may or may not contain dirt, and the agent may be in one location or the other. There are 8 possible world states, as shown in Figure 3.2.
- The agent has three possible actions in this version of the vacuum world: *Left, Right, and Suck*.
- Assume, for the moment, that sucking is 100% effective. The goal is to clean up all the dirt. That is, the goal is equivalent to the state set {7,8}.
- Single state problem
- Multi state problem
- Contingency Problem
- Interleaving problem
- Exploration problem

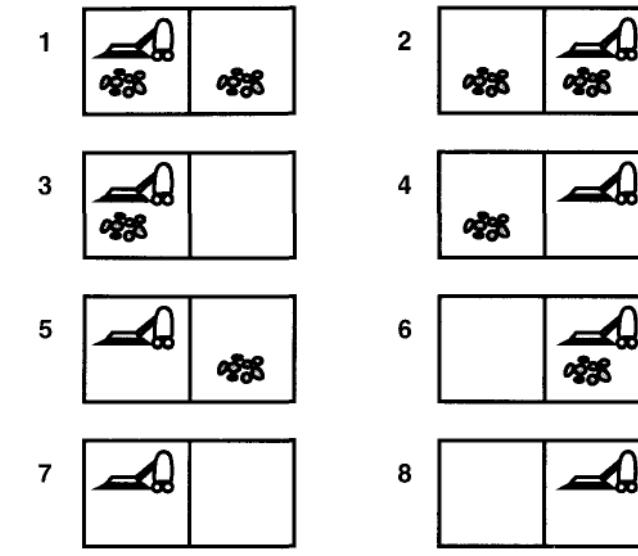


Fig 3.2

The eight possible states of the simplified vacuum world.

Types of Problems

- **Single-state problem:** suppose that the agent's sensors give it enough information to tell exactly which state it is in (i.e., the **world is accessible**); and suppose that it knows exactly what each of its actions does.
- Then it can calculate exactly which state it will be in after any sequence of actions.
- For example, if its initial state is 5, then it can calculate that the action sequence *[Right, Suck]* will get to a goal state (8). This is the simplest case, which we call a **single-state problem**.
- **Multi-state problem:** suppose that the agent knows all the effects of its actions, but has limited access to the world state.
- For example, in the extreme case, it may have no sensors at all. In that case, it knows only that its initial state is one of the set {1,2,3,4,5,6,7,8}.
- it knows what its actions do, it can, for example, calculate that the action *Right will cause it to be in one of the states {2,4,6,8}*.
- In fact, the agent can discover that the action sequence *[Right,Suck,Left,Suck]* is guaranteed to reach a goal state no matter what the start state. When the **world is not fully accessible**, the agent must reason about sets of states that it might get to, rather than single states. This is a **multiple-state problem**.

Types of Problems

- **Contingency Problem:** solving this problem requires **sensing during the execution phase**.
- Notice that the agent must now calculate a **whole tree of actions**, rather than a single action sequence.
- In general, **each branch of the tree deals** with a possible contingency (predication) that might arise a solution state. For this reason, we call this a **contingency problem**.
- Many problems in the real, physical world are contingency problems, because exact prediction is impossible.
- There is **no fixed action sequence** that guarantees a solution to the problem.
- Suppose that sensor tells agent is in one of the state {1,3}, and action [Suck,Right,Suck] sequence might not reach the goal state {7,8}.
- $(\{1,3\}, \text{suck}) = \{5, 7\}$, $(\{5, 7\}, \text{Right}) = \{6, 8\}$, $(\{6\}, \text{suck}) = \{8\}$. But $(\{8\}, \text{suck}) = \text{failure}$
- {1,3}: first suck, then move right, then suck **only if there is dirt there. This is the solution sequence.**

Types of Problems

- **Interleaving Problem:** Agent can act *before it has found a guaranteed plan.*
- This is useful because rather than considering in advance every possible contingency that might arise during execution, it is often better to actually start executing and see which contingencies *do arise*. *The agent can then continue to solve the problem given the additional information.*.:
- **Exploration Problem:** An agent that has no information about the effects of its actions. This is somewhat equivalent to being lost in a strange country with no map at all, and is the hardest task faced by an intelligent agent.
- The agent must *experiment, gradually discovering* what its actions do and what sorts of states exist. This is a kind of search, but a search in the real world rather than in a model thereof.
- Taking a step in the real world, rather than in a model, may involve significant danger for an ignorant agent. If it survives, the agent learns a "map" of the environment, which it can then use to solve subsequent problems.

Video Reference links

S. NO	Description	Video link
1	This video gives the knowledge of Problem formulation steps	https://www.youtube.com/watch?v=KTPmo-KsOis
2	This video describes types of Problems & Problem Solving Strategies.	https://www.youtube.com/watch?v=ftgtzFaHFGE
3	This video provides the insight knowledge of Single State and Multiple State Problem Formulation in Artificial Intelligence.	https://www.youtube.com/watch?v=g5LO4rC_kss

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

Students will be able to

- Apply the knowledge of problem Formulation steps to the real life problems.

Well defined Problem Formulation

A problem is really a collection of information that the agent will use to decide what to do. We will consider only single-state problem.

1. A set of *states* S
2. An *initial state* $s_i \in S$
3. A set of *actions* A
 - $\forall s \text{ } Actions(s) = \text{the set of actions that can be executed in } s,$ that are *applicable* in $s.$
4. *Transition Model:* $\forall s \forall a \in Actions(s) \text{ } Result(s, a) \rightarrow s_r$
 - s_r is called a *successor* of s
 - $\{s_i\} \cup Successors(s_i)^* = \text{state space}$
5. *Goal test Goal(s)*
 - Can be implicit, e.g. $checkmate(x)$
 - s is a *goal state* if $Goal(s)$ is *true*
6. *Path cost* (additive)
 - e.g. sum of distances, number of actions executed, ...
 - $c(x, a, y)$ is the step cost, assumed ≥ 0
 - (where action a goes from state x to state y)

solution

- A *solution* is a **sequence of actions** from the initial state to a goal state.
- *Optimal Solution:* A solution is *optimal* if no solution has a lower path cost.
- **Consider A initial state, G goal state. Distance between cities is 1 units.**

Possible solutions:

- **A-B-C-D-E-F-G = 6**
- **A – B- C – G = 3**
- **A –C-E-F-G = 4**
- **A-D-G = 2**

Optimal solution:

- **A-D-G = 2**

Real World Scenario (A Touring Agent Problem)

- Imagine our agent in the city of Arad, Romania, toward the end of a touring holiday.
- The agent has a ticket to fly out of Bucharest the following day. The ticket is nonrefundable, the agent's visa is about to expire, and after tomorrow, there are no seats available for six weeks.
- In that case, It makes sense for the agent to adopt the **goal** of getting to Bucharest.
- As in fig 3.3, Total solutions = $N * 2^N$
- Romania states $N = 20$. therefore Total solutions = $20 * 2^{20}$
- **Note: A Problem may story, until the problem is converted into the formal notation. You will not be able to get the solution to the problem.**

Road map of Romania

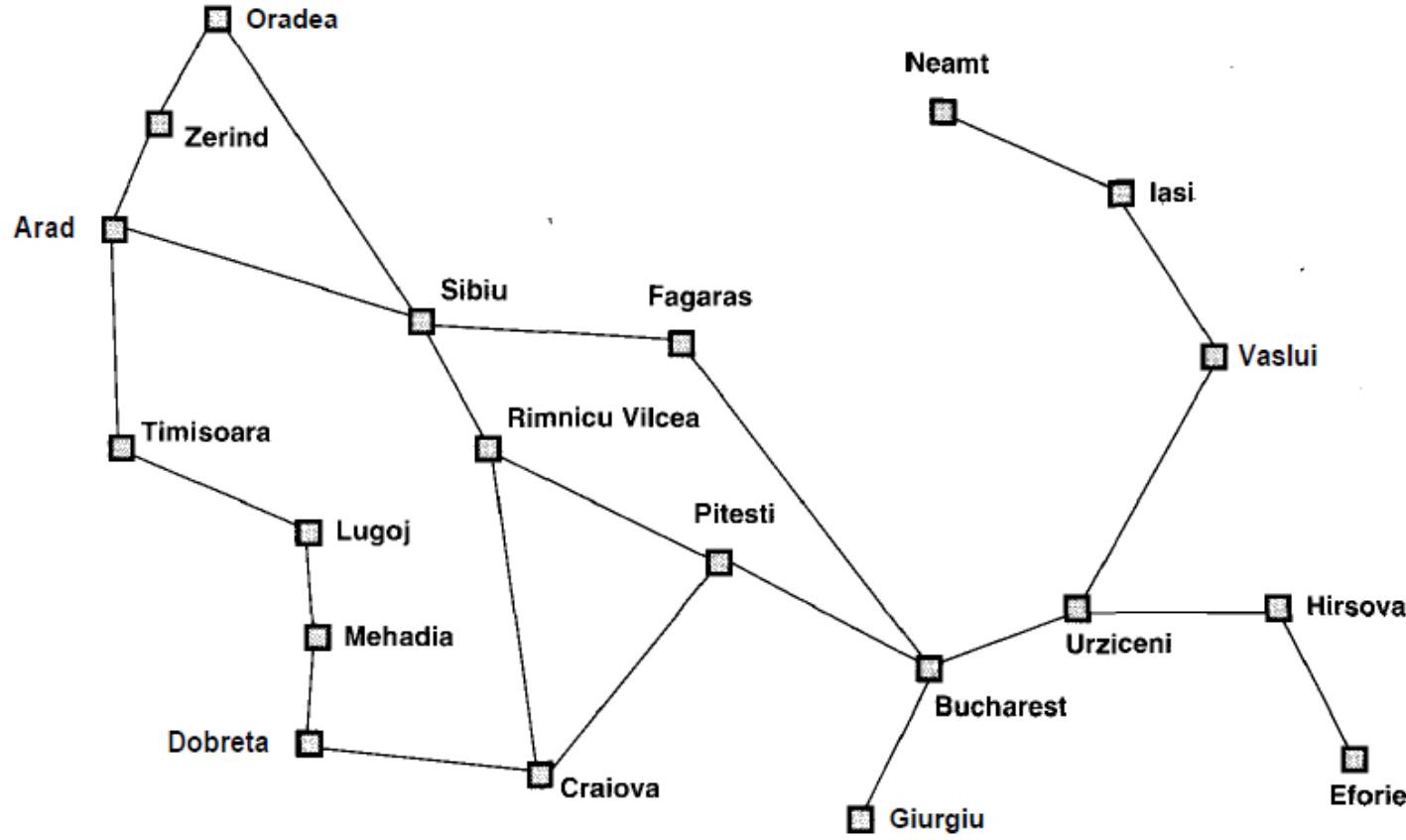


Figure 3.3 A simplified road map of Romania.

Romania Problem Formulation

Steps in problem formulation in Romania:

- 1) Initial state $IN(STATE)$ is $IN(ARAD)$
- 2) Actions (STATE) \Rightarrow GO (STATES)
 $Actions(ARAD) \Rightarrow Go(ZERIND), Go(SIBIU),$
 $Go(TIMISOARA)$
- 3) Transitional Model Result(s, a) $\Rightarrow IN(x)$
 $Result(IN(ARAD), Go(ZERIND)) = IN(ZERIND)$
- 4) Goal Test $IN(x) == \{IN(g)\}$
 $IN(x) == \{IN(Bucharest)\} // \text{in each state check agent reached the Bucharest or not.}$
- 5) Path Cost $C(s, a, x) == p$
 $C(IN(ARAD), Go(ZERIND), IN(ZERIND)) = 75$

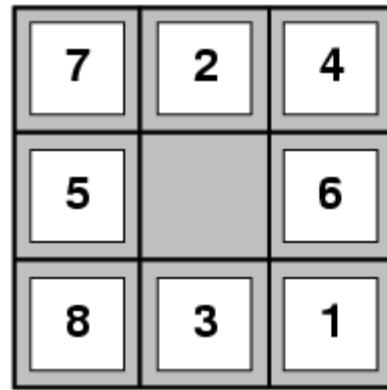
Problem formulation for Toy problems

- 8 – Puzzle Problem
- 8-queens problem
- The vacuum world
- Missionaries and cannibals
- Cryptarithmetic

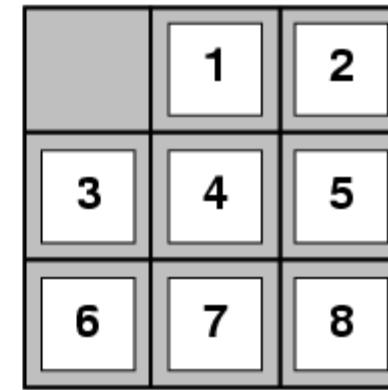
The 8-puzzle

- The **8-puzzle**, an instance of which is shown in Figure 3.4, consists of a **3x3 board with eight** numbered tiles and a blank space. N
- The object is to reach the Goal configuration shown on the right of the figure.
- One important trick is to notice that rather than use operators such as "move the 3 tile into the blank space," it is more sensible to have operators such as "the blank space changes places with the tile to its left."
- **Problem formulation:**
- **States:** a state description specifies the location of each of the eight tiles in one of the nine squares. For efficiency, it is useful to include the location of the blank.
- **Operators/Successors:** blank moves left, right, up, or down.
- **Goal test:** state matches the goal configuration shown in Figure 3.4.
- **Path cost:** each step costs 1, so the path cost is just the length of the path.

8 – puzzle problem



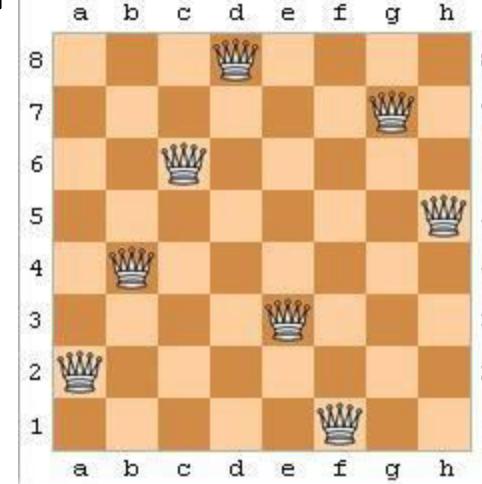
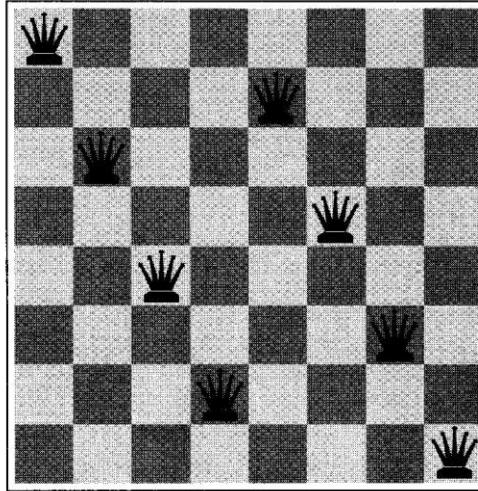
Start State



Goal State

Note: optimal solution of n -Puzzle family is NP-hard]

The 8-queens problem



- The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Figure on left shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at top left. Fig on right is one of the solution to the problem.

states: any arrangement of 0-8 queens on the board is a state

Initial state: no queens on the board

actions: add a queen to any empty square

goal test: 8 queens are on the board, none attacked

$64 \cdot 63 \cdots 57 = 1.8 \times 10^{14}$ possible sequences.

Video Reference links

S. NO	Description	Video link
1	This video describes the Problem formulation of Toy Problem or 8-Puzzle, The vacuum world, and water jug problem.	https://www.youtube.com/watch?v=hWaTrfg00YI

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

Students will be able to

- Apply the knowledge of problem Formulation steps to the real life problems.

Problem formulation for Toy problems

- 8 – Puzzle Problem
- 8-queens problem
- The vacuum world
- Missionaries and cannibals
- Cryptarithmetic

Vacuum world state space graph

- **states** integer dirt and robot location.
 - The agent is in one of two locations, each of which might or might not contain dirt – 8 possible states
- **Initial state:** any state
- **actions** *Left, Right, Suck*
- **goal test** no dirt at all locations
- **path cost** 1 per action

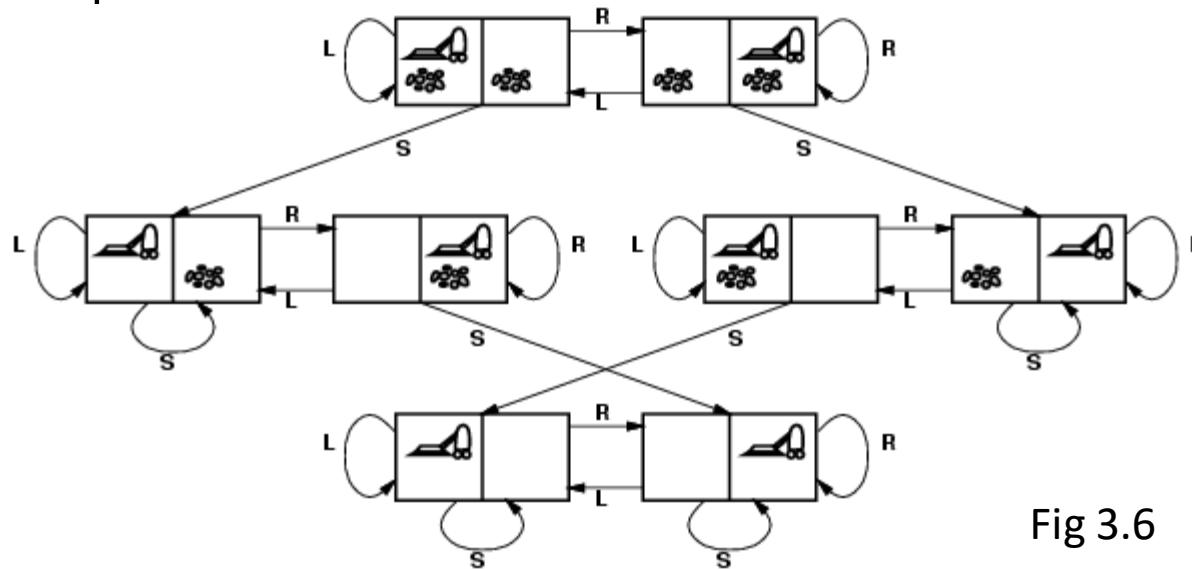


Fig 3.6

Vacuum world state space graph

consider the case where the agent has no sensors, but still has to clean up all the dirt. Because this is a multiple-state problem, we will have the following:

State sets: subsets of states 1-8 shown in Figure 3.2 or 3.6.

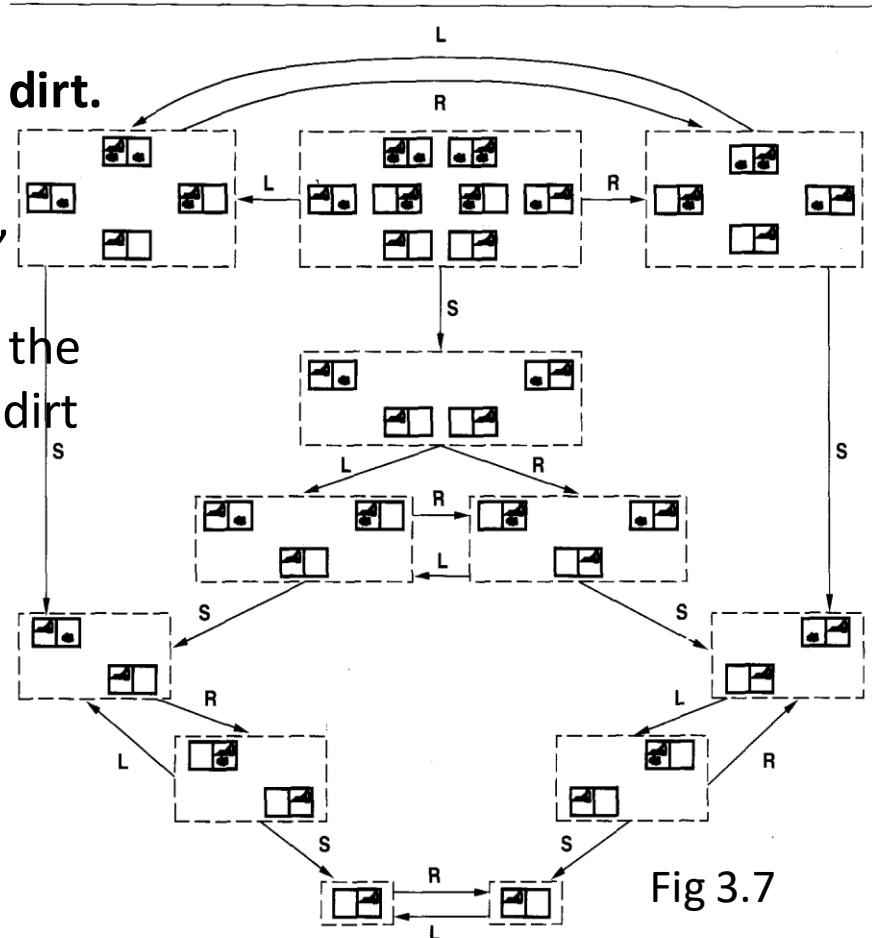
Operators: move left, move right, suck.

Goal test: all states in state set have no dirt.

Path cost: each action costs 1.

The start state set is the set of all states, because the agent has no sensors.

A solution is any sequence leading from the start state set to a set of states with no dirt (see Figure on right).



Missionaries and cannibals

- There are three missionaries and three cannibals on the left bank of a river. They wish to cross over to the right bank using a boat that can only carry two at a time. The number of cannibals on either bank must never exceed the number of missionaries on the same bank, otherwise the missionaries will become the cannibals' dinner! Plan a sequence of crossings that will take everyone safely across.
- **How shall they cross the river?**

Formulation of problem

- **Initial state:** all M, all C, and boat on one bank
- **Actions:** ??
- **Transition Model??**
- **Goal test:** True if all M, all C, and boat on other bank
- **Cost:** ??



Representation:

- **States:** Which properties matter & how to represent
- **Actions & Transition Model:** Which actions are possible & how to represent
- **Path Cost:** Deciding which action is next

Missionaries and cannibals

-
- 1. 1 Missionary and 1 cannibal There are 3 cannibals and 3 missionaries and we have to help them to cross the lake. If there is going to be less missionaries than the cannibals, then the missionaries will be eaten by the cannibals. How are we going to help them?
- 2. Take out the Cannibal and come back
- 3. Take out the missionary and take in 2 cannibals
- 4. Take out 1 cannibal and come back
- 5. Take out the cannibals and take in 2 missionaries
- 6. Take out 1 missionary and take in 1 cannibal
- 7. Take out the cannibal and take in the missionary
- 8. Take out 2 missionaries and take in 1 cannibal
- 9. Take in 1 cannibal
- 10. Take out 1 cannibal
- 11. Take in 1 cannibal
- 12. Take out 2 cannibals

Missionaries and cannibals

- Any permutation of the three missionaries or the three cannibals leads to the same outcome. The considerations lead to the following formal definition of the problem:
- **States:** a state consists of an ordered sequence of three numbers representing the number of missionaries, cannibals, and boats on the bank of the river from which they started. Thus, the start state is (3,3,1).
- **Operators:** from each state the possible operators are to take either one missionary, one cannibal, two missionaries, two cannibals, or one of each across in the boat. Thus, there are at most five operators, although most states have fewer because it is necessary to avoid illegal states.
- Note that if we had chosen to distinguish between individual people then there would be 27 operators instead of just 5.
- **Goal test:** reached state (0,0,0).
- **Path cost:** number of crossings. (1 unit cost per crossing)

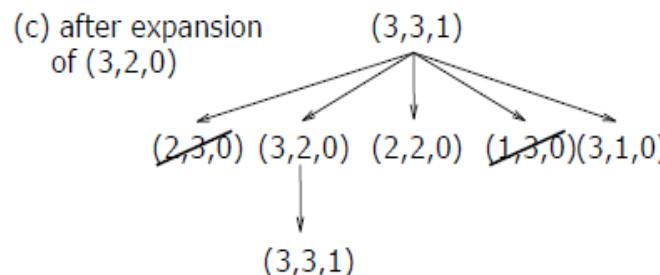
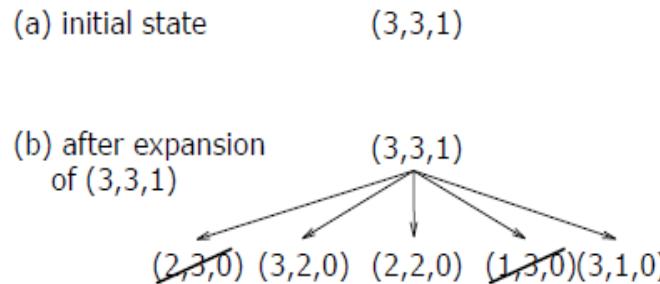
Missionaries and cannibals

- States: (CL, ML, BL)
 - Initial 331 Goal 000
 - Actions:

General Search

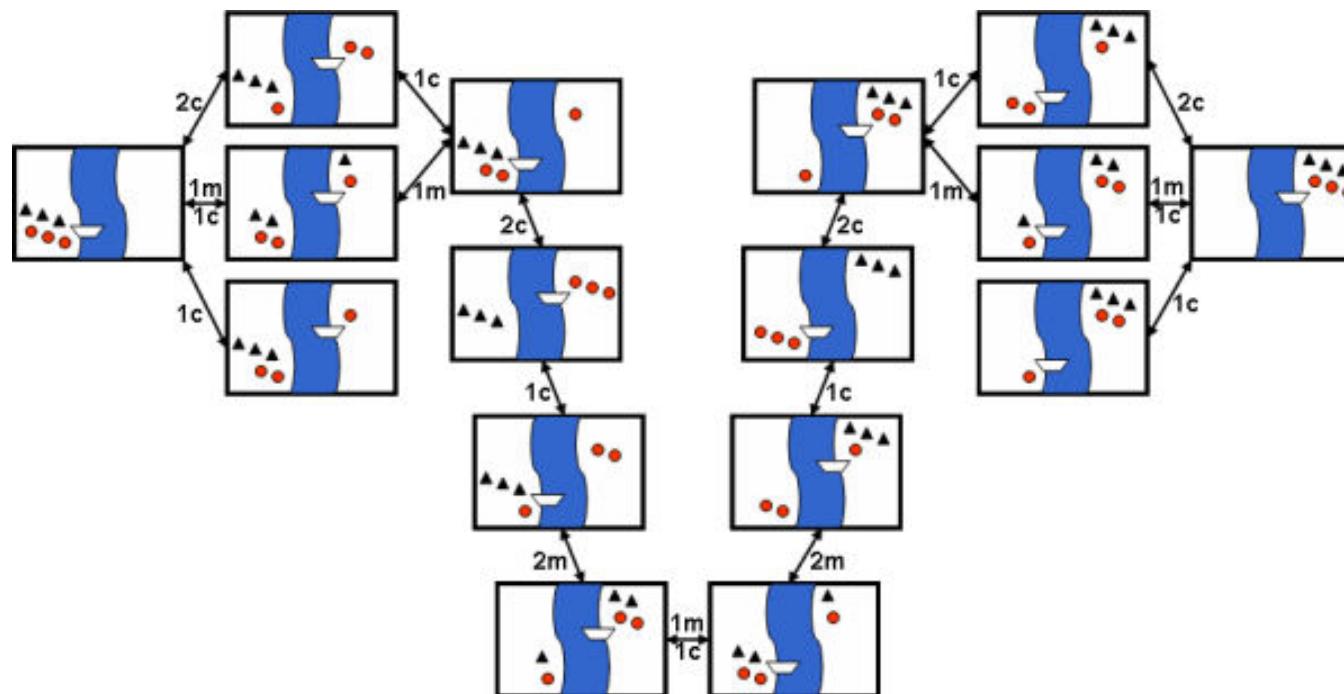
From the initial state, produce all successive states step by step → search tree.

- 101,011,111,201,021



Searching for a Solution

- The initial state is shown to the right here, where black triangles represent missionaries and red circles represent cannibals.
 - This problem can be solved by searching for a solution, which is a sequence of actions that leads from the initial state to the goal state. The goal state is effectively a mirror image of the initial state. The complete search space is shown in below figure .



Cryptarithmetic

- Cryptarithmetic problems are where numbers are replaced with alphabets. By using standard arithmetic rules we need to decipher the alphabet.
- The following formulation is probably the simplest:
- **States:** a cryptarithmetic puzzle with some letters replaced by digits.
- **Operators:** replace all occurrences of a letter with a digit not already appearing in the puzzle.
- **Goal test:** puzzle contains only digits, and represents a correct sum.
- **Path cost: zero.** All solutions equally valid.

• If SEND + MORE = MONEY then find M + O + N + E + Y

S E N D
 M O R E

M O N E Y

Explanation:

Addition of two numbers with 'n' digits, results in a $n+1$ digits, then the left most place always = 1.

So M = 1. Substitute this value.

Now 'o' cannot be 1 as M already 1. It may not be 2 either as $S+1 = 12$ or $1 + S + 1 = 12$ in the both cases S is a two digit number. So 'o' is nothing but zero. Put o = 0.

Now S can be either 8 or 9. If S = 8, then there must be a carry over.

$$E + 0 = 10 + N \text{ or } 1 + E + 0 = 10 + N$$

In the above two cases, $E - N = 10$ is not possible and $E - N = 9$ not possible as as N cannot be zero.

So E = 9.

Now $E + 0 = N$ is not possible as $E = N$. So $1 + E = N$ possible.

$$\begin{array}{r} & & 1 \\ & 9 & E & N & D \\ & 1 & 0 & R & E \\ \hline 1 & 0 & N & E & Y \end{array}$$

The possible cases are, $N + R = 10 + E \dots (1)$ or $1 + N + R = 10 + E \dots (2)$

Substituting $E = N - 1$ in the first equation, $N + R = 10 + N - 1$, we get $R = 9$ which is not possible.

Substituting $E = N - 1$ in the second equation, $1 + N + R = 10 + N - 1$, we get $R = 8$.

We know that N and E are consecutive and N is larger. Take $(N, E) = (7, 6)$ check and substitute, you wont get any unique value for D.

Take $(N, E) = (6, 5)$, Now you get $D = 7, Y = 2$.

$$\begin{array}{r} S & E & N & D \\ M & O & R & E \\ \hline M & O & N & E & Y \end{array} \qquad \begin{array}{r} 9 & 5 & 6 & 7 \\ 1 & 0 & 8 & 5 \\ \hline 1 & 0 & 6 & 5 & 2 \end{array}$$

$$M + O + N + E + Y = 14$$

Video Reference links

S. NO	Description	Video link
1	This video describes the Problem formulation of Toy Problem or 8-Puzzle, The vacuum world, and water jug problem.	https://www.youtube.com/watch?v=hWaTrfg00YI
2	This video demonstrates the missionary and cannibal problem.	https://www.youtube.com/watch?v=CgW67TBN8zQ
3	This video demonstrates the cryptarithmetic problem with an Example SEND + MORE = MONEY	https://www.youtube.com/watch?v=HC6Y49iTg1k

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

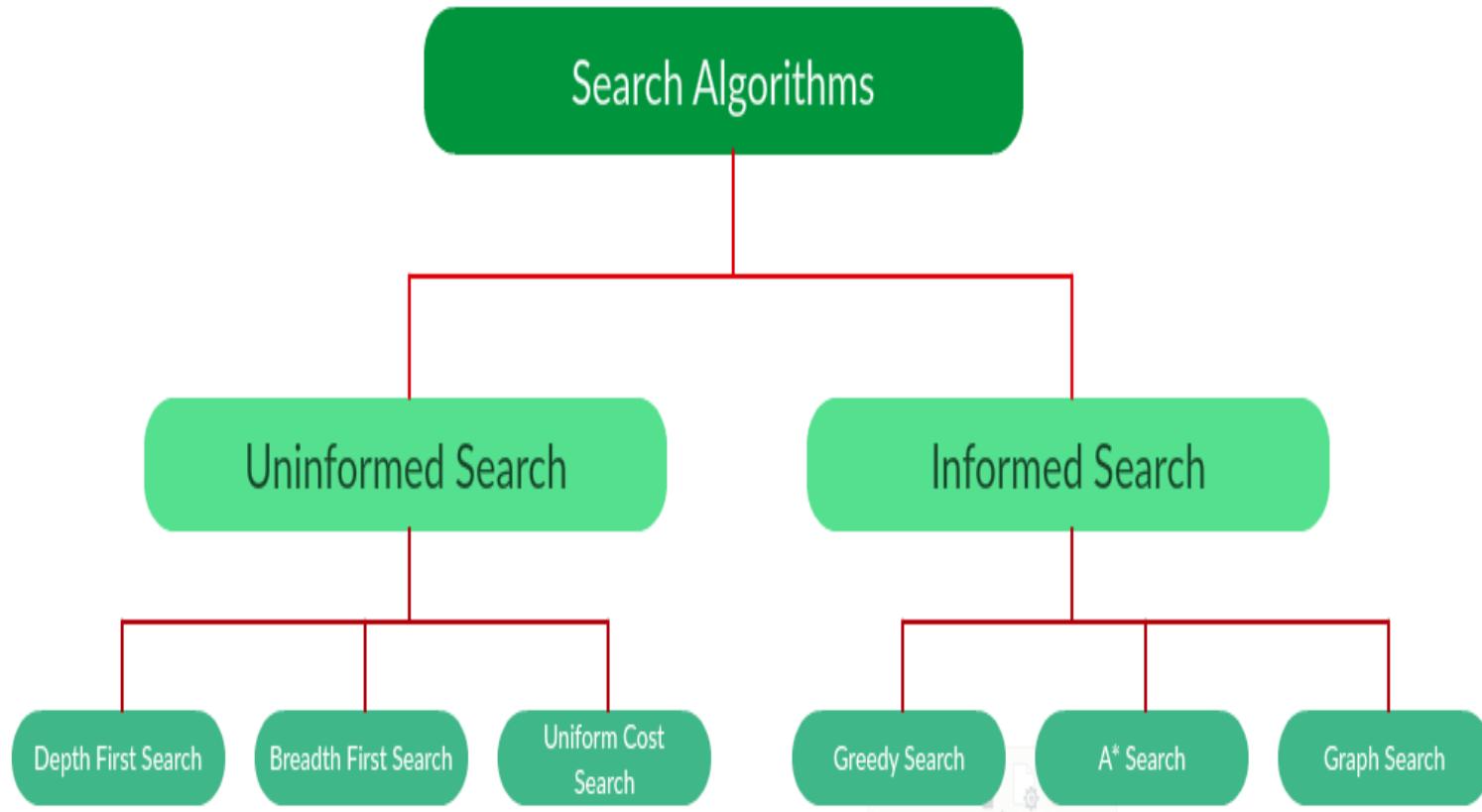
Students will be able to

- Understand the search strategies
- Gain the knowledge of uninformed search Methods.
- Apply the search techniques to solve the problems.

Search for solutions

- **Artificial Intelligence** is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.
- A search problem consists of:
 - **A State Space.** Set of all possible states where you can be.
 - **A Start State.** The state from where the search begins.
 - **A Goal Test.** A function that looks at the current state returns whether or not it is the goal state.
- The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

Search strategies



Four criteria of search strategies

- Finding the right **search strategy for a** problem. To evaluate strategies in terms of four criteria:
- **Completeness:** is the strategy guaranteed to find a solution when there is one?
- **Time complexity:** how long does it take to find a solution?
- **Space complexity:** how much memory does it need to perform the search?
- **Optimality:** does the strategy find the highest-quality solution when there are several different solutions?

Uninformed Search Algorithms

- No additional information on the goal node.
- The plans to reach the goal state from the start state differ only by the order and/or length of actions.
- Uninformed search is also called **Blind search**.

The uninformed search algorithms are

Depth First Search	Depth limited Search
Breath First Search	Iterative Deepening Search
Uniform Cost Search	Bidirectional Search

Each of these algorithms will have:

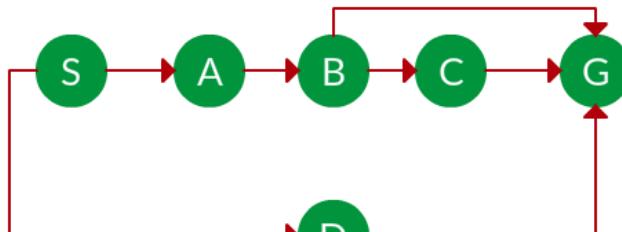
- A problem **graph**, containing the start node S and the goal node G.
- A **strategy**, describing the manner in which the graph will be traversed to get to G .
- A **fringe**, which is a data structure used to store all the possible states (nodes) that you can go from the current states.
- A **tree**, that results while traversing to the goal node.
- A solution **plan**, which the sequence of nodes from S to G.

Generic search algorithm

- Fringe = set of nodes generated but not expanded
- fringe := {initial state}
- loop:
 - if fringe empty, declare failure
 - choose and remove a node v from fringe
 - check if v's state s is a goal state; if so, declare success
 - if not, expand v, insert resulting nodes into fringe
- Key question in search: Which of the generated nodes do we expand next?

Depth-first search (DFS)

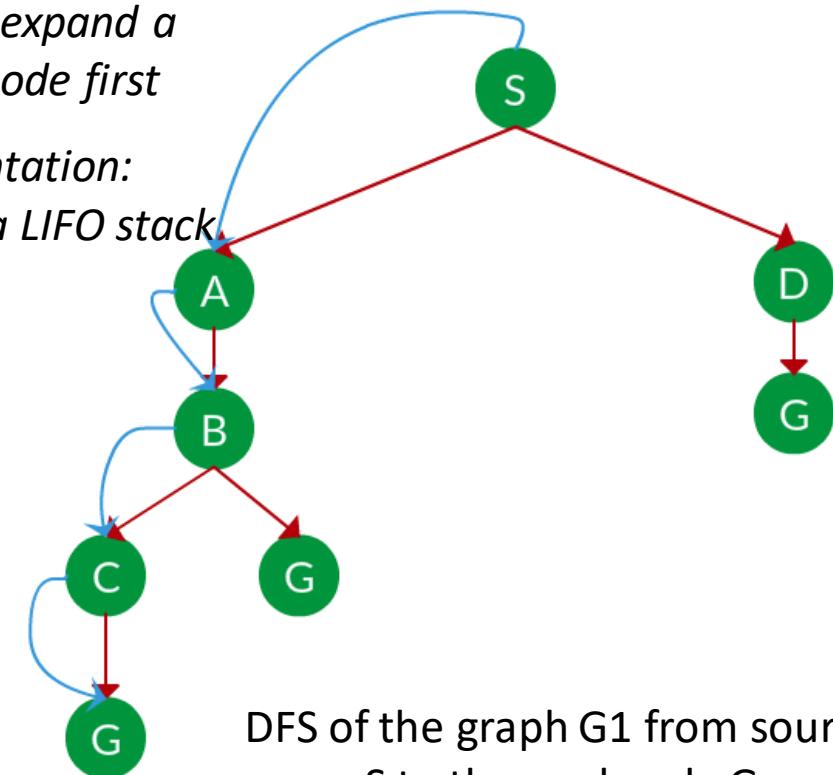
- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.



Graph G1

Strategy: expand a deepest node first

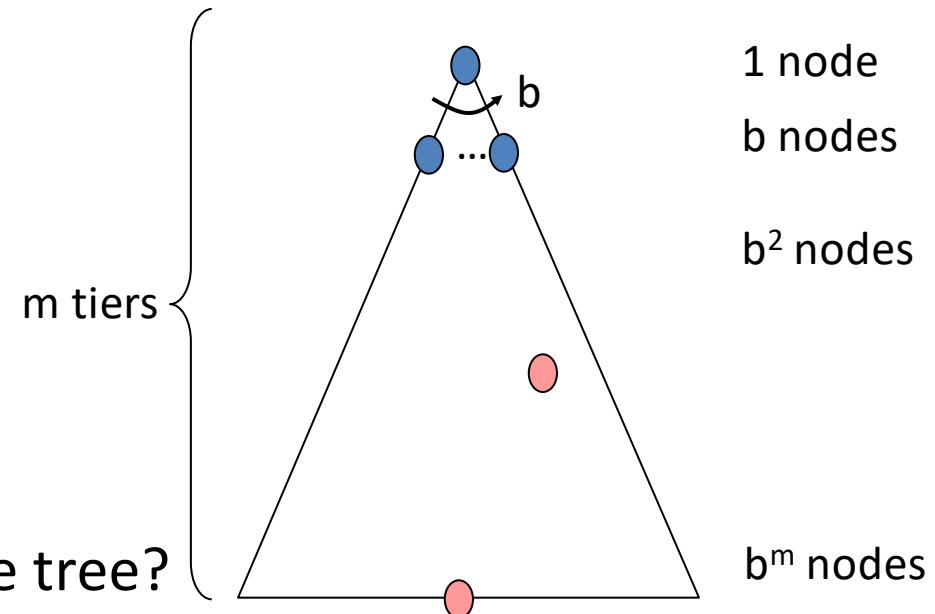
*Implementation:
Fringe is a LIFO stack*



DFS of the graph G1 from source S to the goal node G

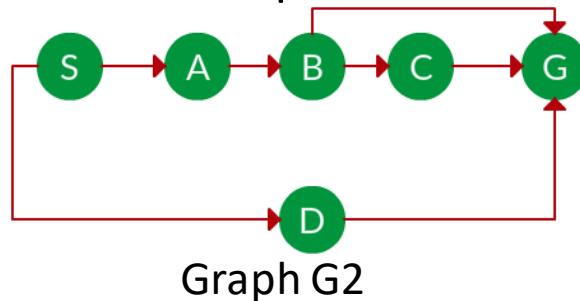
Depth First Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?
- Cartoon of search tree:
 - b is the branching factor
 - m is the maximum depth
 - solutions at various depths
- Number of nodes in entire tree?
 - $1 + b + b^2 + \dots + b^m = O(b^m)$



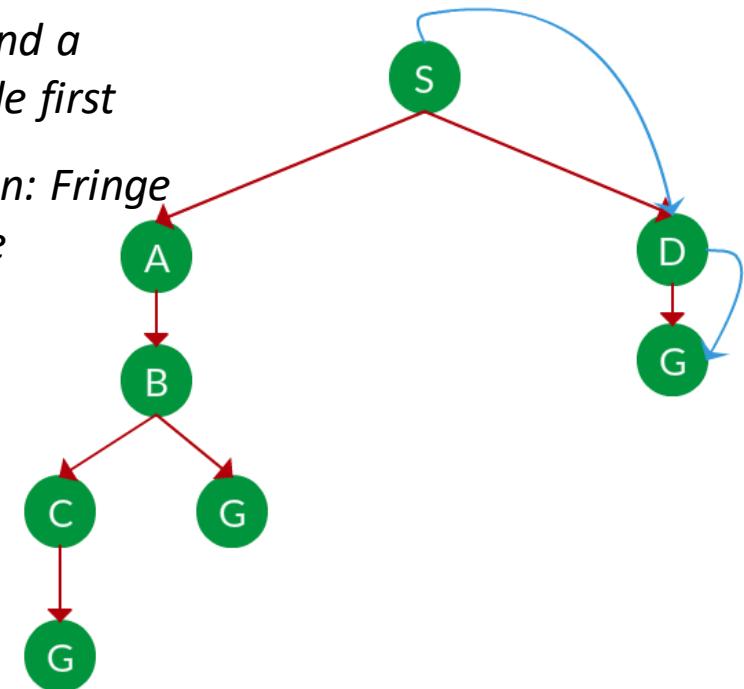
Breadth-first search (BFS)

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a ‘search key’), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.



Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

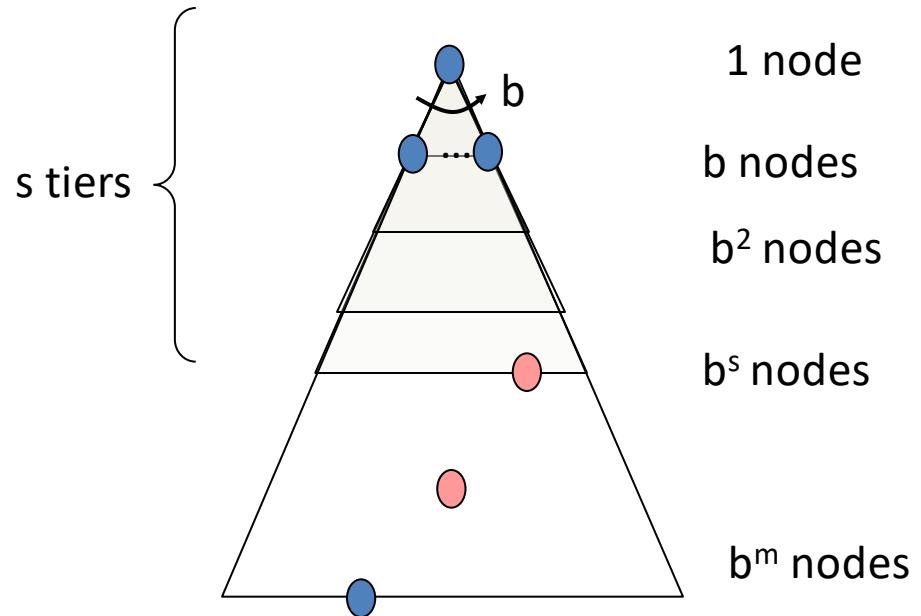


BFS of the graph G2 from source S to the goal node G

FIFO queue \$ is the rear of stack	Is front of the fringe is goal state G	Delete node at the front of fringe, expand the Node.
S\$	No	S
AD\$	No	A
DB\$	No	D
BG\$	No	B
GCG\$	Yes	G; Goal reached stop

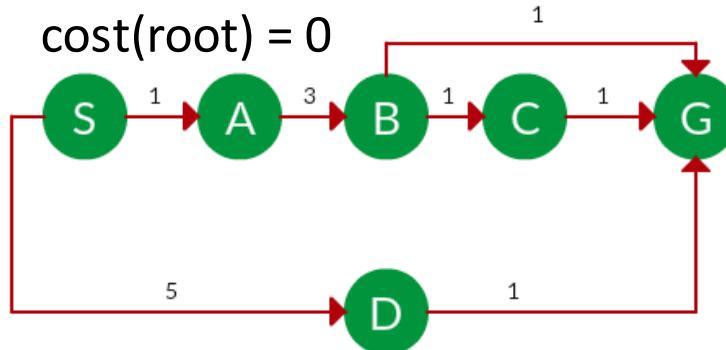
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
 - Processes all nodes above shallowest solution
 - Let depth of shallowest solution be s
 - Search takes time $O(b^s)$
- How much space does the fringe take?
 - Has roughly the last tier, so $O(b^s)$
- Is it complete?
 - s must be finite if a solution exists, so yes!
- Is it optimal?
 - Only if costs are all 1 (more on costs later)



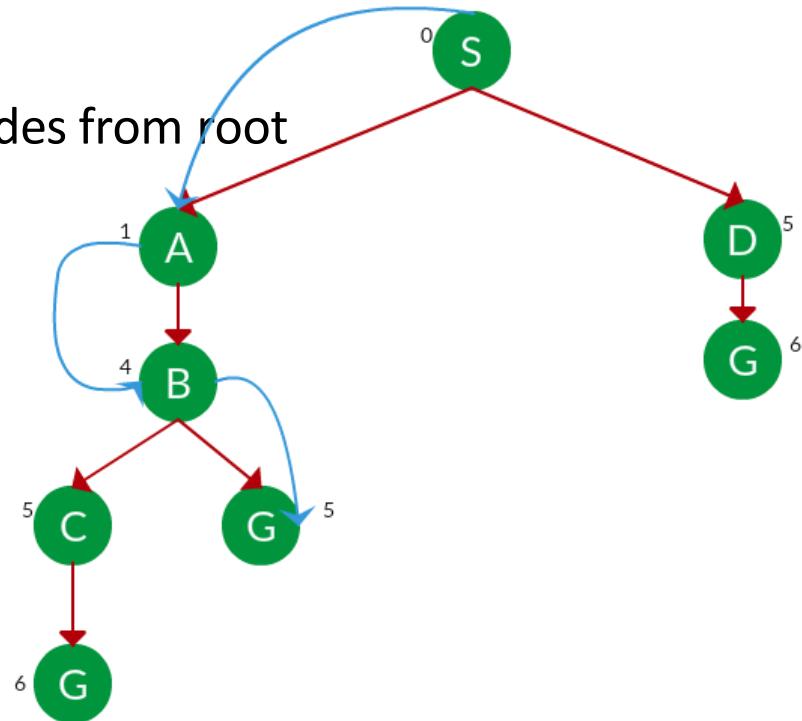
Uniform Cost Search

- UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least. Cost of each node is the cumulative cost of reaching that node from the root. Based on UCS strategy, the path with least cumulative cost is chosen. Many options can be used for fringe.
- Cost of a node** is defined as:
- $\text{cost}(\text{node}) = \text{cumulative cost of all nodes from root}$
- $\text{cost}(\text{root}) = 0$



Strategy: expand a cheapest node first:

*Fringe is a priority queue
(priority: cumulative cost)*



Uniform Cost Search

Priority queue \$ is the rear of stack	Cost(node)	Find the least cumulative cost node in fringe	Delete node with least cumulative cost in fringe, expand the Node.
S\$	$C(S)=0$	S	S
AD\$	$C(A)=1, C(D)=5$	A	A
DB\$	$C(D)=5, C(B)=4$	B	B
DCG\$	$C(D)=5, C(C) =5, C(G)= 5$	No	All nodes with same cumulative cost. Test if one of the nodes is goal node. If yes reach the goal, otherwise delete the node at the front of the queue. Since G is goal, hence G; Goal reached stop

- Limited depth DFS: just like DFS, except never go deeper than some depth d
- Iterative deepening DFS:
 - Call limited depth DFS with depth 0;
 - If unsuccessful, call with depth 1;
 - If unsuccessful, call with depth 2;
 - Etc.
- Complete, finds shallowest solution
- Space requirements of DFS
- May seem wasteful timewise because replicating effort
 - Really not that wasteful because **almost all effort at deepest level**
 - $db + (d-1)b^2 + (d-2)b^3 + \dots + 1b^d$ is $O(b^d)$ for $b > 1$

Depth-limited search algorithm

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.
- **Advantages:**
 - Depth-limited search is Memory efficient.
- **Disadvantages:**
 - Depth-limited search also has a disadvantage of incompleteness.
 - It may not be optimal if the problem has more than one solution.

Depth-limited search properties

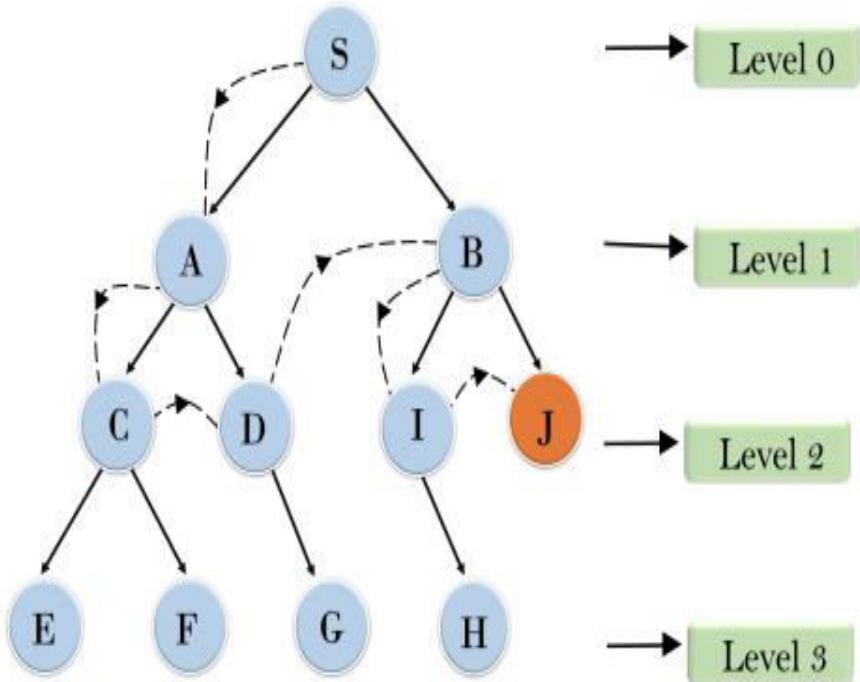
Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is $O(b^{\ell})$.

Space Complexity: Space complexity of DLS algorithm is $O(b \times \ell)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

Depth Limited Search



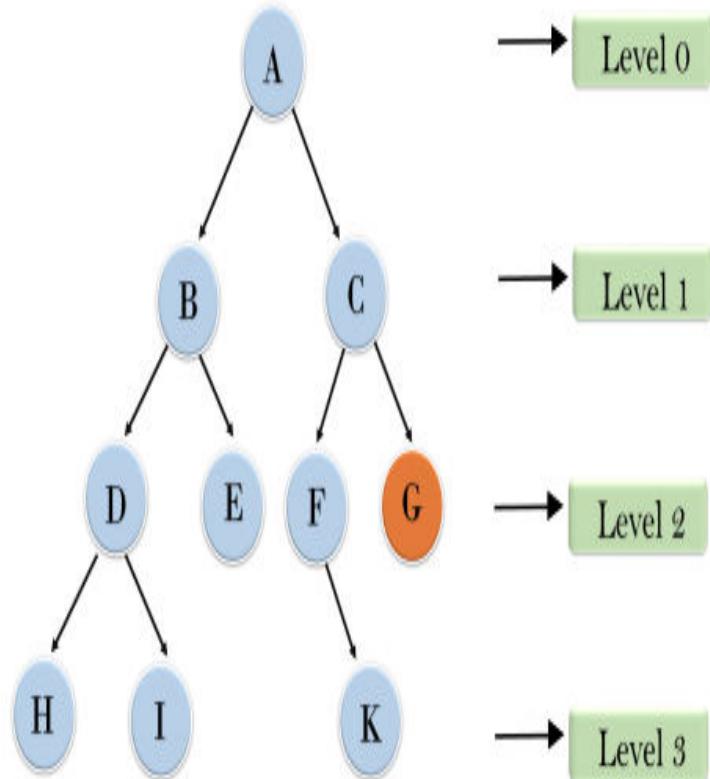
Iterative Deeping Search Algorithm

- The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.
- **Advantages:**
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.
- **Disadvantages:**
- The main drawback of IDDFS is that it repeats all the work of the previous phase.

IDDFS example & properties

- 1'st Iteration----> A
- 2'nd Iteration----> A, B, C
- 3'rd Iteration----->A, B, D, E, C, F, G
- 4'th Iteration----->A, B, D, H, I, E, C, F, K, G
- In the fourth iteration, the algorithm will find the goal node.
- Properties of IDDFS
- **Completeness:**
- This algorithm is complete if the branching factor is finite.
- **Time Complexity:**
- Let's suppose b is the branching factor and depth is d then the worst-case time complexity is $O(b^d)$.
- **Space Complexity:**
- The space complexity of IDDFS will be $O(bd)$.
- **Optimal:**
- IDDFS algorithm is optimal if path cost is a non-decreasing function of the depth of the node.

Iterative deepening depth first search

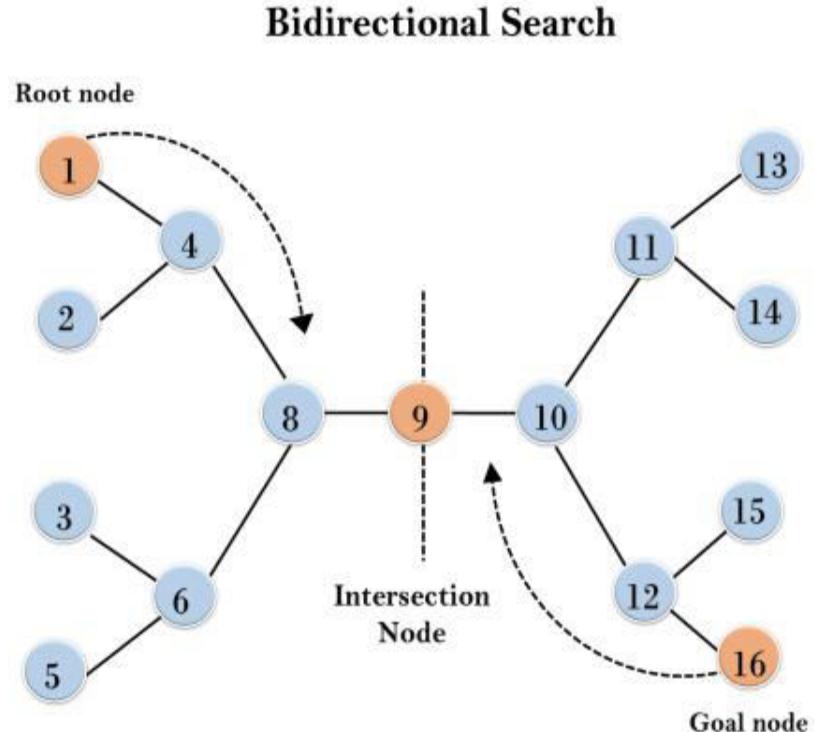


Bidirectional Search

- Bidirectional search algorithm runs two simultaneous searches, one from initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.
- **Advantages:**
 - Bidirectional search is fast.
 - Bidirectional search requires less memory
- **Disadvantages:**
 - Implementation of the bidirectional search tree is difficult.
 - In bidirectional search, one should know the goal state in advance.

Bidirectional Search

- In the example search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.
- The algorithm terminates at node 9 where two searches meet.



Properties:

Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is $O(b^d)$.

Space Complexity: Space complexity of bidirectional search is $O(b^d)$.

Optimal: Bidirectional search is Optimal.

Video Reference links

S. NO	Description	Video link
1	This video gives the knowledge of Artificial Intelligence General Search Algorithm	https://www.youtube.com/watch?v=qp2fVO-toVM
2	This video provides the information of Artificial Intelligence - Uninformed Search Algorithms (Blind Search) 1. Breadth-First Search 2. Depth-First Search 3. Uniform-Cost Search 4. Depth Limited Search 5. Iterative Deepening Depth-First Search 6. Bidirectional Search	https://www.youtube.com/watch?v=5OJv6iHMtVw

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

Students will be able to

- Understand the informed search strategies:
Best first search and A*.
- Apply the informed search techniques to solve
the problems.
- Gain the significance of Avoiding the repeated
states.

informed Search Strategies

- Informed Search algorithms have information on the goal state which helps in more efficient searching.
- This information is obtained by a function that estimates how close a state is to the goal state.
- Example: Greedy Search and Graph Search, A*
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.
- **Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.
- **Admissibility of the heuristic function is given as:** $h(n) \leq h^*(n)$ Here $h(n)$ is heuristic cost, and $h^*(n)$ is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Heuristic Search

- Maintains Two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found.
- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

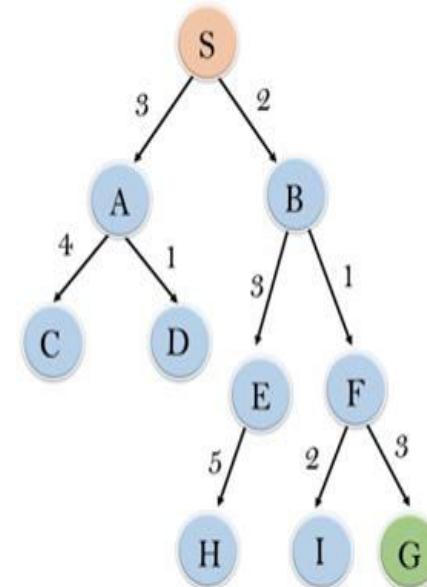
Best-first Search Algorithm (Greedy Search)

- It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search.
- Best-first search allows us to take the advantages of both algorithms.
- The algorithm expands the node which is closest to the goal node and the closest cost is estimated by heuristic function, $f(n) = h(n)$.
- Where, $h(n)$ = estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.
- Best first search algorithm:
 - **Step 1:** Place the starting node into the OPEN list.
 - **Step 2:** If the OPEN list is empty, Stop and return failure.
 - **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
 - **Step 4:** Expand the node n , and generate the successors of node n .
 - **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Best-first Search Algorithm (Greedy Search)

- Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7:** Return to Step 2.
- In the example S-start G-Goal

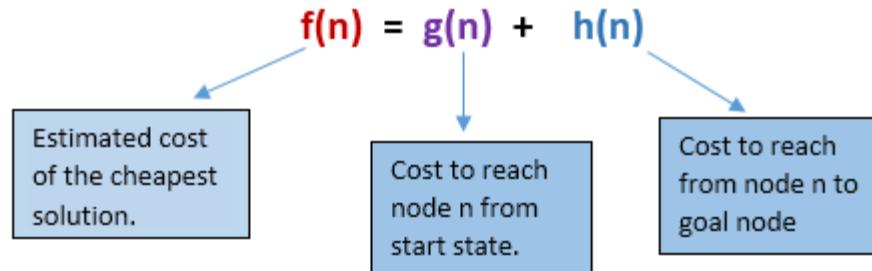
Step	OPEN	$h(n)$ value of nodes in OPEN	CLOSE
1	S	$h(S)=13$	S
2	A B	$h(A)=12, h(B)=4$	S, B
3	A E F	$h(A)=12, h(E)=8$ $h(F)=2$	S, B, F
4	A E I G	$h(A)=12, h(E)=8$ $h(I)=9$ $h(G)=0$	S, B, F, G. G is goal stop



node	$H(n)$
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

A* Search Algorithm:

- A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node n from the start state $g(n)$.

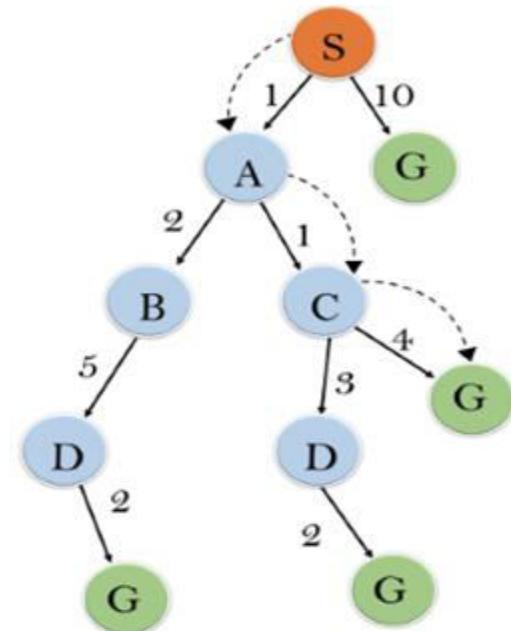
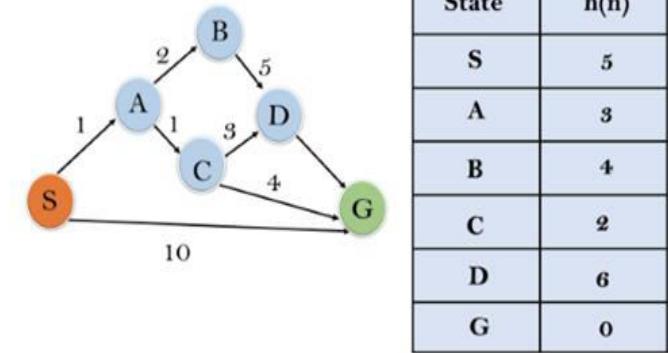


- Algorithm of A* search:
- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function ($g+h$), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

A* Search Algorithm:

- Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.
- Step 6:** Return to Step 2.

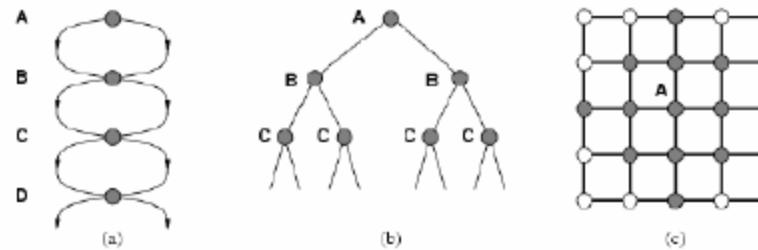
Step	OPEN	$f(n) = g(n)+h(n)$	Successor(n); Where n is best compromised node	CLOSE
1	S	$f(S)=g(S)+h(S) =0+5=5$	$\text{Succ}(S)=\{A, G\}$	S
2	A G	$f(A)=1+3=4$ $f(G)=10+0=10$	$\text{Succ}(A)=\{B, C\}$	A
3	G B C	$f(G)=10+0=10$ $f(B)=3+4=7$ $f(C)=2+2=4$	$\text{Succ}(C)=\{D, G\}$ Attach back pointer to G, it is lowest value.	C
4	G B D	$f(G)=6+0=6$ $f(B)=3+4=7$ $f(D)=5+6=11$	$\text{Succ}(G) = \{\}$	G, Stop as goal is reached



Avoiding repeated states

Problems With Repeated States

- Tree search ignores what happens if nodes are repeatedly visited
 - For example, if actions lead back to already visited states
 - Consider path planning on a grid
- Repeated states may lead to a large (exponential) overhead



- (a) State space with $d+1$ states, where d is the depth
- (b) The corresponding search tree which has 2^d nodes corresponding to the two possible paths!
- (c) Possible paths leading to A

Video Reference links

S. NO	Description	Video link
1	This video provides the description of A* and IDA*	https://www.youtube.com/watch?v=HhDhFsA3aro
2	This video describes the informed search algorithms.	https://www.youtube.com/watch?v=5LMXQ1NGHwU 48min
3	This video describes the iterative search algorithms.	https://www.youtube.com/watch?v=IWt6YX2pQvM 9min

Thank you

UNIT - 2

PROBLEM SOLVING

Dr Devaraj Verma C

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Problem-solving

- **Problem-solving:**

Solving Problems by Searching, Problem-Solving Agents, Formulating Problems, Well-defined problems and solutions, Measuring problem-solving performance, Toy problems, Searching for Solutions, Search Strategies, Avoiding Repeated States, Constraint Satisfaction Search, Informed Search Methods, Best-First Search, Heuristic Functions, Memory Bounded Search, Iterative Improvement Algorithms, Applications in constraint satisfaction problems.

Objectives

Students will be able to

- Understand the informed search strategies.
- Apply the informed search techniques to solve the problems.

informed Search Strategies

- Informed Search algorithms have information on the goal state which helps in more efficient searching.
- This information is obtained by a function that estimates how close a state is to the goal state.
- Example: Greedy Search and Graph Search, A*
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.
- **Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.
- **Admissibility of the heuristic function is given as:** $h(n) \leq h^*(n)$ Here $h(n)$ is heuristic cost, and $h^*(n)$ is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Heuristic Search

- Maintains Two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found.
- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

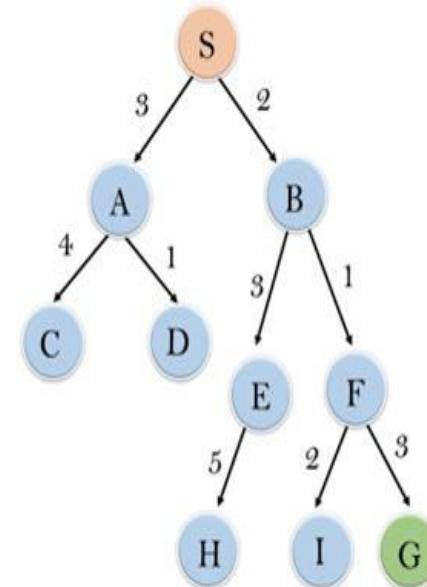
Best-first Search Algorithm (Greedy Search)

- It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search.
- Best-first search allows us to take the advantages of both algorithms.
- The algorithm expands the node which is closest to the goal node and the closest cost is estimated by heuristic function, $f(n) = h(n)$.
- Where, $h(n)$ = estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.
- Best first search algorithm:
 - **Step 1:** Place the starting node into the OPEN list.
 - **Step 2:** If the OPEN list is empty, Stop and return failure.
 - **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
 - **Step 4:** Expand the node n , and generate the successors of node n .
 - **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Best-first Search Algorithm (Greedy Search)

- Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7:** Return to Step 2.
- In the example S-start G-Goal

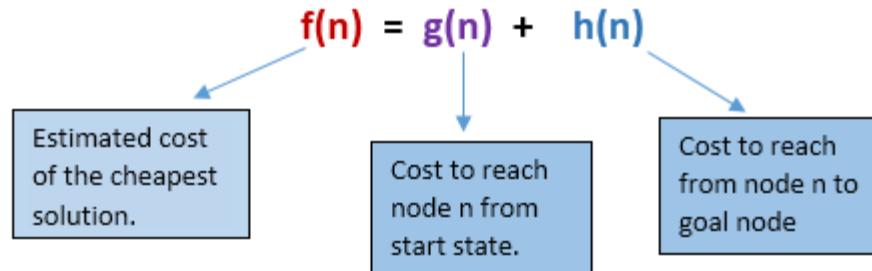
Step	OPEN	$h(n)$ value of nodes in OPEN	CLOSE
1	S	$h(S)=13$	S
2	A B	$h(A)=12, h(B)=4$	S, B
3	A E F	$h(A)=12, h(E)=8$ $h(F)=2$	S, B, F
4	A E I G	$h(A)=12, h(E)=8$ $h(I)=9$ $h(G)=0$	S, B, F, G. G is goal stop



node	$H(n)$
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

A* Search Algorithm:

- A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node n from the start state $g(n)$.

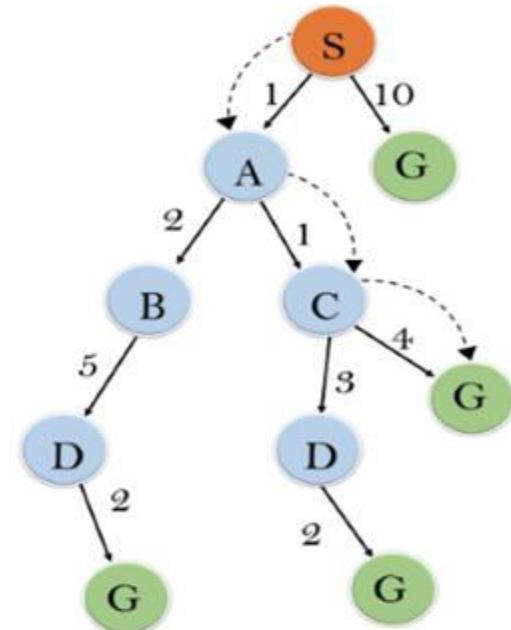
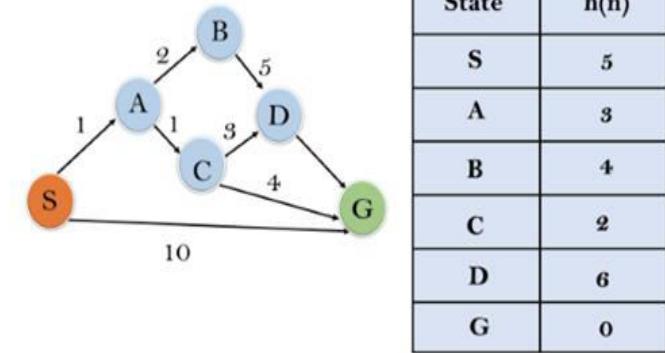


- Algorithm of A* search:
- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function ($g+h$), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

A* Search Algorithm:

- Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.
- Step 6:** Return to Step 2.

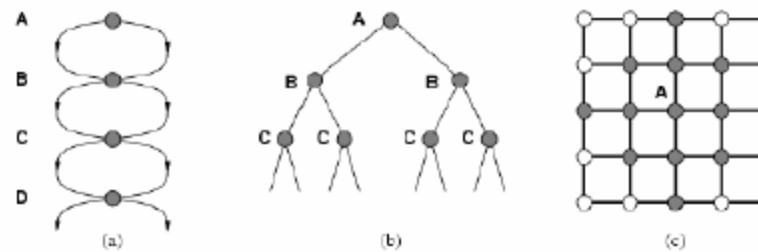
Step	OPEN	$f(n) = g(n)+h(n)$	Successor(n); Where n is best compromised node	CLOSE
1	S	$f(S)=g(S)+h(S) =0+5=5$	$\text{Succ}(S)=\{A, G\}$	S
2	A G	$f(A)=1+3=4$ $f(G)=10+0=10$	$\text{Succ}(A)=\{B, C\}$	A
3	G B C	$f(G)=10+0=10$ $f(B)=3+4=7$ $f(C)=2+2=4$	$\text{Succ}(C)=\{D, G\}$ Attach back pointer to G, it is lowest value.	C
4	G B D	$f(G)=6+0=6$ $f(B)=3+4=7$ $f(D)=5+6=11$	$\text{Succ}(G) = \{\}$	G, Stop as goal is reached



Avoiding repeated states

Problems With Repeated States

- Tree search ignores what happens if nodes are repeatedly visited
 - For example, if actions lead back to already visited states
 - Consider path planning on a grid
- Repeated states may lead to a large (exponential) overhead

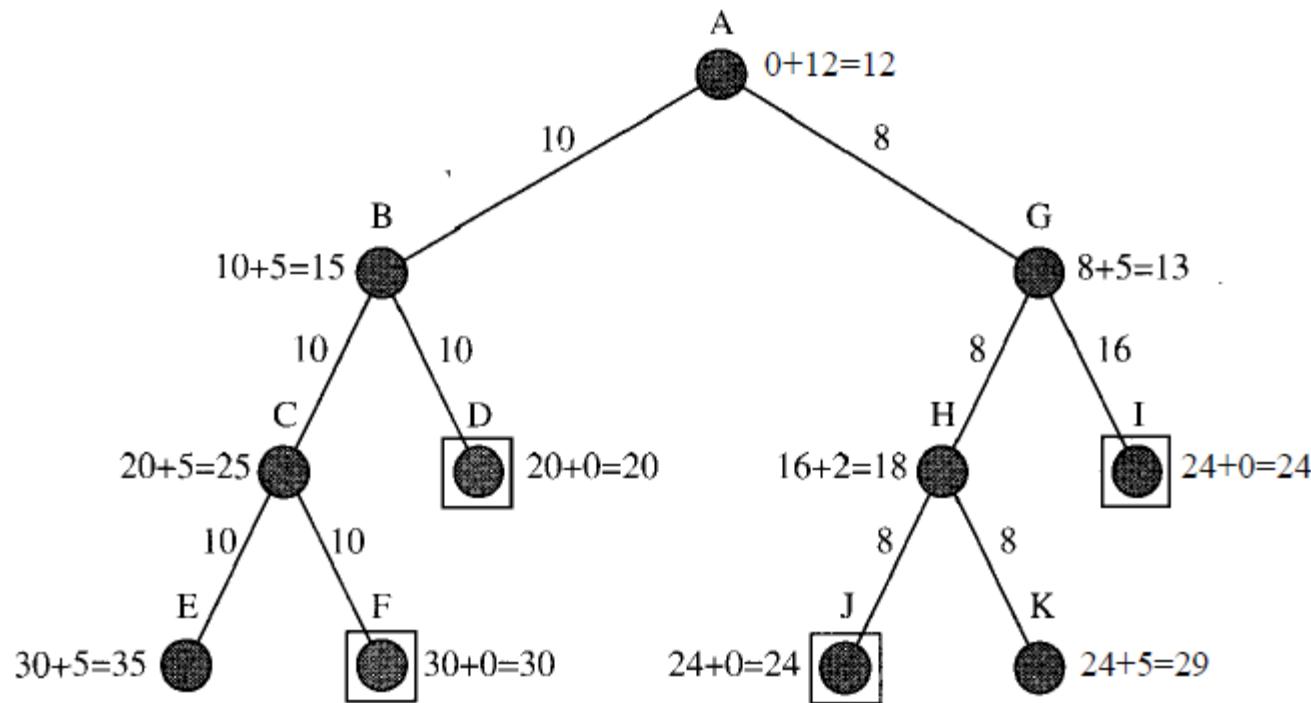


- (a) State space with $d+1$ states, where d is the depth
- (b) The corresponding search tree which has 2^d nodes corresponding to the two possible paths!
- (c) Possible paths leading to A

Memory Bounded Search Algorithms

- The first, **IDA***, is a logical extension of ITERATIVE-DEEPENING-SEARCH to use heuristic information.
- The second,(**Simplified Memory A***) SMA*, is similar to A*, but restricts the queue size to fit into the available memory.
- **Iterative Deepening A* - IDA***
- Similar to the iteratively deepening search.
- The limit for expanding a state is not given by the depth but by the f-value.
- It starts with the limit being the f-value of the initial state.
- If the goal is not found, the limit is increased by a constant or to the lowest f-value found in the previous search that exceeds the previous limit, and the depth-first search restarts.
- It can require more time than A*, but they are asymptotically similar. IDA* needs a lot less memory.
- Apply the IDA* algorithm for the graph in the next slide.

Iterative Deepening A* - IDA*



1. Start with A, $f(A) = 12$. Open={A}. Close={}
2. Open={B, G}, Close={A}, min f-score of Open is $f(G) = 13$
3. Open={B, H, I}, close={A, G}, min f-score is

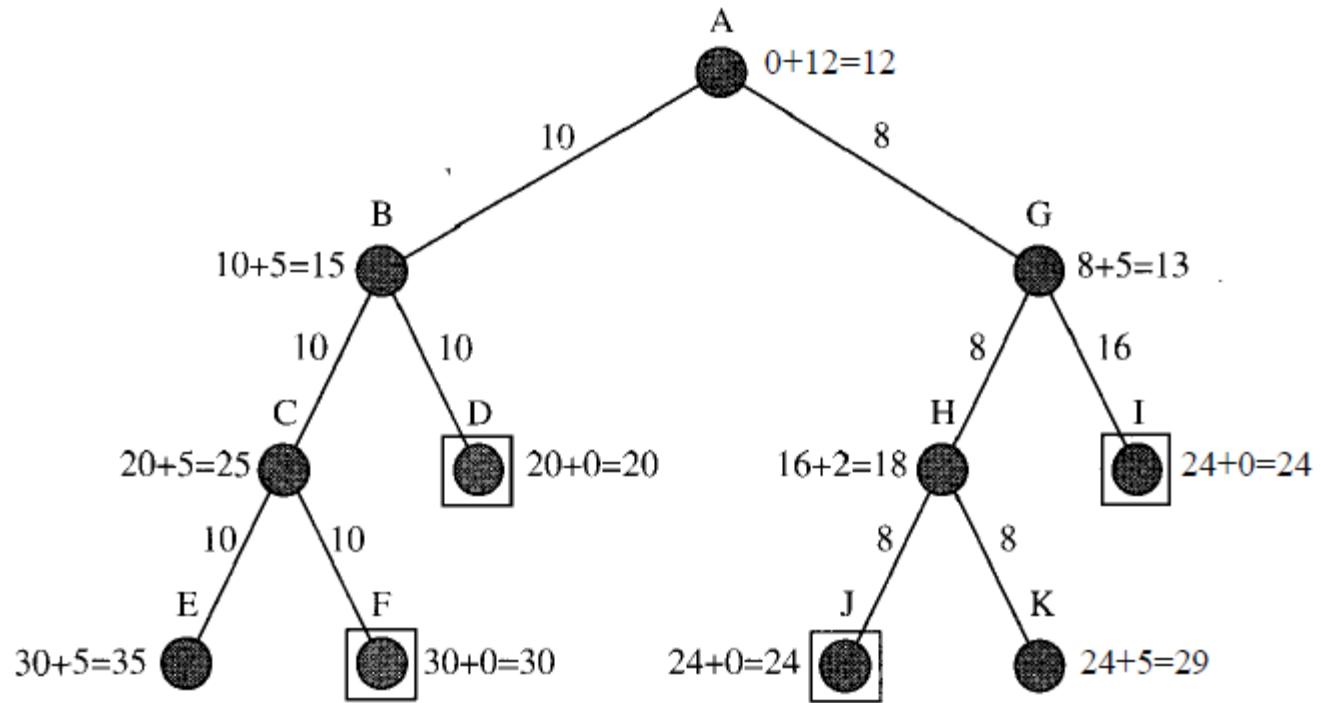
Simplified Memory A*- SMA*

- Utilization of allocated memory efficiently – No memory leak.
- Avoids the repeated states.
- It is complete if available memory is sufficient enough to store the shallowest path solution.
- It is optimal if enough memory is available to store the shallowest optimal solution path.
- Otherwise, it returns the best solution that can be reached with the available memory.
- When enough memory is available for search, then it is optimally efficient.
- When it needs to generate a successor but has no memory left, it will need to make space on the queue. To do this, it drops a node from the queue. Nodes that are dropped from the queue in this way are called **forgotten nodes**. It prefers to drop nodes that are unpromising—that is, nodes with highest cost.
- Example in figure Each node is labeled with $g + h — f$ values, and the goal nodes (D, F, I, J) are shown in squares. **The aim is to find the lowest-cost goal node with enough memory for only three nodes.**

Example – graph; with memory limit for 3 nodes

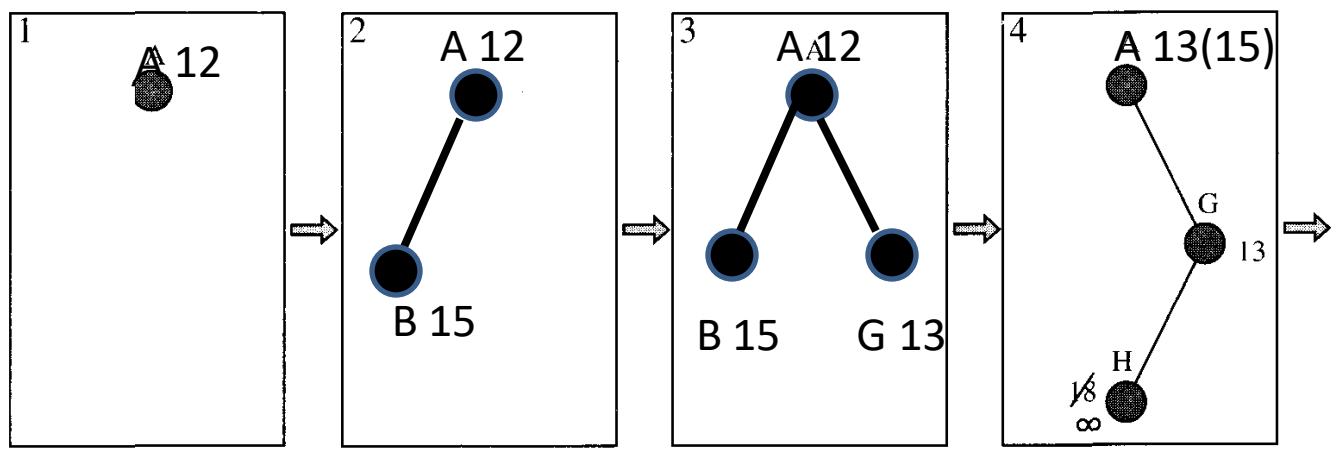
Node n	$h(n)$
A	12
B	5
C	5
D	0
E	5
F	0
G	5
H	2
I	0
J	0
K	5

f-score of node $f(n) = g(n) + h(n)$



SMA*; with memory limit for 3 nodes

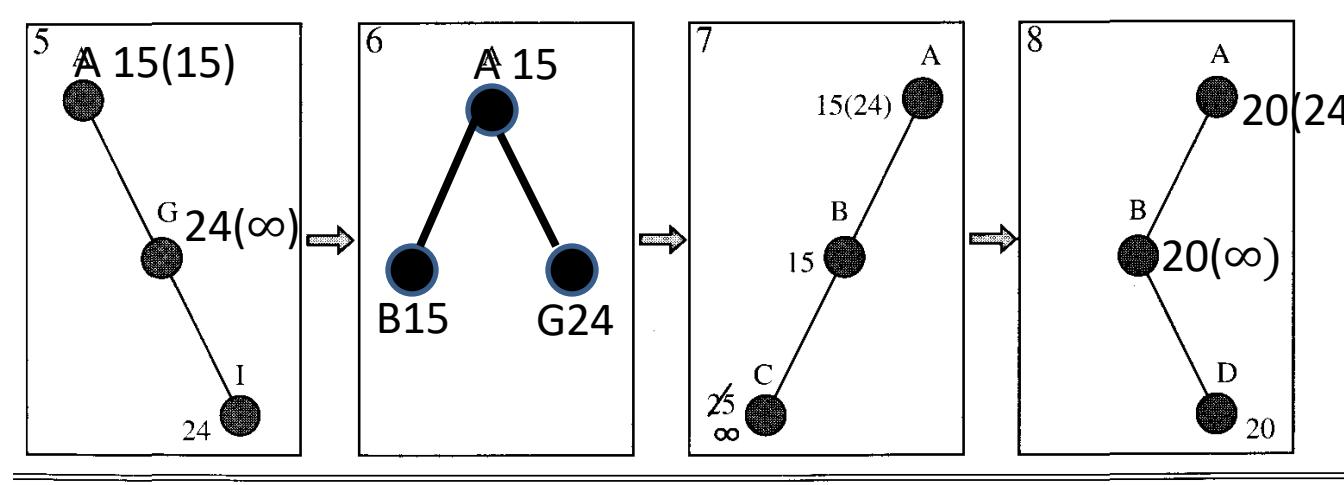
1. Start with root node A, f-score 12. add to queue Q with f-score.
2. At each stage, one successor is added to the deepest lowest cost node that has some successors not currently in the tree. The left child B is added to the root A.



3. Children of A have f-score values 15 & 13.
4. The minimum child f-score value replaced for f-score of parent node i,e A. f-score for B is larger, the node B is forgotten and its f-score written in parenthesis.

SMA*; with memory limit for 3 nodes

5. $f(A)$ becomes 15, the minimum of 15 (forgotten successor value) and 24. Notice that I is a goal node, but it might not be the best solution because A's/-cost is only 15.
6. A is once again the most promising node, so B is generated for the second time. We have found that the path through G was not so great after all. Hence G forgotten.
7. C, the first successor of B, is a non-goal node at the maximum depth and $f(C) = \infty$. Hence forgotten
8. To look at the second successor, D, we first drop C. Then $f(D) = 20$, and this value is inherited by B and A.



9. Now the deepest, lowest-/-cost node is D. D is therefore selected, and because it is a goal node, the search terminates.

Iterative improvement Algorithms

1. Also called as optimal algorithms. The idea of iterative improvement is to move around the landscape trying to find the **highest peaks**, which are the optimal solutions.
2. Iterative improvement algorithms usually keep track of only the **current state**, and **do not look ahead beyond** the immediate neighbors of that state.
3. For some problems path to the solution is irrelevant; just want path to the solution.
4. Start with the initial state and change it iteratively to improve it. (find a “best” or a “minimum” value.)
5. Examples: a) Finding the optimal way of assigning the dates within n people. B) Traveling Sales Man problem c) knapsack problem
6. Iterative improvement algorithms divide into two major classes.
7. **Hill-climbing (or, alternatively, gradient descent if we view the evaluation function as a cost rather than a quality)** algorithms always try to make changes that improve the current state.
8. **Simulated annealing** algorithms can sometimes make changes that make things worse, at least temporarily.

Hill climbing Algorithm

1. Simple Hill climbing

: It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.

```

function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
          next, a node

  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    next  $\leftarrow$  a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current  $\leftarrow$  next
  end

```

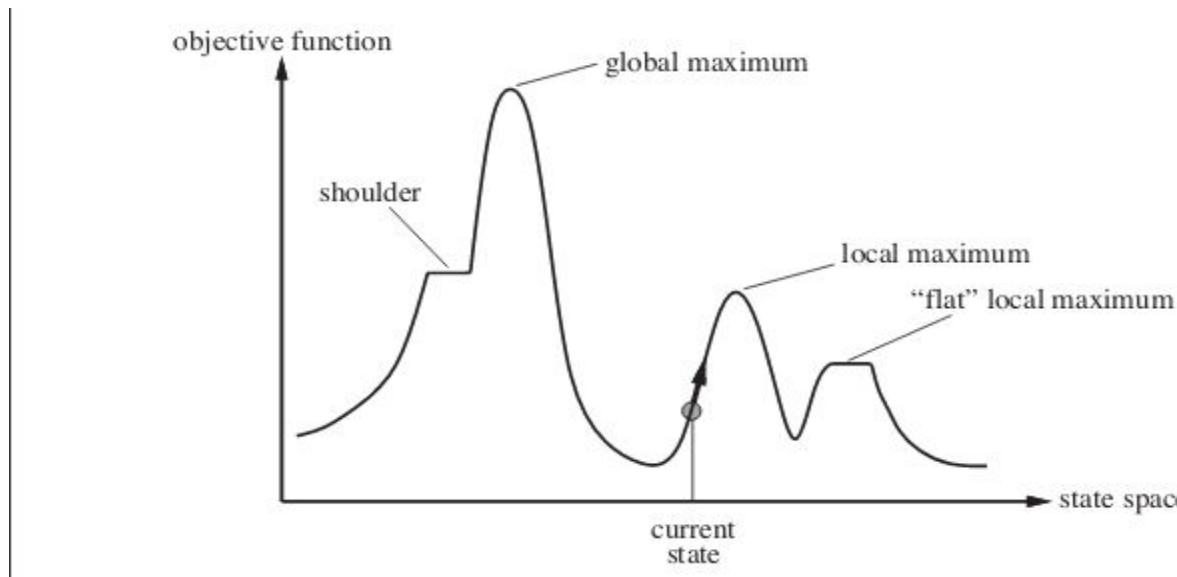
Hill climbing Algorithm

- *Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.*
- *Step 2 : Loop until the solution state is found or there are no new operators present which can be applied to the current state.*
- *a) Select a state that has not been yet applied to the current state and apply it to produce a new state.*
- *b) Perform these to evaluate new state*
 - i. If the current state is a goal state, then stop and return success.*
 - ii. If it is better than the current state, then make it current state and proceed further.*
 - iii. If it is not better than the current state, then continue in the loop until a solution is found.*

Step 3 : Exit.

State Space diagram for Hill Climbing

- State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).
- **X-axis** : denotes the state space ie states or configuration our algorithm may reach.
- **Y-axis** : denotes the values of objective function corresponding to a particular state.
- The best solution will be that state space where objective function has maximum value(global maximum).

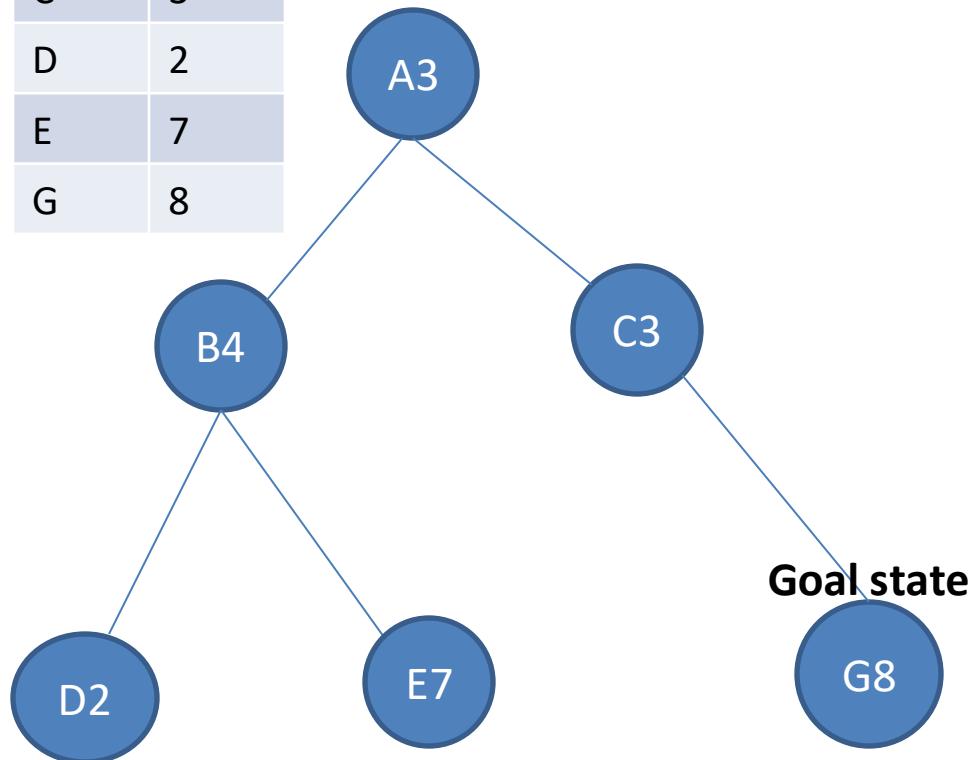


Different regions in the State Space Diagram

- **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
- **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
- **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
- **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
- **Current state :** The region of state space diagram where we are currently present during the search.
- **Shoulder :** It is a plateau that has an uphill edge.

Example of Hill climbing

Node N	$h(N)$
A	3
B	4
C	3
D	2
E	7
G	8



Step	Open	Close
1	[]	[]
2	[A3]	[]
3	[B4, C3]	[A3]
4	[E7, D2, C3]	[A3, B4]
5	[D2, C3]	[A3, B4, E7]
6	[C3]	[A3, B4, E7, D2]
7	[G8]	[A3, B4, E7, D2, C3]
8	[]	[A3, B4, E7, D2, C3, G8]

Path cost: A-B-E-D-C-G = 5

Choose the best adjacent node N based on the heuristic value $h(N)$;
evaluation function is $\max(h(N))$

Simulated Annealing:

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum.
- And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.
- **Simulated Annealing** is an algorithm which yields both efficiency and completeness.
- In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state.
- The physical property of the metal changes because the internal structure changes while cooling the metal gradually.
- The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move.
- If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.
- Solution states in a problem \Leftrightarrow states in a physical system
- Cost of a solution \Leftrightarrow energy of a state

Simulated Annealing Algorithm

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to "temperature"

static: *current*, a node

next, a node

T, a "temperature" controlling the probability of downward steps

current \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

for *t* \leftarrow 1 to ∞ **do**

T \leftarrow *schedule*[*t*]

if *T*=0 **then return** *current*

next \leftarrow a randomly selected successor of *current*

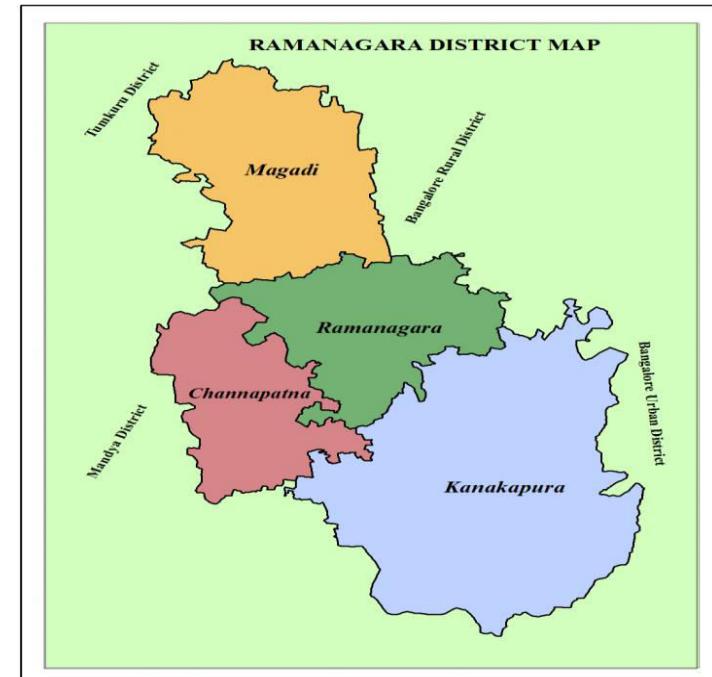
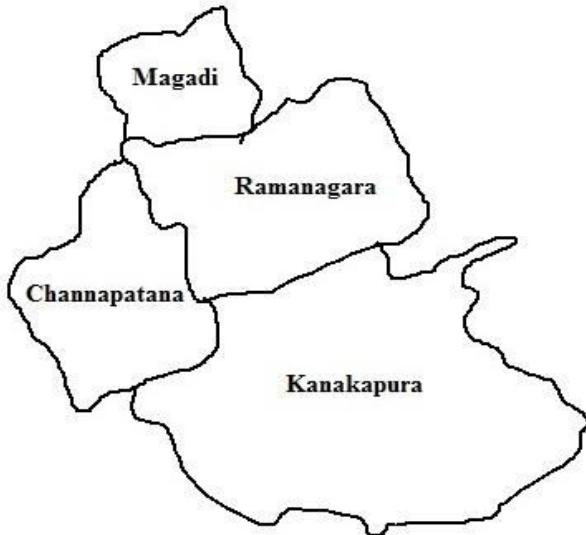
$\Delta E \leftarrow$ VALUE[*next*] - VALUE[*current*]

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

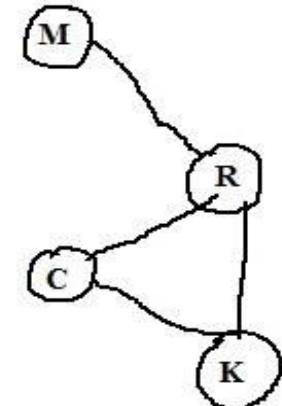
constraint satisfaction problems

- A **constraint satisfaction problem (CSP)** is a problem that requires its solution within some limitations or conditions also known as constraints. It consists of the following:
- A finite set of **variables** which stores the solution ($V = \{V_1, V_2, V_3, \dots, V_n\}$)
- A set of **discrete** values known as **domain** from which the solution is picked ($D = \{D_1, D_2, D_3, \dots, D_n\}$)
- A finite set of **constraints** ($C = \{C_1, C_2, C_3, \dots, C_n\}$)
- **Example Map Coloring Problem**
- **Constraint: Each city colored differently**



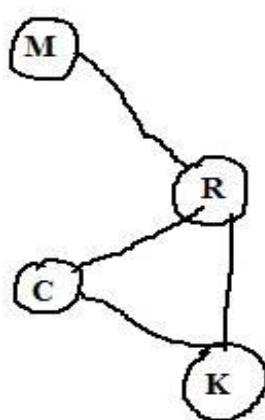
Varieties of constraint

- Unary constraint involve single variable.
eg.: Magadi # Green
- Binary constraint involve pair of variables.
eg.: Magadi # Ramanagara
- Higher order Constraint involve more than three or more variables.
Eg. Cryptarithmetic column constraints
- Preferences (soft constraints) eg. red is better than green often represent-able by a cost for each variable assignment
- Constraint Graph – Nodes are the variables; arcs shows the constraints between the nodes.
- M- Magadi; R-Ramanagar; C- chennapatana
- K- Kanakapura
- **Constraint=** No two adjacent cities have the same color.



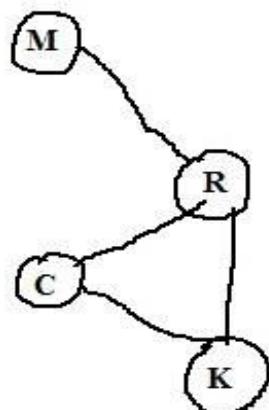
Map coloring problem

- Consider the map in the previous slide color the cities in the map with the four color {yellow, Green, pink, blue}. Assume the constraints as given
- A) all cities are colored differently
- B) No two adjacent cities colored with the same color
- A- solution**
- Variable set V = { M,C, R, K }**
- Domain set D = {D1, D2, D3, D4}**
- Initially each Di = {Y=yellow, G=Green, P=pink, B=blue} where i=1, 2, 3,4.**
- Constraint set C = { Color(Ci)# Color(Cj) / For any i # j; i and j are any two cities}**



Step	M	C	R	K
Initial Step	D1={Y,G,P,B}	D2={Y, G, P, B}	D3={Y, G, P, B}	D4={Y,G,P, B}
1	Y	D2 = {G, P, B}	D3 = {G, P, B}	D4 = {G, P, B}
2	Y	G	D3 = {P, B}	D4 = {P, B}
3	Y	G	P	D4 = {B}
4	Y	G	P	B

- **B- solution**
- **Variable set V = { M,C, R, K}**
- **Domain set D = {D1, D2, D3, D4}**
- **Initially each Di = {Y=yellow, G=Green, P=pink, B=blue} where i=1, 2, 3,4.**
- **Constraint set C = { Color(Ci)# Color(Cj) / For any i # j; i and j are any two adjacent cities}**



Step	M	C	R	K
Initial Step	D1={Y,G,P,B}	D2={Y, G, P, B}	D3={Y, G, P, B}	D4={Y,G,P, B}
1	Y	D2 = {Y,G, P, B}	D3 = {G, P, B}	D4 = {Y,G,P,B}
2	Y	Y	D3 = {G,P, B}	D4 = {G,P, B}
3	Y	Y	G	D4 = {P,B}
4	Y	Y	G	P OR
5	Y	Y	G	B

Video Reference links

S. NO	Description	Video link
1	This video provides the description of A* and IDA*	https://www.youtube.com/watch?v=HhDhFsA3aro
2	This video describes the SMA* algorithm.	https://www.youtube.com/watch?v=2tmXWpVHW2w
3	This video describes the informed search algorithms.	https://www.youtube.com/watch?v=5LMXQ1NGHwU 48min
4	This video describes the iterative search algorithms.	https://www.youtube.com/watch?v=lWt6YX2pQvM 9min
5	This video demonstrate the concept of hill climbing.	https://www.youtube.com/watch?v=SSdnuu6LbD0
6	This video provides the knowledge of simulated Annealing Algorithm.	https://www.youtube.com/watch?v=RX2oS3hTXyM https://www.youtube.com/watch?v=jR9RtOBHAAo

Thank you

Agents that Reason Logically (UNIT-3)

Knowledge-based agents
Propositional logic

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

Some material adopted from notes
by Andreas Geyer-Schulz
and Chuck Dyer

UNIT-3

- **Knowledge and reasoning:** (2 Hours)
A Knowledge-Based Agent, Representation, Reasoning, and Logic, Prepositional Logic, An Agent for the Wumpus World, Problems with the propositional agent, First-Order Logic, Syntax and Semantics, Extensions and Notational Variations. Chap 6

The Wumpus World environment

- ▶ The Wumpus computer game
- ▶ The agent explores a cave consisting of rooms connected by passageways.
- ▶ Lurking somewhere in the cave is **the Wumpus**, a beast that eats any agent that enters its room.
- ▶ Some rooms contain **bottomless pits** that *trap any agent* that wanders into the room.
- ▶ Occasionally, there is a **heap of gold** in a room.
- ▶ The goal is:
 - to collect the gold and
 - exit the world
 - without being eaten

Jargon file on “Hunt the Wumpus”

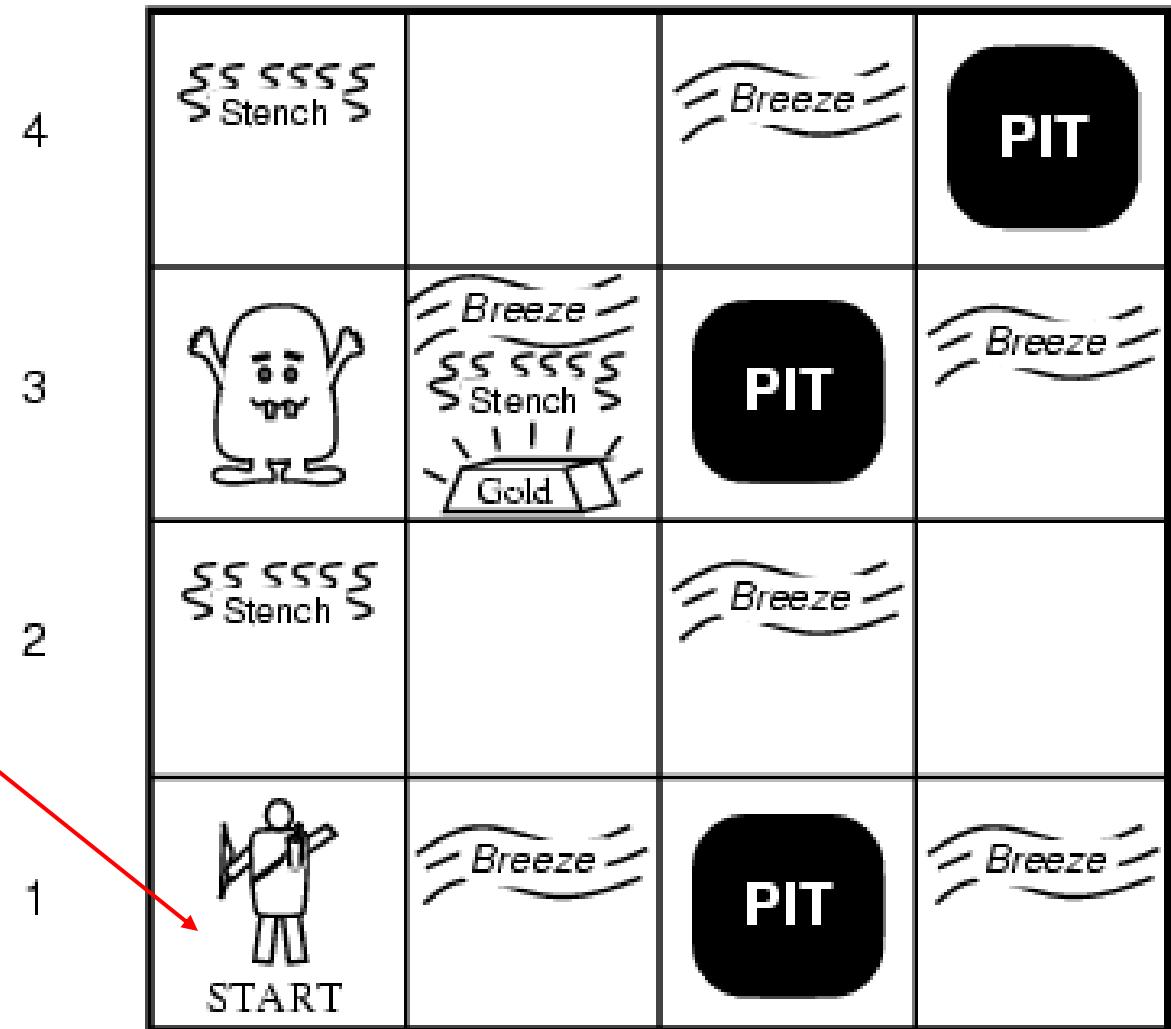
- **WUMPUS** /wuhm'p*s/ n. The central monster (and, in many versions, the name) of a famous family of very early computer games called “Hunt The Wumpus,” dating back at least to 1972
- The wumpus lived somewhere in a cave with the topology of a dodecahedron's edge/vertex graph
 - (later versions supported other topologies, including an icosahedron and Möbius strip).
- The player started somewhere at random in the cave with five “crooked arrows”;
 - these could be shot through up to three connected rooms, and would kill the wumpus on a hit
 - (later versions introduced the wounded wumpus, which got very angry).

Jargon file on “Hunt the Wumpus” (cont)

- ▶ Unfortunately for players, the movement necessary to map the maze was made **hazardous** not merely by the wumpus
 - (which would eat you if you stepped on him)
- ▶ There are also **bottomless pits** and **colonies of super bats** that would pick you up and drop you at a random location
 - (later versions added “anaerobic termites” that ate arrows, bat migrations, and earthquakes that randomly change pit locations).
- ▶ This game appears to have been the first to use a **non-random graph-structured map** (as opposed to a rectangular grid like the even older Star Trek games).
- ▶ In this respect, as in the dungeon-like setting and its terse, amusing messages, it prefigured **ADVENT** and **Zork**.
- ▶ It was directly ancestral to both.
 - (Zork acknowledged this heritage by including a super-bat colony.)
 - Today, a port is distributed with SunOS and as freeware for the Mac.
 - A C emulation of the original Basic game is in circulation as freeware on the net.

A typical Wumpus world

- The agent always starts in the field [1,1].
- The task of the agent is to find the gold, return to the field [1,1] and climb out of the cave.



Agent in a Wumpus world: Percepts

► The agent perceives

- a **stench** in the square containing the wumpus and in the adjacent squares (not diagonally)
- a **breeze** in the squares adjacent to a pit
- a **glitter** in the square where the gold is
- a **bump**, if it walks into a wall
- a **woeful scream** everywhere in the cave, if the wumpus is killed

► The percepts will be given as a **five-symbol list**:

- If there is a stench, and a breeze, but no glitter, no bump, and no scream, the percept is
[Stench, Breeze, None, None, None]

► The agent can not perceive its own location.

The actions of the agent in Wumpus game

- ▶ **go forward**
- ▶ **turn right** 90 degrees
- ▶ **turn left** 90 degrees
- ▶ **grab** means pick up an object that is in the same square as the agent
- ▶ **shoot** means fire an arrow in a straight line in the direction the agent is looking.
 - The arrow continues until it either hits and kills the wumpus or hits the wall.
 - The agent has only one arrow.
 - Only the first shot has any effect.
- ▶ **climb** is used to leave the cave.
 - Only effective in start field.
- ▶ **die**, if the agent enters a square with a pit or a live wumpus.
 - (No take-backs!)

The agent's goal

The agent's goal is to **find the gold** and **bring it back to the start** as quickly as possible, **without getting killed**.

- (+1000) points reward for climbing out of the cave with the gold
- (-1)point deducted for every action taken
- (-10000) points penalty for getting killed

The Wumpus agent's first step

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A OK	2,1 OK	3,1	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

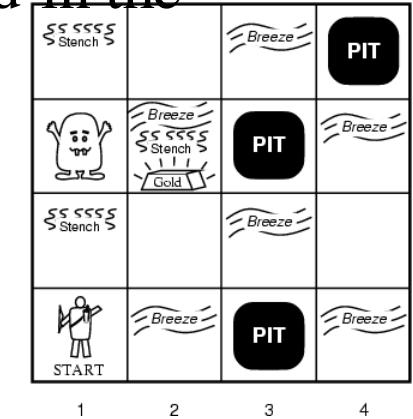
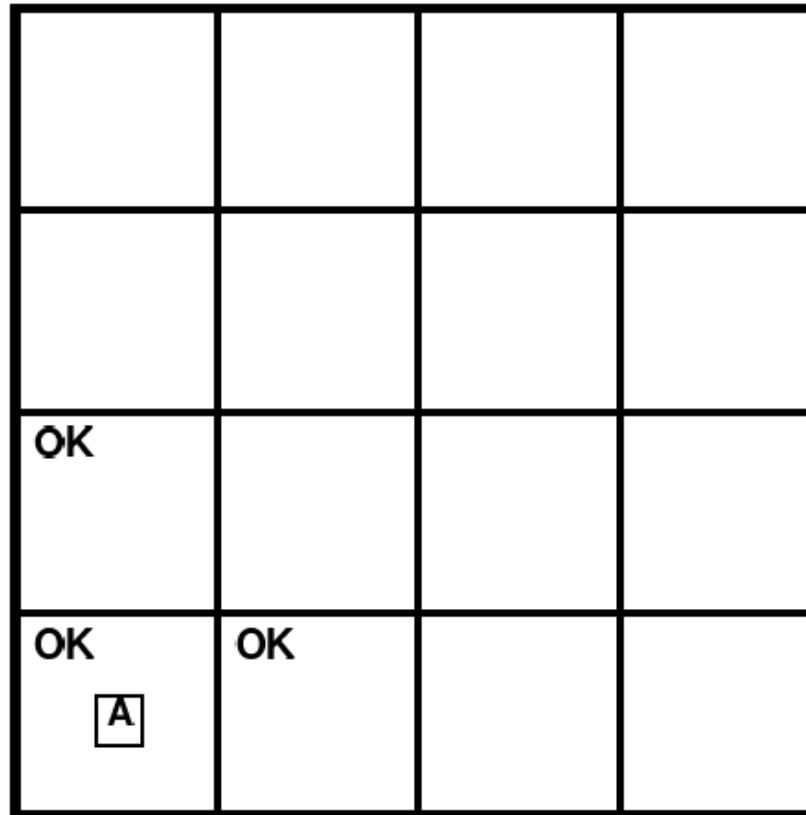
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P? OK	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

Exploring a wumpus world

a **stench** in the square containing the wumpus and in the adjacent squares (not diagonally)

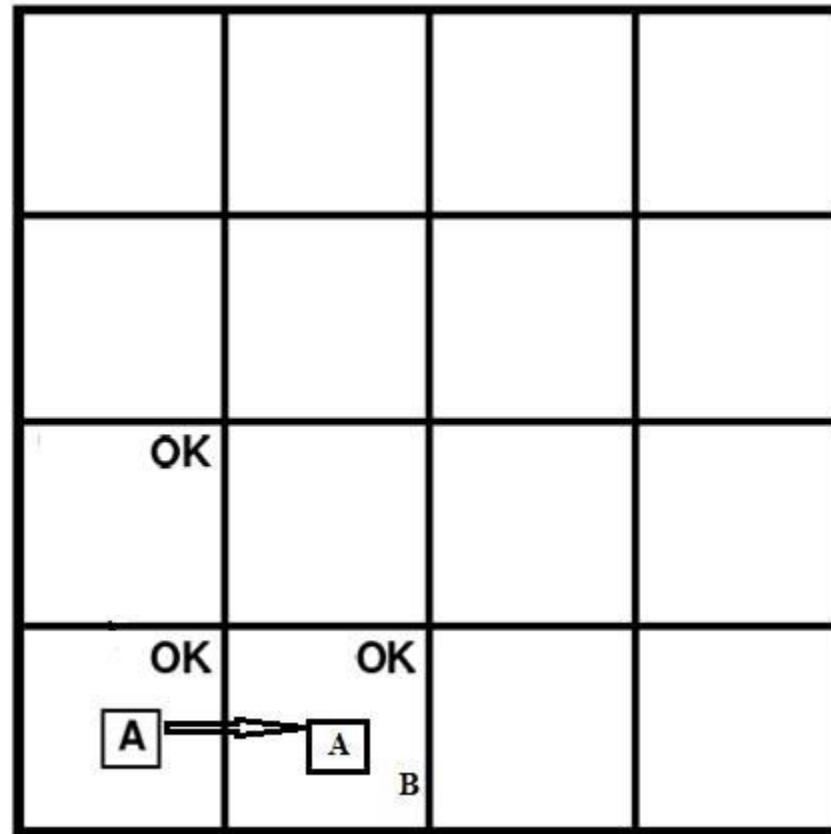
a **breeze** in the squares adjacent to a pit



Agent Precept sequence: [stench, breeze, glitter, bump, scream]
 [Stench, Breeze, None, None, None]

Exploring a wumpus world

a **breeze** in the squares adjacent to a pit



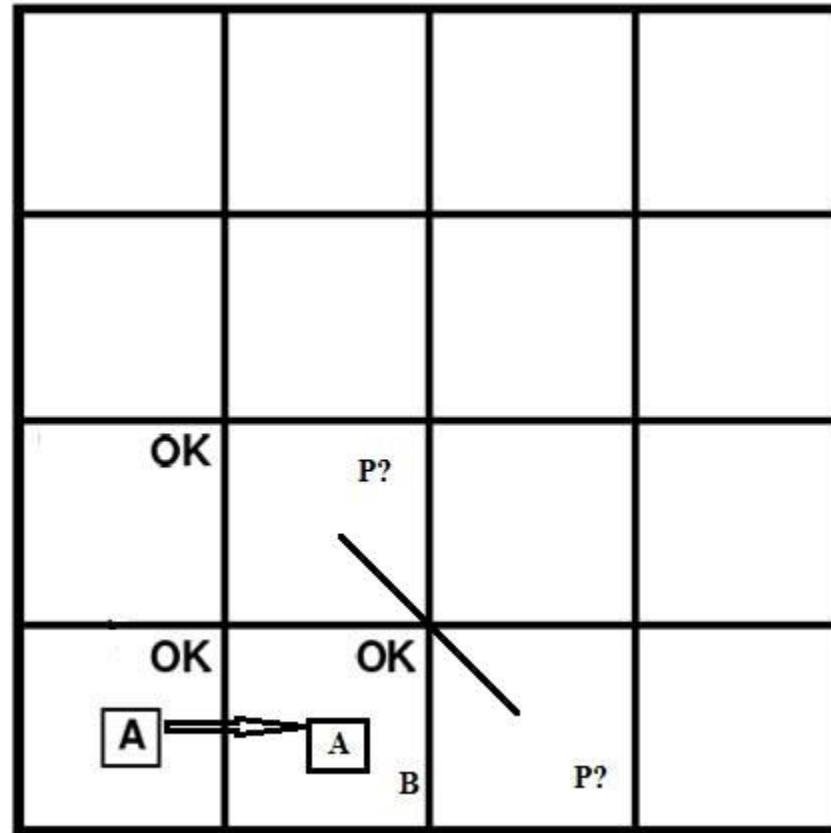
4	Stench		Breeze	PIT
3	Wumpus	Breeze Stench Gold	PIT	Breeze
2	Stench		Breeze	
1	START	Breeze	PIT	Breeze
	1	2	3	4

Agent Precept sequence: [stench, breeze, glitter, bump, scream]
 [None, None, None, None, None]

Exploring a wumpus world

Agent A back track from [1,2] to [1,1], moves 90 degree right, then moves to [2,1] safe state.

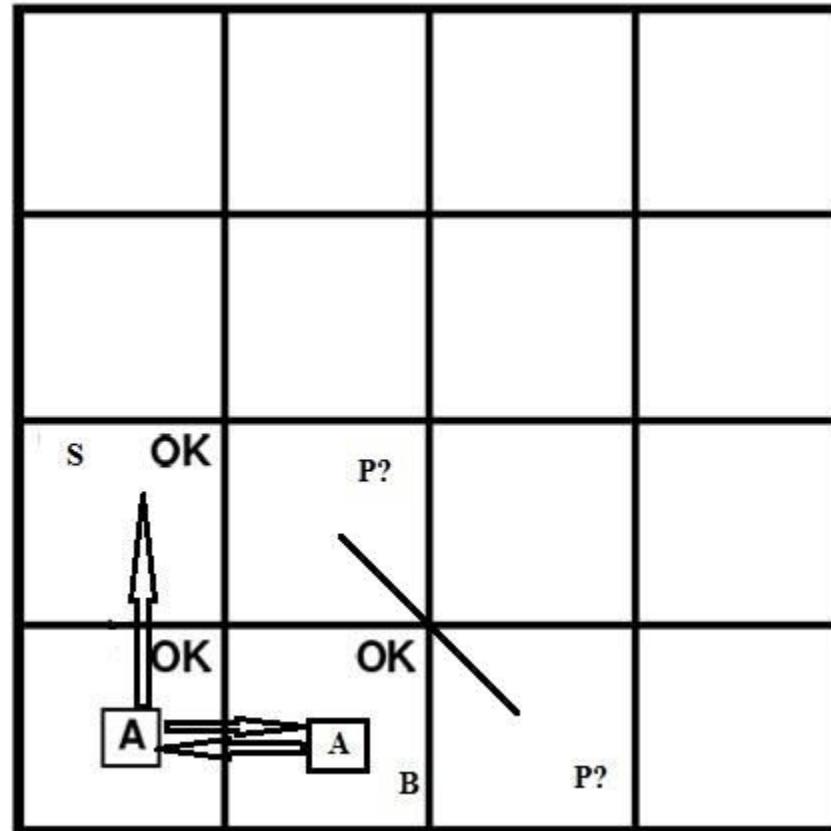
[row, col]



1	2	3	4

Exploring a wumpus world

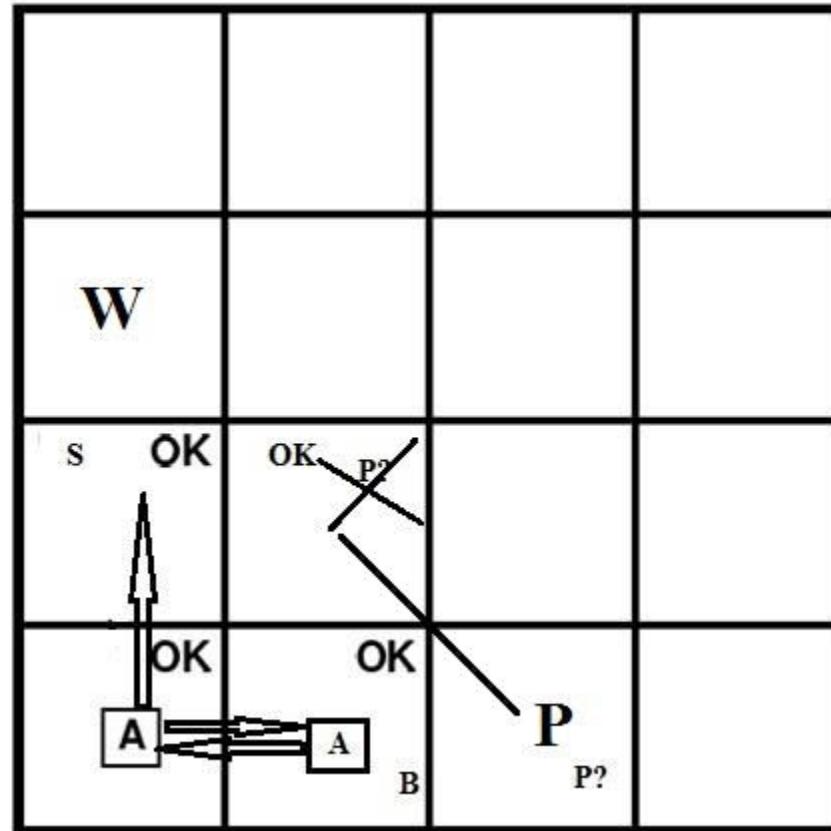
Since [2,1] has stench smell, adjacent to [2,1] must be wumpus.



 Stench		 Breeze	 PIT
	 Breeze	 Stench	 PIT
 Stench		 Breeze	
 START	 Breeze	 PIT	 Breeze
1	2	3	4

Exploring a wumpus world

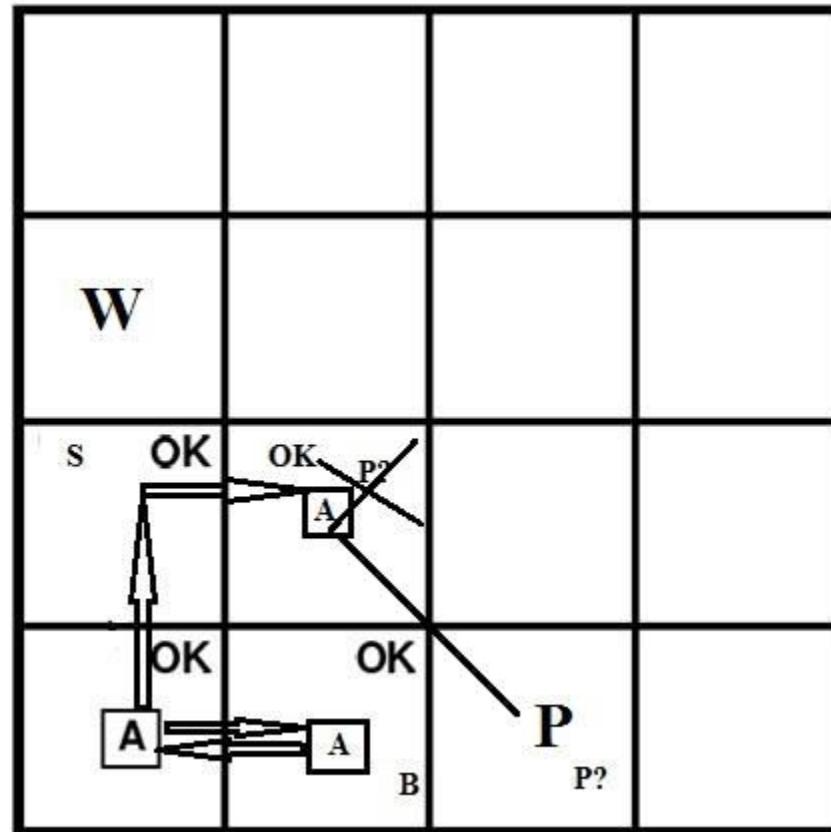
Agent [2,1] moves to safe state [2,2]



$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench		Breeze	PIT
	Breeze		Breeze
	$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench	Gold	
$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench	Breeze		
	Breeze		Breeze

Exploring a wumpus world

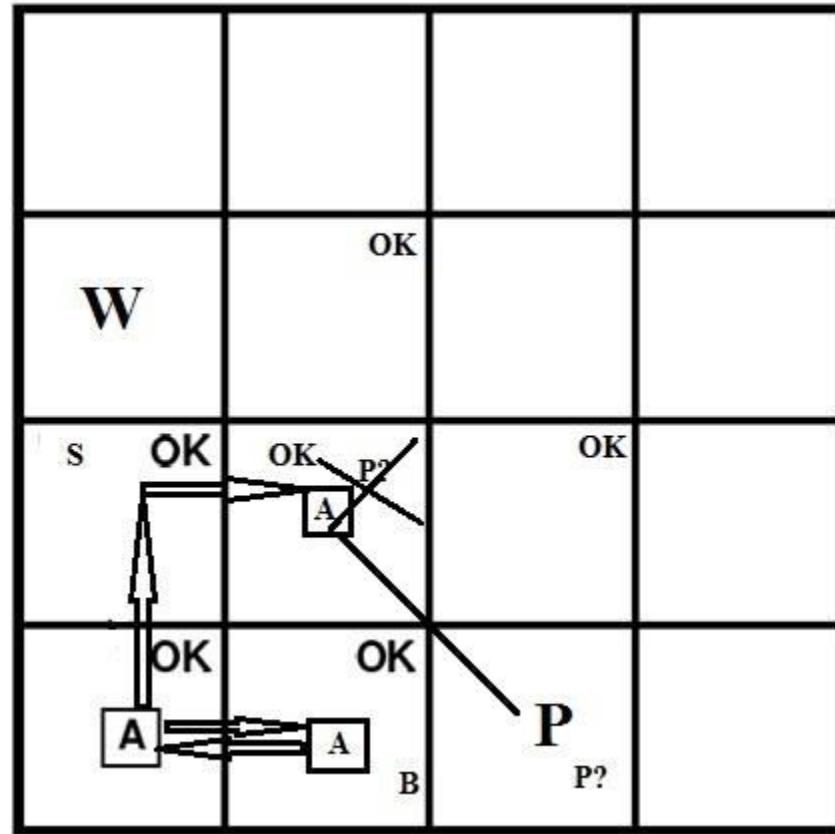
Agent perceives BGS in [3,2] and B in [2,3], both states are safe.



$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench		Breeze	PIT
Wumpus	Breeze	PIT	Breeze
$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench	Breeze		
Agent	Breeze	PIT	Breeze
1 2 3 4			

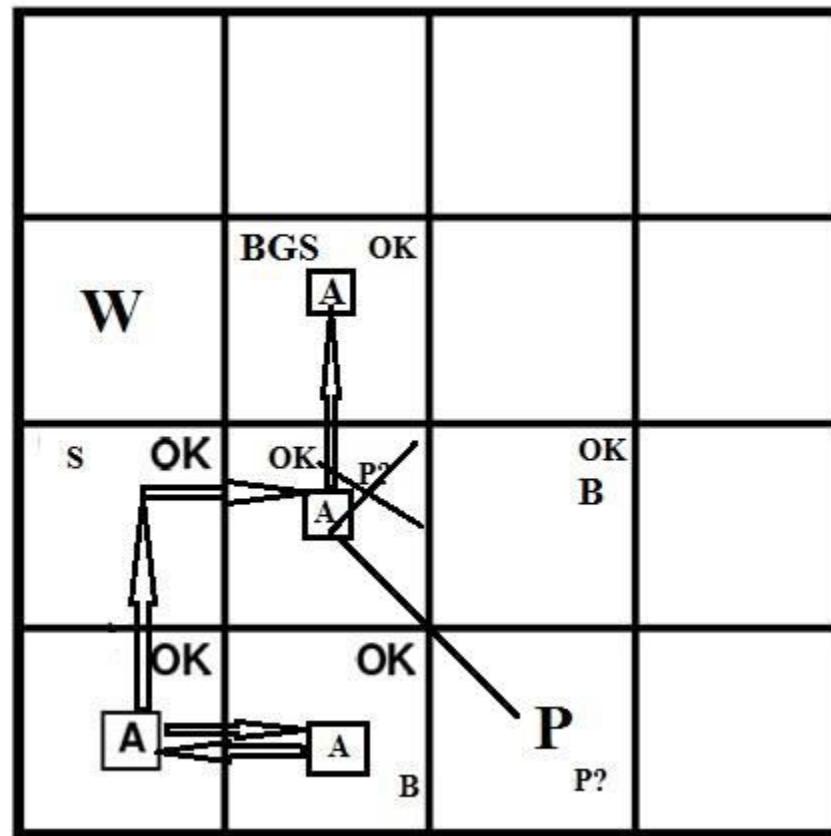
Exploring a wumpus world

Since glitter smell in state [3,2] agent A moves to [3,2].
And we know that room containing gold has glitter smell.



 Stench		 Breeze	PIT
	 Stench  Gold	PIT	
 Stench		 Breeze	
	 Breeze	PIT	
START			
1	2	3	4

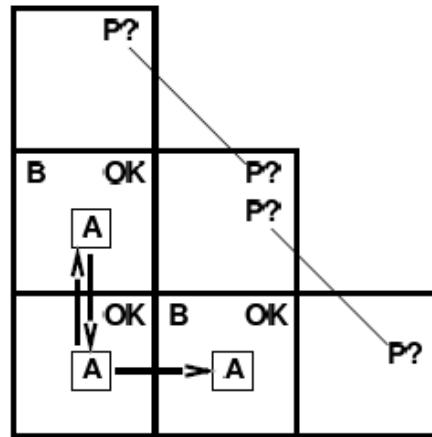
Exploring a wumpus world



$\gg \gg \gg \gg$ Stench		$\sim \sim$ Breeze	PIT
$\gg \gg \gg \gg$ Stench	$\sim \sim$ Breeze	PIT	$\sim \sim$ Breeze
$\gg \gg \gg \gg$ Stench	$\gg \gg \gg \gg$ Stench	PIT	$\sim \sim$ Breeze
START	$\sim \sim$ Breeze	PIT	$\sim \sim$ Breeze
1	2	3	4

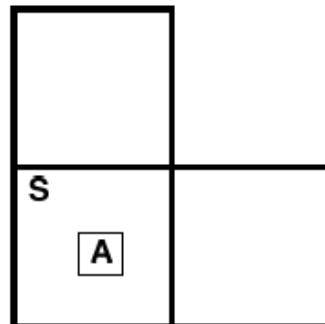
Exploring a wumpus world

Other tight spots



Breeze in (1,2) and (2,1)
 \Rightarrow no safe actions

Assuming pits uniformly distributed,
(2,2) has pit w/ prob 0.86, vs. 0.31



Smell in (1,1) \Rightarrow cannot move
Can use a strategy of coercion:
shoot straight ahead
wumpus was there \Rightarrow dead \Rightarrow safe
wumpus wasn't there \Rightarrow safe

Later

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(a)

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(b)

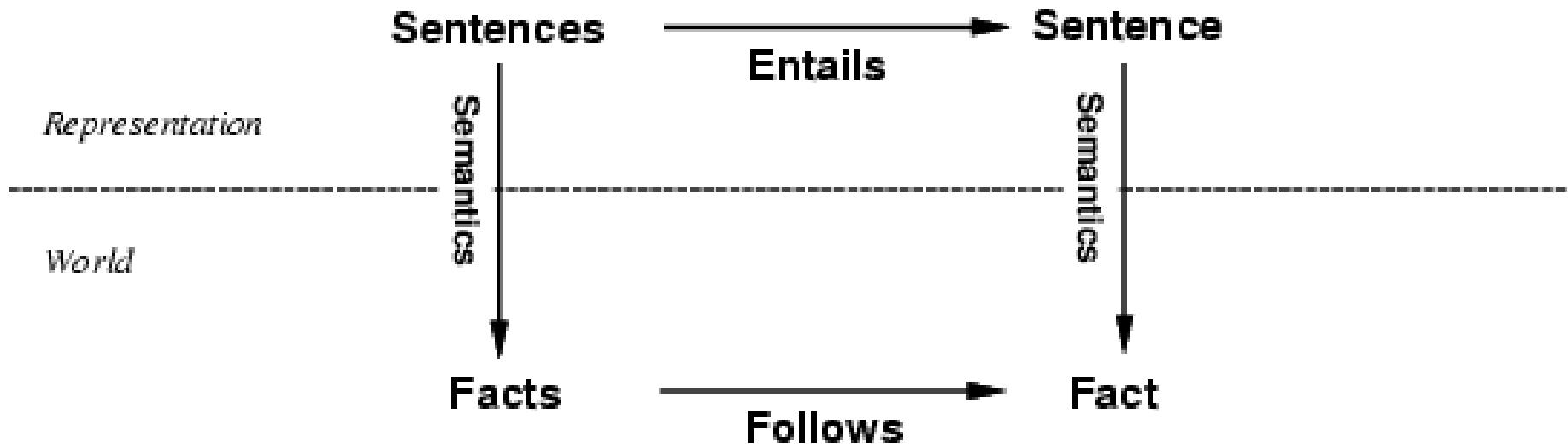
World-wide web wumpuses

- <http://scv.bu.edu/wcl>
- <http://216.246.19.186>
- <http://www.cs.berkeley.edu/~russell/code/doc/overview-AGENTS.html>

Representation, reasoning, and logic

- The object of *knowledge representation* is to express knowledge in a **computer-tractable form**, so that agents can perform well.
- A **knowledge representation language** is defined by:
 - its **syntax**, which defines all possible sequences of symbols that constitute sentences of the language.
 - Examples: Sentences in a book, bit patterns in computer memory.
 - its **semantics**, which determines the facts in the world to which the sentences refer.
 - Each sentence makes a claim about the world.
 - An agent is said to believe a sentence about the world.

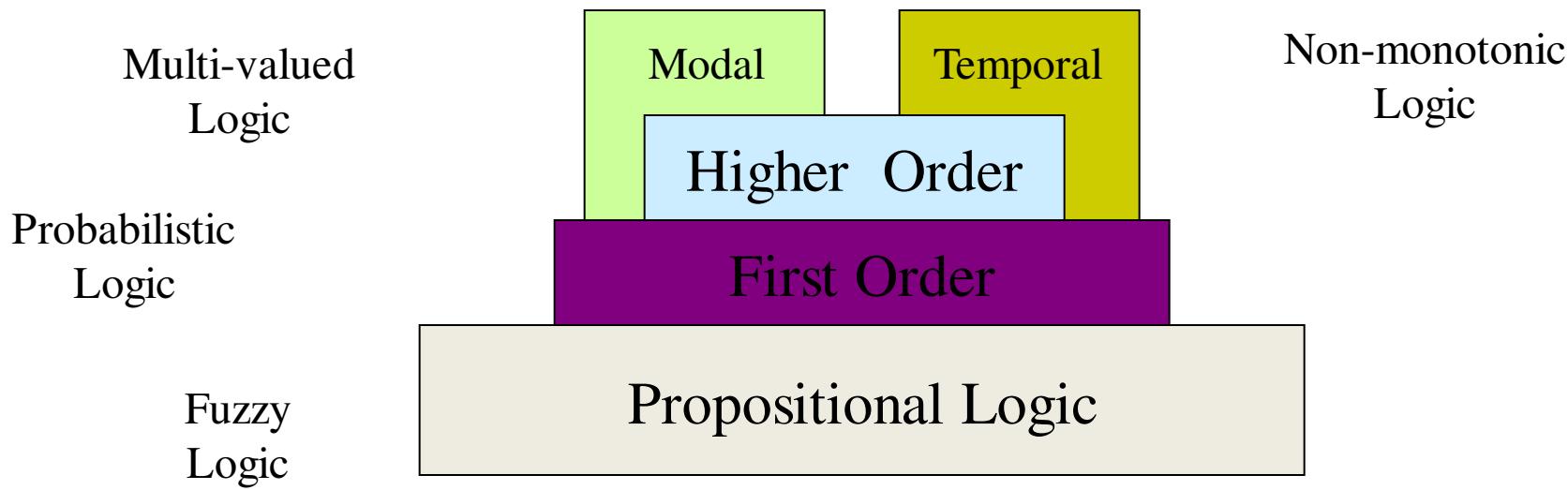
The connection between sentences and facts



Semantics maps sentences in logic to facts in the world.

The property of *one fact following from another* is mirrored by the property of *one sentence being entailed by another*.

Logic as a Knowledge-Representation (KR) language



Ontology and epistemology

- **Ontology** is the study of what there is,
 - an inventory of what exists.
- An ontological commitment is a commitment to an existence claim.
- **Epistemology** is major branch of philosophy that concerns the forms, nature, and preconditions of knowledge.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0...1
Fuzzy logic	degree of truth	degree of belief 0...1

Types of logic

- ▶ **Fuzzy logic** is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) **Boolean logic** on which the modern computer is based.
- ▶ **Temporal logic** is an approach to the semantics of expressions with tense, that is, expressions with qualifications of when. Some expressions, such as ' $2 + 2 = 4$ ', are true at all times, while tensed expressions such as 'John is happy' are only true sometimes.
- ▶ The expressions true in the past may not be true in present and vice versa.
- ▶ Truth value (semantics) of expression vary over time.
- ▶ **Multi-valued logics** are **logical calculi** in which there are more than two possible truth values. Truth values can be true(T), false(F) and Indeterminate(I).
- ▶ A **proposition** is a collection of declarative statements that has either a truth value "true" or a truth value "false". **Propositional Logic** is concerned with statements to which the truth values, "true" and "false", can be assigned. A propositional logic consists of propositional variables and connectives

Propositional logic

- **Logical constants:** true, false
- **Propositional symbols:** P, Q, S, ...
- Wrapping **parentheses:** (...)
- Sentences are combined by **connectives:**

\wedge ...and

\vee ...or

\Rightarrow ...implies

\Leftrightarrow ..is equivalent

\neg ...not

Propositional logic (PL)

- ▶ A **simple language** useful for showing key ideas and definitions
- ▶ User defines a **set of propositional symbols**, like P and Q.
- ▶ User defines the **semantics of each** of these symbols, e.g.:
 - P means "It is hot"
 - Q means "It is humid"
 - R means "It is raining"
- ▶ A **sentence** (aka **formula**, **well-formed formula**, **wff**) defined as:
 - A symbol
 - If **S** is a sentence, then $\sim S$ is a sentence (e.g., "not")
 - If **S** is a sentence, then so is (S)
 - If **S** and **T** are sentences, then $(S \vee T)$, $(S \wedge T)$, $(S \Rightarrow T)$, and $(S \Leftrightarrow T)$ are sentences (e.g., "or," "and," "implies," and "if and only if")
 - A finite number of applications of the above

Examples of PL sentences

- $(P \wedge Q) \Rightarrow R$
“If it is hot and humid, then it is raining”
- $Q \Rightarrow P$
“If it is humid, then it is hot”
- Q
“It is humid.”
- A better way:
 - H_o = “It is hot”
 - H_u = “It is humid”
 - R = “It is raining”

A BNF grammar of sentences in propositional logic

```

S := <Sentence> ;
<Sentence> := <AtomicSentence> | <ComplexSentence> ;
<AtomicSentence> := "TRUE" | "FALSE" |
                     "P" | "Q" | "S" ;
<ComplexSentence> := "(" <Sentence> ")" | 
                     <Sentence> <Connective> <Sentence> | 
                     "NOT" <Sentence> ;
<Connective> := "NOT" | "AND" | "OR" | "IMPLIES" | 
                     "EQUIVALENT" ;

```

Terms: semantics, interpretation, model, tautology, contradiction, entailment

- ▶ The meaning or **semantics** of a sentence determines its interpretation.
- ▶ Given the truth values of all of symbols in a sentence, it can be “evaluated” to determine its **truth value** (True or False).
- ▶ A **model** for a KB is a “possible world” in which each sentence in the KB is True.
- ▶ A **valid sentence** or **tautology** is a sentence that is True under all interpretations, no matter what the world is actually like or what the semantics is.
 - Example: “It’s raining or it’s not raining.”
- ▶ An **inconsistent sentence** or **contradiction** is a sentence that is False under all interpretations.
 - The world is never like what it describes, as in “It’s raining and it’s not raining.”
- ▶ **P entails Q**, written $P \models Q$, means that whenever P is True, so is Q.
 - In other words, **all models of P are also models of Q**.

Truth tables

And

P	q	$P \cdot q$
T	T	T
T	F	F
F	T	F
F	F	F

Or

P	q	$P \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

If . . . then

P	q	$P \supset q$
T	T	T
T	F	F
F	T	T
F	F	T

Not

P	$\sim P$
T	F
F	T

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., $A \vee B$, 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

Truth tables II

The five logical connectives:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>
<i>True</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>
<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>	<i>True</i>

A complex sentence: $((P \vee H) \wedge \neg H) \Rightarrow P$ is Valid or tautology

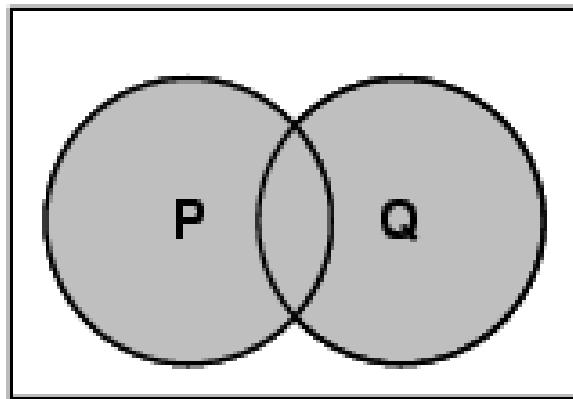
P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>

Models / Semantics

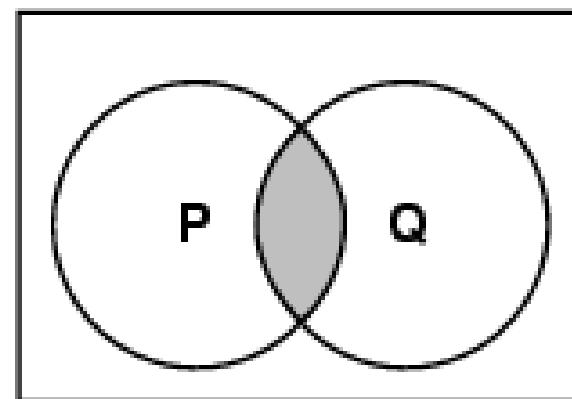
- A **model** specifies which of the proposition symbols are true and which are false
- Given a model, I should be able to tell you whether a sentence is true or false
- **Truth table** defines semantics of operators:
 - ▶ Refer the previous slide for the truth table of the following operations:
 - ▶ and (\wedge)
 - ▶ or (\vee)
 - ▶ implication (\Rightarrow)
 - ▶ not (\sim)
 - ▶ equivalence or biconditional (\Leftrightarrow)

Models of complex sentences

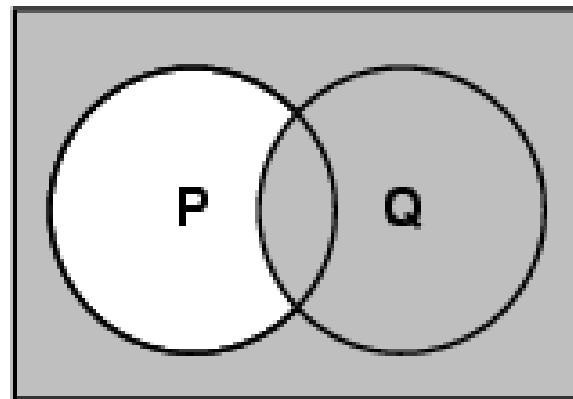
$P \vee Q$



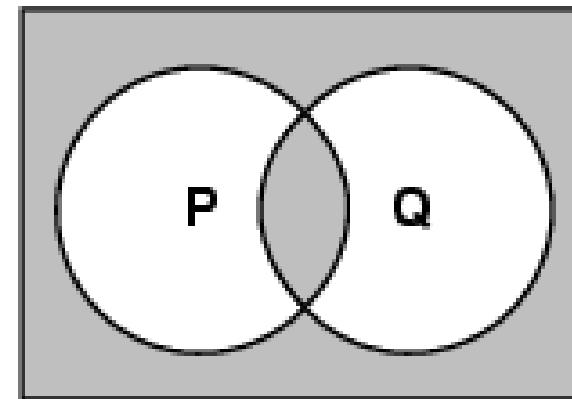
$P \wedge Q$



$P \Rightarrow Q$



$P \Leftarrow Q$



Tautologies

- A sentence is a **tautology** if it is true for any setting of its propositional symbols

P	Q	P OR Q	NOT(P) AND NOT(Q)	(P OR Q) OR (NOT(P) AND NOT(Q))
false	false	false	true	true
false	true	true	false	true
true	false	true	false	true
true	true	true	false	true

- $(P \text{ OR } Q) \text{ OR } (\text{NOT}(P) \text{ AND } \text{NOT}(Q))$ is a tautology

Is this a tautology?

- $(P \Rightarrow Q) \text{ OR } (Q \Rightarrow P)$
- *Left as an exercise*

Logical equivalences

- Two sentences are **logically equivalent** if they have the same truth value for every setting of their propositional variables

P	Q	P OR Q	NOT(NOT(P)) AND NOT(Q))
false	false	false	false
false	true	true	true
true	false	true	true
true	true	true	true

- P OR Q and NOT(NOT(P)) AND NOT(Q)) are logically equivalent
- Tautology = logically equivalent to True

Famous Logical Equivalence

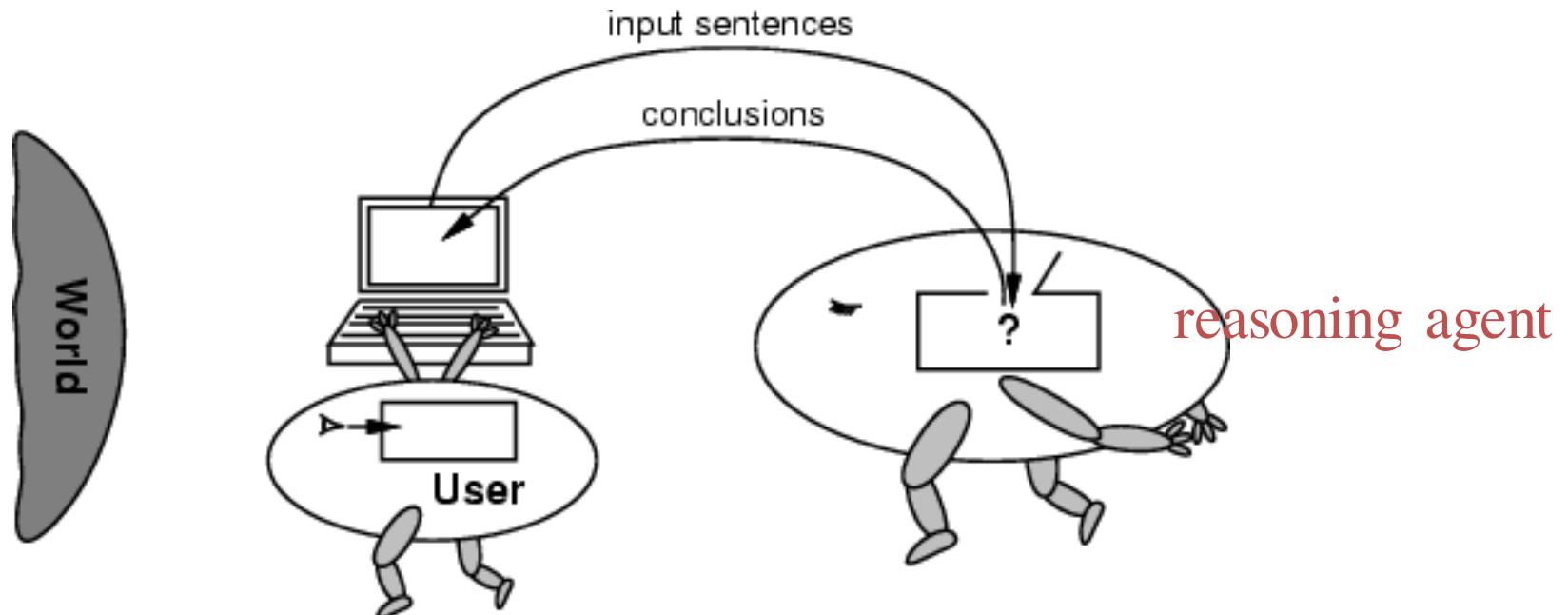
Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of \wedge
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of \vee
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of \wedge
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of \vee
- $\neg(\neg\alpha) \equiv \alpha$ double-negation elimination
- $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition
- $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination
- $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of \wedge over \vee
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of \vee over \wedge

Agents have no independent access to the world

- The reasoning agent often gets its knowledge about the facts of the world **as a sequence of logical sentences**.
- It must draw conclusions **only from them**, without independent access to the world.
- Thus it is very important that the **agent's reasoning is sound!**



Inference rules

- ▶ **Logical inference** is used to create **new sentences** that logically follow from a given set of predicate calculus sentences (KB).
- ▶ An inference rule is **sound** if every sentence X produced by an inference rule operating on a KB logically follows from the KB.
 - (That is, the inference rule does not create any contradictions)
- ▶ An inference rule is **complete** if it is able to produce every expression that logically follows from the KB.
 - We also say - "expression is entailed by KB".
 - Please note the analogy to **complete search algorithms**.

Sound rules of inference

- ▶ Here are some examples of **sound rules of inference**.
- ▶ Each can be shown to be sound **using a truth table**:
 - A rule is sound if **its conclusion is true whenever the premise is true**.

RULE	PREMISE	CONCLUSION
Modus Ponens	$A, A \Rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\sim\sim A$	A
Unit Resolution	$A \vee B, \sim B$	A
Resolution	$A \vee B, \sim B \vee C$	$A \vee C$

Sound Inference Rules (deductive rules)

- Here are some examples of **sound rules of inference**.
- Each can be shown to be sound using a truth table -- **a rule is sound if its conclusion is true whenever the premise is true.**

<u>RULE</u>	<u>PREMISE</u>	<u>CONCLUSION</u>
Modus Tollens	$\sim B, A \Rightarrow B$	$\sim A$
Or Introduction	A	$A \vee B$
Chaining	$A \Rightarrow B, B \Rightarrow C$	$A \Rightarrow C$

Soundness of modus ponens

premise	conclusion	premise	OK?
A	B	$A \rightarrow B$	
True	True	True	✓
True	False	False	✓
False	True	True	✓
False	False	True	✓

Soundness of the resolution inference rule

α	β	γ	$\alpha \vee \beta$	$\neg \beta \vee \gamma$	$\alpha \vee \gamma$
<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>	<u>True</u>	<u>False</u>
<u>False</u>	<u>False</u>	<u>True</u>	<u>False</u>	<u>True</u>	<u>True</u>
<u>False</u>	<u>True</u>	<u>False</u>	<u>True</u>	<u>False</u>	<u>False</u>
<u>False</u>	<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>
<u>True</u>	<u>False</u>	<u>False</u>	<u>True</u>	<u>True</u>	<u>True</u>
<u>True</u>	<u>False</u>	<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>
<u>True</u>	<u>True</u>	<u>False</u>	<u>True</u>	<u>False</u>	<u>True</u>
<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>

Whenever premise is true, the conclusion is also true

But it may be also in other cases

► Resolution rule

Unit Resolution

$$A \vee B, \sim A \vdash B$$

Resolution

$$A \vee B, \sim B \vee C \vdash A \vee C$$

Let $\alpha_1 \dots \alpha_i, \beta, \gamma_1 \dots \gamma_k$ be literals. Then

$$\alpha_1 \vee \dots \vee \alpha_i \vee \beta, \sim \beta, \gamma_1 \vee \dots \vee \gamma_k \vdash \alpha_1 \vee \dots \vee \alpha_i \vee \gamma_1 \vee \dots \vee \gamma_k$$

- Operates on **two disjunctions of literals**
- The pair of two opposite literals (β and $\sim \beta$) **cancel each other**, all other literals from the two disjuncts are combined to form a new disjunct as the inferred sentence
- Resolution rule can replace all other inference rules

Modus Ponens

$$A, \sim A \vee B \vdash B$$

Modus Tollens

$$\sim B, \sim A \vee B \vdash \sim A$$

Chaining

$$\sim A \vee B, \sim B \vee C \vdash \sim A \vee C$$



These old
rules are
special
cases of
resolution

Proving things: what are proofs and theorems?

- ▶ A **proof** is a sequence of sentences, where each sentence is *either a premise or a sentence derived from earlier sentences* in the proof by one of the rules of inference.
- ▶ The last sentence is the **theorem** (also called goal or query) that we want to prove.
- ▶ Example for the “**weather problem**” given above.

1 Hu	Premise	“It is humid”
2 Hu=>Ho	Premise	“If it is humid, it is hot”
3 Ho	Modus Ponens(1,2)	“It is hot”
4 (Ho^Hu)=>R	Premise	“If it’s hot & humid, it’s raining”
5 Ho^Hu	And Introduction(1,2)	“It is hot and humid”
6 R	Modus Ponens(4,5)	“It is raining”

Proof by resolution / Resolution algorithm

- ▶ Given formula in conjunctive normal form, repeat:
- ▶ Find two clauses with complementary literals,
- ▶ Apply resolution,
- ▶ Add resulting clause (if not already there)
- ▶ If the **empty** clause results, formula is **not satisfiable**
 - Must have been obtained from P and $\text{NOT}(P)$
- ▶ Otherwise, if we get stuck (and we will **eventually**), the formula is guaranteed to be satisfiable.

Proof by resolution method example

- ▶ Given set of premises
 - ▶ 1) “It is humid”
 - ▶ 2) “If it is humid, it is hot”
 - ▶ 3) “If it’s hot & humid, it’s raining”
-

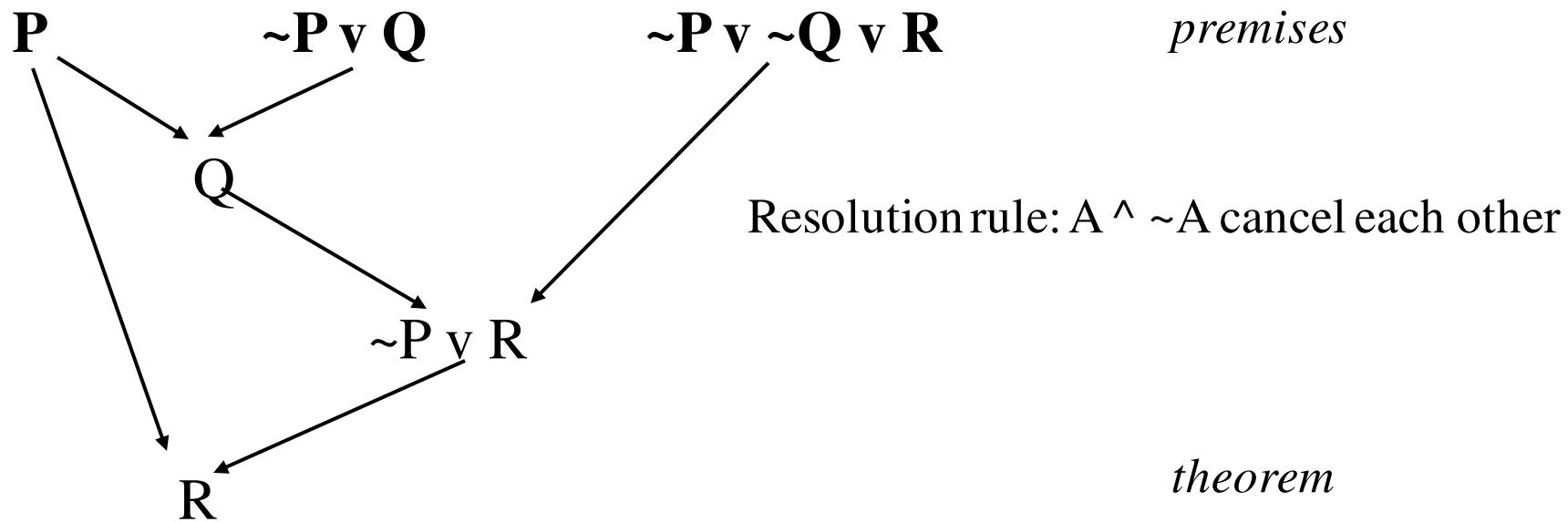
- ▶ Prove that “It is raining”
- ▶ Sol: Let P: It is humid Q: it is hot R: it’s raining

- 1) P
- 2) $P \Rightarrow Q$
- 3) $(P \wedge Q) \Rightarrow R$

We have to prove $(1 \wedge 2 \wedge 3) \Rightarrow R$

- 4) $\neg P \vee Q$ (2) \Rightarrow elimination rule
- 5) $\neg(P \wedge Q) \vee R$ (3) \Rightarrow elimination rule
- 6) $\neg P \vee \neg Q \vee R$ (5) demorgan law

► Proof by resolution



- Theorem proving **as search**
 - **Start node:** the set of given premises/axioms (KB + Input)
 - **Operator:** inference rule (add a new sentence into parent node)
 - **Goal:** a state that contains the theorem asked to prove
 - **Solution:** a path from start node to a goal

Normal forms of PL sentences

► Disjunctive normal form (DNF)

- Any sentence can be written as a disjunction of conjunctions of literals.
- Examples: $\underline{P \wedge Q \wedge \sim R}$; $\underline{A \wedge B} \vee \underline{C \wedge D} \vee \underline{P \wedge Q \wedge R}$; \underline{P}
- Widely used in logical circuit design (simplification)

► Conjunctive normal form (CNF)

- Any sentence can be written as a **conjunction of disjunctions** of literals.
- Examples: $\underline{P} \vee \underline{Q} \vee \sim \underline{R}$; $(\underline{A} \vee \underline{B}) \wedge (\underline{C} \vee \underline{D}) \wedge (\underline{P} \vee \underline{Q} \vee \underline{R})$; \underline{P}

► Normal forms can be obtained by applying equivalence laws

$$\begin{aligned}
 & [(A \vee B) \Rightarrow (C \vee D)] \Rightarrow P \\
 & \equiv \sim [\sim (A \vee B) \vee (C \vee D)] \vee P && // \text{ law for implication} \\
 & \equiv [\sim \sim (A \vee B) \wedge \sim (C \vee D)] \vee P && // \text{ de Morgan's law} \\
 & \equiv [(A \vee B) \wedge (\sim C \wedge \sim D)] \vee P && // \text{ double negation and de Morgan's law} \\
 & \equiv (A \vee B \vee P) \wedge (\sim C \wedge \sim D \vee P) && // \text{ distribution law} \\
 & \equiv (A \vee B \vee P) \wedge (\sim C \vee P) \wedge (\sim D \vee P) && // \text{ a CNF}
 \end{aligned}$$

Formal Proof Example: A story

- ▶ You roommate comes home; he/she is completely wet
- ▶ You know the following things:
 - Your roommate is wet
 - If your roommate is wet, it is because of rain, sprinklers, or both
 - If your roommate is wet because of sprinklers, the sprinklers must be on
 - If your roommate is wet because of rain, your roommate must not be carrying the umbrella
 - The umbrella is not in the umbrella holder
 - If the umbrella is not in the umbrella holder, either you must be carrying the umbrella, or your roommate must be carrying the umbrella
 - You are not carrying the umbrella
- ▶ Can you conclude that the sprinklers are on?
- ▶ Can AI conclude that the sprinklers are on?
- ▶ **Prove by formal method i,e Use rules of inferences or proof systems**

Knowledge base for the story

- ▶ RoommateWet
- ▶ RoommateWet => (RoommateWetBecauseOfRain OR RoommateWetBecauseOfSprinklers)
- ▶ RoommateWetBecauseOfSprinklers => SprinklersOn
- ▶ RoommateWetBecauseOfRain =>
NOT(RoommateCarryingUmbrella)
- ▶ UmbrellaGone
- ▶ UmbrellaGone => (YouCarryingUmbrella OR RoommateCarryingUmbrella)
- ▶ NOT(YouCarryingUmbrella)

Formal proof that the sprinklers are on

- 1) RoommateWet
- 2) RoommateWet => (RoommateWetBecauseOfRain OR RoommateWetBecauseOfSprinklers)
- 3) RoommateWetBecauseOfSprinklers => SprinklersOn
- 4) RoommateWetBecauseOfRain => NOT(RoommateCarryingUmbrella)
- 5) UmbrellaGone
- 6) UmbrellaGone => (YouCarryingUmbrella OR RoommateCarryingUmbrella)
- 7) NOT(YouCarryingUmbrella)
- 8) YouCarryingUmbrella OR RoommateCarryingUmbrella (*modus ponens on 5 and 6*)
- 9) NOT(YouCarryingUmbrella) => RoommateCarryingUmbrella (*equivalent to 8*)
- 10) RoommateCarryingUmbrella (*modus ponens on 7 and 9*)
- 11) NOT(NOT(RoommateCarryingUmbrella)) (*equivalent to 10*)
- 12) NOT(NOT(RoommateCarryingUmbrella)) => NOT(RoommateWetBecauseOfRain) (*equivalent to 4 by contraposition*)
- 13) NOT(RoommateWetBecauseOfRain) (*modus ponens on 11 and 12*)
- 14) RoommateWetBecauseOfRain OR RoommateWetBecauseOfSprinklers (*modus ponens on 1 and 2*)
- 15) NOT(RoommateWetBecauseOfRain) => RoommateWetBecauseOfSprinklers (*equivalent to 14*)
- 16) RoommateWetBecauseOfSprinklers (*modus ponens on 13 and 15*)
- 17) SprinklersOn (*modus ponens on 16 and 3*)

Prove that the sprinklers are on

- By resolution Method
- Left as an Exercise.

Horn sentences

- A **Horn sentence** or **Horn clause** has the form:

$$P_1 \wedge P_2 \wedge P_3 \dots \wedge P_n \Rightarrow Q$$

or alternatively

$$\neg P_1 \vee \neg P_2 \vee \neg P_3 \dots \vee \neg P_n \vee Q$$

$$(P \Rightarrow Q) = (\neg P \vee Q)$$

where Ps and Q are **non-negated atoms**

- To get a proof for Horn sentences, apply **Modus Ponens** repeatedly until nothing can be done
- We will use the **Horn clause form** later

Special case: Horn clauses

- ▶ Horn clauses are implications with only positive literals
- ▶ $x_1 \text{ AND } x_2 \text{ AND } x_4 \Rightarrow x_3 \text{ AND } x_6$
- ▶ $\text{TRUE} \Rightarrow x_1$
- ▶ Try to figure out whether some x_j is entailed
- ▶ Simply follow the implications (modus ponens) as far as you can, see if you can reach x_j
- ▶ x_j is entailed if and only if it can be reached (can set everything that is not reached to false)

Entailment and derivation

► Entailment: $\text{KB} \models Q$

- Q is entailed by KB (a set of premises or assumptions) if and only if Q is true in every logically possible world in which all the premises in KB are true.
- Or, stated positively, Q is entailed by KB if and only if the conclusion is true in every logically possible world in which all the premises in KB are true.

► Derivation: $\text{KB} \vdash Q$

- We can derive Q from KB if there is a proof consisting of a sequence of valid inference steps starting from the premises in KB and resulting in Q

Two important properties for inference

Soundness: If $\text{KB} \vdash Q$ then $\text{KB} \models Q$

- If Q is derived from a set of sentences KB using a given set of rules of inference, then Q is entailed by KB .
- Hence, inference produces only real entailments,
 - or any sentence that follows deductively from the premises is valid.

Completeness: If $\text{KB} \models Q$ then $\text{KB} \vdash Q$

- If Q is entailed by a set of sentences KB , then Q can be derived from KB using the rules of inference.
- Hence, inference produces all entailments,
 - or all valid sentences can be proved from the premises.

Propositional logic is a weak language

- ▶ Hard to identify "**individuals.**" E.g., Mary, 3
 - Individuals cannot be PL sentences themselves.
- ▶ Can't directly talk about properties of individuals or relations between individuals. (hard to connect individuals to class properties).
 - E.g., property of being a human implies property of being mortal
 - E.g. "Bill is tall"
- ▶ Generalizations, patterns, regularities can't easily be represented.
 - E.g., **all** triangles have 3 sides
 - **All** members of a class have this property
 - **Some** members of a class have this property
- ▶ A **better representation is needed to capture the relationship** (and distinction) *between objects and classes*, including properties belonging to classes and individuals.

Confusius Example: weakness of PL

- Consider the problem of representing the following information:
 - *Every person is mortal.*
 - *Confucius is a person.*
 - *Confucius is mortal.*
- How can these sentences be represented so that we can infer the third sentence from the first two?

Example continued

- ▶ In PL we have to create propositional symbols to stand for all or part of each sentence. For example, we might do:
P = “person”; Q = “mortal”; R = “Confucius”
- ▶ so the above 3 sentences are represented as:
P => Q; R => P; R => Q
- ▶ Although the third sentence is entailed by the first two, we needed an explicit symbol, **R**, to represent an individual, Confucius, **who is a member of the classes “person” and “mortal.”**
- ▶ To represent other individuals we must introduce separate symbols for each one, with means for representing the fact that all individuals who are “people” are also “mortal.”

PL is Too Weak a Representational Language

- ▶ Consider the problem of representing the following information:
 - Every person is mortal. (S1)
 - Confucius is a person. (S2)
 - Confucius is mortal. (S3)
- ▶ S3 is clearly a logical consequence of S1 and S2.
 - But how can these sentences be represented using PL so that we can infer the third sentence from the first two?
- ▶ We can use symbols P, Q, and R to denote the three propositions,
- ▶ but this leads us to nowhere because knowledge important to infer R from P and Q
- ▶ (i.e., relationship between being a human and mortality, and the membership relation between Confucius and human class) is not expressed in a way that can be used by inference rules

- ▶ Alternatively, we can use symbols for parts of each sentence
 - $P = \text{"person"}; M = \text{"mortal"}; C = \text{"Confucius"}$
 - The above 3 sentences can be roughly represented as:
 $S_2: C \Rightarrow P; S_1: P \Rightarrow M; S_3: C \Rightarrow M.$
 - Then S_3 is entailed by S_1 and S_2 by the chaining rule.
- ▶ Bad semantics
 - “Confucius” (and “person” and “mortal”) are not PL sentences (**not a declarative statement**) and cannot have a truth value.
 - What does $P \Rightarrow M$ mean?
- ▶ We need **infinite distinct symbols X for individual persons**, and **infinite implications to connect these X with P (person) and M (mortal)** because **we need a unique symbol for each individual.**

$\text{Person_1} \Rightarrow P; \text{person_1} \Rightarrow M;$

$\text{Person_2} \Rightarrow P; \text{person_2} \Rightarrow M;$

...
 $\text{Person_n} \Rightarrow P; \text{person_n} \Rightarrow M$

The “Hunt the Wumpus” agent

- Some Atomic Propositions

S_{12} = There is a stench in cell (1,2)

B_{34} = There is a breeze in cell (3,4)

W_{22} = The Wumpus is in cell (2,2)

V_{11} = We have visited cell (1,1)

OK_{11} = Cell (1,1) is safe.

etc

- Some rules

(R1) $\sim S_{11} \Rightarrow \sim W_{11} \wedge \sim W_{12} \wedge \sim W_{21}$

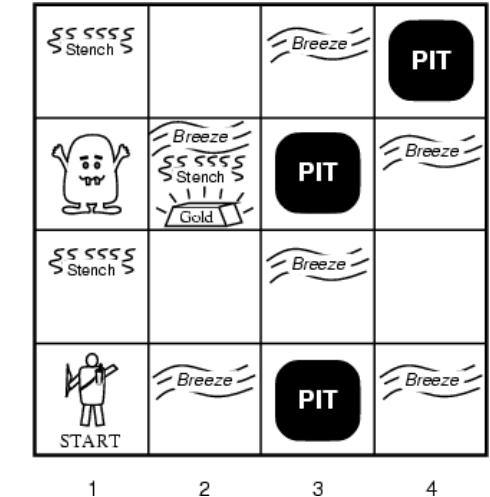
(R2) $\sim S_{21} \Rightarrow \sim W_{11} \wedge \sim W_{21} \wedge \sim W_{22} \wedge \sim W_{31}$

(R3) $\sim S_{12} \Rightarrow \sim W_{11} \wedge \sim W_{12} \wedge \sim W_{22} \wedge \sim W_{13}$

(R4) $S_{12} \Rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

etc

- Note that the **lack of variables** requires us to give similar rules for each cell.



After the third move

- We can prove that the Wumpus is in (1,3) using the four rules given.
- See R&N section 6.5

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

Proving W13

- ▶ Apply MP with $\sim S11$ and R1:

$\sim W11 \wedge \sim W12 \wedge \sim W21$

- ▶ Apply And-Elimination to this we get 3 sentences:

$\sim W11, \sim W12, \sim W21$

- ▶ Apply MP to $\sim S21$ and R2, then applying And-elimination:

$\sim W22, \sim W21, \sim W31$

- ▶ Apply MP to S12 and R4 we obtain:

$W13 \vee W12 \vee W22 \vee W11$

- ▶ Apply Unit resolution on $(W13 \vee W12 \vee W22 \vee W11)$ and $\sim W11$

$W13 \vee W12 \vee W22$

- ▶ Apply Unit Resolution with $(W13 \vee W12 \vee W22)$ and $\sim W22$

$W13 \vee W12$

- ▶ Apply UR with $(W13 \vee W12)$ and $\sim W12$

$W13$

- ▶ QED

(R1) $\sim S11 \Rightarrow \sim W11 \wedge \sim W12 \wedge \sim W21$

$\sim S11$

$\sim W11 \wedge \sim W12 \wedge \sim W21$

$\sim S21$

$\sim W22, \sim W21, \sim W31$

R2

$(W13 \vee W12 \vee W22 \vee W11)$

$\sim W11$

S12

R4

$(W13 \vee W12 \vee W22)$

$\sim W22$

$\sim W12$

$W13 \vee W12$

W13

Problems with the propositional Wumpus hunter

- ▶ **Lack of variables** prevents stating more general rules.
 - E.g., we need a set of similar rules for each cell
- ▶ **Change of the KB over time** is **difficult to represent**
 - Standard technique is to index facts with the time when they're true
 - This means we have a **separate KB for every time point.**

Summary I

- ▶ Intelligent agents need **knowledge about the world** for making good decisions.
- ▶ The knowledge of an agent is **stored in a knowledge base** in the form of **sentences** in a **knowledge representation language**.
- ▶ A knowledge-based agent needs a **knowledge base** and an **inference mechanism**.
 - It operates by storing sentences in its knowledge base,
 - inferring new sentences with the inference mechanism,
 - and using them to deduce which actions to take.
- ▶ A **representation language** is defined by its syntax and semantics, which specify the structure of sentences and how they relate to the facts of the world.
- ▶ The **interpretation** of a sentence is the fact to which it refers.
 - If this fact is part of the actual world, then the sentence is true.

Summary II

- ▶ The process of deriving new sentences from old one is called **inference**.
 - **Sound** inference processes derives true conclusions given true premises.
 - **Complete** inference processes derive all true conclusions from a set of premises.
- ▶ **A valid sentence** is true in all worlds under all interpretations.
- ▶ If an implication sentence can be shown to be valid, then - given its premise - its consequent can be derived.
- ▶ Different logics make different **commitments** about what the world is made of and what kind of beliefs we can have regarding the facts.
 - Logics are useful for the commitments they do not make because lack of commitment gives the knowledge base write more freedom.
- ▶ **Propositional logic** commits only to the existence of facts that may or may not be the case in the world being represented.
 - It has a simple syntax and a simple semantic. It suffices to illustrate the process of inference.
 - Propositional logic quickly becomes impractical, even for very small worlds.

Video Reference links

S. NO	Description	Video link
1	This video describes the wumpus world problem : Example on propositional logic	https://www.youtube.com/watch?v=fnwjsO9ky9o
2	This video describes the wumpus world problem : Example on propositional logic	https://www.youtube.com/watch?v=wy2IyZ6ayCY
3	This video explains the knowledge representation in propositional logic.	https://www.youtube.com/watch?v=hUuCBQKH-Cg
4	This video explains the Rules of Inference - Definition & Types of Inference Rules	https://www.youtube.com/watch?v=HcS4lqXxrV4
5	This video demonstrates the INFERENCE THEORY PROBLEM S.	https://www.youtube.com/watch?v=7J9e-leElhU

Thank you

UNIT – 3(2)

First Order Logic

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-3(2)

- **First-Order Logic:** (2 Hours)

Using First-Order Logic, A Simple Reflex Agent, Deducing Hidden Properties of the World, Toward a Goal-Based Agent, Building a Knowledge Base, Knowledge Engineering, Inference Rules Involving Quantifiers, Generalized Modus Ponens, Forward and Backward Chaining, Completeness, Resolution: A Complete Inference Procedure, Completeness of resolution

Limitations of Propositional Logic

- Cannot deal properly with general statements of the form
- “All men are mortal”
- Cannot derive the conjunction of “All men are mortal” and “Socrates is a man” that “Socrates is mortal”
- Example”
- If All men are mortal = P
- Socrates is a Man = Q
- Socrates is mortal = R
- Then $(P \& Q) \Rightarrow R$ is not valid Proposition.
- Conclusion: Socrates is mortal, cannot be inferred.
- Objects and relationships cannot be represented in PL.

First Order Logic

- Propositional logic assumes the world consists of atomic facts
- First-order logic assumes the world contains objects, relations, and functions.
- The world consists of **objects**, that is, things with individual PROPERTIES identities and **properties that distinguish them from other objects**.
- Among these objects, various **relations hold. Some of these relations are functions**—
- **FUNCTIONS** are the relations in which there is only one "value" for a given "input."
- Examples
- **Objects:** people, houses, numbers, theories, Ronald McDonald, colors, baseball games ...
- **Relations:** brother of, bigger than, inside, part of, has color, occurred after, owns ...
- **Properties:** red, round, bogus, prime, multistoried
- **Functions:** father of, best friend, third inning of, one more than ...

Identify Objects, Properties or relations and functions

- "One plus two equals three"
- **Objects:** one, two, three, one plus two; **Relation:** equals; **Function:** plus.
(One plus two is a name for the object that is obtained by applying the function plus to the objects one and two. Three is another name for this object.)
- **Equals(plus(1,2), 3).**
- "Squares neighboring the wumpus are smelly."
- **Objects:** wumpus, square; **Property:** smelly; **Relation:** neighboring.
- Neighbor(Square, wumpus) => smelly(square)
- "Evil King John ruled England in 1200."
- **Objects:** John, England, 1200; **Relation:** ruled; **Properties:** evil, king.

Syntax of FOL

Sentence — *AtomicSentence*
 | *Sentence Connective Sentence*
 | *Quantifier Variable, . . . Sentence*
 | \neg *Sentence*
 | *(Sentence)*

AtomicSentence — *Predicate(Term, . . .)* | *Term = Term*

Term — \rightarrow *Function(Term, . . .)*
 | *Constant*
 | *Variable*

Connective — \Rightarrow | \wedge | \vee | \Leftrightarrow

Quantifier — \forall | \exists

Constant — *A* | *X* | *John* | \dots

Variable — *a* | *x* | *s* | \dots

Predicate — *Before* | *HasColor* | *Raining* | \dots

Function — *Mother* | *LeftLegOf* | \dots

Syntax of FOL

Constants:	John, Sally, 2, ...
Variables:	x, y, a, b, ...
Predicates:	Person(John), Siblings(John, Sally), IsOdd(2), ...
Functions:	MotherOf(John), Sqrt(x), ...
Connectives:	\neg , \wedge , \vee , \Rightarrow , \Leftrightarrow
Equality:	=
Quantifiers:	\forall , \exists
Term:	Constant or Variable or Function(Term ₁ , ..., Term _n)
Atomic sentence:	Predicate(Term ₁ , ..., Term _n) or Term ₁ = Term ₂
Complex sentence:	made from atomic sentences using connectives and quantifiers

Constants, predicate and functions

- **Constant symbols:** *A, B, C, John ...*
- An interpretation must specify which object in the world is referred to by each constant symbol.
- Each constant symbol names exactly one object, but not all objects need to have names, and some can have several names. An interpretation John might refer to the evil King John.
- **Predicate symbols:** *Round, Brother,...*
- An interpretation specifies that a predicate symbol refers to a particular relation in the model.
- For example, the *Brother symbol* might refer to the *relation of brotherhood*.
- *Brother is a binary predicate symbol.*
- **Function symbols:** *Cosine, FatherOf, LeftLegOf...*
- Some relations are functional—that is, any given object is related to exactly one other object by the relation.
- For example, any angle has only one number that is its cosine; any person has only one person that is his or her father

Term, atomic sentence

- A **term** is a logical expression that refers to an object. Constants, variables and functions can be terms.
- An **atomic sentence** is formed from a predicate symbol followed by a parenthesized list of terms.
- For example, *Brother(Richard, John)* states, under the interpretation given before, that Richard the Lionheart is the brother of King John.
- Atomic sentences can have arguments that are complex terms:
- *Married(FatherOf(Richard), MotherOf(John))*
- states that Richard the Lionheart's father is married to King John's mother (again, under a suitable interpretation).
- *An atomic sentence is true if the relation referred to by the predicate symbol holds between the objects referred to by the arguments*

Semantics of FOL

- Sentences are true with respect to a **model** and an **interpretation**
- Model contains objects (**domain elements**) and relations among them
- Interpretation specifies referents for
 - constant symbols → objects
 - predicate symbols → relations
 - function symbols → functional relations
- An atomic sentence **Predicate(Term₁, ..., Term_n)** is true iff the objects referred to by **Term₁, ..., Term_n** are in the relation referred to by predicate

Quantifiers

- **Quantifier:** Tell how many objects have a certain property.
- First-order logic contains two standard quantifiers, called *universal and existential*.

Universal quantification

- $\forall x P(x)$
- Example: “Everyone at UNC is smart”
- $\forall x At(x,UNC) \Rightarrow Smart(x)$
- Why not $\forall x At(x,UNC) \wedge Smart(x)$?
- Roughly speaking, equivalent to the conjunction of all possible instantiations of the variable:
- $At(John, UNC) \Rightarrow Smart(John) \wedge \dots$
- $At(Richard, UNC) \Rightarrow Smart(Richard) \wedge \dots$
- **$\forall x P(x)$ is true in a model m iff $P(x)$ is true with x being each possible object in the model**

Existential quantification

- $\exists x P(x)$
- Example: “Someone at UNC is smart”
- $\exists x At(x,UNC) \wedge Smart(x)$
- Why not $\exists x At(x,UNC) \Rightarrow Smart(x)$?
- Roughly speaking, equivalent to the disjunction of all possible instantiations:
- $[At(John,UNC) \wedge Smart(John)] \vee$
- $[At(Richard,UNC) \wedge Smart(Richard)] \vee \dots$
- **$\exists x P(x)$ is true in a model m iff $P(x)$ is true with x being some possible object in the model**

Universal Quantifier \forall existential \exists or 3 or \exists

- Examples
- universal quantifier: “for every object x in the universe, $x > 1$ ” written as: $\forall x x > 1$
- existential quantifier: "for some object x in the universe, $x > 1$ " written as: $\exists x x > 1$
- $\forall x \text{Cat}(x) \Rightarrow \text{Mammal}(x)$
- $\forall X P$, where P is any logical expression, as being equivalent to the conjunction (i.e., the A) of all the sentences obtained by substituting the name of an object for the **variable x** wherever it appears in P . The preceding sentence is therefore equivalent to

$\text{Cat}(\text{Spot}) \Rightarrow \text{Mammal}(\text{Spot})$ A

$\text{Cat}(\text{Rebecca}) \Rightarrow \text{Mammal}(\text{Rebecca})$ A

$\text{Cat}(\text{Felix}) \Rightarrow \text{Mammal}(\text{Felix})$ A

$\text{Cat}(\text{Richard}) \Rightarrow \text{Mammal}(\text{Richard})$ A

$\text{Cat}(\text{John}) \Rightarrow \text{Mammal}(\text{John})$ A

Existential Quantifier

- A statement about *some object in the universe without naming it, by using an existential quantifier.*
- for example, that Spot has a sister who is a cat, we write
- **$\exists x \ Sister(x, Spot) \wedge Cat(x)$**
- \exists is pronounced "There exists ...". In general, $\exists x P$ is true if P is true for some object in the universe.
- It therefore can be thought of as equivalent to the disjunction (i.e., the \vee) of all the sentences obtained by substituting the name of an object for the variable*. Doing the substitution for the above sentence, we would get
- $(Sister(Spot, Spot) \wedge Cat(Spot)) \vee$
- $(Sister(Rebecca, Spot) \wedge Cat(Rebecca)) \vee$
- $(Sister(Felix, Spot) \wedge Cat(Felix)) \vee$
- $(Sister(Richard, Spot) \wedge Cat(Richard)) \vee$
- $(Sister(John, Spot) \wedge Cat(John)) \vee$

Nested quantifiers

- Express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type.
- For example, "For all x and all y, if x is the parent of y then y is the child of x" becomes
- $\forall x, y \ Parent(x, y) \Rightarrow Child(y, x)$
- *Similarly, the fact that a person's brother has that person as a sibling is expressed by*

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

- "Everybody loves somebody" means that for every person, there is someone that person loves:

$\forall x \exists y \ Loves(x, y)$

- There is someone who is loved by everyone

$\exists y \forall x \ Loves(x, y)$

Connections between V and 3

- Because V is really a conjunction over the universe of objects and 3 is a disjunction, it should not be surprising that they obey De Morgan's rules. The De Morgan rules for quantified and unqualified sentences are as follows:

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

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

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

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

$$\neg P \wedge \neg Q \equiv \neg(P \vee Q)$$

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

Everyone likes ice cream"

$\forall x Likes(x, IceCream)$, is equivalent to $\neg \exists x \neg Likes(x, IceCream)$

$\Delta \vdash \ldots$

Properties of quantifiers

- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\exists x \forall y$ is not the same as $\forall y \exists x$
- $\exists x \forall y \text{ Loves}(x,y)$

“There is a person who loves everyone”

- $\forall y \exists x \text{ Loves}(x,y)$

“Everyone is loved by at least one person”

- Quantifier duality: each quantifier can be expressed using the other with the help of negation
- $\forall x \text{ Likes}(x, \text{IceCream}) \quad \neg \exists x \neg \text{Likes}(x, \text{IceCream})$
- $\exists x \text{ Likes}(x, \text{Broccoli}) \quad \neg \forall x \neg \text{Likes}(x, \text{Broccoli})$

Equality

- **Term1 = Term2** is true under a given model if and only if **Term1** and **Term2** refer to the same object
- E.g., definition of **Sibling** in terms of **Parent**:
 - $\forall x, y \text{ Sibling}(x,y) \Leftrightarrow$
 - $[\neg(x = y) \wedge \exists m,f \neg(m = f) \wedge \text{Parent}(m,x) \wedge \text{Parent}(f, x) \wedge \text{Parent}(m, y) \wedge \text{Parent}(f, y)]$

EXTENSIONS AND NOTATIONAL VARIATIONS

- Higher-order logic allows us to quantify over relations and functions as well as over objects.
- For example, in higher-order logic we can say that two objects are equal if and only if all properties applied to them are equivalent:

$$\forall x, y (x = y) \Leftrightarrow (\forall p p(x) \Leftrightarrow p(y))$$

- Or we could say that two functions are equal if and only if they have the same value for all arguments:

$$\forall f, g (f = g) \Leftrightarrow (\forall x f(x) = g(x))$$

λ -expression

- To turn the term $x^2 - y^2$ into a function, we need to say what its arguments are: is it the function where you square the first argument and subtract the square of the second argument, or vice versa.
- The operator λ (the Greek letter lambda) is traditionally used for this purpose. The function that takes the difference of the squares of its first and second argument is written as

$$(\lambda x, y \ x^2 - y^2)(25, 24) = 25^2 - 24^2 = 49$$

The uniqueness quantifier 3!

- use 3 to say that *some objects exist, but there is no concise way to say that a unique object satisfying some predicate exists.*
- *Some authors use the notation 3! To represent uniqueness quantifier.*
- $3! x \text{King}(x)$
- to mean "there exists a unique object x satisfying $\text{King}(x)$ " or more informally, "**there's exactly one King.**"

The uniqueness operator ι .

- It is convenient to use $\exists!$ to state uniqueness, but sometimes it is even more convenient to have a term representing the unique object directly.
- The notation $\iota xP(x)$ is commonly used for this.
- (The symbol ι is the Greek letter iota.) To say that "***the unique ruler of Freedonia is dead***"
- or
- equivalently "the r that is the ruler of Freedonia is dead," we would write: $\text{Dead}(\iota r \text{Ruler}(r, \text{Freedonia}))$
- $\exists! r \text{ Ruler}(r, \text{Freedonia}) \wedge \text{Dead}(r)$ Equivalent to

Notational variations

Syntax item	This book	Others
Negation (not)	$\neg P$	$\sim P \quad P$
Conjunction (and)	$P \wedge Q$	$P \& Q \quad P \quad Q \quad PQ \quad P, Q$
Disjunction (or)	$P \vee Q$	$P \mid Q \quad P; Q \quad P + Q$
Implication (if)	$P \Rightarrow Q$	$P \rightarrow Q \quad P \supset Q$
Equivalence (iff)	$P \Leftrightarrow Q$	$P \equiv Q \quad P \leftarrow Q$
Universal (all)	$\forall x \ P(x)$	$(\forall x)P(x) \quad \bigwedge x P(x) \quad P(x)$
Existential (exists)	$\exists x \ P(x)$	$(\exists x)P(x) \quad \bigvee x P(x) \quad P(Skolem_i)$
Relation	$R(x, y)$	$(R x y) \quad Rxy \quad xRy$

Examples of FOPL

- 1. All birds fly.
 $\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$
- 2. Every man respects his parent.
 $\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$
- 3. Some boys play cricket.
 $\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$
- 4. Not all students like both Mathematics and Science.
 $\neg \forall (x) [\text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})].$
- 5. Only one student failed in Mathematics.
 $\exists(x) [\text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [\neg(x==y) \wedge \text{student}(y) \rightarrow \neg\text{failed}(x, \text{Mathematics})]].$

Examples of FOPL

- $\forall x \cdot Man(x) \Rightarrow Mortal(x)$

‘For all x , if x is a man, then x is mortal.’

(i.e. all men are mortal)

- $\forall x \cdot Man(x) \Rightarrow \exists!y \cdot Woman(y) \wedge MotherOf(x, y)$

‘For all x , if x is a man, then there exists exactly one y such that y is a woman and the mother of x is y .’

(i.e., every man has exactly one mother).

- $\exists m \cdot Monitor(m) \wedge MonitorState(m, ready)$

‘There exists a monitor that is in a ready state.’

- $\forall r \cdot Reactor(r) \Rightarrow \exists t \cdot (100 \leq t \leq 1000) \wedge temp(r) = t$

‘Every reactor will have a temperature in the range 100 to 1000.’

Examples of FOPL

- $\exists n \cdot posInt(n) \wedge n = (n * n)$

“Some positive integer is equal to its own square.”

- $\exists c \cdot EUcountry(c) \wedge Borders(c, Albania)$

“Some EU country borders Albania.”

- $\forall m, n \cdot Person(m) \wedge Person(n) \Rightarrow \neg Superior(m, n)$

“No person is superior to another.”

- $\forall m \cdot Person(m) \Rightarrow \neg \exists n \cdot Person(n) \wedge Superior(m, n)$

(same as previous)

Video Reference links

S. NO	Description	Video link
1	This simple video covers the very basics of predicate logic (first order logic) used in knowledge representation	https://www.youtube.com/watch?v=4JNApj1wjsw&pp=QAA%3D
2	This video is in Continuation to Part 1 of Predicate logic (first order logic). Contains more examples on predicate logic and knowledge representation. please go through part 1 before starting this video.	https://www.youtube.com/watch?v=wzqrjYEJ6_w

Thank you

UNIT - 5

First Order Logic

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

UNIT-5

- **First-Order Logic:** (3 Hours)

Using First-Order Logic, A Simple Reflex Agent, Deducing Hidden Properties of the World, Toward a Goal-Based Agent, Building a Knowledge Base, Knowledge Engineering, Inference Rules Involving Quantifiers, Generalized Modus Ponens, Forward and Backward Chaining, Completeness, Resolution: A Complete Inference Procedure, Completeness of resolution

using first-order logic

- To represent family relationships and mathematical sets
- **The kinship domain** - domain of family relationships, or kinship.
- **The domain of sets** - A “predicate” is a statement involving variables over a specified “domain” (set).

The kinship domain

- This domain includes **facts** such as "Elizabeth is the mother of Charles" and "Charles is the father of William,"
- **rules** such as "If x is the mother of y and y is a parent of z, then x is a grandmother of z."
- **objects** in our domain are **people**.
- Properties / Predicates - **two unary predicates**, *Male* and *Female*.
- **binary predicates**: *Parent*, *Sibling*, *Brother*, *Sister*, *Child*, *Daughter*, *Son*, *Spouse*, *Wife*, *Husband*, *Grandparent*, *Grandchild*, *Cousin*, *Aunt*, *Uncle*.
- **functions** - *Mother* and *Father*, because every person has exactly one of each of these (at least according to nature's design).
- one's mother is one's female parent:
 $\forall m, c \ Mother(c)=m \Leftrightarrow Female(m) \wedge Parent(m, c)$
- One's husband is one's male spouse:
 $\forall w, h \ Husband(h, w) \Leftrightarrow Male(h) \wedge Spouse(h, w)$

The kinship domain

- Male and female are disjoint categories:

$$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

- Parent and child are inverse relations:

$$\forall p, c \text{ Parent}(p,c) \Leftrightarrow \text{Child}(c,p)$$

- A grandparent is a parent of one's parent:

$$\forall g, c \text{ Grandparent}(g,c) \Leftrightarrow \exists p \text{ Parent}(g,p) \wedge \text{Parent}(p,c)$$

- A sibling is another child of one's parents:

$$\forall x, y \text{ Sibling}(x,y) \Leftrightarrow x \neq y \wedge \exists p \text{ Parent}(p,x) \wedge \text{Parent}(p,y)$$

- Sibling hood is a symmetric relation

$$\forall x, y \text{ Sibling}(x,y) \Leftrightarrow \text{Sibling}(y,x)$$

Mathematical Facts

- **Axiom:** axioms to capture the basic facts about a domain.
- **Definition:** define other concepts in terms of those basic facts.
- axioms and definitions to prove **theorems**.
- In AI, **sentences** that are in the knowledge base initially are sometimes called "axioms," and it is common to talk of "definitions." as facts.

The domain of sets

- *EmptySet is a **constant**;*
- *Member and Subset are **predicates**;*
- *Intersection, Union, and Adjoin are **functions**.*
- *Adjoin is the function that adds one element to a set.*
- *Set is a **predicate** that is true only of sets.*
- *The following axioms provide the sets:*
- The only sets are the empty set and those made by adjoining something to a set.

$$\forall s \ Set(s) \Leftrightarrow (s = \text{EmptySet}) \vee (\exists x, s_2 \ Set(s_2) \wedge s = \text{Adjoin}(x, s_2))$$

- The empty set has no elements adjoined into it.

$$\neg \exists x, s \text{ Adjoin}(x, s) = \text{EmptySet}$$

- Adjoining an element already in the set has no effect:

$$\forall x, s \ Member(x, s) \Leftrightarrow s = \text{Adjoin}(x, s)$$

The domain of sets

- A set is a subset of another if and only if all of the first set's members are members of the second set.

$$\forall s_1, s_2 \ Subset(s_1, s_2) \Leftrightarrow (\forall x \ Member(x, s_1) \Rightarrow Member(x, s_2))$$

- Two sets are equal if and only if each is a subset of the other.

$$\forall s_1, s_2 \ (s_1 = s_2) \Leftrightarrow (Subset(s_1, s_2) \wedge Subset(s_2, s_1))$$

- An object is a member of the intersection of two sets if and only if it is a member of each of the sets.

$$\begin{aligned} \forall x, s_1, s_2 \ Member(x, Intersection(s_1, s_2)) \Leftrightarrow \\ Member(x, s_1) \text{ A } Member(x, s_2) \end{aligned}$$

- An object is a member of the union of two sets if and only if it is a member of either set.

$$\begin{aligned} \forall x, s_1, s_2 \ Member(x, Union(s_1, s_2)) \Leftrightarrow \\ Member(x, s_1) \text{ V } Member(x, s_2) \end{aligned}$$

Set notations in Predicate logic

$\emptyset = \text{EmptySet}$

$[] = \text{Nil}$

$\{x\} = \text{Adjoin}(x, \text{EmptySet})$

$[x] = \text{Cons}(x, \text{Nil})$

$\{x, y\} = \text{Adjoin}(x, \text{Adjoin}(y, \text{EmptySet}))$

$[x, y] = \text{Cons}(x, \text{Cons}(y, \text{Nil}))$

$\{x, y, s\} = \text{Adjoin}(x, \text{Adjoin}(y, s))$

$[x, y][l] = \text{Cons}(x, \text{Cons}(y, l))$

$r \cup s = \text{Union}(r, s)$

$r \cap s = \text{Intersection}(r, s)$

$x \in s = \text{Member}(x, s)$

$r \subset s = \text{Subset}(r, s)$

Simple reflex agents

Agent

Fast but too simple

- *NO MEMORY
- *Fails if environment
- is partially observable / works in fully observable
- *No previous history
- *Decision based only on current situation

example: vacuum cleaner world,
yan of a person, if room_temp>45
then switch on Ac and set
room_temp to 26 degree.

Sensors

What the world
is like now

Actuators

Environment

A SIMPLE REFLEX AGENT

- An agent has rules directly connecting percepts to actions.
- These rules resemble reflexes or instincts.
- For example, if the agent sees a glitter, it should do a grab in order to pick up the gold:

$$\forall s, b, u, c, t \text{ } Percept([s, b, \text{Glitter}, u, c], t) \Rightarrow Action(Grab, t)$$

- The connection between percept and action can be mediated by rules for perception

$$\forall b, g, u, c, t \text{ } Percept([Stench, b, g, u, c], t) \Rightarrow Stench(t)$$

$$\forall s, g, u, c, t \text{ } Percept([s, Breeze, g, u, c], t) \Rightarrow Breeze(t)$$

$$\forall s, b, u, c, t \text{ } Percept([s, b, \text{Glitter}, u, c], t) \Rightarrow AtGold(t)$$

- Then a connection can be made from these predicates to action choices:

$$\forall t \text{ } AtGold(t) \Rightarrow Action(Grab, t)$$

DEDUCING HIDDEN PROPERTIES OF THE WORLD

- Once the agent knows where it is, it can associate qualities with the *places rather than just the situations*.
- for example, one might say that if the agent is at a place and perceives a breeze, then that place is breezy,

$$\forall l, s \text{At}(\text{Agent}, l, s) \wedge \text{Breeze}(s) \rightarrow \text{Breezy}(l)$$

- if the agent perceives a stench, then that place is smelly:

$$\forall l, s \text{At}(\text{Agent}, l, s) \wedge \text{Stench}(s) \rightarrow \text{Smelly}(l)$$

- Having discovered which places are breezy or smelly (and, very importantly, *not smelly or not breezy*), *the agent can deduce where the pits are, and where the wumpus is*.
- Furthermore, it can deduce which squares are safe to move to (we use the predicate OK to represent this), and can use this information to hunt for the gold.*

Synchronic rules

- **synchronic ("same time") rules**, are the rules they relate properties of a world state to other properties of the same world state.
- There are two main kinds of synchronic rules:
- **Causal rules:**
- Causal rules reflect the assumed direction of causality in the world: some hidden property of the world causes certain percepts to be generated.
- For example, we might have rules stating that squares adjacent to wumpuses are smelly and squares adjacent to pits are breezy:

$$\forall l_1, l_2, s \ At(Wumpus, l_1, s) \wedge \text{Adjacent}(l_1, l_2) \Rightarrow \text{Smelly}(l_2)$$

$$\forall l_1, l_2, s \ At(Pit, l_1, s) \wedge \text{Adjacent}(l_1, l_2) \Rightarrow \text{Breezy}(l_2)$$

- Systems that reason with causal rules are called **model-based reasoning systems**.

Synchronic rules

- **Diagnostic rules:**
- Diagnostic rules infer the presence of hidden properties directly from the percept-derived information.
- To deduce the presence of breezy at location L.

$$\forall l, s \text{At}(\text{Agent}, l, s) \wedge \text{Breeze}(s) \rightarrow \text{Breezy}(l)$$

- To deduce the presence of smelly at location L

$$\forall l, s \text{At}(\text{Agent}, l, s) \wedge \text{Stench}(s) \rightarrow \text{Smelly}(l)$$

- For deducing the presence of wumpuses, a diagnostic rule: if a location is smelly, then the wumpus must either be in that location or in an adjacent location:

$$\begin{aligned} \forall l_1, s \text{ Smelly}(l_1) \Rightarrow \\ (\exists l_2 \text{At}(\text{Wumpus}, l_2, s) \wedge (l_2 = l_1 \vee \text{Adjacent}(l_1, l_2))) \end{aligned}$$

- a square can be OK even when smells and breezes abound.

$$\forall x, t \ (\neg \text{At}(\text{Wumpus}, x, t) \wedge \neg \text{Pit}(x)) \Leftrightarrow \text{OK}(x)$$

TOWARD A GOAL-BASED AGENT

- Once the gold is found, the policies need to change radically. The aim now is to return to the start square as quickly as possible.
- We infer that the agent now has the **goal of being at location [1,1]**:
 $\forall s \text{ Holding(Gold, } s) \rightarrow \text{GoalLocation}([1,1], s)$
- The presence of an explicit goal allows the agent to work out a sequence of actions that will achieve the goal. There are at least **three ways** to find such a sequence:
- Inference:** write axioms that will allow us to ASK the *KB* for a sequence of actions that is guaranteed to achieve the goal safely.
- Search:** We can use a best-first search procedure to find a path to the goal.
- Planning:** This involves the use of special-purpose reasoning systems designed to reason about actions.

knowledge engineering.

- The process of building a knowledge base is called **knowledge engineering**.
- A knowledge engineer is someone who investigates a particular domain, determines what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain.
- Often, the knowledge engineer is trained in representation but is not an expert in the domain at hand, be it circuit design, space station mission scheduling, or whatever.
- The knowledge engineer will usually interview the real experts to become educated about the domain and to elicit the required knowledge, in a process called **knowledge acquisition**.

Ontological Engineering

- The ways to represent time, change, objects, substances, events, actions, money, measures, and so on in more general representation as in supermarket domain.
- These are important because they show up in one form or another in every domain.
- Representing these very general concepts is sometimes called **ontological engineering**.
- Steps in Knowledge Engineering**

<i>Knowledge Engineering</i>	<i>Programming</i>
(1) Choosing a logic	Choosing a programming language
(2) Building a knowledge base	Writing a program
(3) Implementing the proof theory	Choosing or writing a compiler
(4) Inferring new facts	Running a program

General way to represent good KB

- In a good knowledge base $BearOfVerySmallBrain(Pooh)$ would be replaced by something like the following:
- 1. Pooh is a bear; bears are animals; animals are physical things.

$Bear(Pooh)$

$\forall b \ Bear(b) \Rightarrow Animal(b)$

$\forall a \ Animal(a) \Rightarrow PhysicalThing(a)$

- 2. Pooh has a very small brain.
- $RelativeSize(BrainOf(Pooh), BrainOf(TypicalBear)) = very(Small)$

Knowledge Engineer must understand

- domain – objects and their relationship.
- representational language – to encode facts.
- implementation of inference procedure – answer queries within reasonable time.

Five step Methodology – to develop the KB.

- 1) *Decide what to talk about*
- 2) *Decide on a vocabulary of predicates, functions, and constants.*
- 3) *Encode general knowledge about the domain.*
- 4) *Encode a description of the specific problem instance.*
- 5) *Pose queries to the inference procedure and get answers.*

KE-Five step Methodology

- *1) Decide what to talk about. Understand the domain well enough to know which objects and facts need to be talked about, and which can be ignored.*
- *2) Decide on a vocabulary of predicates, functions, and constants. That is, translate the important domain-level concepts into logic-level names.*
- *Size be a function or a predicate? Bigness be a better name than Size?*
- *Small be a constant or a predicate?*
- Is *Small* a measure of relative size or absolute size?
- Once the choices have been made, the result is a vocabulary that is known as the **ontology of the domain**.

KE-Five step Methodology

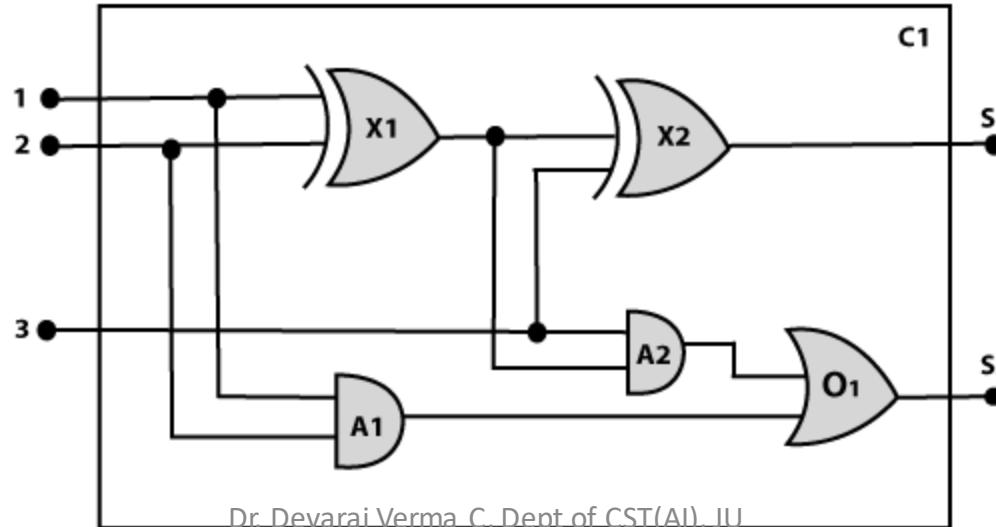
- *3) Encode general knowledge about the domain. The ontology is an informal list of the concepts in a domain.*
- By writing logical sentences or **axioms about the terms in the ontology**, we accomplish two goals:
 - 1) we make the terms more precise so that humans will agree on their interpretation.
 - 2) we make it possible to run inference procedures to automatically derive consequences from the knowledge base.
- Once the **axioms are in place**, we can say that a **knowledge base has been produced**.
- *4) Encode a description of the specific problem instance. If the ontology is well thought out, this step will be easy.*
- It will mostly involve writing simple atomic sentences about instances of concepts that are already part of the ontology.

KE-Five step Methodology

- *5) Pose queries to the inference procedure and get answers.*
- *This is where the reward is:* we can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing.

KE Process – Example

- A digital circuit C1 (one bit full adder ckt) , with three inputs and two outputs, containing two XOR gates, two AND gates and one OR gate.
- The inputs are bit values to be added, and the outputs are the sum bit and the carry bit.
- The first output is the sum(s_1), and the second output is a carry bit (s_2) for the next adder.
- The goal is to provide an analysis that determines if the circuit is in fact an adder, and that can answer questions about the value of current flow at various points in the circuit



KE Process – Example

- **1) Decide what to talk about:** we need to talk about *circuits, their terminals (in and out), and the signals at the terminals.*
- *To determine what these signals will be, we need to know about individual gates, and gate types: AND, OR, XOR, and NOT.*
- **2) Decide on a vocabulary:** The next step is to choose functions, predicates, and constants to name them. We will start from individual gates and move up to circuits.
- Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**.
- The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**.
- Circuits will be identified by a predicate: **Circuit (C1)**.
- For the terminal, we will use predicate: **Terminal(x)**.
- For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.
- The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

KE Process – Example

- The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.
- We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.
- **3) Encode general rules:** good ontology sign => general rules that need to be specified.
- A sign that we have a good vocabulary is that each rule can be stated clearly and concisely.
- With our example, we need only seven simple rules to define the axioms / sentences:
- If two terminals are connected then they have the same input signal, it can be represented as:
- $\forall t_1, t_2 \text{ Terminal}(t_1) \wedge \text{Terminal}(t_2) \wedge \text{Connect}(t_1, t_2) \rightarrow \text{Signal}(t_1) = \text{Signal}(t_2)$.
- Signal at every terminal will have either value 0 or 1, it will be represented as:
- $\forall t \text{ Terminal}(t) \rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0$.

KE Process – Example

- Connect predicates are commutative:
- $\forall t_1, t_2 \text{Connect}(t_1, t_2) \rightarrow \text{Connect}(t_2, t_1)$.
- Representation of types of gates:
- $\forall g \text{Gate}(g) \wedge r = \text{Type}(g) \rightarrow r = \text{OR} \vee r = \text{AND} \vee r = \text{XOR} \vee r = \text{NOT}$.
- Output of AND gate will be zero if and only if any of its input is zero.
- $\forall g \text{Gate}(g) \wedge \text{Type}(g) = \text{AND} \rightarrow \text{Signal}(\text{Out}(1, g)) = 0 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 0$.
- Output of OR gate is 1 if and only if any of its input is 1:
- $\forall g \text{Gate}(g) \wedge \text{Type}(g) = \text{OR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 1$
- Output of XOR gate is 1 if and only if its inputs are different:
- $\forall g \text{Gate}(g) \wedge \text{Type}(g) = \text{XOR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{In}(2, g))$.
- Output of NOT gate is invert of its input:
- $\forall g \text{Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{Out}(1, g))$.

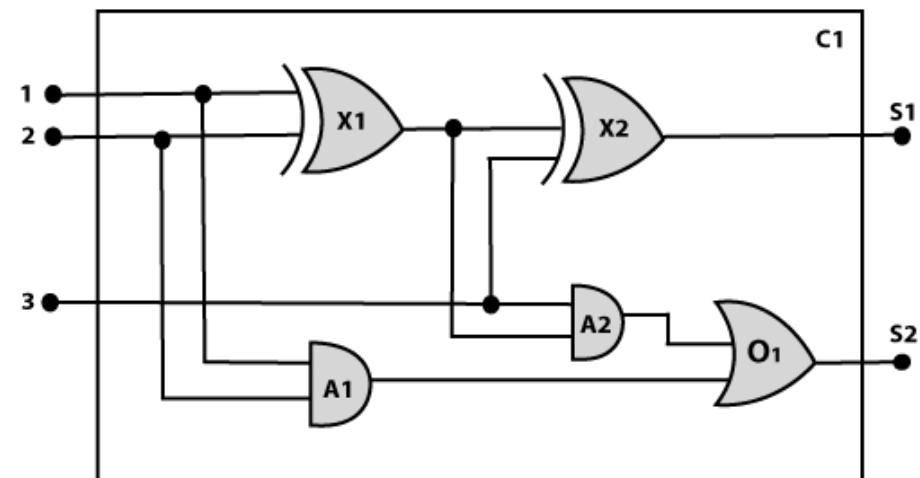
KE Process – Example

- All the gates in the above circuit have two inputs and one output (except NOT gate).
- $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Arity}(g, 1, 1)$
- $\forall g \text{ Gate}(g) \wedge r = \text{Type}(g) \wedge (r = \text{AND} \vee r = \text{OR} \vee r = \text{XOR}) \rightarrow \text{Arity}(g, 2, 1)$.
- All gates are logic circuits:
- $\forall g \text{ Gate}(g) \rightarrow \text{Circuit}(g)$.
- **4) Encode the specific instance**
- This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.
- For the given circuit C1, we can encode the problem instance in atomic sentences as below:
- Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:
 - For XOR gate: $\text{Type}(x1) = \text{XOR}$, $\text{Type}(x2) = \text{XOR}$
 - For AND gate: $\text{Type}(A1) = \text{AND}$, $\text{Type}(A2) = \text{AND}$
 - For OR gate: $\text{Type}(O1) = \text{OR}$.

KE Process – Example

connections between all the gates:

- $\text{Connected}(\text{Out}(1, X1), \text{In}(1, X2))$
- $\text{Connected}(\text{Out}(1, X1), \text{In}(2, A2))$
- $\text{Connected}(\text{Out}(1, A2), \text{In}(1, O1))$
- $\text{Connected}(\text{Out}(1, A1), \text{In}(2, O1))$
- $\text{Connected}(\text{Out}(1, X2), \text{Out}(1, CO))$
- $\text{Connected}(\text{Out}(1, O1), \text{Out}(2, CO))$
- $\text{Connected}(\text{In}(1, C1), \text{In}(1, X1))$
- $\text{Connected}(\text{In}(1, C1), \text{In}(1, A1))$
- $\text{Connected}(\text{In}(2, C1), \text{In}(2, X1))$
- $\text{Connected}(\text{In}(2, CO), \text{In}(2, A1))$
- $\text{Connected}(\text{In}(3, C1), \text{In}(2, X2))$
- $\text{Connected}(\text{In}(3, C1), \text{In}(1, A2))$



KE Process – Example

- **5) Pose queries to the inference procedure**
- In this step, we will find all the possible set of values of all the terminal for the adder circuit.
- The first query will be:
- **Q1)** What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?
- $\exists i_1, i_2, i_3 \text{ Signal } (\text{In}(1, C1))=i_1 \wedge \text{Signal } (\text{In}(2, C1))=i_2 \wedge \text{Signal } (\text{In}(3, C1))=i_3 \wedge \text{Signal } (\text{Out}(1, C1))=0 \wedge \text{Signal } (\text{Out}(2, C1))=1$
- The answer is
 - $(i_1 = \text{On} \wedge i_2 = \text{On} \wedge i_3 = \text{Off}) \vee$
 - $(i_1 = \text{On} \wedge i_2 = \text{off} \wedge i_3 = \text{On}) \vee$
 - $(i_1 = \text{Off} \wedge i_2 = \text{On} \wedge i_3 = \text{On})$

Video Reference links

S. NO	Description	Video link
1	This simple video explains the AI and Knowledge engineer components	https://www.youtube.com/watch?v=7Rz47HnTZ_k
2	This video provides the Information on Steps of knowledge engineering applied to specific problem. Reference:1 before starting this video.	https://www.youtube.com/watch?v=pZx2Hus4B6s
3	This video gives the in depth understanding of Forward and Backward Chaining in Artificial Intelligence with example.	https://www.youtube.com/watch?v=Dso6EKzvXHo

Thank you

First-Order Logic

Limitations of propositional logic

- Suppose you want to say “All humans are mortal”
 - In propositional logic, you would need ~6.7 billion statements
- Suppose you want to say “Some people can run a marathon”
 - You would need a disjunction of ~6.7 billion statements

First-order logic

- Propositional logic assumes the world consists of atomic **facts**
- First-order logic assumes the world contains objects, relations, and functions

Syntax of FOL

- **Constants:** John, Sally, 2, ...
- **Variables:** x, y, a, b, ...
- **Predicates:** Person(John), Siblings(John, Sally), IsOdd(2), ...
- **Functions:** MotherOf(John), Sqrt(x), ...
- **Connectives:** \neg , \wedge , \vee , \Rightarrow , \Leftrightarrow
- **Equality:** =
- **Quantifiers:** \forall , \exists

- **Term:** Constant or Variable or Function(Term₁, ..., Term_n)
- **Atomic sentence:** Predicate(Term₁, ..., Term_n) or Term₁ = Term₂
- **Complex sentence:** made from atomic sentences using connectives and quantifiers

Semantics of FOL

- Sentences are true with respect to a **model** and an **interpretation**
- Model contains objects (**domain elements**) and relations among them
- Interpretation specifies referents for
 - constant symbols → objects
 - predicate symbols → relations
 - function symbols → functional relations
- An atomic sentence **Predicate(Term₁, … , Term_n)** is true iff the **objects** referred to by **Term₁, … , Term_n** are in the **relation** referred to by predicate

Universal quantification

- $\forall x P(x)$
- Example: “Everyone at UNC is smart”
 $\forall x \text{At}(x, \text{UNC}) \Rightarrow \text{Smart}(x)$
Why not $\forall x \text{At}(x, \text{UNC}) \wedge \text{Smart}(x)$?
- Roughly speaking, equivalent to the conjunction of all possible instantiations of the variable:
 $\text{At}(\text{John}, \text{UNC}) \Rightarrow \text{Smart}(\text{John}) \wedge \dots$
 $\text{At}(\text{Richard}, \text{UNC}) \Rightarrow \text{Smart}(\text{Richard}) \wedge \dots$
- $\forall x P(x)$ is true in a model m iff $P(x)$ is true with x being each possible object in the model

Existential quantification

- $\exists x P(x)$
- Example: “Someone at UNC is smart”
 $\exists x \text{At}(x, \text{UNC}) \wedge \text{Smart}(x)$
Why not $\exists x \text{At}(x, \text{UNC}) \Rightarrow \text{Smart}(x)$?
- Roughly speaking, equivalent to the **disjunction** of all possible instantiations:
 $[\text{At}(\text{John}, \text{UNC}) \wedge \text{Smart}(\text{John})] \vee$
 $[\text{At}(\text{Richard}, \text{UNC}) \wedge \text{Smart}(\text{Richard})] \vee \dots$
- $\exists x P(x)$ is true in a model m iff $P(x)$ is true with x being some possible object in the model

Properties of quantifiers

- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\exists x \forall y$ is **not** the same as $\forall y \exists x$
 $\exists x \forall y \text{ Loves}(x,y)$
“There is a person who loves everyone”
 $\forall y \exists x \text{ Loves}(x,y)$
“Everyone is loved by at least one person”
- **Quantifier duality:** each quantifier can be expressed using the other with the help of negation
 $\forall x \text{ Likes}(x,\text{IceCream})$
 $\exists x \text{ Likes}(x,\text{Broccoli})$

Equality

- **Term₁ = Term₂** is true under a given model if and only if **Term₁** and **Term₂** refer to the same object

- E.g., definition of **Sibling** in terms of **Parent**:

$$\forall x,y \text{ Sibling}(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f \neg(m = f) \wedge \text{Parent}(m,x) \wedge \text{Parent}(f,x) \wedge \text{Parent}(m,y) \wedge \text{Parent}(f,y)]$$

Using FOL: The Kinship Domain

- Brothers are siblings
 $\forall x,y \text{ Brother}(x,y) \Rightarrow \text{Sibling}(x,y)$
- “Sibling” is symmetric
 $\forall x,y \text{ Sibling}(x,y) \Leftrightarrow \text{Sibling}(y,x)$
- One's mother is one's female parent
 $\forall m,c (\text{Mother}(c) = m) \Leftrightarrow (\text{Female}(m) \wedge \text{Parent}(m,c))$

Why “First order”?

- FOL permits quantification over variables
- Higher order logics permit quantification over functions and predicates:

$$\forall P, x [P(x) \vee \neg P(x)]$$

$$\forall x, y (x=y) \Leftrightarrow [\forall P (P(x) \Leftrightarrow P(y))]$$

Inference in FOL

- All rules of inference for propositional logic apply to first-order logic
- We just need to reduce FOL sentences to PL sentences by instantiating variables and removing quantifiers

Reduction of FOL to PL

- Suppose the KB contains the following:

$$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

King(John) Greedy(John) Brother(Richard,John)

- How can we reduce this to PL?
- Let's instantiate the universal sentence in all possible ways:

$$\text{King}(John) \wedge \text{Greedy}(John) \Rightarrow \text{Evil}(John)$$

$$\text{King}(Richard) \wedge \text{Greedy}(Richard) \Rightarrow \text{Evil}(Richard)$$

King(John) Greedy(John) Brother(Richard,John)

- The KB is *propositionalized*

- Proposition symbols are King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction of FOL to PL

- What about existential quantification, e.g.,
 $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$?
- Let's instantiate the sentence with a new constant that doesn't appear anywhere in the KB:
 $\text{Crown}(\text{C}_1) \wedge \text{OnHead}(\text{C}_1, \text{John})$

Substitution

- **Substitution** of variables by *ground terms*:

SUBST({v/g},P)

- Result of $\text{SUBST}(\{x/\text{Harry}, y/\text{Sally}\}, \text{Loves}(x,y))$:
 $\text{Loves}(\text{Harry}, \text{Sally})$
- Result of $\text{SUBST}(\{x/\text{John}\}, \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x))$:
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

Universal instantiation (UI)

- A universally quantified sentence entails every instantiation of it:

$$\frac{\forall v P(v)}{\text{SUBST}(\{v/g\}, P(v))}$$

for any variable v and ground term g

- E.g., $\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$ yields:
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
 $\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$

Existential instantiation (EI)

- An existentially quantified sentence entails the instantiation of that sentence with a new constant:

$$\frac{\exists v P(v)}{\text{SUBST}(\{v/C\}, P(v))}$$

for any sentence P , variable v , and constant C that does not appear elsewhere in the knowledge base

- E.g., $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$ yields:

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

provided C_1 is a new constant symbol, called a *Skolem constant*

Propositionalization

- Every FOL KB can be *propositionalized* so as to preserve entailment
 - A ground sentence is entailed by the new KB iff it is entailed by the original KB
- **Idea:** propositionalize KB and query, apply resolution, return result
- **Problem:** with function symbols, there are infinitely many ground terms
 - For example, $\text{Father}(X)$ yields $\text{Father}(\text{John})$, $\text{Father}(\text{Father}(\text{John}))$, $\text{Father}(\text{Father}(\text{Father}(\text{Father}(\text{John}))))$, etc.

Propositionalization

- **Theorem** (Herbrand 1930):
 - If a sentence α is entailed by an FOL KB, it is entailed by a *finite* subset of the propositionalized KB
- **Idea:** For $n = 0$ to Infinity do
 - Create a propositional KB by instantiating with depth- n terms
 - See if α is entailed by this KB
- **Problem:** works if α is entailed, loops if α is not entailed
- **Theorem** (Turing 1936, Church 1936):
 - Entailment for FOL is **semidecidable**: algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence

Review: FOL inference

Propositionalization

- Example KB:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

King(John)

$\forall y \text{ Greedy}(y)$

Brother(Richard, John)

- What will propositionalization produce?

King(John) \wedge Greedy(John) \Rightarrow Evil(John)

King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)

Greedy(John) Greedy(Richard)

King(John) Brother(Richard, John)

- But what if all we want is to prove Evil(John)?

Generalized Modus Ponens (GMP)

$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)$

such that $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$ for all i

$\text{SUBST}(\theta, q)$

- Used with **definite clauses** (exactly one positive literal)
- All variables assumed universally quantified
- **Example:**

p_1' is King(John) p_1 is King(x)

p_2' is Greedy(y) p_2 is Greedy(x)

θ is {x/John,y/John}

q is Evil(x)

$\text{SUBST}(\theta, q)$ is Evil(John)

Unification

UNIFY(α, β) = θ means that **SUBST(θ, α) = SUBST(θ, β)**

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,Mary)	
Knows(John,x)	Knows(y,z)	

- Standardizing apart eliminates overlap of variables
- Most general unifier

Inference with GMP

$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)$

such that $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$ for all i

$\text{SUBST}(\theta, q)$

- **Forward chaining**
 - Like search: keep proving new things and adding them to the KB until we can prove q
- **Backward chaining**
 - Find p_1, \dots, p_n such that knowing them would prove q
 - Recursively try to prove p_1, \dots, p_n

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove that Col. West is a criminal

Example knowledge base

It is a crime for an American to sell weapons to hostile nations:

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

Nono has some missiles

$$\exists x \text{ Owns}(Nono,x) \wedge \text{Missile}(x)$$

$$\text{Owns}(Nono,M_1) \wedge \text{Missile}(M_1)$$

All of its missiles were sold to it by Colonel West

$$\text{Missile}(x) \wedge \text{Owns}(Nono,x) \Rightarrow \text{Sells}(West,x,Nono)$$

Missiles are weapons:

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

An enemy of America counts as “hostile”:

$$\text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x)$$

West is American

$$\text{American}(West)$$

The country Nono is an enemy of America

$$\text{Enemy}(Nono,\text{America})$$

Forward chaining proof

American(West)

Missile(M₁)

Owns(Nono,M₁)

Enemy(Nono,America)

$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

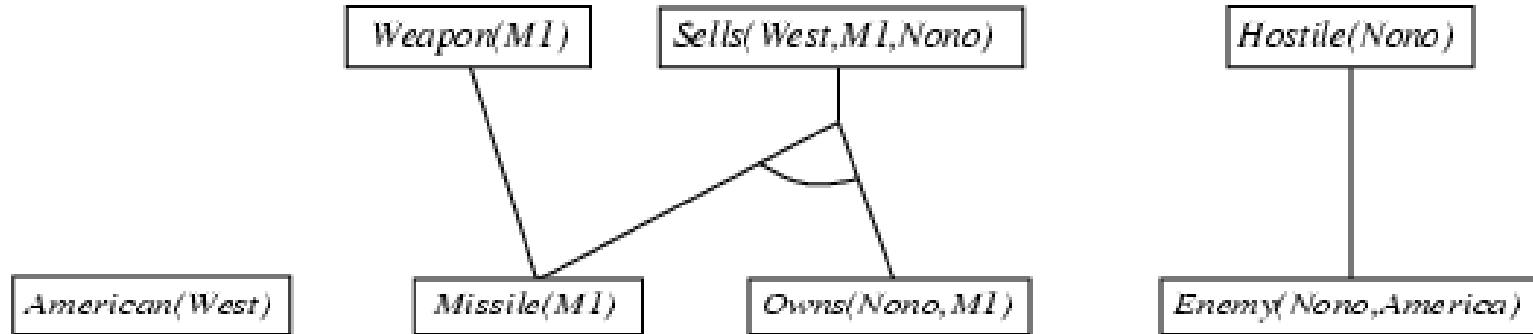
$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Forward chaining proof



$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

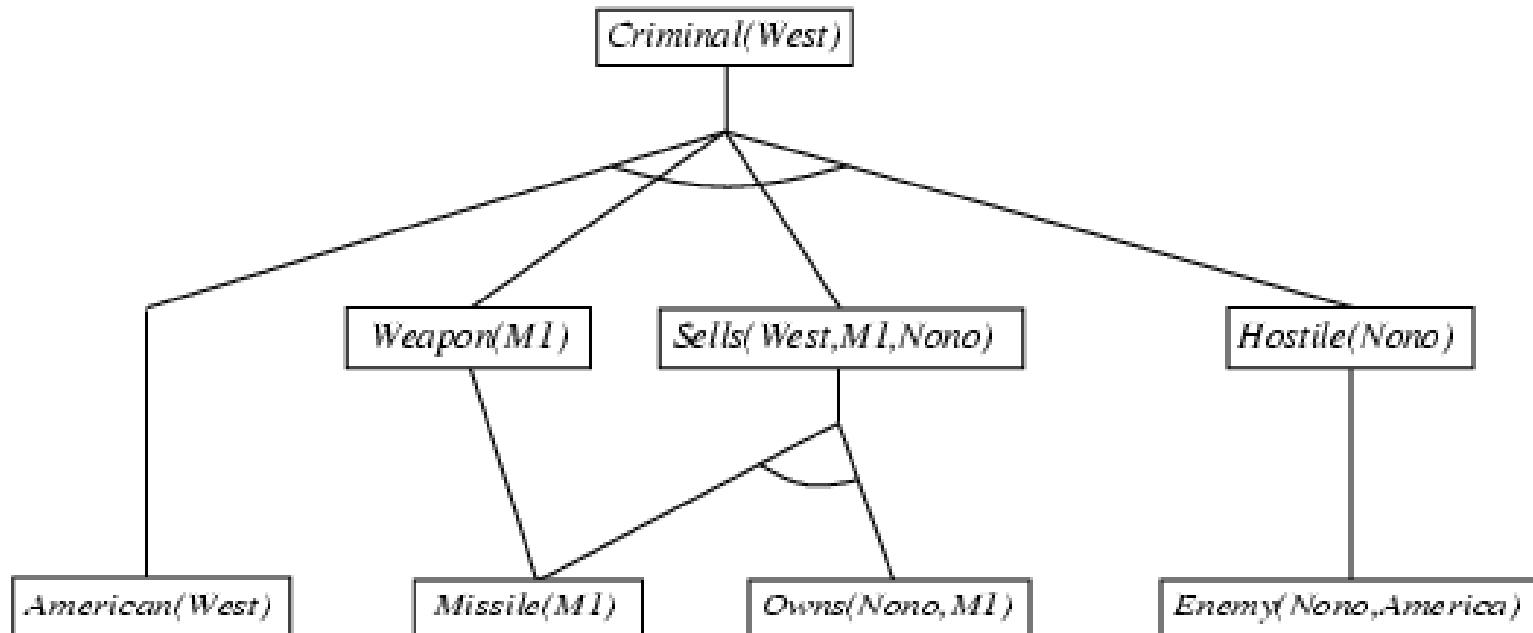
$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Forward chaining proof



$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Backward chaining example

Criminal(West)

American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)

Owns(Nono,M₁) \wedge Missile(M₁)

Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)

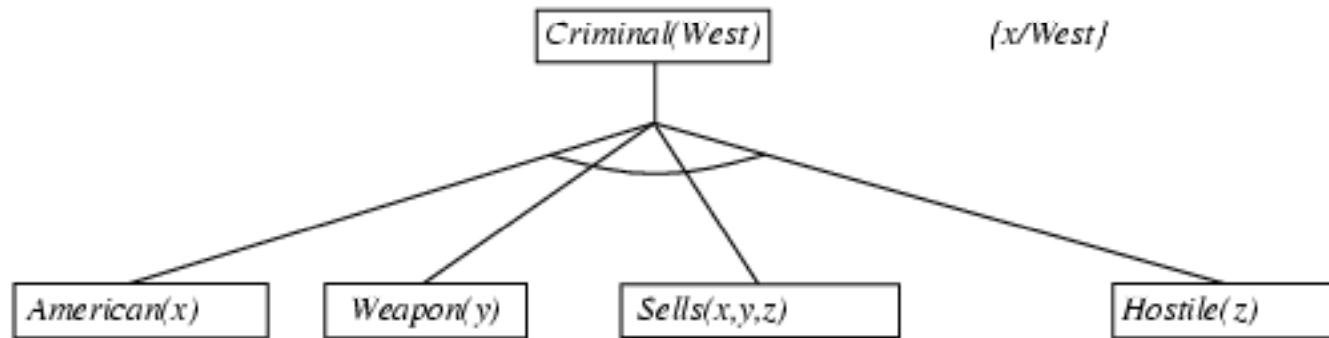
Missile(x) \Rightarrow Weapon(x)

American(West)

Enemy(x,America) \Rightarrow Hostile(x)

Enemy(Nono,America)

Backward chaining example



$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono,M_1) \wedge Missile(M_1)$

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

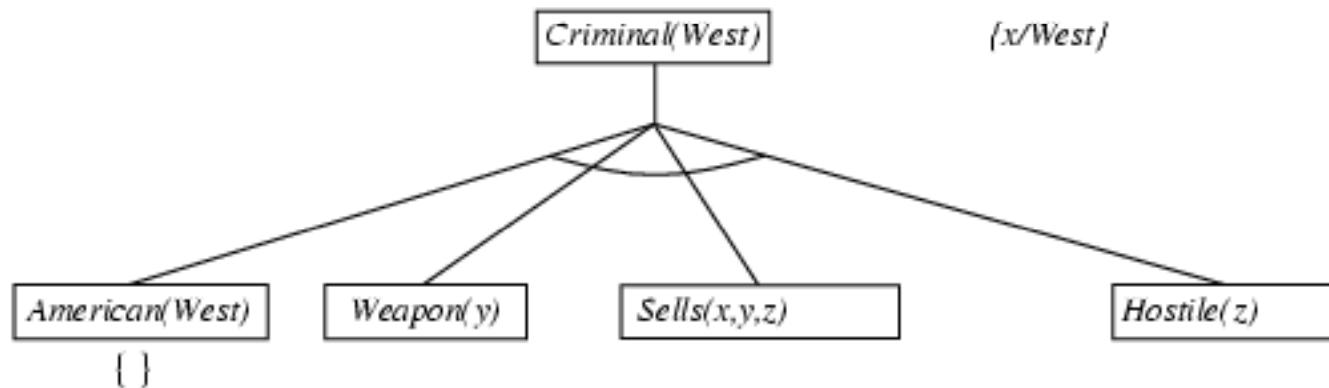
$Missile(x) \Rightarrow Weapon(x)$

$American(West)$

$Enemy(x,America) \Rightarrow Hostile(x)$

$Enemy(Nono,America)$

Backward chaining example



$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

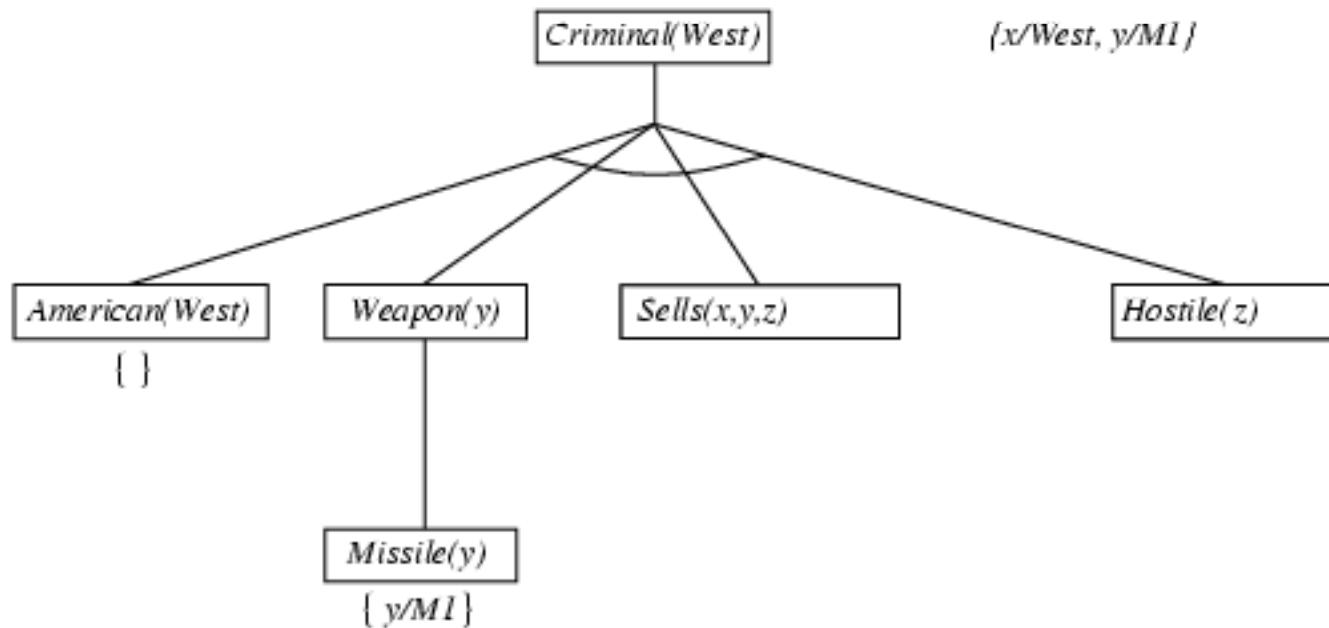
$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Backward chaining example



$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

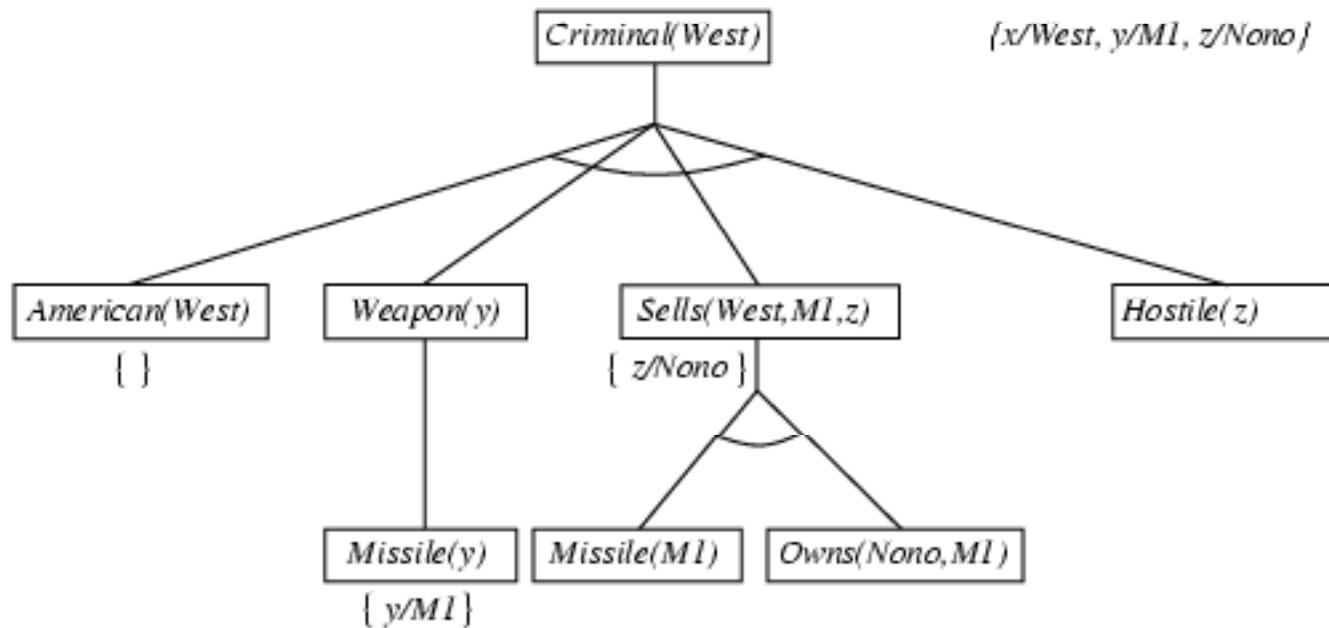
$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Backward chaining example



$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1) \wedge Missile(M_1)$

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

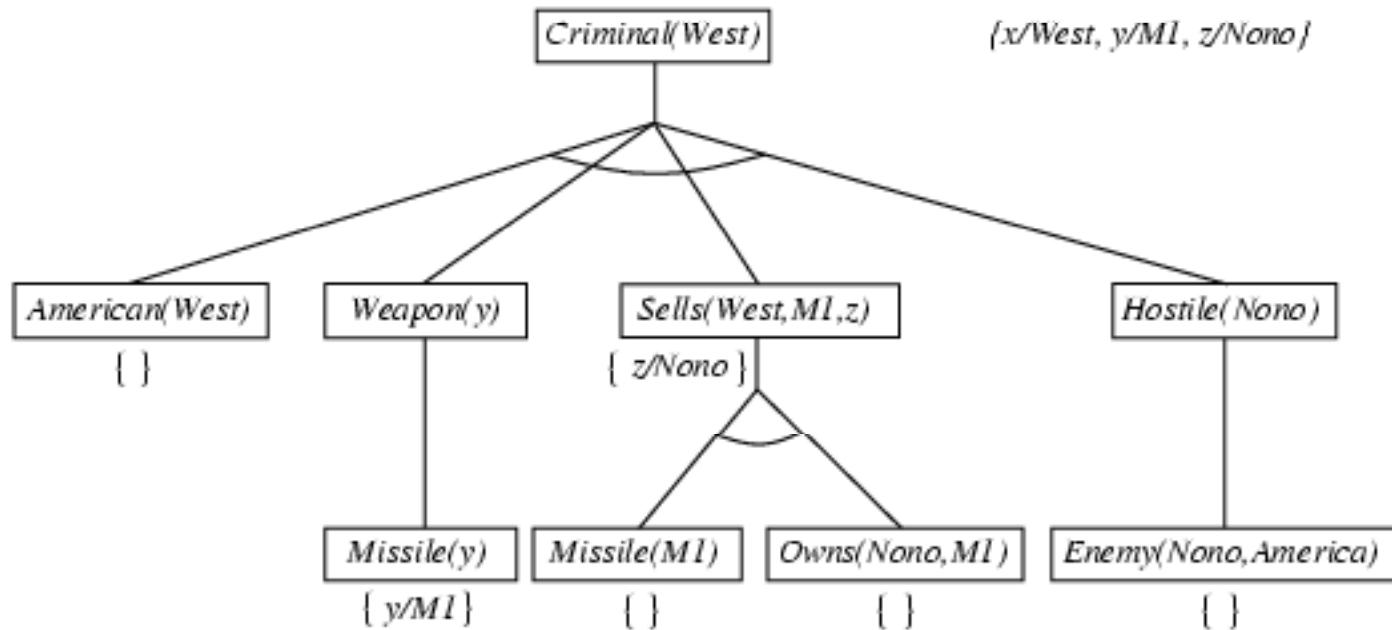
$Missile(x) \Rightarrow Weapon(x)$

$American(West)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$Enemy(Nono, America)$

Backward chaining example



$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x,y,z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

$\text{Owns}(\text{Nono}, M_1) \wedge \text{Missile}(M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

$\text{American}(\text{West})$

$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

$\text{Enemy}(\text{Nono}, \text{America})$

Backward chaining algorithm

function FOL-BC-ASK($KB, goals, \theta$) **returns** a set of substitutions

inputs: KB , a knowledge base

$goals$, a list of conjuncts forming a query (θ already applied)

θ , the current substitution, initially the empty substitution $\{ \}$

local variables: $answers$, a set of substitutions, initially empty

if $goals$ is empty **then return** $\{\theta\}$

$q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$

for each sentence r **in** KB

 where STANDARDIZE-APART(r) = $(p_1 \wedge \dots \wedge p_n \Rightarrow q)$

 and $\theta' \leftarrow \text{UNIFY}(q, q')$ succeeds

$new_goals \leftarrow [p_1, \dots, p_n | REST(goals)]$

$answers \leftarrow \text{FOL-BC-ASK}(KB, new_goals, \text{COMPOSE}(\theta', \theta)) \cup answers$

return $answers$

Resolution: FOL version

$$\begin{array}{ll} p_1 \vee \dots \vee p_k, & q_1 \vee \dots \vee q_n \\ \text{such that } \text{UNIFY}(p_i, \neg q_j) = \theta \end{array}$$

$$\text{SUBST}(\theta, p_1 \vee \dots \vee p_{i-1} \vee p_{i+1} \vee \dots \vee p_k \vee q_1 \vee \dots \vee q_{j-1} \vee q_{j+1} \vee \dots \vee q_n)$$

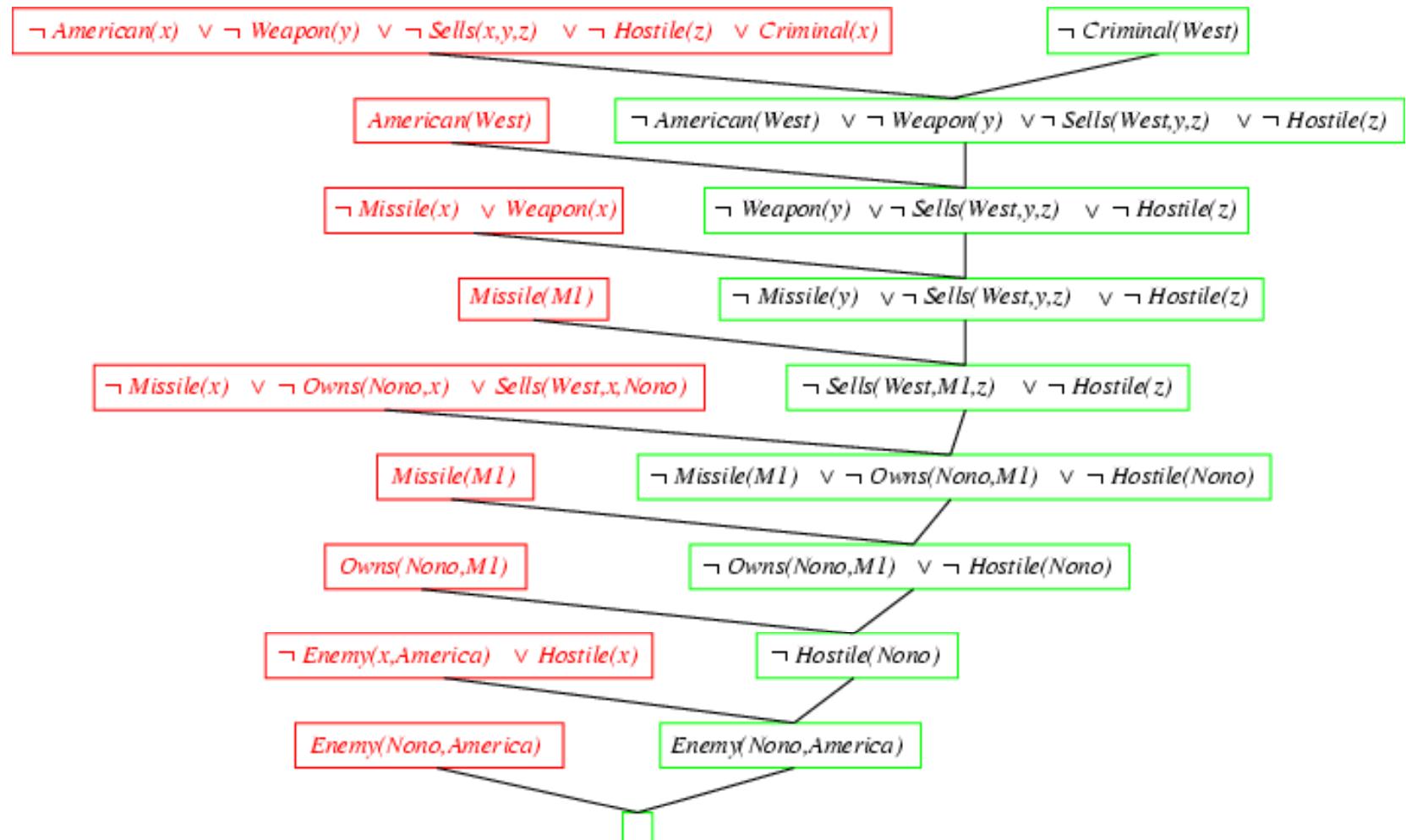
- For example,

$$\frac{\begin{array}{c} \neg \text{Rich}(x) \vee \text{Unhappy}(x) \\ \text{Rich(Ken)} \end{array}}{\text{Unhappy(Ken)}}$$

with $\theta = \{x/\text{Ken}\}$

- Apply resolution steps to $\text{CNF}(\text{KB} \wedge \neg a)$; complete for FOL

Resolution proof: definite clauses



UNIT - 4

ACTING LOGICALLY - PLANNING

Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall

UNIT-4

- **Acting logically:** **(9 Hours)**
- A Simple Planning Agent, From Problem Solving to Planning, Planning in Situation Calculus, Basic Representations for Planning, A Partial-Order Planning Algorithm, Planning with Partially Instantiated Operators, Knowledge Engineering for Planning, Practical Planners, Hierarchical Decomposition, Analysis of Hierarchical Decomposition, More Expressive Operator Descriptions, Resource Constraints, Planning and Acting, Conditional Planning, A Simple Re-planning Agent, Fully Integrated Planning and Execution. Chap 11, Chap 12, Chap 13

A Simple Planning Agent

- planning agent that is very similar to a problem-solving agent in that it constructs plans that achieve its goals, and then executes them.
- partial-order planning (POP) algorithm, which searches through the space of plans to find one that is guaranteed to succeed.
- partially ordered plan representation allows a planning agent to handle quite complicated domains.

A SIMPLE PLANNING AGENT

- IDEAL-PLANNER can be any of the planners (planning algorithm).
- function STATE-DESCRIPTION, which takes a percept as input and returns an initial state description in the format required by the planner.
- function MAKE-GOAL-QUERY, which is used to ask the knowledge base what the next goal should be.
- Agent must deal with the case where the goal is infeasible and
- case where the complete plan is in fact empty, because the goal is already true in the initial state.
- The agent interacts with the environment in a minimal way—it uses its percepts to define the initial state and thus the initial goal, but thereafter it simply follows the steps in the plan it has constructed.

Simple planning agent

```

function SIMPLE-PLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, a plan, initially NoPlan
          t, a counter, initially 0, indicating time
  local variables: G, a goal
                      current, a current state description

  TELL(KB,MAKE-PERCEPT-SENTENCE(percept, t))
  current  $\leftarrow$  STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
    G  $\leftarrow$  ASK(KB, MAKE-GOAL-QUERY(t))
    p  $\leftarrow$  IDEAL-PLANNER(current, G, KB)
  if p = NoPlan or p is empty then action  $\leftarrow$  NoOp
  else
    action  $\leftarrow$  FIRST(p)
    p  $\leftarrow$  REST(p)
  TELL(KB,MAKE-ACTION-SENTENCE(action,t))
  t  $\leftarrow$  t + 1
  return action

```

PROBLEM SOLVING TO PLANNING

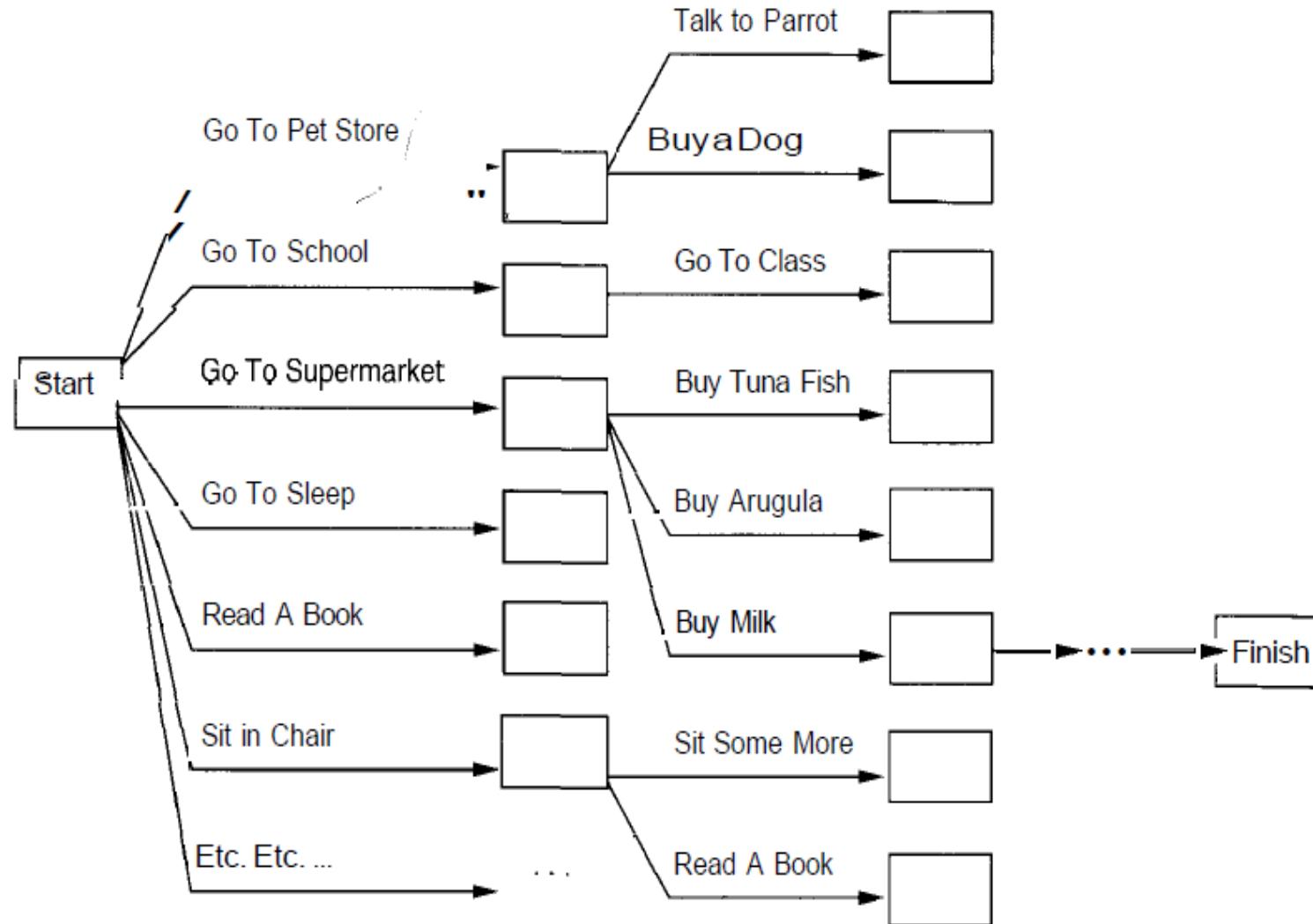
Basic elements of a search-based problem-solver:

- **Representation of actions.** Actions are described by programs that generate successor state descriptions.
- **Representation of states.** In problem solving, a complete description of the initial state is given, and actions are represented by a program that generates complete state descriptions,
- Therefore, all state representations are complete. Eg: state is the position of the agent in a route-finding problem.
- **States are used** only for successor generation, heuristic function evaluation, and goal testing.
- **Representation of goals.** The only information that a problem-solving agent has about its goal is in the form of the goal test and the heuristic function.
- Both of these can be applied to states to decide on their desirability, but they are used as "black boxes."
- That is, the problem-solving agent cannot "look inside" to select actions that might be useful in achieving the goal.

PROBLEM SOLVING TO PLANNING

- **Representation of plans.** In problem solving, a solution is a sequence of actions, such as
- "Go from Arad to Sibiu to Fagaras to Bucharest."
- During the construction of solutions, search algorithms consider only **unbroken sequences of actions** beginning from the initial state to Goal state.
- Consider **an example** to Get a quart of milk and a bunch of bananas and a variable-speed cordless drill.“
- **initial state:** the agent is at home but without any of the desired objects,
- **operator set:** all the things that the agent can do.
- **heuristic function:** perhaps the number of things that have not yet been acquired.

Supermarket shopping problem



Draw backs of search space approach

- actual branching factor - thousands or millions
- length of the solution – many steps.
- too many actions and too many states to consider.
- heuristic evaluation function can only choose among states to decide which is closer to the goal; it cannot eliminate actions from consideration.

Key ideas of planning

- **1) "open up " the representation of states, goals, and actions:**
- *Formal languages*: first-order logic or a subset thereof.
- States and goals are represented by sets of sentences, and
- Actions are represented by logical descriptions of preconditions and effects.
- *Have(Milk): Buy(x) => Have(x),*
- *Planner need not consider irrelevant actions such as Buy(WhippingCream) or GoToSleep.*
- **2) The planner is free to add actions to the plan wherever they are needed, rather than in an incremental sequence starting at the initial state.**
- *For example, the agent may decide that it is going to have to Buy(Milk), even before it has decided where to buy it, how to get there, or what to do afterwards.*
- There is no necessary connection between the order of planning and the order of execution.

Key ideas of planning

- The planner can reduce the branching factor for future choices and reduce the need to backtrack over arbitrary decisions.
- For example, when adding the action *Buy(Milk)* to the plan, the agent can represent the state in which the action is executed as, say, *At(Supermarket)*.
- ***3) Most parts of the world are independent of most other parts:***
- *This makes it feasible to take a conjunctive goal like "get a quart of milk and a bunch of bananas and a variable-speed cordless drill" and solve it with a **divide-and-conquer** strategy.*
- A subplan involving going to the supermarket can be used to achieve the first two conjuncts, and another subplan (e.g., either going to the-hardware store or borrowing from a neighbor) can be used to achieve the third.
- The supermarket subplan can be further divided into a milk subplan and a bananas subplan.

PLANNING IN SITUATION CALCULUS

- A planning problem is represented in situation calculus by logical sentences that describe the **three main parts** of a problem:
- **Initial state:** An arbitrary logical sentence about a situation S_0 . For the shopping problem, this might be

$\text{At}(\text{Home}, S_0) \wedge \neg \text{Have}(\text{Milk}, S_0) \wedge \neg \text{Have}(\text{Bananas}, S_0) \wedge \neg \text{Have}(\text{Drill}, S_0)$

- **Goal state:** A logical query asking for suitable situations. For the shopping problem, the query would be
- **Operators:** A set of descriptions of actions. For example, here is a successor-state axiom involving the *Buy(Milk)* action:

$$\begin{aligned} \forall a, s \text{ Have}(\text{Milk}, \text{Result}(a, s)) \Leftrightarrow & [(a = \text{Buy}(\text{Milk}) \wedge \text{At}(\text{Supermarket}, s)) \\ & \vee (\text{Have}(\text{Milk}, s) \wedge a \neq \text{Drop}(\text{Milk}))] \end{aligned}$$

PLANNING IN SITUATION CALCULUS

- situation calculus is based on the idea that actions **transform states**:
- ***Result(a, s)*** names the situation resulting from executing action *a* in situation *s*.
- ***Result'(l, s)*** to mean the situation resulting from executing the sequence of actions / starting in *s*.
- ***Result'*** is defined by saying that an empty sequence of actions has no effect on a situation.
- The result of a **nonempty sequence of actions** is the same as applying the first action, and then applying the rest of the actions from the resulting situation:

$$\forall s \ Result'([], s) = s$$

$$\forall a, p, s \ Result'([a|p], s) = Result'(p, Result(a, s))$$

solution to the shopping problem

- Plan p that when applied to the start state S_0 yields a situation satisfying the goal query.

$At(Home, Result'(p, S_0)) \wedge Have(Milk, Result'(p, S_0)) \wedge Have(Bananas, Result'(p, S_0))$
 $\wedge Have(Drill, Result'(p, S_0))$

- If have this query to ASK, we end up with a solution such as
- $p = [Go(SuperMarket), Buy(Milk), Buy(Banana), Go(HardwareStore), Buy(Drill), Go(Home)]$

BASIC REPRESENTATIONS FOR PLAN

- STRIPS language – to represent states, goals and actions. (using STRIP operators).
- STRIPS stands for "STanford Research Institute Problem Solver.
- **Representations for states and goals:**
- In the STRIPS language, states are represented by conjunctions of function-free ground literals,
- that is, predicates applied to constant symbols, possibly negated.
- For example, the **initial state** for the milk-and-bananas problem might be described as

At(Home) $\wedge \neg$ Have(Milk) $\wedge \neg$ Have(Bananas) $\wedge \neg$ Have(Drill) $\wedge \dots$

- **Goals** are also described by conjunctions of literals. For example, the shopping goal might be represented as

At(Home) A Have(Milk) A Have(Bananas) A Have(Drill)

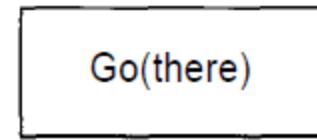
- Goals can also contain variables. For example, the goal of being at a store that sells milk would be represented as *At(x) \wedge Sells(x, Milk)*

STRIPS Language

- **Representations for actions**
- Our STRIPS operators consist of three components:
- **The action description** is what an agent actually returns to the environment in order to do something.
- Within the planner it serves only as a **name for a possible action**.
- **The precondition** is a conjunction of atoms (positive literals) that says what must be true before the operator can be applied.
- **The effect of an operator** is a conjunction of literals (positive or negative) that describes how the situation changes when the operator is applied.
- For eg: STRIPS operator for going from one place to another:

*Op(**ACTION**: Go(*there*), **PRECOND**: At(*here*) \wedge Path(*here, there*),
EFFECT: At(*there*) \wedge At(*here*))*

At(*here*), Path(*here, there*)



- operator *Go(there)*. *Preconditions (above)*
- *Effects (below)*

Example of Go operator in super market

- For example, if the initial situation includes the literals
At(Home), Path(Home, Supermarket),...
- then the action *Go(Supermarket)* is applicable, and the resulting situation contains the literals
¬At(Home), At(Supermarket), Path(Home, Supermarket),...
- **situation space planner** because it searches through the space of possible situations, and a **progression planner** because it searches forward from the initial situation to the goal situation.
- The main problem with this approach is the **high branching factor** and thus the **huge size of the search space**.
- One way to try to **cut the branching factor** is to **search backwards**, from the goal state to the initial state; such a search is called **regression planning / Planner**.
- **This approach is possible because the operators contain enough information to regress from a partial description of a result state to a partial description of the state before an operator is applied.**

Search through Space of Plans

- **Partial Plan:** start with a simple, incomplete plan. Search through space of plans.
- Then we consider **ways of expanding the partial plan** until we come up with a complete plan that solves the problem.
- The operators in this search are operators on plans: adding a step, imposing an ordering that puts one step before another, instantiating a previously unbound variable, and so on.
- The solution is the final plan, and the path taken to reach it is irrelevant.
- **Operations on plans come in two categories:** **Refinement operators take a partial plan** and add constraints to it. Refinement operators eliminate some plans from this set, but they never add new plans to it.
- **Modification operator.** Some planners work by constructing potentially incorrect plans, and then "debugging" them using modification operators.

Basic representation of Plan

Example 2: putting on a pair of shoes

- Representations for plans
- If we are going to search through a space of plans, we need to be able to represent them.
- consider **partial plans** for a simple problem: **putting on a pair of shoes**. The goal is the conjunction of *RightShoeOn* & *LeftShoeOn*, the initial state has no literals at all, and the **four operators** are:

Op(ACTION:RightShoe,PRECOND:RightSockOn, EFFECT: RightShoeOn)

Op(ACTION:RightSock, EFFECT:RightSockOn)

Op(ACTION:LeftShoe,PRECOND:LeftSockOn, EFFECT: LeftShoeOn)

Op(ACTION:LeftSock,EFFECT: LeftSockOn)

- A partial plan has two steps ***RightShoe and LeftShoe***.
- The **principle of least commitment**, which says that one should only make choices about things that you currently care about, leaving the other choices to be worked out later.
- A least commitment planner could leave the ordering of the two steps unspecified.

Partial and Total order planner

- A planner that can represent plans in which some steps are ordered (before or after) with respect to each other and other steps are unordered is called a **partial order** planner.
- For eg: putting on the right sock comes before putting on the right shoe, but no order for among putting first leftshoe or rightshoe.
- $\text{POP} = \{\text{leftshoe}, \text{rightshoe}\}; \{\text{rightshoe}, \text{leftshoe}\}$
- The alternative is a **total order planner**, in which plans consist of a **simple list of steps**.
- A totally ordered plan that is derived from a plan P by *adding ordering constraints* is called a **linearization of P** .
- **Total order planner $P = \{ \text{leftshock}, \text{leftshoe}\}, \{ \text{rightshock}, \text{rightshoe}\}$.**
- Plans in which every variable is bound to a constant are called **fully instantiated plans**.

An example of fully instantiated plan

- Consider Goal is to *Have(Milk)*, and you have the action *Buy(item, store)*.
- A *sensible commitment* is to choose this action with the variable item bound to Milk.
- Binding for *store*, so the principle of least commitment says to leave it unbound and make the choice later.
- Binding the variable *store* to the specialty store where the milk is also available.
- By delaying the commitment to a particular store, we allow the planner to make a good choice later.
- If milk not available in store, Backtrack and get the milk from the specialty store.

Defining Plan formally

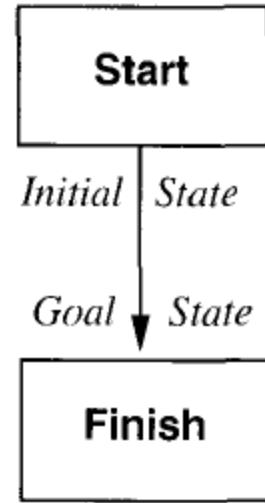
- A **plan** is formally defined as a data structure consisting of the following four components:
- A set of plan steps. Each step is one of the operators for the problem.
- A set of step ordering constraints. Each ordering constraint is of the form $S_i \prec S_j$ which is read as "*Si* before *Sj* and means that step *Si* must occur sometime before step *Sj* (but not necessarily immediately before).
- A set of variable binding constraints. Each variable constraint is of the form $v = x$, where v is a variable in some step, and x is either a constant or another variable.
- A set of causal links. A causal link is written as $S_i \xrightarrow{c} S_j$ and read as "*Si* achieves *c* for *Sj*". Causal links serve to record the purpose(s) of steps in the plan: here a purpose of *Si* is to achieve the precondition *c* of *Sj*.

Planner steps

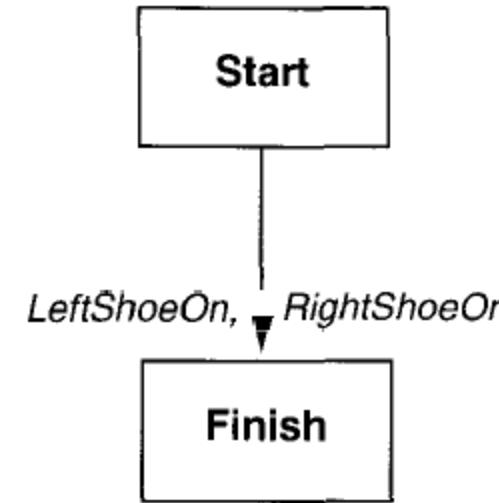
- The **initial plan**, before any refinements have taken place, simply describes the unsolved problem.
- It consists of two steps, called ***Start*** and ***Finish***, with the ordering constraint ***Start X Finish***.
- *Both Start and Finish have null actions associated with them, so when it is time to execute the plan, they are ignored.*
- The ***Start*** step has ***no preconditions***, and its ***effect*** is to add all the ***propositions*** that are ***true*** in the initial state.
- The ***Finish*** step has the goal state as its ***precondition***, and ***no*** effects.
- planners can start with the initial plan and manipulate (refine) it until they come up with a plan that is a solution.
- The shoes-and-socks problem is defined by the four operators given earlier and an initial plan that we write as follows:

Planner for shoes-and-socks problem

*Plan(STEPS:{ S₁: Op(ACTION:Start),
S₂: Op(ACTION:Finish,
 PRECOND:RightShoeOn A LeftShoeOn}),
ORDERINGS: {S₁ < S₂},
BINDINGS: {}},
LINKS: {})*



(a)



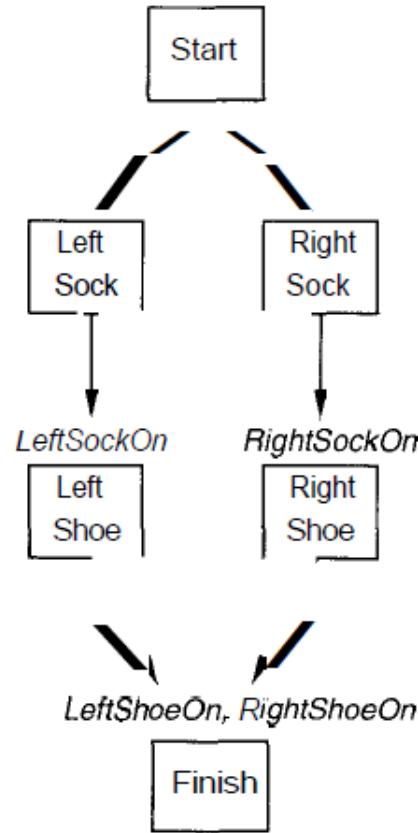
(b)

- a. Partial plans containing only *Start* and *Finish* steps. Ordering constraints are shown as arrows between boxes. b. The initial plan for the shoes-and-socks problem.

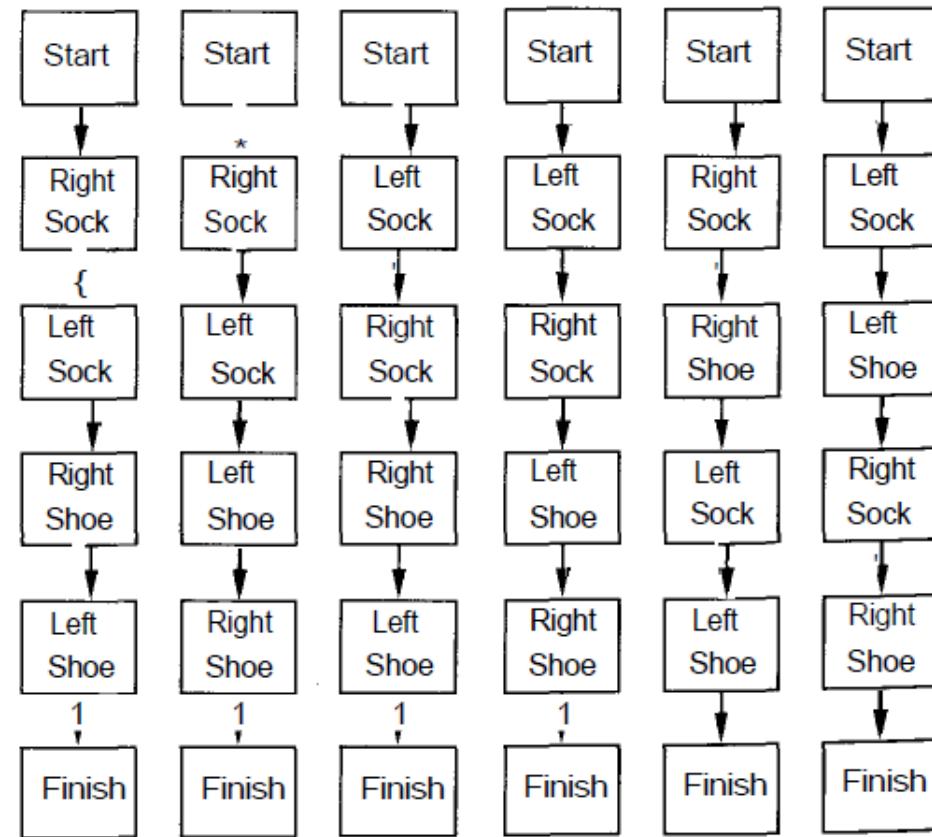
Planner's for shoes-and-socks problem

A **partial-order plan** for putting on shoes and socks (including preconditions on steps), and the **six possible linearizations (Total order)** of the plan.

Partial Order Plan:



Total Order Plans:

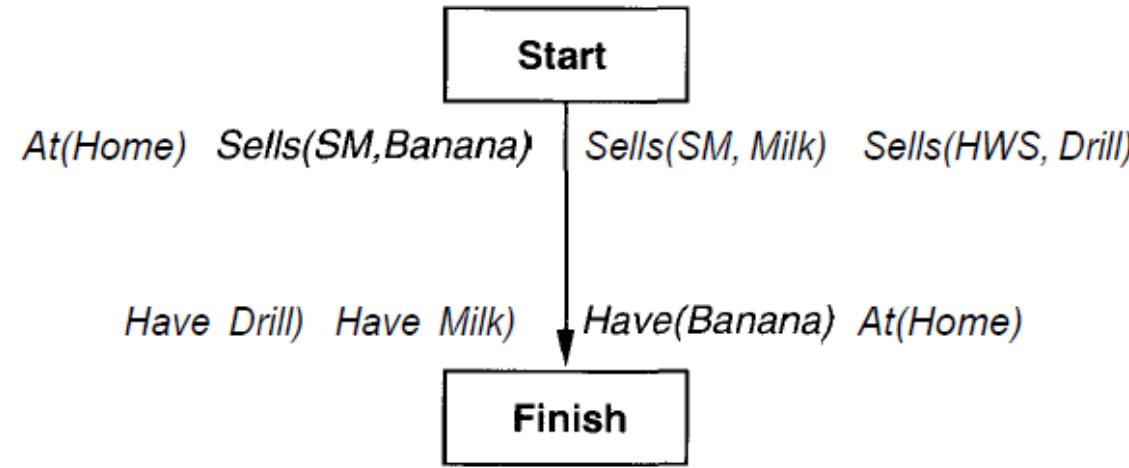


Solutions

- A **solution** is a plan that an agent can execute, and that guarantees achievement of the goal.
- only **fully instantiated, totally ordered** plans can be solutions.
- A solution is a **complete, consistent plan**.
- A **complete plan** is one in which every precondition of every step is achieved by some other step.
- formally, a step S_i , achieves a precondition c of the step S_j
 $if (1) \ S_i < S_j \ and \ c \in EFFECTS(S_i); \ and \ (2) \ there \ is \ no \ step \ S_k \ such \ that \ (\sim c) \in EFFECTS(S_k), \ where \ S_i < S_k < S_j \ in \ some \ linearization \ of \ the \ plan.$
- A **consistent plan** is one in which there are **no contradictions** in the ordering or binding constraints.
- A **contradiction occurs** when both $S_i < S_j$ and $S_j < S_i$ hold or both $v = A$ and $v = B$ hold (for two different constants A and B). Both $<$ and $=$ are **transitive**, so, for example, a plan with, $S_1 < S_2, S_2 < S_3$ and $S_3 < S_1$ is inconsistent.

A PARTIAL-ORDER PLANNING EXAMPLE

- Partial-order regression planner that searches through plan space.
- The planner starts with an initial plan representing the start and finish steps, and on each iteration adds one more step.
- Consider Problem of getting some milk, a banana, and a drill, and bringing them back home.
- **Go action**- travel between any two locations.
- **Buy action – ignores money**
- **HWS means hardware store and SM means supermarket**



11.6 The initial plan for the shopping problem.

The initial plan for the shopping problem.

- initial state

$Op(\text{ACTION:}Start, \text{EFFECT:}At(Home) \wedge Sells(HWS, Drill)$
 $\wedge Sells(SM, Milk), Sells(SM, Banana))$

- goal state is defined by a *Finish step*

$Op(\text{ACTION:}Finish,$
 $\text{PRECOND:}Have(Drill) \wedge Have(Milk) \wedge Have(Banana) \wedge At(Home))$

- Actions are defined as

$Op(\text{ACTION:}Go(there), \text{PRECOND:}At(here),$
 $\text{EFFECT:}At(there) \wedge \neg At(here))$

$Op(\text{ACTION:}Buy(x), \text{PRECOND:}At(store) \wedge Sells(store, x),$
 $\text{EFFECT: } Have(x))$

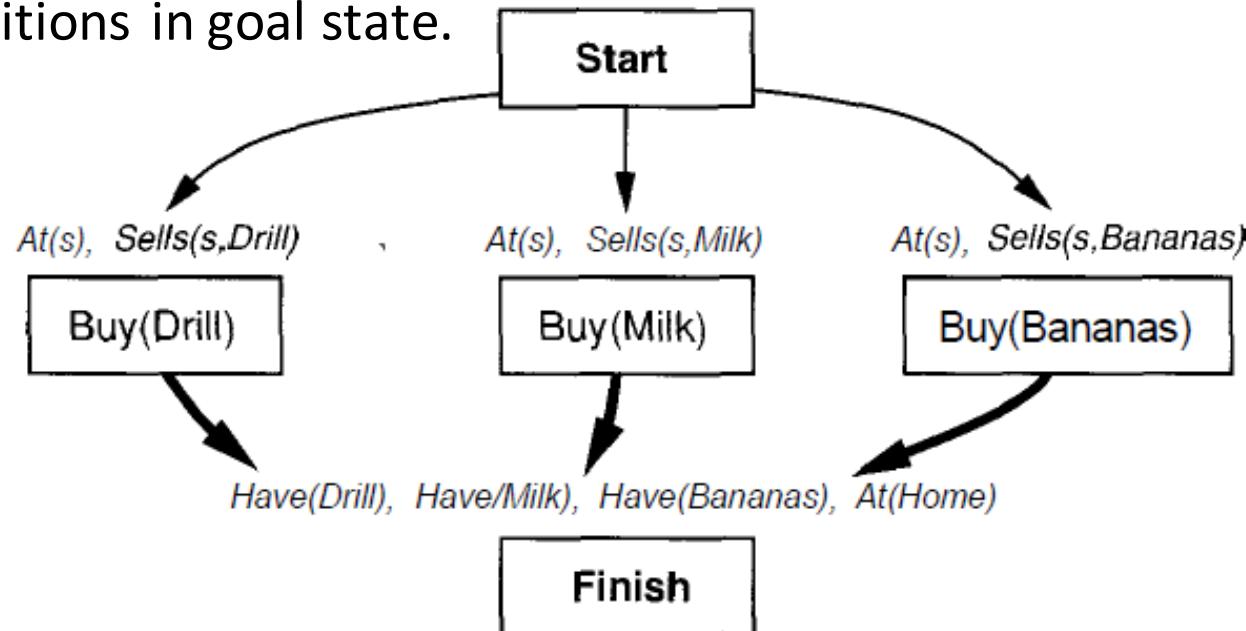
STRIPS PLAN FOR SHOPPING PROBLEM

Shopping Problem

- Actions
 - $\text{Buy}(x, \text{store})$
 - Pre: $\text{At}(\text{store})$, $\text{Sells}(\text{store}, x)$
 - Eff: $\text{Have}(x)$
 - $\text{Go}(x, y)$
 - Pre: $\text{At}(x)$
 - Eff: $\text{At}(y)$, $\neg\text{At}(x)$
- Goal
 - $\text{Have}(\text{Milk}) \wedge \text{Have}(\text{Banana}) \wedge \text{Have}(\text{Drill})$
- Start
 - $\text{At}(\text{Home}) \wedge \text{Sells}(\text{SM}, \text{Milk}) \wedge \text{Sells}(\text{SM}, \text{Banana}) \wedge \text{Sells}(\text{HW}, \text{Drill})$

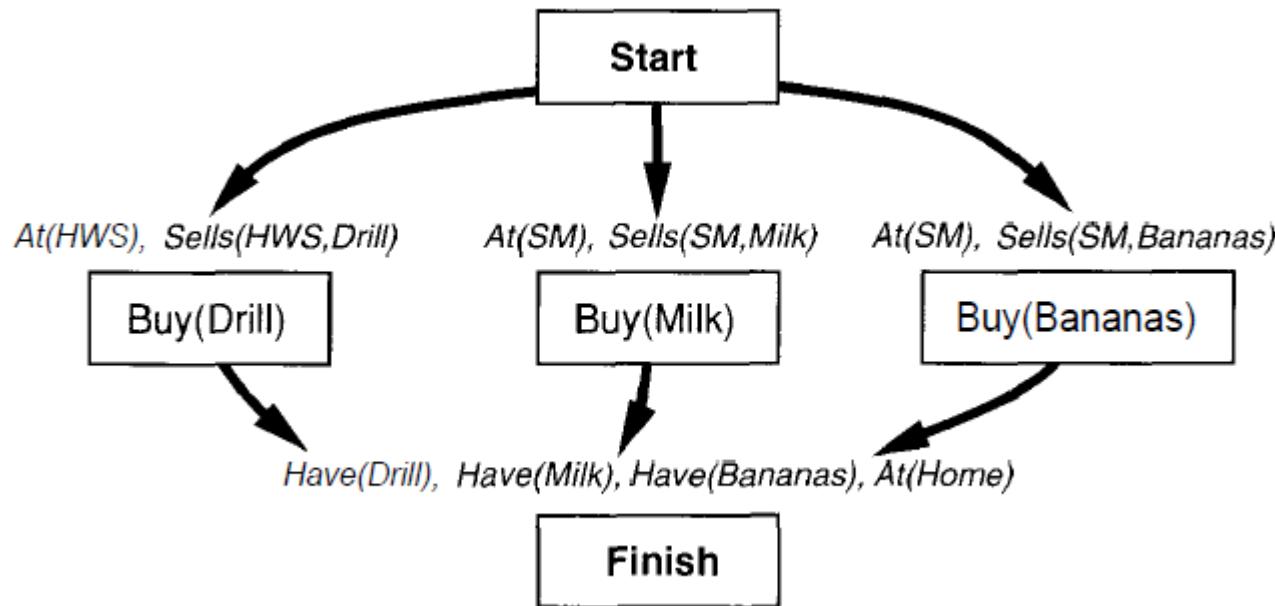
Partial Plan – Initial step (Step1)

- A partial plan that achieves three of the four preconditions of *Finish*. The heavy arrows show causal links.
- **bold arrows** in the figure are **causal links**. Light arrows in the figure show **ordering constraints**. Assume light arrows are underneath the causal links.
- The POP adds **Buy(drill)**, **Buy(Milk)**, **Buy(Bananas)** actions to achieve the preconditions in goal state.



Partial Plan – refining (Step2)

Refining the partial plan by adding causal links to achieve the *Sells preconditions of the Buy steps.*



Extending Partial Plan (Step3)

- ❖ In Figure 11.8, we extend the plan by choosing two *Go actions* to get us to the hardware store and supermarket, thus achieving the *At preconditions* of the *Buy actions*.
- ❖ The agent cannot be *At two places at the same time*. Each *Go action* has a precondition *At(x)*, where *x* is the location that the agent was at before the *Go action*.

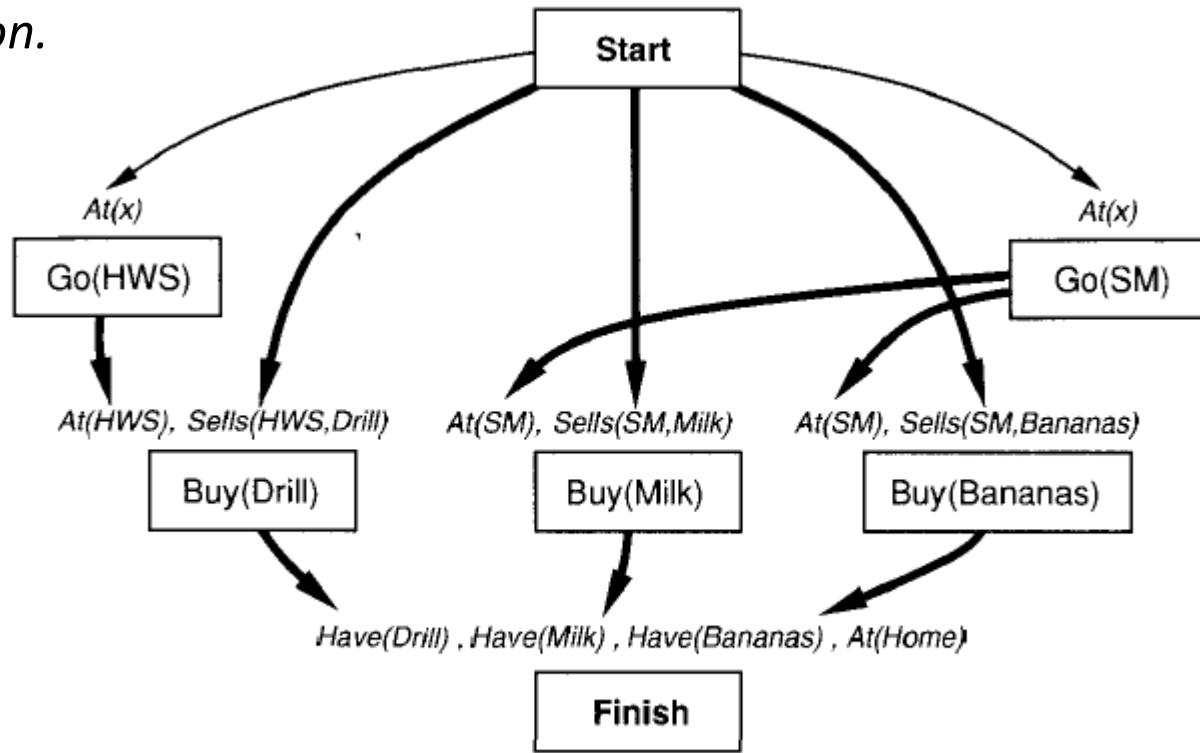
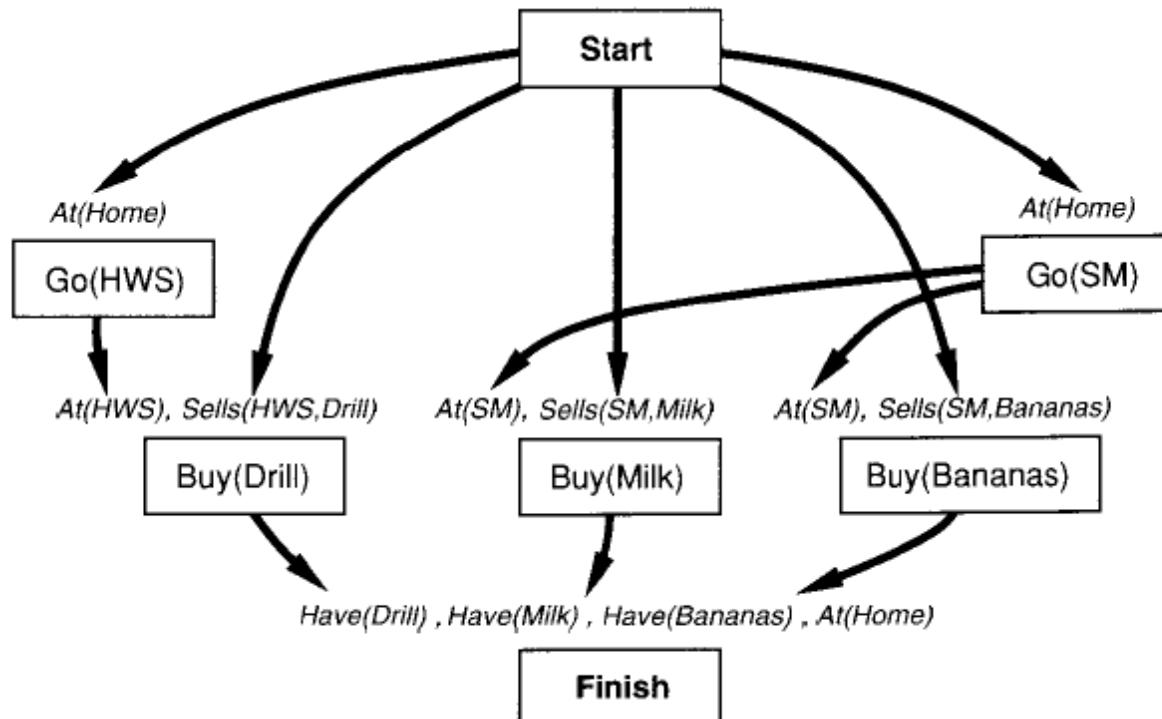


Figure 11.8 A partial plan that achieves A? preconditions of the three *Buy* actions.

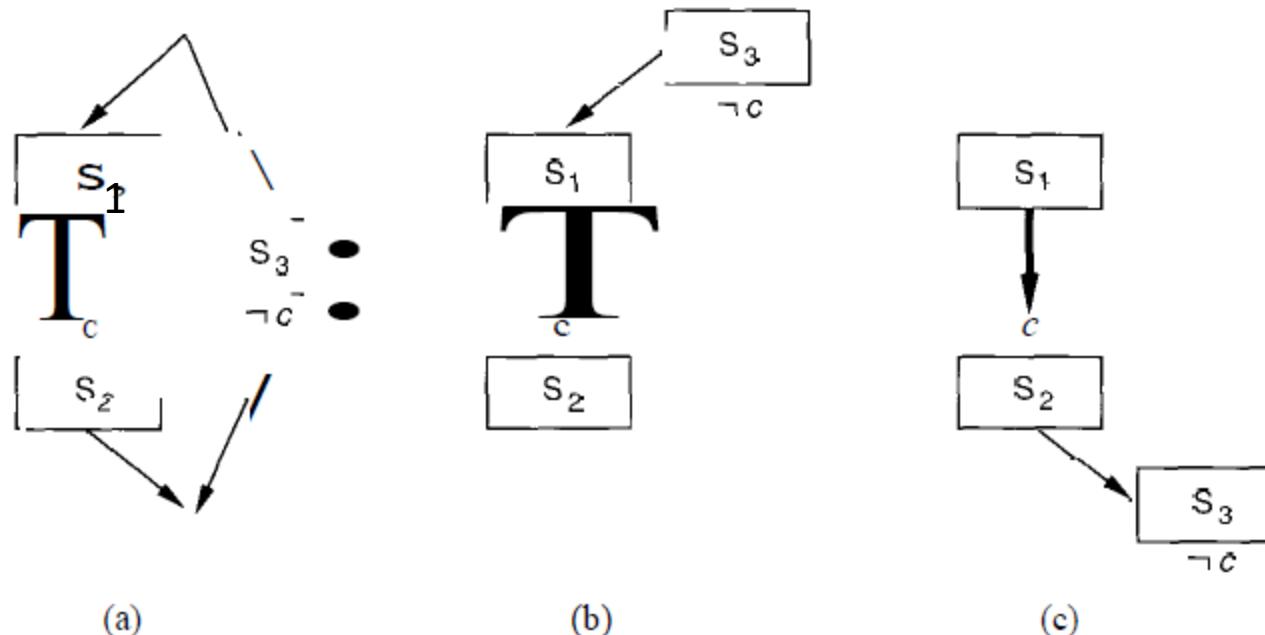
Planner achieve Preconditions of Go(HWS) and Go(SM) (Step4)

- The planner tries to achieve the preconditions of *Go(HWS)* and *Go(SM)* by *linking them to the : At(Home) condition in the initial state*.
- if the agent goes to the hardware store, it can no longer go from home to the supermarket and vice-versa (threats in POP). **dead end in the search for a solution.**



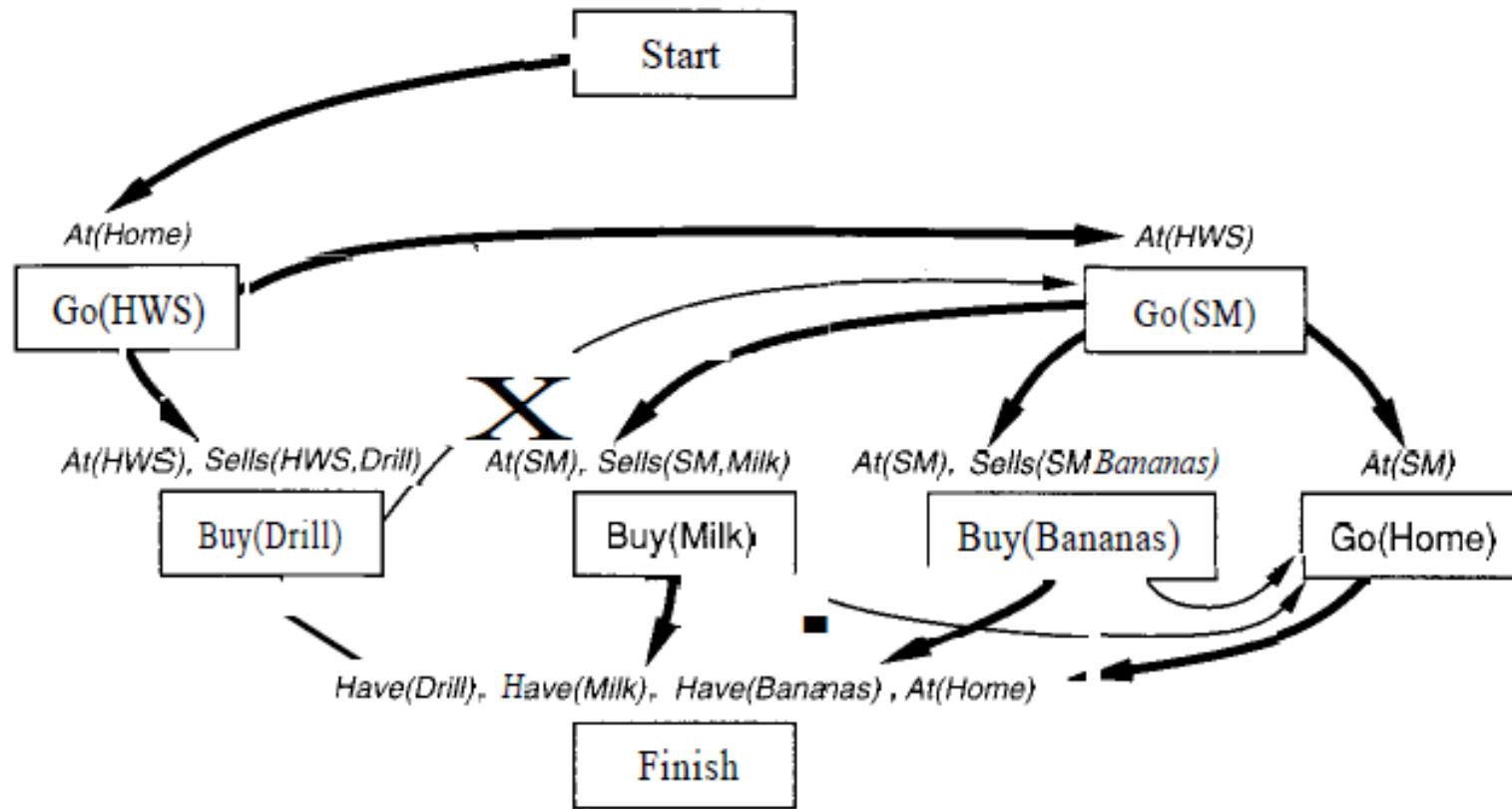
Protected links

- The causal links in a partial plan are **protected links**.
- Threats**—that is, steps that might delete the **protected condition**.
- A causal link is protected by ensuring that **threats** are ordered to come before or after the protected link.



In (a), the step S_3 **threatens** a condition c that is established by S_1 and protected by the causal link from S_1 to S_2 . In (b), S_3 has **been demoted** to come before S_1 , and in (c) it has been **promoted** to come after S_2 .

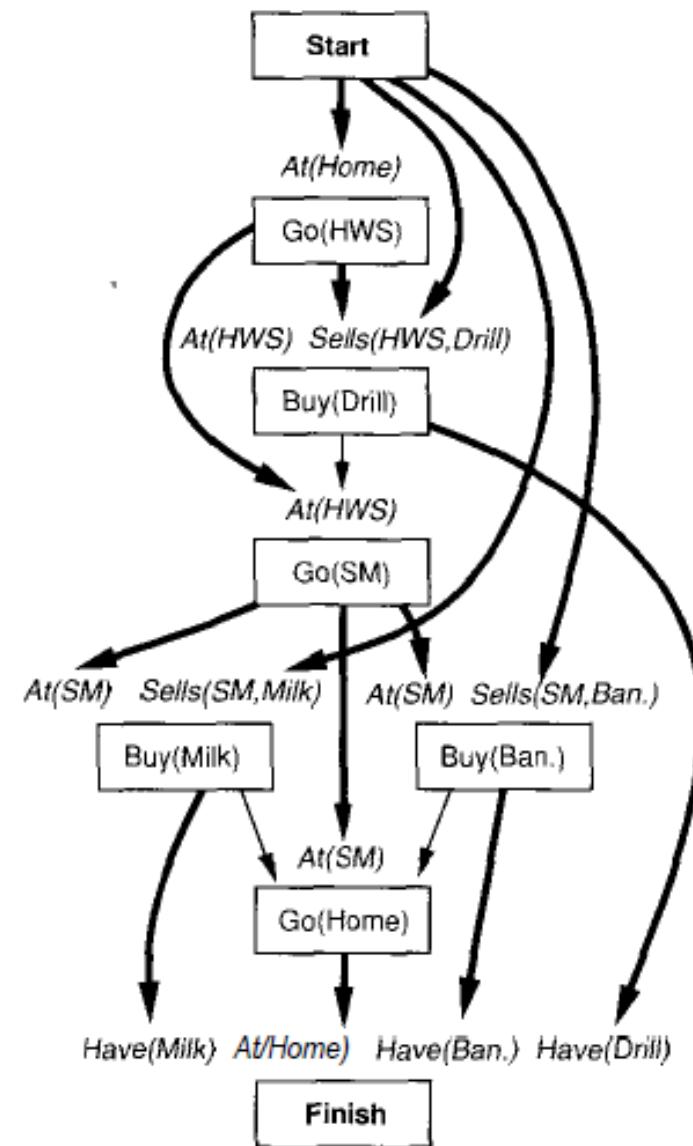
Causal link protection in the shopping plan.



The $\text{Go(HWS)} \xrightarrow{\text{At(HWS)}} \text{Buy(Drill)}$ causal link is protected ordering the Go(SM) step after Buy(Drill) , and the $\text{Go(SM)} \xrightarrow{\text{At(HWS)}} \text{Buy(Milk or Banana)}$ link is protected by ordering Go(Home) after Buy(Milk) and Buy(Bananas) .

Complete Plan of Shopping

- complete solution plan, with the steps redrawn to reflect the ordering constraints.
- The result is an almost totally ordered plan; the only ambiguity is that *Buy(Milk)* and *Buy(Bananas)* can come in either order.



POP planner for shopping problem

- **Start/Initial state**
- At(home) \wedge ~Have(Milk) \wedge ~Have(Banana) \wedge ~Have(Drill)
- **Go(HWS)**
- At(HWS) \wedge ~At(home) \wedge ~Have(Milk) \wedge ~Have(Banana) \wedge ~Have(Drill)
- **Buy(Drill)**
- At(HWS) \wedge ~At(home) \wedge ~Have(Milk) \wedge ~Have(Banana) \wedge Have(Drill)
- **Go(SM)**
- At(SM) \wedge ~At(home) \wedge ~Have(Milk) \wedge ~Have(Banana) \wedge Have(Drill)
- **Buy(Milk)**
- At(SM) \wedge ~At(home) \wedge Have(Milk) \wedge ~Have(Banana) \wedge Have(Drill)
- **Buy(Banana)**
- At(SM) \wedge ~At(home) \wedge Have(Milk) \wedge Have(Banana) \wedge Have(Drill)
- **Go(home)**
- At(SM) \wedge At(home) \wedge Have(Milk) \wedge Have(Banana) \wedge Have(Drill)
- **Goal/Finish state**

POP algorithm

```

function POP(initial, goal, operators) returns plan
    plan  $\leftarrow$  MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
         $S_{need}, c \leftarrow$  SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, Sneed, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns Sneed, c
    pick a plan step  $S_{need}$  from STEPS(plan)
    with a precondition c that has not been achieved
    return  $S_{need}, c$ 

procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
    choose a step  $S_{add}$  from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link  $S_{add} \xrightarrow{c} S_{need}$  to LINKS(plan)
    add the ordering constraint  $S_{add} \prec S_{need}$  to ORDERINGS(plan)
    if  $S_{add}$  is a newly added step from operators then
        add  $S_{add}$  to STEPS(plan)
        add Start  $\prec S_{add} \prec Finish to ORDERINGS(plan)
    procedure RESOLVE-THREATS(plan)
        for each  $S_{threat}$  that threatens a link  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
            choose either
            Promotion: Add  $S_{threat} \prec S_i$  to ORDERINGS(plan)
            Demotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
            if not CONSISTENT(plan) then fail
        end$ 
```

Steps:

1. POP starts with a minimal partial plan.
2. On each step extend the plan by achieving a precondition *c* of a step S_{need} .
3. choosing some operator - from exiting plan or Pool of operators – satisfies preconditions.
4. Record the causal links for this newly achieved preconditions.
5. Resolve any threats to causal links.
The **new step** may threaten an existing causal link or an **existing step** may threaten the new causal link.
6. If algorithm fails to find a relevant operator or resolve a threat, it backtracks to a previous choice point and SELECT-SUB GOAL.
7. Once it has achieved all the preconditions of all the steps, it is done; it has a solution.
POP *is sound and complete*.

Figure 11.13 The partial-order planning algorithm, POP.

POP with Partially instantiated operators

```
procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
    choose a step  $S_{add}$  from operators or STEPS(plan) that has  $c_{add}$  as an effect
        such that  $\mu = \text{UNIFY}(C, c_{add}, \text{BINDINGS}(\text{plan}))$ 
    if there is no such step
        then fail
    add  $\mu$  to BINDINGS(plan)
    add  $S_{add} \xrightarrow{c} S_{need}$  to LINKS(plan)
    add  $S_{add} \prec S_{need}$  to ORDERINGS(plan)
    if  $S_{add}$  is a newly added step from operators then
        add  $S_{add}$  to STEPS(plan)
    add Start  $\prec S_{add} \prec \text{Finish}$  to ORDERINGS(plan)
```

```
procedure RESOLVE-THREATS(plan)
    for each  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
        for each  $S_{threat}$  in STEPS(plan) do
            for each  $c'$  in EFFECT(Sthreat) do
                if  $\text{SUBST}(\text{BINDINGS}(\text{plan}), c) = \text{SUBST}(\text{BINDINGS}(\text{plan}), \neg c')$  then
                    choose either
                        Promotion: Add  $S_{threat} \prec S_i$  to ORDERINGS(plan)
                        Demotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
                if not CONSISTENT(plan)
                    then fail
            end
        end
    end
```

There are **three main approaches** to dealing with **possible threats**:

1. Resolve now with an equality constraint: **Modify RESOLVE-THREATS so that it resolves** all possible threats as soon as they are recognized. For eg: Planner has operator $\sim\text{At}(x)$, if $x = \text{HWS}$, will not threat the $\text{At}(\text{Home})$.

2. Resolve now with an inequality constraint: Extend the language of variable binding constraints to allow the constraint $x \neq \text{home}$

3. Resolve later: The third possibility is to ignore possible threats, and only deal with them when they become *necessary threats*. That is, *RESOLVE-THREATS* would not consider $\sim\text{At}(x)$, to be a threat to $\text{At}(\text{Home})$ if $x = \text{HWS}$. If $X = \text{Home}$ then threat has to be resolved.

Figure 11.14 Support for partially instantiated operators in POP.

- Decide what to talk about.
- Decide on a vocabulary of conditions (literals), operators, and objects.
- Encode operators for the domain.
- Encode a description of the specific problem instance.
- Pose problems to the planner and get back plans.

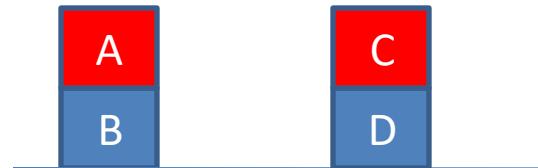
Each of these five steps, demonstrating them in two domains.

The blocks world

Shakey's world

Blocks World Problem

- **What to talk about:**
- This domain consists of a set of cubic blocks sitting on a table.
- The blocks can be stacked, but only one block can fit directly on top of another.
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block.
- The arm can only pick up one block at a time, so it cannot pick up a block that has another one on it.
- The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks.
- For example, a goal might be to make two stacks, one with block *A* on *B*, and the other with *C* on *D*.



Blocks World Problem

- **Vocabulary:**
- The **objects** in this domain are the **blocks** and the **table**. They are represented by constants.
- *On(b, x)* to indicate that block b is on x, where x is either another block or the table.
- *Move(b,x,y)* operator for moving block b from a position on top of x to a position on top y.
- **preconditions** on moving b is that no other block is on it.

$\neg \exists x \text{ On}(x, b) \text{ or } \forall x \neg \text{On}(x, b)$

- *Clear(x)* to mean that nothing is on x.
- **Operators:** The operator *Move* moves a block b from x to y if both b and y are clear, and once the move is made, x becomes clear but y is clear no longer. The formal description of Move is as follows:

Op(ACTION:Move(b,x, y),

PRECOND:On(b, x) A Clear(b) A Clear(y),

EFFECT: On(b,y) A Clear(x) A \neg On(b, x) A \neg Clear(y))

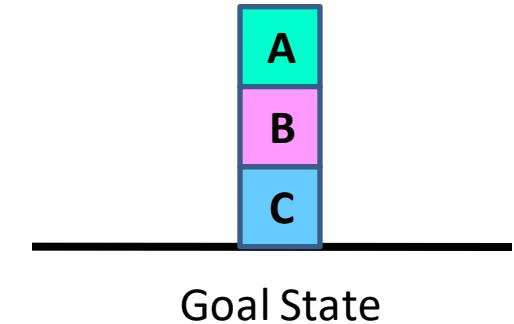
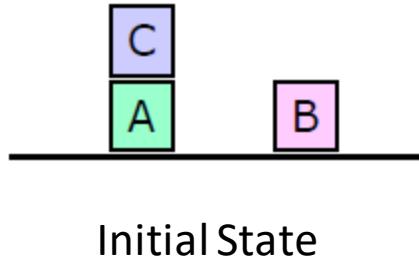
MOVE(b,x,y)
STACK(A,B),
UNSTACK(A,B)
PICKUP(A)

Blocks World Problem

- operator to move a block b from x to the table:

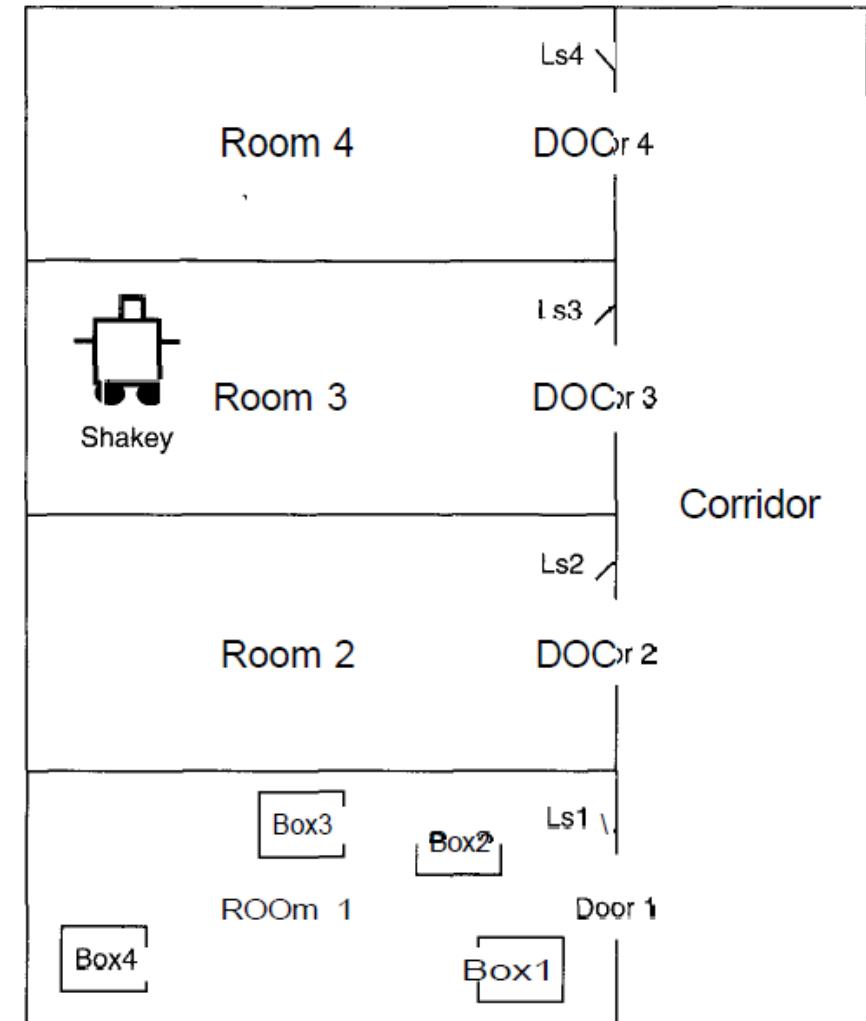
*Op(ACTION:MoveToTable(b, x),
PRECOND: $On(b, x) \wedge Clear(b)$,
EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x)$)*

- Logic Representation of initial and goal state



Initial state: $On(C, A) \wedge On(B, \text{table})$
Goal State: $On(A, B) \wedge On(B, C)$

Shakey's World Problem



Shakey's world.

Shakey's World Problem

- **Domain:** Shakey can move from place to place, push movable objects (such as boxes), climb on and off of rigid objects (such as boxes), and turn light switches on and off.
- **Vocabulary:**
- **Go(y):** Go from current location to location y
- **In(x, r) In(y, r):** x and y be *In the same room r.*
- **Push(b, x, y):** Push an object b from *location x to location y*
- **Climb(b):** Climb up onto a box.
- **Down(b):** Climb down from a box
- **TurnOn(ls):** Turn a light switch on.
- **TurnOff(ls):** Turn a light switch off

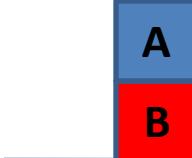
Goal stack Planning – example1

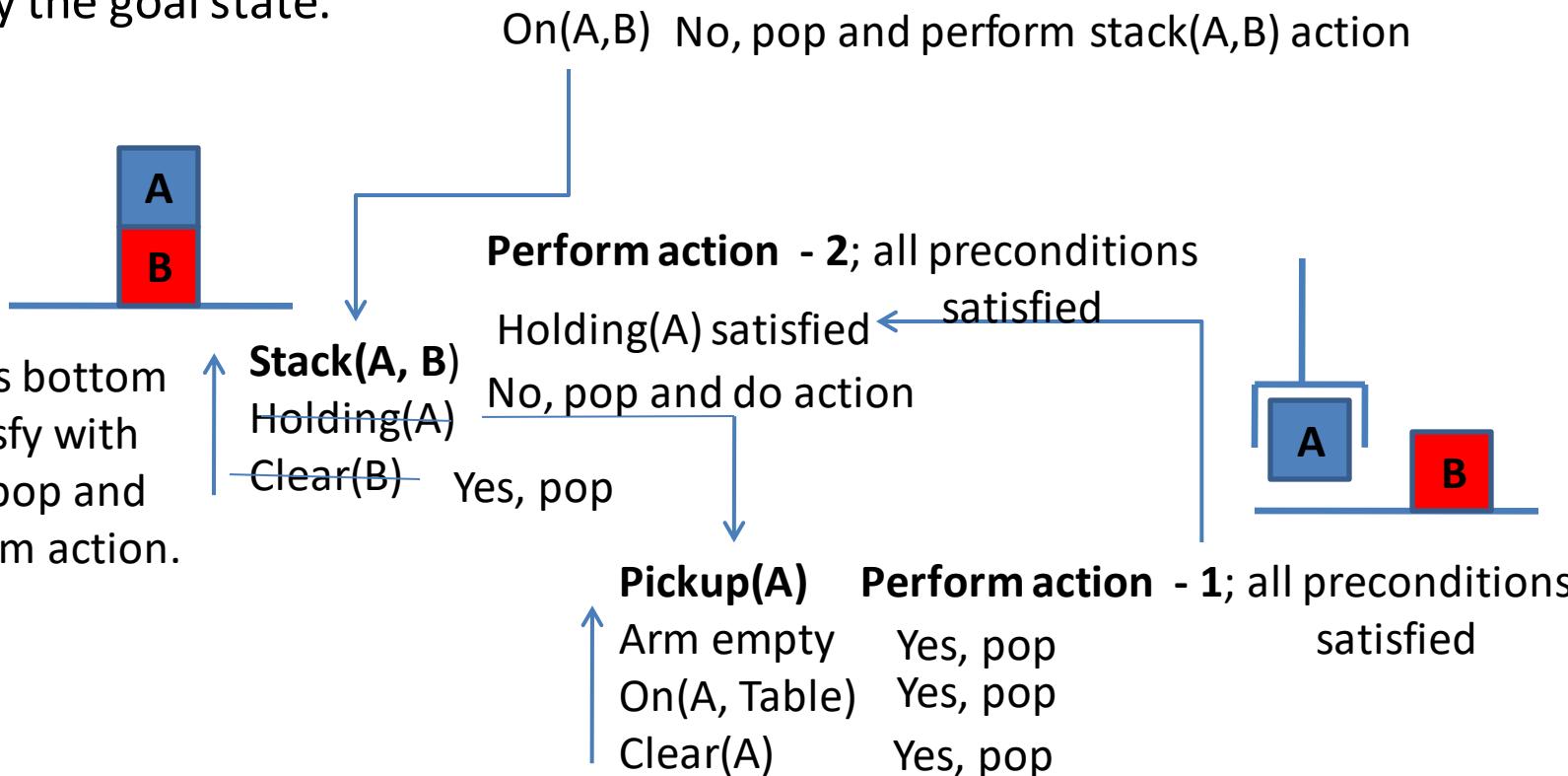


- Logic Representation of states
- Initial state: $\text{On}(A, \text{Table})$, $\text{On}(B, \text{Table})$
- Goal state: $\text{On}(A, B)$

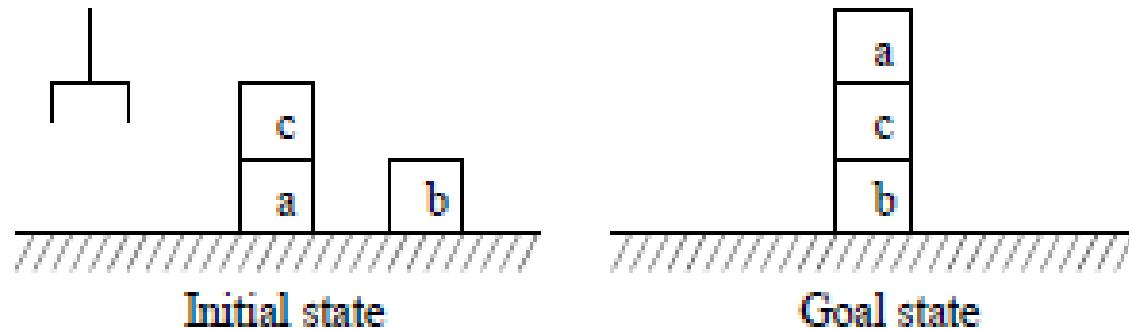
Goal stack Planning – example1

- Consider goal state $\text{On}(A,B)$ and push onto the stack – There is **no matching** initial state. **Pop** the stack and **perform the operation / action** to satisfy the goal state.


 Check Actions bottom
 To top. If satisfy with
 initial state, pop and
 do not perform action.
 - Clear(B)



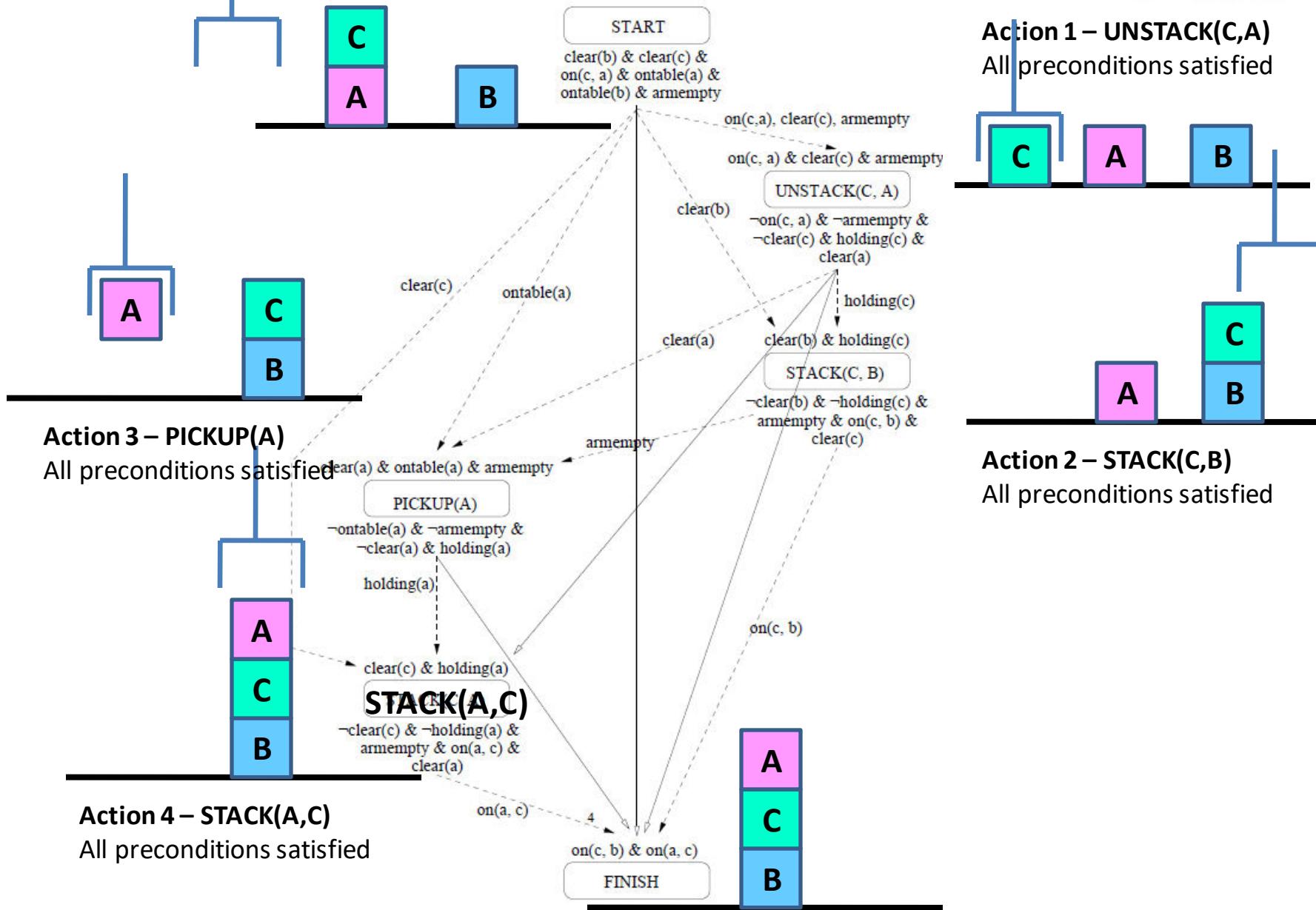
Blocks World Problem – example 2



These would be represented in POP as the following initial plan:

```
Plan(STEPS: {S1: Op( ACTION: Start,  
                      EFFECT: clear(b) ∧ clear(c) ∧  
                             on(c, a) ∧ ontable(a) ∧  
                             ontable(b) ∧ armempty),  
                 S2: Op( ACTION: Finish,  
                      PRECOND: on(c, b) ∧ on(a, c))},  
ORDERINGS: {S1 < S2},  
LINKS: {})
```

Solution to Blocks World Problem



Practical Planners

- planners that operate in complex, realistic domains.
- **Spacecraft assembly, integration, and verification** - OPTIMUM-AIV Planner
- **Job shop scheduling** - O-PLAN Planner
- **Scheduling for space missions** - PlanERS-1, Hubble space telescope (HST).
- **Buildings, aircraft carriers, and beer factories** - SIPE (System for Interactive Planning and Execution monitoring) Planner.

OPTIMUM-AIV Planner

- OPTIMUM-AIV is a planner that is used by the European Space Agency to help in the assembly, integration, and verification (AIV) of spacecraft.
- The system is used both to generate plans and to monitor their execution.
- During monitoring, the system reminds the user of upcoming activities, and can suggest repairs to the plan when an activity is performed late, cancelled, or reveals something unexpected.
- In fact, the **ability to quickly replan** is the principal objective of OPTIMUM-AIV.
- The system does not execute the plans; that is done by humans with standard construction and test equipment.

STRIPS Cannot be used AIV Domain

- The STRIPS language is insufficient for the AIV domain because it cannot express four key concepts:
- **Hierarchical plans:** launching a spacecraft is more complicated.
- One way to handle the increased complexity is to specify plans at varying levels of detail.
- There might be a dozen intermediate levels before we finally get down to the level of executable actions: *insert nut A into hole B and fasten with bolt C.*
- **Complex conditions:** STRIPS operators are essentially prepositional and unconditional.
- we cannot express the fact that if all systems are go, then the *Launch* will put the spacecraft into orbit.
- **Time:** Because the STRIPS language is based on situation calculus it assumes that all actions occur instantaneously, and that one action follows another with no break in between.
- Real world projects need a better model of time.

STRIPS Cannot be used AIV Domain

- **Resources:** A project normally has a **budget** that cannot be exceeded, so the plan must be **constrained** to spend no more money than is available.
- Similarly, there are **limits on the number of workers** that are available, and on the **number of assembly and test stations**.
- Resource limitations may be placed on the number of things that may be used at one time (e.g., people) or on the total amount that may be used (e.g., money).
- Action descriptions must incorporate resource consumption and generation, and
- planning algorithms must be **able to handle constraints on resources efficiently.**

O-PLAN

- **Job shop scheduling:** The problem that a factory solves is to take in raw materials and components, and assemble them into finished products.
- The problem can be divided into a planning task (deciding what assembly steps are going to be performed) and a scheduling task (deciding when and where each step will be performed).
- **O-PLAN** is being used by Hitachi for job shop planning and scheduling in a system called TOSCA.
- A typical problem involves a product line of 350 different products, 35 assembly machines, and over 2000 different operations.
- The planner comes up with a 30-day schedule for three 8-hour shifts a day.

Scheduling for space missions

- Planning and scheduling systems have been used extensively in planning space missions as well as in constructing spacecraft.
- There are two main reasons for this. **First**, spacecraft are very expensive and sometimes contain humans, and any mistake can be costly and irrevocable.
- **Second**, space missions take place in space, which does not contain many other agents to mess up the expected effects of actions.
- **PlanERS-1 is an automatic planner** based on O-PLAN that produces observation plans for the ERS-1 space craft.

Buildings, aircraft carriers, and beer factories

- SIPE (System for Interactive Planning and Execution monitoring) was the **first planner** to deal with the problem of **replanning**, and the first to take some important steps toward expressive operators.
- SIPE allows deductive rules to operate over states, so that the user does not have to specify all relevant literals as effects of each operator.
- It allows for an inheritance hierarchy among object classes and for an expressive set of constraints.

HIERARCHICAL DECOMPOSITION

- High-level description/ abstractions used of what to do, but it is a long way from the type of instructions that can be fed directly to the agent's effectors.
- A plan such as

$[Go(Supemarket), \quad Buy(Milk), \quad Buy(Bananas),$
 $Go(Home)]$

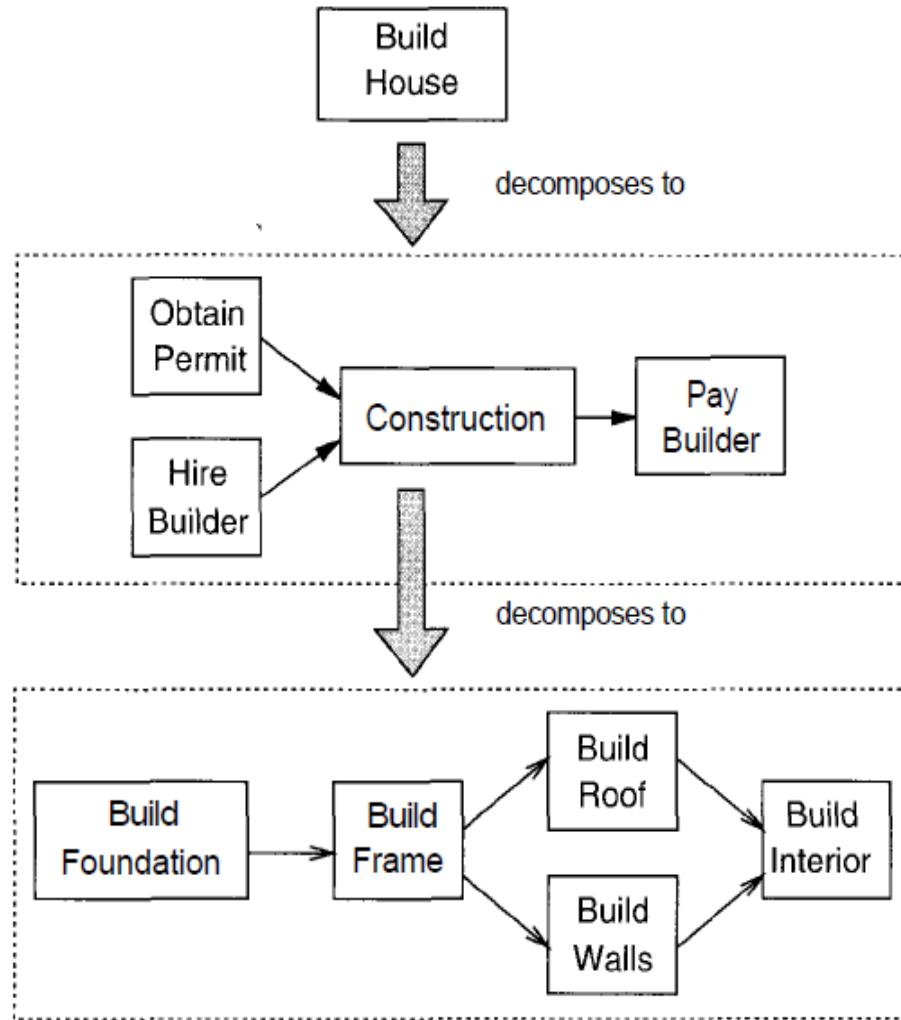
- is a good high level abstraction plan, which can be decomposed into sub plans to solve sub goals.
- **hierarchical decomposition:** An abstract operator can be decomposed into a group of steps that forms a plan that implements the operator.
- These decompositions can be stored in a library of plans and retrieved as needed.

HIERARCHICAL DECOMPOSITION

- Consider the problem of building a frame house.
- **Abstract operator** - can be decomposed into a plan that consists of the few steps.
- The plan is complete when every step is a **primitive operator**—one that can be directly executed by the agent and cannot be decomposed further.
- Hierarchical planning follow two things.
- (1) Provide an extension to the STRIPS language to allow for nonprimitive operators.
- (2) Modify the planning algorithm to allow the replacement of a nonprimitive operator with its decomposition.
- Decomposition of *Construction* as in the Figure in the next slide.
- **operator Build(House)** can be decomposed into a plan that consists of the four steps *Obtain Permit, Hire Builder, Construction, and Pay Builder,*
- **Build Walls operator** can be decomposed into a plan involving wood, bricks, concrete, or vinyl.
- **Hammer(Nail)** to be primitive and **Build(House)** to be nonprimitive

Hierarchical decomposition of the operator

Build House into a partial-order plan.



HIERARCHICAL DECOMPOSITION

- To incorporate hierarchical decomposition, we have to make two additions to the description of each problem domain.
- First, we partition the set of operators into **primitive and nonprimitive operators**. *Hammer(Nail)* to be primitive and *Build(House)* to be nonprimitive.
- Second, we add a set of decomposition methods. Each method is an expression of the form ***Decompose(o, p)***, which means that a nonprimitive operator that unifies with *o* can be decomposed into a plan *p*. The decomposition of *Construction* is given below

Decompose(Construction,

*Plan(STEPS:{S₁ : Build(Foundation) S₂ : Build(Frame),
 S₃ : Build(Roof), S₄: Build(Walls),
 S₅ : Build(Interior)})*

ORDERINGS:{S₁ < S₂ < S₃ < S₅, S₂ < S₄ < S₅},

BINDINGS:{},

LINKS:{S₁ $\xrightarrow{\text{Foundation}}$ S₂, S₂ $\xrightarrow{\text{Frame}}$ S₃, S₂ $\xrightarrow{\text{Frame}}$ S₄, S₃ $\xrightarrow{\text{Roof}}$ S₅, S₄ $\xrightarrow{\text{Walls}}$ S₅}})

Decomposition method

- A *decomposition method* is like a subroutine or macro definition for an operator.
- A plan P correctly implements an operator o if it is a complete and consistent plan for the problem of achieving the effects of o given the preconditions of o:
- 1. p must be consistent. (There is no contradiction in the ordering or variable binding constraints of p.)
- 2. Every effect of o must be asserted by at least one step of p (and is not denied by some other, later step of p).
- 3. Every precondition of the steps in p must be achieved by a step in p or be one of the preconditions of o.

HD-POP Algorithm

function HD-POP(*plan*, *operators*, *methods*) **returns** *plan*

inputs: *plan*, an abstract plan with start and goal steps (and possibly other steps)

loop do

if SOLUTION?(*plan*) **then return** *plan*

S_{need} , *c* \leftarrow SELECT-SUB-GOAL(*plan*)

CHOOSE-OPERATOR(*plan*, *operators*, S_{need} , *c*)

$S_{nonprim} \leftarrow$ SELECT-NONPRIMITIVE(*plan*)

CHOOSE-DECOMPOSITION(*plan*, *methods*, $S_{nonprim}$)

RESOLVE-THREATS(*plan*)

end

On each iteration of the loop we first achieve an unachieved condition (CHOOSE-OPERATOR), then Decompose a nonprimitive operator (CHOOSE-DECOMPOSITION), then resolve threats.

HD-POP Working Principle,

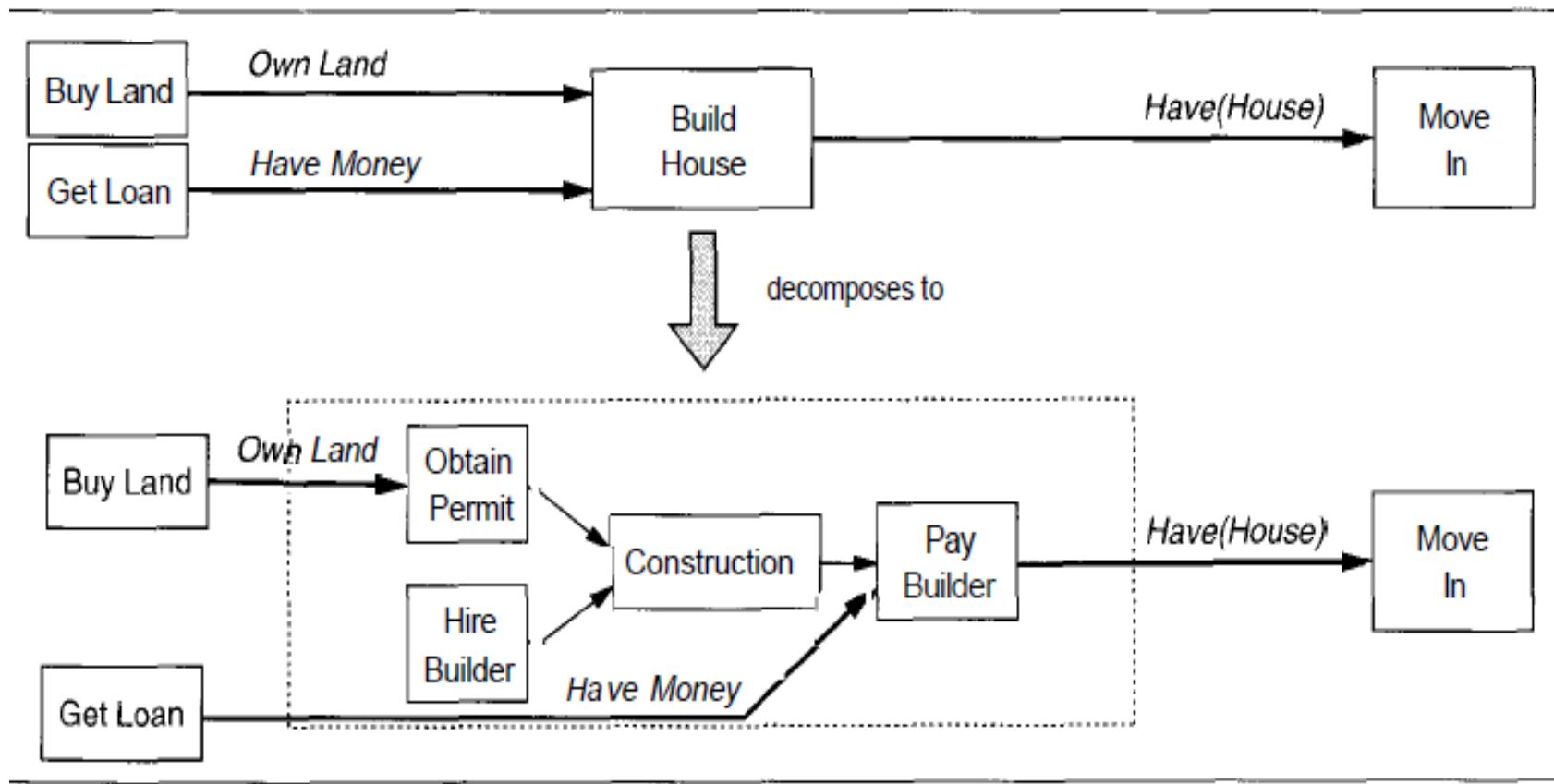
- Check that every step of the plan is primitive.
- **SELECT-NONPRIMITIVE** arbitrarily selects a nonprimitive step from the plan.
- The function **CHOOSE-DECOMPOSITION** picks a decomposition method for the plan and applies it.
- If *method* is chosen as the decomposition for the step $S_{nonprim}$, then the fields of the plan are altered as follows:
 - **STEPS:** Add all the steps of *method* to the plan, but remove $S_{nonprim}$.
 - **BINDINGS:** Add all the variable binding constraints of *method* to the plan. Fail if this introduces a contradiction.
 - **ORDERINGS:** Following the principle of least commitment, Replace each ordering constraint of the form $S_a < S_{nonprim}$ with constraint(s) that order S_a before the latest step(s) of *method*. That is, if S_m is a step of *method*, and there is no other S_j in *method* such that $S_m < S_j$, then add the constraint $S_a < S_m$.

HD-POP Working Principle

- Similarly, replace each constraint of the form $S_{nonprim} < Sz$ with constraint(s) that order Sz after the earliest step(s) of method.
- call **RESOLVE-THREATS** to add any additional ordering constraints that may be needed.
- **LINKS:** It is easier to match up causal links with the right substeps of method than it is to match up ordering constraints.
- If $S_i \xrightarrow{c} S_{nonprim}$ was a causal link in plan,
- replace it by a set of links $S_i \xrightarrow{c} S_m$, where each S_m is a step of method that has c as a precondition, and there is no earlier step of method that has c as a precondition.
- If there are several such steps with c as a precondition, then put in a causal link for each one. If there are none, then the causal link from S_i , can be dropped, because c was an unnecessary precondition of $S_{nonprim}$.
- Similarly, for each link $S_{nonprim} \xrightarrow{c} S_j$ in plan,
- replace it with a set of links $S_m \xrightarrow{c} S_j$, where S_m is a step of method that has c as an effect and there is no later step of method with c as an effect.

Detailed decomposition

one of the causal links that leads into the **nonprimitive step *Build House*** ends up being attached to the first step of the decomposition, but the other causal link is attached to a later step.

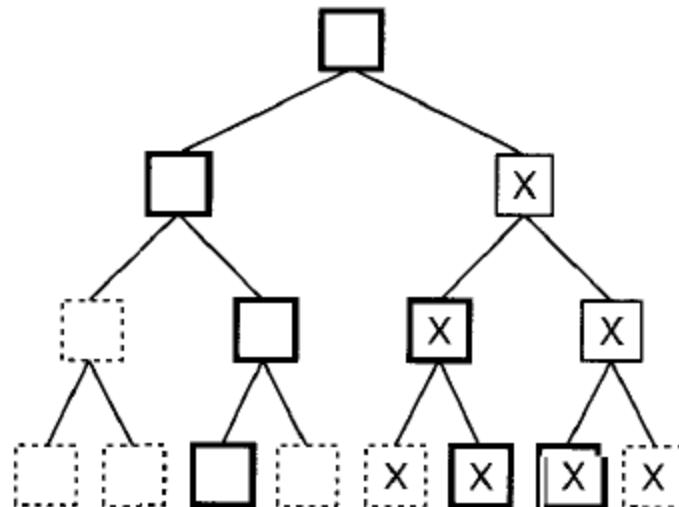


Finding abstract solution

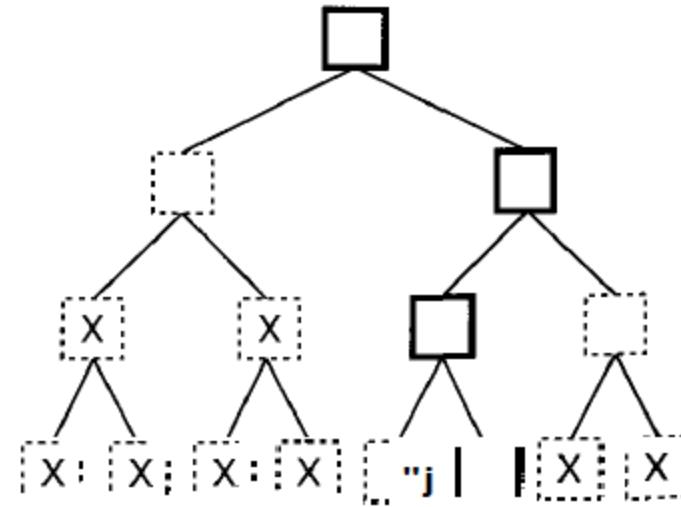
- **abstract solution**—a plan that contains abstract operators, but is consistent and complete.
- Finding a small abstract solution, if one exists, should not be too expensive.
- finding an abstract solution, and in **rejecting** other **abstract plans as inconsistent**.
- The following properties to hold:
- If p is an **abstract solution**, then there is a primitive solution of which p is an abstraction. If this property holds, then once an abstract solution is found we can prune away all other abstract plans from the search tree. This property is the **downward solution property**.
- If an abstract plan is **inconsistent**, then there is no primitive solution of which it is an abstraction. If this property holds, then we can prune away all the descendants of any inconsistent abstract plan. This is called the **upward solution property because it also** means that all complete abstractions of primitive solutions are abstract solutions.

Downward and upward solution

- ❖ Each box represents an entire plan (not just a step).
- ❖ Each arc represents a decomposition step in which an abstract operator is expanded.
- ❖ At the top is a very abstract plan, and at the bottom are plans with only primitive steps.
- ❖ The boxes with bold outlines are (possibly abstract) solutions.
- ❖ The boxes with dotted outlines are inconsistent.
- ❖ Plans marked with an "X" need not be examined by the planning algorithm.



(a) Downward Solution Property



(b) Upward Solution Property

Decomposition versus approximation

- **Hierarchical decomposition**, is the idea that an abstract nonprimitive operator can be decomposed into a more complex network of steps.
- The abstraction here is on the granularity with which operators interact with the world.
- **Abstraction hierarchies** capture the idea that a single operator can be planned with at different levels of abstraction.
- At the primitive level, the operator has a full set of preconditions and effects; at higher levels, the planner ignores some of these details.
- An **approximation hierarchy** planner takes an operator and partitions its preconditions according to their **criticality level**, for example:

*Op(ACTION:*Buy*(*x*),
EFFECT:*Have*(*x*) A \neg *Have*(*Money*),
PRECOND: 1:*Sells*(*store*,*x*) A
2:*At*(*store*) A
3: *Have*(*Money*))*

Decomposition versus approximation

- An **approximation hierarchy planner** first solves the problem using only the **preconditions of criticality 1**.
- The solution would be a plan that buys the right things at the right stores, but does not worry about how to travel between stores or how to pay for the goods.
- The idea is that it will be easy to find an abstract solution like this, because most of the pesky details are ignored. Once a solution is found, it can be **expanded** by considering the preconditions at **criticality level 2**.

POP-DUNC

- Partial-Order Planner with Disjunction, Universal quantification, Negation and Conditional effects.
- **Conditional effects:** Operators with conditional effects have different effects depending on what the world is like when they are executed.
- simple blocks world has only one real **action**—moving a block from one place to another. **Move(b,x,y)** and **MoveToTable(b,x)**

Op(ACTION:Move(b, x, v),

PRECOND:On(b, x) A Clear(b) A Clear(y),

EFFECT: On(b, y) A Clear(x) A \neg On(b, x) A \neg Clear(y))

Op(ACTION:MoveToTable(b, x),

PRECOND:On(b, x) A Clear(b),

EFFECT: On($b, Table$) A Clear(x) A \neg On(b, x))

- Suppose that the initial situation includes *On(A,B)* and we have the goal *Clear(B)*.
- We can achieve the goal by moving A off B, but *unfortunately we are forced to choose whether we want to move A to the table or to somewhere else.*

Effect when condition

- syntax "*effect when condition*" used, where *effect* and *condition* are both literals or conjunctions of literals.
- the *effect part* refers to the situation that is the result of the operator, and the *condition part* refers to the situation before the operator is applied.
- Thus, "**Q when P**" is not the same as the logical statement $P \Rightarrow Q'$, rather it is equivalent to the situation calculus statement $P(s) = Q(\text{Result}(act, s))$.
- *The conditional Move operator is written as follows:*

*Op(ACTION:Move(b, x, y),
 PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y)$,
 EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x)$
 $\wedge \neg Clear(y)$ when $y \neq Table$)*

- **SELECT-SUB-GOAL**, if a precondition c in a conditional effect of the form e when c should be considered as a candidate for selection. The answer is that if the effect e supplies a condition that is protected by a causal link, then we should consider selecting c , but not until the causal link is there.

Effect when condition

- **RESOLVE-THREAT:** Any step that has the effect ($\sim c'$ when p) is a possible threat to the causal link $S_i \xrightarrow{c} S_j$, whenever c and c' unify.
- **resolve the threat** by making sure that p does not hold. This technique **confrontation**.
- **Example of Confrontation:**
- In the blocks world, if we need a given block to be clear in order to carry out some step, then it is possible for the *Move(b , x , y) operator to threaten this condition if y is uninstantiated (i.e where is Y ? on top of some or on table).*
- *However, the threat only occurs if $y \neq Table$;*
- *confrontation removes the threat by setting $y = Table$.*

RESOLVE-THREATS with the confrontation technique

procedure RESOLVE-THREATS(*plan*)

```
for each  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
    for each  $S_{threat}$  in STEPS(plan) do
        for each  $c'$  in EFFECT( $S_{threat}$ ) do
            if SUBST(BINDINGS(plan),  $c$ ) = SUBST(BINDINGS(plan),  $\neg c'$ ) then
                choose either
                    Promotion: Add  $S_{threat} \prec S_i$  to ORDERINGS(plan)
                    Demotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
                    Confrontation: if  $c'$  is really of the form ( $c' \text{ when } p$ ) then
                        CHOOSE-OPERATOR(plan, operators,  $S_{threat}$ ,  $\neg p$ )
                        RESOLVE-THREATS(plan)
                if not CONSISTENT(plan) then fail
            end
        end
    end
```

Negated and disjunctive goals

- The confrontation technique calls **CHOOSE-OPERATOR** with the goal $\sim p$.
- unification function allows p to match $\sim p$.
- goal of the form $\sim p$ can be matched either by an explicit effect that unifies with $\sim p$ or by the initial state, if it does not contain p .
- **disjunctive precondition:** **SELECT-SUB-GOAL**, if we choose a step with a precondition of the form $p \vee q$, then we nondeterministically choose to return either p or q and reserve the other one as a backtrack point.
- **Any operator** with the precondition $p \vee q$ could be replaced with two operators, one with p as a precondition and one with q , but then the planner would have to commit to one or the other.
- $(p \vee q) a E$, split as $p a E, q a E$ a is action and E is Effect.
- **disjunctive effects are very difficult** to incorporate. They change the environment from deterministic to nondeterministic.
- A disjunctive effect is used to model random effects, or effects that are not determined by the preconditions of the operator.
- For ex: *Flip(coin)* have the disjunctive effect *Heads(coin) V Tails(coin)*.

Universal quantification

- **universally quantified preconditions:**
- Instead of writing $\text{Clear}(b)$ as a precondition,
- we can $\forall x \text{ Block}(x) \Rightarrow \neg \text{On}(x, b)$ instead.
- **universally quantified effects:**
- In shopping domain we could define the operator $\text{Carry}(\text{bag}, x, y)$ so that it has the effect that all objects that are in the bag are at y and are no longer at x . we can define this operator with universal quantification.

*Op(ACTION: $\text{Carry}(\text{bag}, x, y)$,
 PRECOND: $\text{Bag}(\text{bag}) \wedge \text{At}(\text{bag}, x)$,
 EFFECT: $\text{At}(\text{bag}, y) \wedge \neg \text{At}(\text{bag}, x)$ A
 $\forall i \text{ Item}(i) \Rightarrow (\text{At}(i, y) \wedge \neg \text{At}(i, x))$ when $\text{In}(i, \text{bag})$)*

- Expand universally quantified goals to eliminate the quantifier.
- $$\forall x \text{ } T(x) \Rightarrow C(x)$$
- where T is a unary type predicate on x , and C is a condition involving x .
 - expand this form into an equivalent conjunctive expression with no quantifiers: $\text{MX } T(x) \Rightarrow C(x) = C(x_1) \wedge \dots \wedge C(x_n)$

Universal quantification

- where x_1, \dots, x_n are the objects in the initial state that satisfy $T(x)$. Here is an example:

Initial State: $\text{Bag}(B) \wedge \text{Milk}(M_1) \wedge \text{Milk}(M_2) \wedge \text{Milk}(M_3)$

Expression: $\forall x \text{ Milk}(x) \Rightarrow \text{In}(x, B)$

Expansion: $\text{In}(M_1, B) \wedge \text{In}(M_2, B) \wedge \text{In}(M_3, B)$

- POP-DUNC Algorithm**

function SELECT-SUB-GOAL(*plan*) *returns plan, precondition conjunct*

pick a plan step S_{need} from STEPS(*plan*) with a precondition conjunct c that has not been achieved

if c is a universally quantified expression **then**

return S_{need} , EXPANSION(c)

else if c is a disjunction $c_1 \vee c_2$ **then**

return S_{need} , choose(c_1, c_2)

else **return** S_{need}, c

Choose Operator

procedure CHOOSE-OPERATOR($plan$, $operators$, S_{need} , c)

choose a step S_{add} from $operators$ or $\text{STEPS}(plan)$ that has c_{add} as an effect
such that $u = \text{UNIFY}(c, c_{add}, \text{BINDINGS}(plan))$

if there is no such step then fail

$u' \leftarrow u$ without the universally quantified variables of c_{add}

add u' to $\text{BINDINGS}(plan)$

add $S_{add} \xrightarrow{c} S_{need}$ to $\text{LINKS}(plan)$

add $S_{add} \prec S_{need}$ to $\text{ORDERINGS}(plan)$

if S_{add} is a newly added step from $operators$ then

add S_{add} to $\text{STEPS}(plan)$

add $Start \prec S_{add} \prec Finish$ to $\text{ORDERINGS}(plan)$

Resolve Threats

procedure RESOLVE-THREATS(*plan*)

```

for each  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
    for each  $S_{threat}$  in STEPS(plan) do
        for each  $c'$  in EFFECT( $S_{threat}$ ) do
            if SUBST(BINDINGS(plan),  $c$ ) = SUBST(BINDINGS(plan),  $\neg c'$ ) then
                choose either
                    Promotion: Add  $S_{threat} \prec S_i$  to ORDERINGS(plan)
                    Demotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
                    Confrontation: if  $c'$  is really of the form ( $c' \text{ when } p$ ) then
                        CHOOSE-OPERATOR(plan, operators,  $S_{threat}$ ,  $\neg p$ )
                        RESOLVE-THREATS(plan)
                    if not CONSISTENT(plan) then fail
                end
            end
        end
    
```

RESOURCE CONSTRAINTS

- how to handle money and other resources properly in problem domain.
- Precondition $\text{Have}(\$1.89)$ in shopping domain to shop an item i.
- **Using numeric – valued measures in planning**, such as money, volume, liter ..etc
- Measures can be referred to by logical terms such as $\$(1.50)$ or $\text{Gallons}(6)$ or GasLevel .
- *Measure functions such as Volume apply to objects such as GasInCar to yield measures: $\text{GasLevel} = \text{Volume}(\text{GasInCar}) = \text{Gallons}(6)$.*
- In planning problems, we are usually interested in **amounts that change over time**.
- A situation calculus representation would therefore include a situation argument (e.g., $\text{GasLevel}(s)$), but as usual in planning we will leave the situation implicit.
- expressions such as GasLevel ; **measure fluents**.

Example - 1 of measure

- For example, in a shopping problem, we might want to state that the amount of money the agent has, *Cash*, must be nonnegative;
- amount of gas* in the tank, *GasLevel*, can range up to 15 gallons;
- price of gas ranges from \$1.00 to \$1.50 per gallon; and
- price of milk ranges from \$ 1.00 to \$1.50 per quart:

$$$(0) < \text{Cash}$$

$$\text{Gallons}(0) < \text{GasLevel} < \text{Gallons}(15)$$

$$$(1.00) < \text{UnitPrice(Gas)} \times \text{Gallons}(1) < $(1.50)$$

$$$(1.00) < \text{UnitPrice(Milk)} \times \text{Quarts}(1) < $(1.50)$$

- Cash* and *GasLevel*, are treated as *resources* that can be produced and consumed.
- initial state is described by the effects of the start action:

Op(ACTION:Start,

EFFECT: Cash—\$(12.50)A

GasLevel ← Gallons(5)A

Example - 1 of measure

The *Buy* action reduces the amount of Cash one has:

Op(ACTION:Buy(x, store),
EFFECT:Have(x)A Cash ← Cash — Price(x, store))

Getting gas can be described by an abstract *Fillup* operator:

Op(ACTION:Fillup(GasLevel),
EFFECT: GasLevel — Gallons(15)A
Cash ← Cash - (UnitPrice(Gas) x (Gallons(15) - GasLevel)))

- ❖ **Temporal constraints:** The initial state specifies a start time for the plan. for example, *Time* \leftarrow 8:30.
- ❖ how much time each operation consumes. (In the case of time, of course, consumption means *adding to the amount*.)
- ❖ Suppose it takes 10 seconds to pump each gallon of gas, and 3 minutes to do the rest of the *Fillup action*. Then an effect of *Fillup* is
- ❖ *Time* \leftarrow *Time* + Minutes(3) + (Seconds(10)/Gallons(1))*x* (Gallons(15) - *GasLevel*)
- ❖ It is handy to provide the operator *Wait(x)*, which has the effect *Time* \leftarrow *Time* + *x* and no other preconditions or effects

Video Reference links

S. NO	Description	Video link
1	This video describes the Drawback of state space search. Why Planning in AI	https://youtu.be/MklNx77CDZc
2	This video gives the knowledge of the Partial order planner	https://youtu.be/-rYpkxm3J3A
3	This video provides the Goal stack planning - actions, preconditions, effects	https://youtu.be/WlG0H0u8aCg
4	This video demonstrates the Goal stack planning- example	https://youtu.be/kHwjLOZe_jY
5	This video explains the Heuristic based goal stack planning	https://youtu.be/-NrzPWUnYIE
6	This video demonstrates the POP quiz example	https://youtu.be/uPeEfd6CZe8
7	This video enhances the knowledge of the State space Planner	https://youtu.be/ngVOPThLc2A
8	This video explains the Hierarchical Decomposition	https://youtu.be/xzkAQ_KPlgY
9	This video describe the importance of HP. Due to more number of steps in POP. Why Hierarchical planning	https://youtu.be/7L3tcoFMR7w
10	This video explains the Hierarchical planning.	https://youtu.be/FSZT07cLIWg

Thank you

UNIT - 5

GENERALIZED MODELS

**Ref: Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig,
3rd Edition, Prentice Hall**

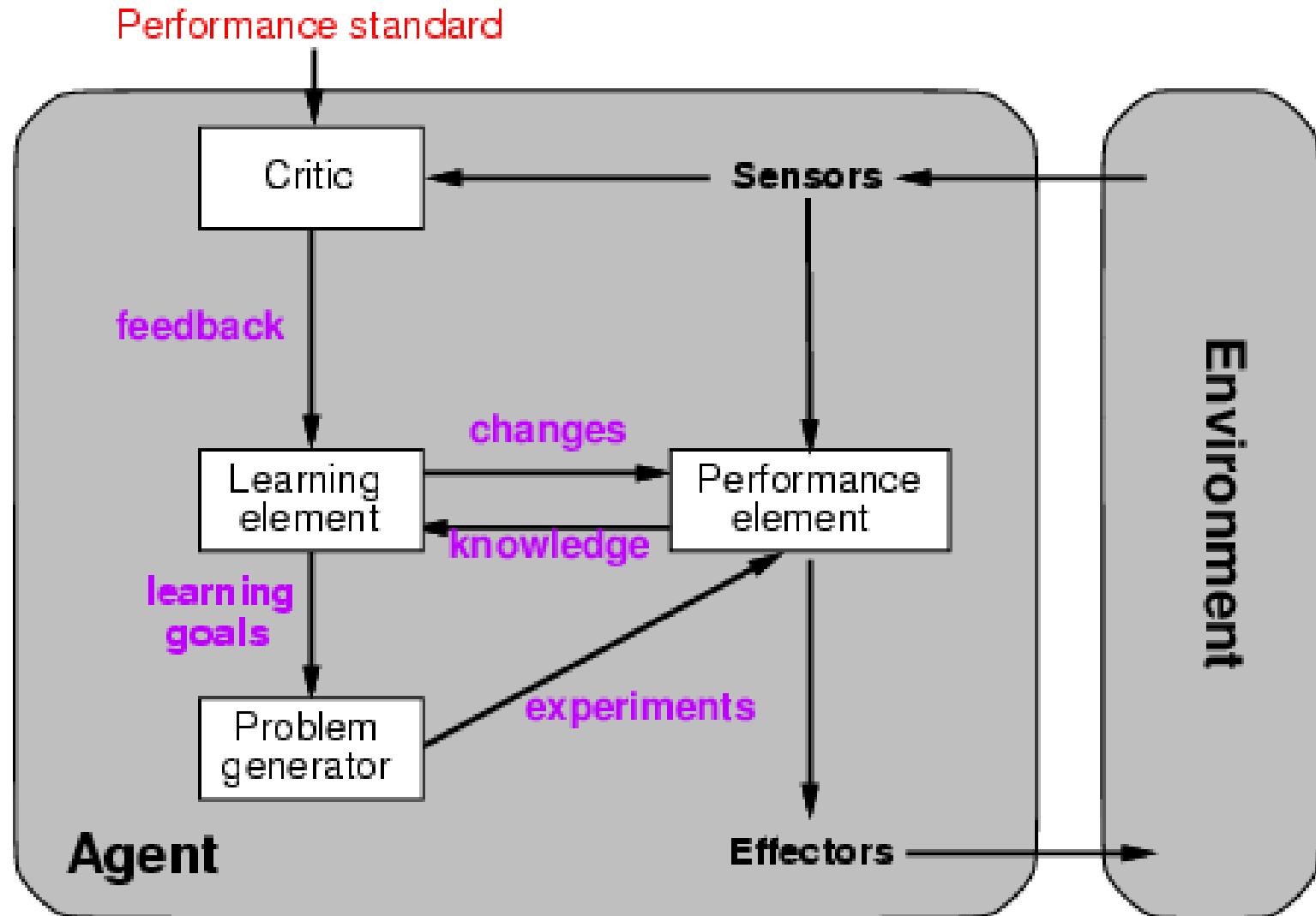
UNIT-5

- **Unit V** **(8 Hours)**
- **Generalized Models**
- A General Model of Learning Agents, Components of the performance element, Representation of the components, Inductive Learning, Learning Decision Trees, Using Information Theory, Learning General Logical Descriptions, Computational Learning Theory, Learning in Neural and Belief Networks, Neural Networks, Perceptrons, Multilayer Feed-Forward Networks, Bayesian Methods for Learning Belief Networks, Reinforcement Learning, Passive Learning in a Known Environment, Passive Learning in an Unknown Environment, Generalization in Reinforcement Learning.

Learning

- The idea behind learning is that **percepts** should be used **not only for acting**, but also for improving the agent's ability to act in the future.
- Learning takes place as a result of the **interaction** between the **agent and the world**, and from **observation** by the agent of its **own decision-making processes**.
- Learning is essential for unknown environments,
 - i.e., when designer lacks omniscience
- Learning is useful as a system construction method,
 - i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to improve performance

Generic Model of Learning agents



Components of Learning Agents

- A learning agent can be divided into four conceptual components:
- **Learning element**, which is responsible for making improvements.
- The learning element takes some knowledge about the Performance element and some feedback on how the agent is doing, and determines how the performance element should be modified to (hopefully) do better in the future.
- The design of the learning element depends very much on the design of the performance element.
- **Performance element**, which is responsible for selecting external actions.
- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- The **critic is designed to tell the learning element how well the agent is doing. The critic** employs a fixed standard of performance.
- This is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program may receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; otherwise agent can improve its behavior.

Components of Learning Agents

- **Problem generator.** It is responsible for suggesting actions that will lead to new and informative experiences.
- The problem generator's job is to suggest these exploratory actions to improve the performance of an agent.
- **speedup learning:** solve the same problem faster next time.
- For example, when asked to make a trip to a new destination, the taxi might take a while to consult its map and plan the best route. But the next time a similar trip is requested, the planning process should be much faster.
- The **design of the learning element** is affected by **four major issues**:
- Which *components* of the performance element are to be improved.
- What *representation* is used for those components.
- What *feedback* is available.
- What *prior information* is available.

Design of learning element

- The **components** of the *performance element* can include the following:
- 1. A **direct mapping** from conditions on the current state to actions.
- 2. A means to **infer** relevant properties of the world from the percept sequence.
- 3. Information about the way the **world evolves**.
- 4. Information about the results of **possible actions** the agent can take.
- 5. **Utility information** indicating the desirability of world states.
- 6. **Action-value information** indicating the desirability of particular actions in particular states.
- 7. **Goals** that describe classes of states whose achievement maximizes the agent's utility.

Design of Learning element

- **Representation of the components**
- Any of these components can be represented using any of the representation schemes.
- **deterministic descriptions** such as linear weighted polynomials.
- **propositional and first-order logical** sentences for all of the components in a logical agent.
- **probabilistic descriptions** such as belief networks
- **Type of feedback:**
 - **Supervised learning**: correct answers for each example
 - **Unsupervised learning**: correct answers not given
 - **Reinforcement learning**: occasional reward
- **Prior knowledge:**
- Most human learning takes place in the context of a good deal of background knowledge.
- Some psychologists and linguists claim that even newborn babies exhibit knowledge of the world.
- Whatever the truth of this claim, there is no doubt that prior knowledge can help enormously in learning.

INDUCTIVE LEARNING

- This involves the process of *learning by example* -- where a system tries to induce a general rule from a set of observed instances.
- This involves classification -- assigning, to a particular input, the name of a class to which it belongs. Classification is important to many problem solving tasks.
- A learning system has to be capable of evolving its own class descriptions:
 - Initial class definitions may not be adequate.
 - The world may not be well understood or rapidly changing.
- The task of constructing class definitions is called *induction* or *concept learning*
- More formally, we say an **example is a pair $(x, f(x))$** ,
- where x is the input and $f(x)$ is the output of the function applied to x .
- **The task of pure inductive inference (or induction) is this: given a collection of examples of f , return a function h that approximates f . The function h is called a hypothesis.**

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

(This is a highly simplified model of real learning:

- Ignores prior knowledge
- Assumes examples are given)

simple reflex learning agent

global *examples* — { }

function REFLEX-PERFORMANCE-ELEMENT(*percept*) **returns** an action

if (*percept*, *a*) *in* *examples* **then return** *a*

else

h \leftarrow INDUCE(*examples*)

return *h*(*percept*)

procedure REFLEX-LEARNING-ELEMENT(*percept*, *action*)

inputs: *percept*, feedback *percept*

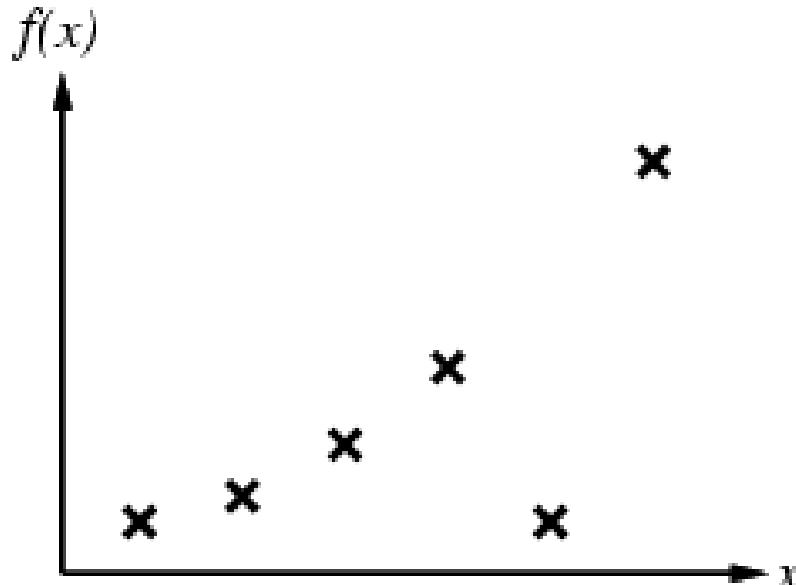
action, feedback *action*

examples \leftarrow *examples* \cup {(*percept*,*action*)}

- ❖ The learning element just stores each example percept/action pair.
- ❖ The performance element either does whatever was done last time for a given percept, or it induces an action from similar percepts.

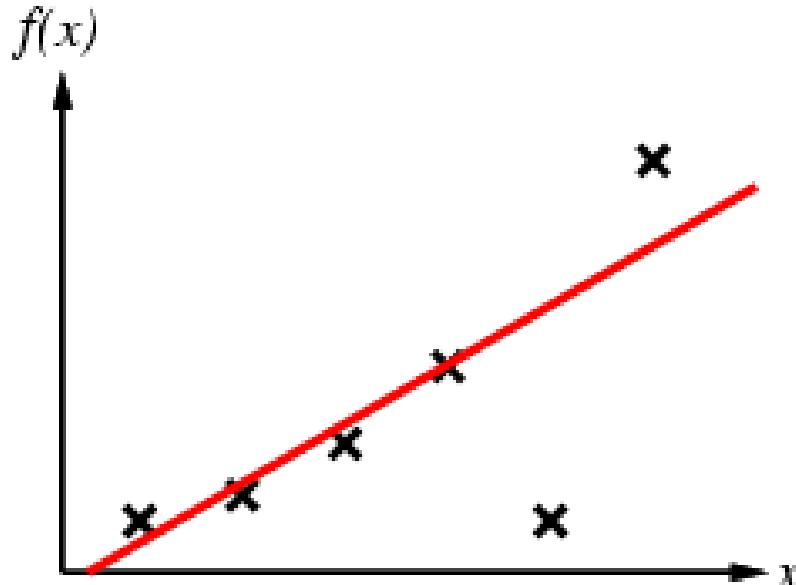
Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:
-



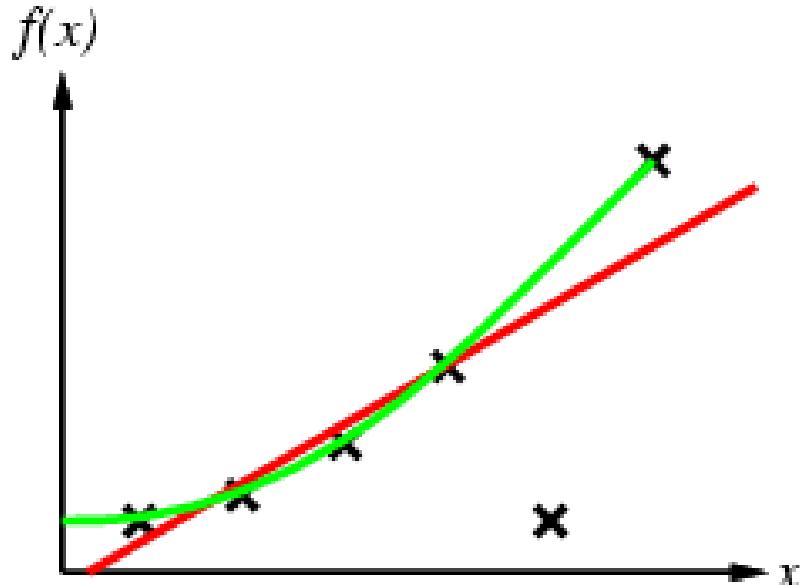
Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:
-



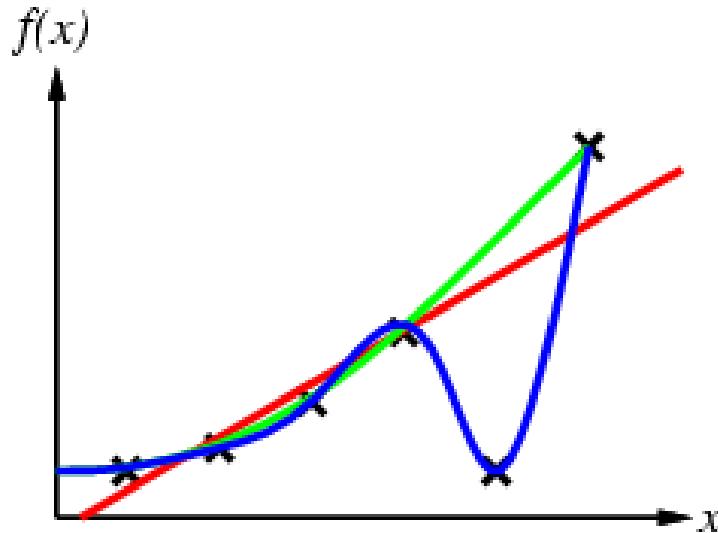
Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:
-



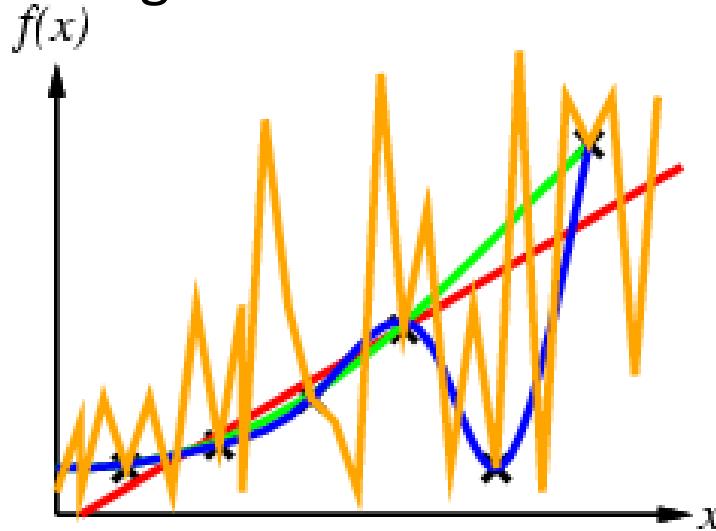
Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:
-



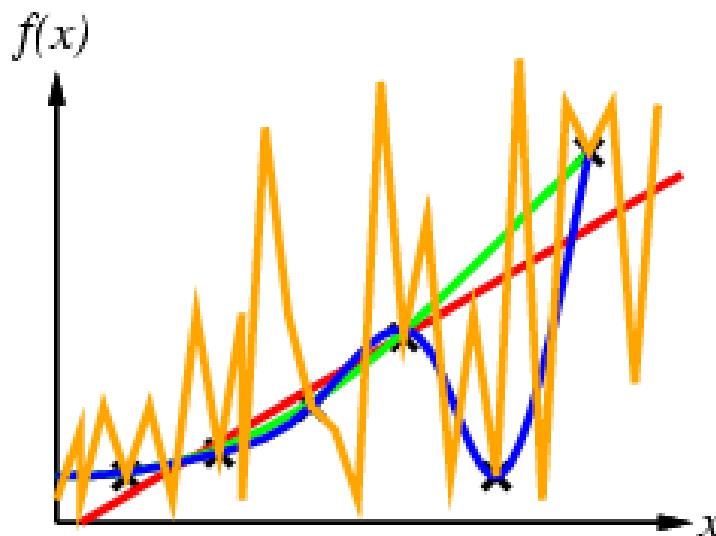
Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:



Inductive learning method

- Construct/adjust h to agree with f on training set
- (h is **consistent** if it agrees with f on all examples)
- E.g., curve fitting:



Any preference for one hypothesis over another, beyond mere consistency with the examples, is called a **bias**.

- Ockham's razor: prefer the simplest hypothesis consistent with data

Inductive Learning Approaches

- Two approaches to learning logical sentences: **decision tree methods and version-space approach.**
- A **decision tree** takes as input an object or situation described by a set of properties, and outputs a yes/no "decision."
- **version-space approach:** The original hypothesis space can be viewed as a disjunctive sentence

$$H_1 \vee H_2 \vee H_3 \dots \vee H_n$$

- As various hypotheses are found to be **inconsistent with the examples**, this disjunction shrinks, retaining only those hypotheses not ruled out.
- Assuming that the original hypothesis space does in fact contain the right answer, the reduced disjunction must still contain the right answer because only incorrect hypotheses have been removed.
- The set of hypotheses remaining is called the **version space, and the learning algorithm is called the version space learning algorithm** (also the **candidate elimination algorithm**).

THE ILA ALGORITHM

- Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm which is used for generating a set of a classification rule, which produces rules of the form “IF-THEN”, for a set of examples, producing rules at each iteration and appending to the set of rules.
- **General requirements at start of the algorithm:-**
- list the examples in the form of a table ‘T’ where each row corresponds to an example and each column contains an attribute value.
- create a set of m training examples, each example composed of k attributes and a class attribute with n possible decisions.
- create a rule set, R, having the initial value false.
- initially all rows in the table are unmarked.
- **Note: Examples to General rules**

Steps in the ILA algorithm

- **1:** divide the table 'T' containing m examples into n sub-tables (t_1, t_2, \dots, t_n). One table for each possible value of the class attribute. (repeat steps 2-8 for each sub-table).
- **2:** Initialize the attribute combination count 'j' = 1.
- **3:** For the sub-table on which work is going on, divide the attribute list into distinct combinations, each combination with 'j' distinct attributes.
- **4:** For each combination of attributes, count the number of occurrences of attribute values that appear under the same combination of attributes in unmarked rows of the sub-table under consideration, and at the same time, not appears under the same combination of attributes of other sub-tables. Call the first combination with the maximum number of occurrences the max-combination 'MAX'.
- **5:** If 'MAX' = null , increase 'j' by 1 and go to Step 3.
- **6:** Mark all rows of the sub-table where working, in which the values of 'MAX' appear, as classified.
- **7:** Add a rule (IF attribute = "XYZ" \rightarrow THEN decision is YES/ NO) to R.
- **8:** If all rows are marked as classified, then move on to process another sub-table and go to Step 2. else, go to Step 4. If no sub-tables are available, exit with the set of rules obtained till then.

An example showing the use of ILA

EXAMPLE NO.	PLACE TYPE	WEATHER	LOCATION	DECISION
I)	hilly	winter	kullu	Yes
II)	mountain	windy	Mumbai	No
III)	mountain	windy	Shimla	Yes
IV)	beach	windy	Mumbai	No
V)	beach	warm	goa	Yes
VI)	beach	windy	goa	No
VII)	beach	warm	Shimla	Yes

Creating training examples with class attribute

subset 1

S.NO	PLACE TYPE	WEATHER	LOCATION	DECISION
1	hilly	winter	kullu	Yes
2	mountain	windy	Shimla	Yes
3	beach	warm	goa	Yes
4	beach	warm	Shimla	Yes

subset 2

S.NO	PLACE TYPE	WEATHER	LOCATION	DECISION
5	mountain	windy	Mumbai	No
6	beach	windy	Mumbai	No
7	beach	windy	goa	No

Create General rules set from examples

- **Finally we get the rule set :-**
- **Rule Set**
- row 3 & 4 column weather is selected and row 3 & 4 are marked.
- **Rule 1:** IF the weather is warm THEN the decision is yes.
- row 1 column place type is selected and row 1 is marked.
- **Rule 2:** IF place type is hilly THEN the decision is yes.
- row 2 column location is selected and row 2 is marked.
- **Rule 3:** IF location is Shimla THEN the decision is yes.
- row 5&6 column location is selected and row 5&6 are marked.
- **Rule 4:** IF location is Mumbai THEN the decision is no.
- row 7 column place type & the weather is selected and row 7 is marked.
- **Rule 5:** IF place type is beach AND the weather is windy THEN the decision is no.

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range (\$, \$\$, \$\$\$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

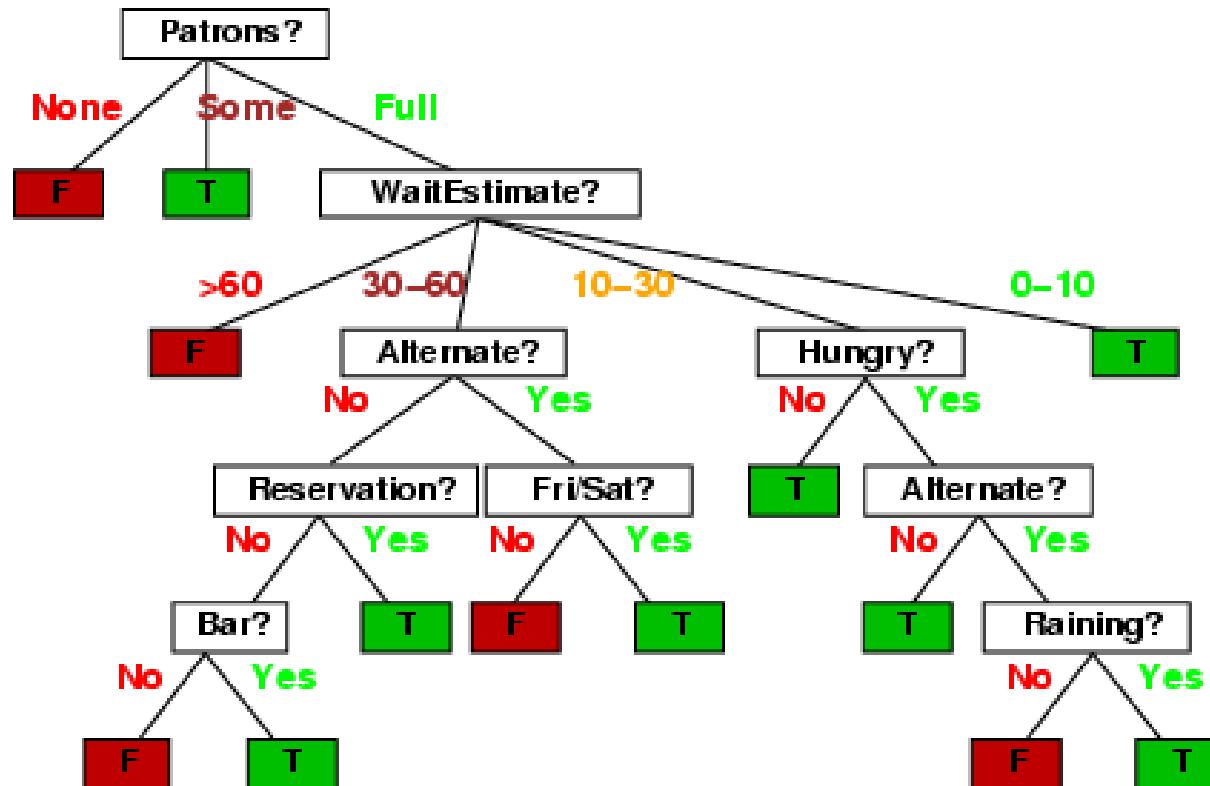
- Examples described by **attribute values** (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target Wait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

- Classification of examples is **positive (T)** or **negative (F)**

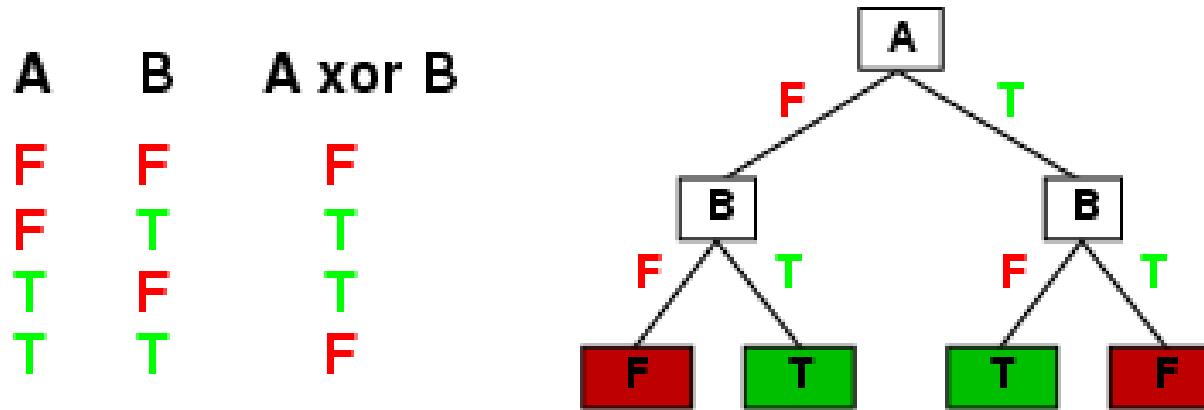
Decision trees

- One possible representation for hypotheses
- E.g., here is the “true” tree for deciding whether to wait:



Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless f nondeterministic in x) but it probably won't generalize to new examples
- Prefer to find more **compact** decision trees

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., *Hungry* \wedge \neg *Rain*)?

- Each attribute can be in (positive), in (negative), or out
 $\Rightarrow 3^n$ distinct conjunctive hypotheses
- More expressive hypothesis space
 - increases chance that target function can be expressed
 - increases number of hypotheses consistent with training set
 \Rightarrow may get worse predictions

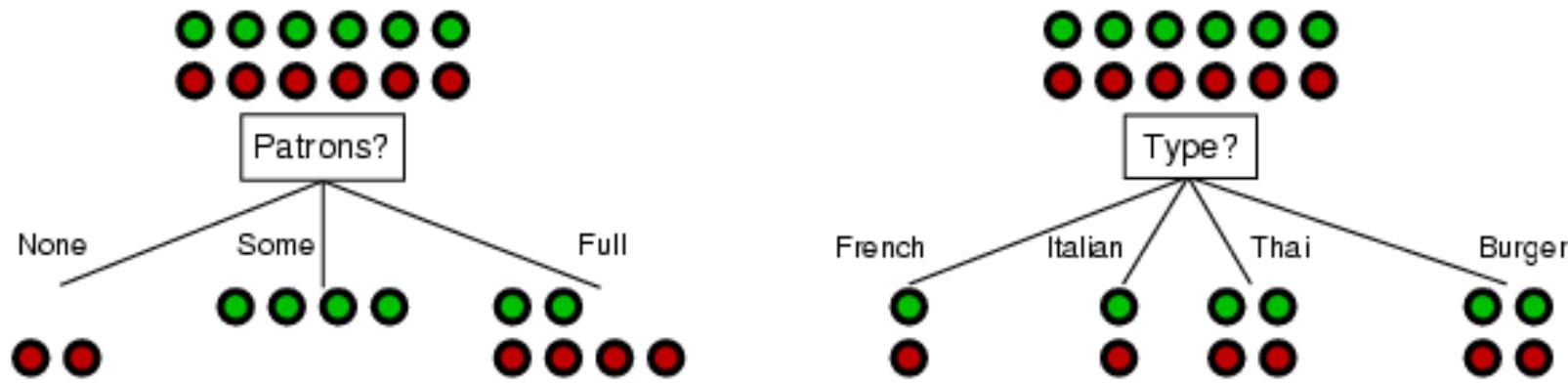
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value vi of best do
      examplesi ← {elements of examples with best = vi}
      subtree ← DTL(examplesi, attributes - best, MODE(examples))
      add a branch to tree with label vi and subtree subtree
  return tree
```

Choosing an attribute

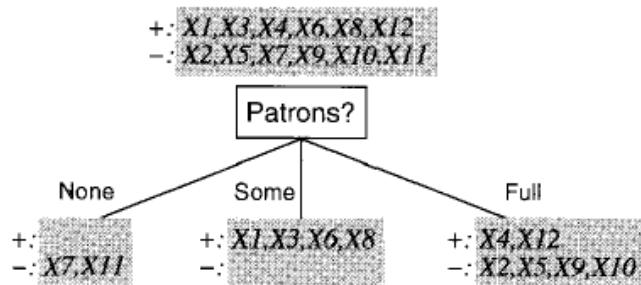
- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



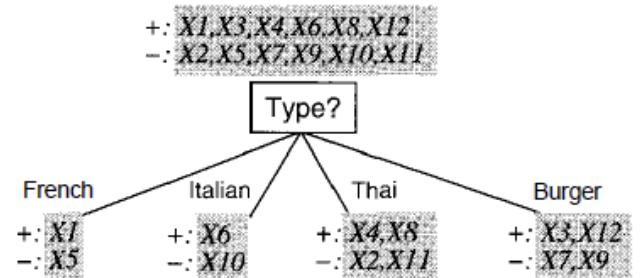
- Patrons?* is a better choice

Splitting the examples by testing on attributes.

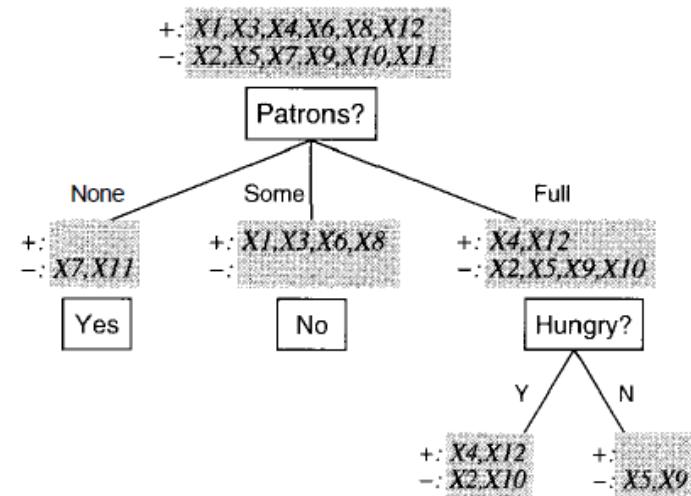
(a)



(b)



(c)



In (a), we see that *Patrons* is a good attribute to test first; in (b), we see that *Type* is a poor one; and in (c), we see that *Hungry* is a fairly good second test, given that *Patrons* is the first test.

Using information theory

- To implement Choose-Attribute in the DTL algorithm
- Information Content (Entropy):

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1 \text{ to } n} -P(v_i) \log_2 P(v_i)$$

- For a training set containing p positive examples and n negative examples:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Information gain

- A chosen attribute A divides the training set E into subsets E_1, \dots, E_v according to their values for A , where A has v distinct values.

$$\text{remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p + n}, \frac{n}{p + n}\right) - \text{remainder}(A)$$

- Choose the attribute with the largest IG

To compute IG an attribute

$$IG(\text{Patron}) = I(6/12, 6/12) - \text{remainder}(\text{patron});$$

$$\begin{aligned}I(6/12, 6/12) &= -(1/2)*\log_2(1/2) - (1/2) * \log_2(1/2) = -1/2[\log_2 1 - \log_2 2] - \frac{1}{2} * [\log_2 1 - \log_2 2] \\&= (-1/2)[0-1]-1/2[0-1] = \frac{1}{2} + \frac{1}{2} = 1\end{aligned}$$

Patron has three distinct values some, full and None.

$$\begin{aligned}\text{remainder}(\text{patron}) &= \frac{0+4}{6+6} I\left(\frac{0}{0+4}, \frac{4}{0+4}\right) + \frac{4+2}{6+6} I\left(\frac{4}{4+2}, \frac{2}{4+2}\right) + \frac{2+0}{6+6} I\left(\frac{2}{2+0}, \frac{0}{2+0}\right) \\&= \frac{4}{12} I\left(\frac{0}{4}, \frac{4}{4}\right) + \frac{6}{12} I\left(\frac{4}{6}, \frac{2}{6}\right) + \frac{2}{12} I\left(\frac{2}{2}, \frac{0}{2}\right) \\&= \frac{4}{12} I(0,1) + \frac{6}{12} I\left(\frac{4}{6}, \frac{2}{6}\right) + \frac{2}{12} I(1,0)\end{aligned}$$

$$I(0,1) = -0 * \log_2(0) - 1 * \log_2(1) = 0$$

$$I(1,0) = -1 * \log_2(1) - 0 * \log_2(0) = 0$$

i	Patron value (A)	No. of positive samples	No. of Negative samples
1	Some	0	4
2	Full	4	2
3	None	2	0

$$\begin{aligned}I\left(\frac{4}{6}, \frac{2}{6}\right) &= -\frac{4}{6} * \log_2\left(\frac{4}{6}\right) - \frac{2}{6} * \log_2\left(\frac{2}{6}\right) = -\frac{4}{6} * (\log_2(4) - \log_2(6)) - \frac{2}{6} * (\log_2(2) - \log_2(6)) \\&= -\frac{4}{6} * (2 - 2.58) - \frac{2}{6} * (1 - 2.58) = -\frac{4}{6} * (-0.58) - \frac{2}{6} * (-1.58) = \frac{4 * 0.58 + 2 * 1.58}{6} = 0.913\end{aligned}$$

$$\text{remainder}(\text{patron}) = 0 + 0.913 + 0 = 0.913$$

$$\text{Therefore } IG(\text{patron}) = 1 - \text{remainder}(\text{patron}) = 1 - 0.913 = 0.087$$

Information gain

For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

Consider the attributes *Patrons* and *Type* (and others too):

$$IG(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .087 \text{ bits}$$

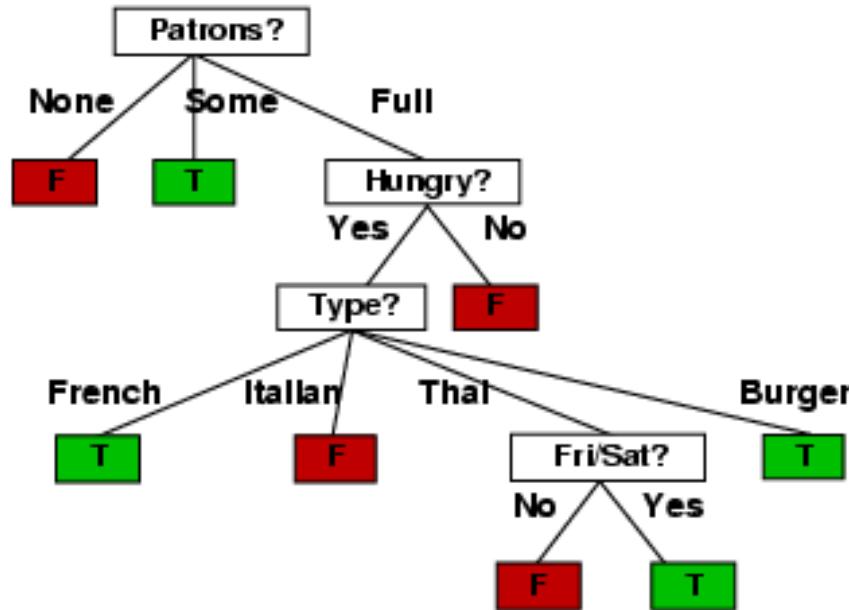
$$IG(Type) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

i	Type value (A)	No. of positive samples	No. of Negative samples
1	Thai	2	2
2	French	1	1
3	Burger	2	2
4	Italian	1	1

Example contd.

- Decision tree learned from the 12 examples:



- Substantially simpler than “true” tree---a more complex hypothesis isn’t justified by small amount of data

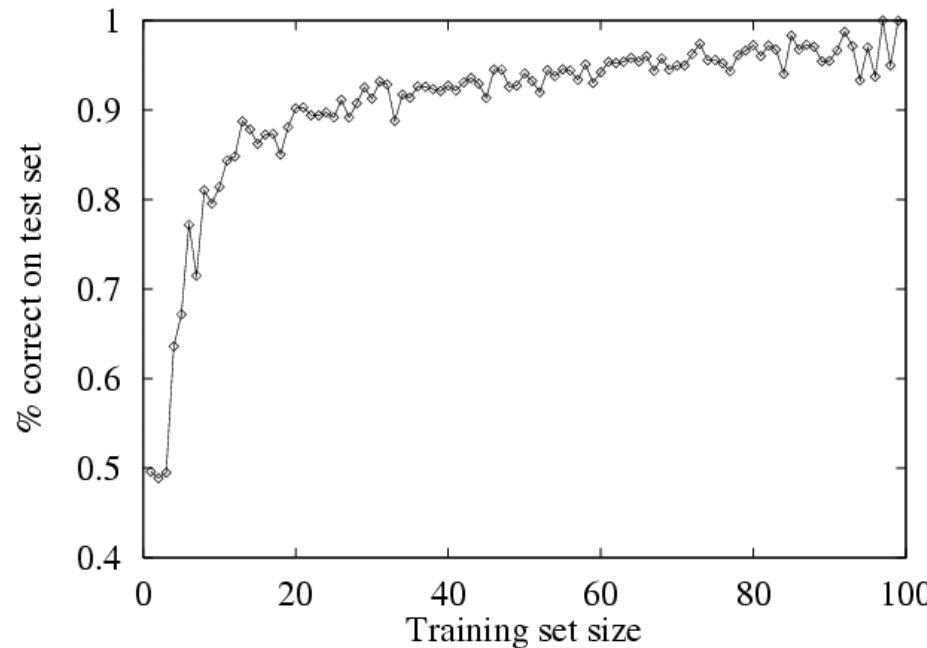
Assessing the performance of the learning algorithm

- A learning algorithm is good if it produces hypotheses that do a good job of predicting the classifications of unseen examples.
- it is more convenient to adopt the following methodology:
- 1. Collect a large set of examples.
- 2. Divide it into two disjoint sets: the **training set and the test set**.
- 3. Use the learning algorithm with the training set as examples to generate a hypothesis H .
- 4. Measure the percentage of examples in the test set that are correctly classified by H .
- *5. Repeat steps 1 to 4 for different sizes of training sets and different randomly selected training sets of each size.*
- The result of this is a set of data that can be processed to give the average prediction quality as a function of the size of the training set.
- This can be plotted on a graph, giving what is LEARNING CURVE called the **learning curve for the algorithm on the particular domain**.

Performance measurement

- How do we know that $h \approx f$?
 1. Use theorems of computational/statistical learning theory
 2. Try h on a new **test set** of examples
(use **same** distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size



Summary

- Learning needed for unknown environments, lazy designers
- Learning agent = performance element + learning element
- For supervised learning, the aim is to find a simple hypothesis approximately consistent with training examples
- Decision tree learning using information gain
- Learning performance = prediction accuracy measured on test set

Neural Networks

Dr. Devaraj Verma C,
Associate Professor, Dept of
CST(AI)

Courtesy: Shilpa S

Agenda

- Introduction to Neural Networks
- Single Perceptron
- Multilayer Neural Networks
- Bayesian Methods for Learning Belief N/W
- Recurrent Neural Networks

Introduction to Neural Networks

The simple arithmetic computing elements correspond to neurons—the cells that perform information processing in the brain—and the network as a whole corresponds to a collection of interconnected neurons are called neural networks.

The neuron, or nerve cell, is the fundamental functional unit of all nervous system tissue, including the brain. Each neuron consists of a cell body, that contains a cell nucleus.

Introduction to Neural Networks

Comparing brains with digital computers

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates 10^9 bits RAM, 10^{10} bits	10^{11} neurons 10^{11} neurons, 10^{14} synapses
Storage units	disk	
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Introduction to Neural Networks

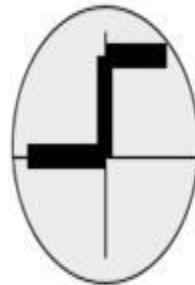
- A neural network is composed of a number of nodes, or units, connected by links.
- Each link has a numeric weight associated with it. Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights.
- Some of the units are connected to the external environment, and can be designated as input or output units.

Introduction-ANN

What is an artificial neural network?

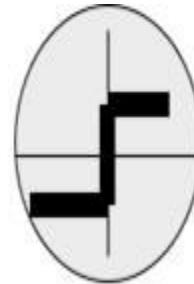
- Artificial neural networks are one of the main tools used in machine learning. As the “neural” part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn.
- Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use

Units of ANN



Step function
(Linear Threshold Unit)

$\text{step}(x) = 1, \text{ if } x \geq \text{threshold}$
 $0, \text{ if } x < \text{threshold}$



Sign function

$$\begin{aligned} \text{sign}(x) = +1, & \text{ if } x \geq 0 \\ & -1, \text{ if } x < 0 \end{aligned}$$



Sigmoid function

$$\text{sigmoid}(x) = 1/(1+e^{-x})$$

PERCEPTRONS

- Perceptron is a single layer neural network.
 - A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise
 - Given inputs x_1 through x_n , the output $\mathbf{O}(x_1, \dots, x_n)$ is defined as
- $$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \dots + w_nx_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$
- where each w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output.
 - $-w_0$ is a threshold that the weighted combination of inputs $w_1x_1 + \dots + w_nx_n$ must surpass in order for the perceptron to output a 1.

PERCEPTRONS

Sometimes, the perceptron function is written as,

$$O(\vec{x}) = \text{sgn} (\vec{w} \cdot \vec{x})$$

Where,

$$\text{sgn}(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Learning a perceptron involves choosing values for the weights w_0, \dots, w_n . Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

Why do we need Weights and Bias?

Weights shows the strength of the particular node.

A bias value allows you to shift the activation function curve up or down

PERCEPTRONS

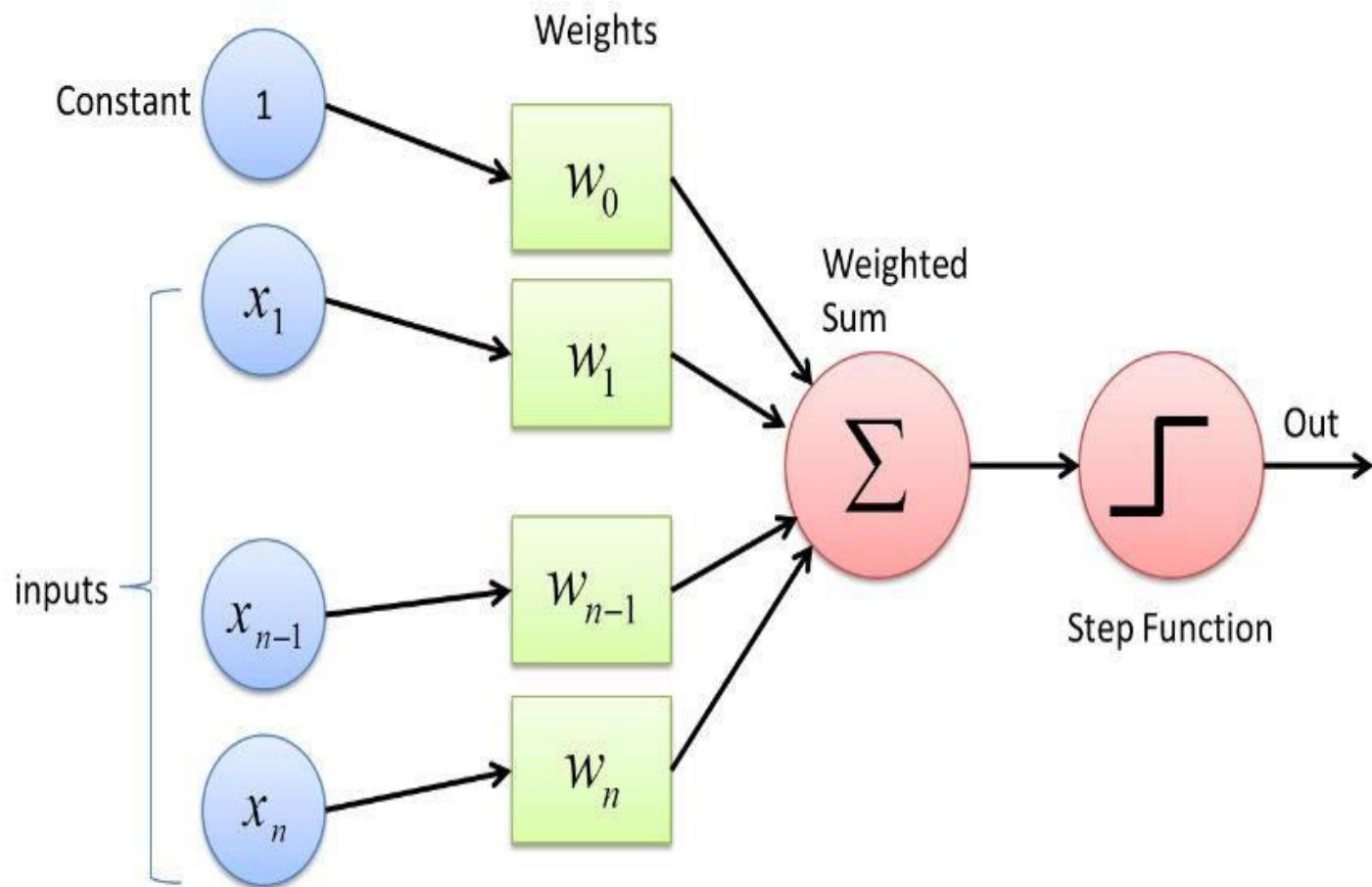
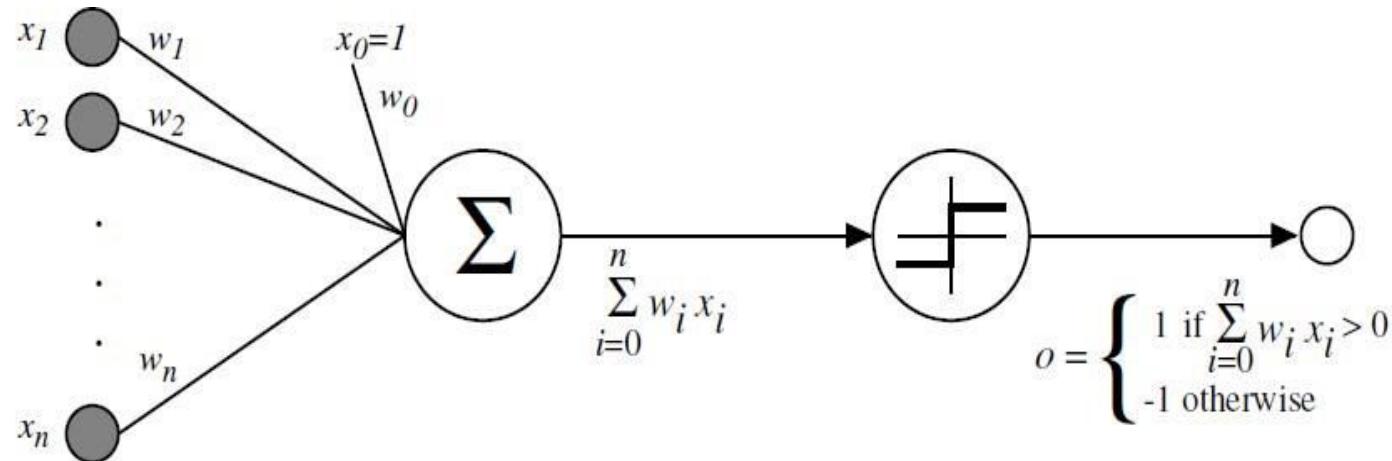


Fig : Perceptron

PERCEPTRONS

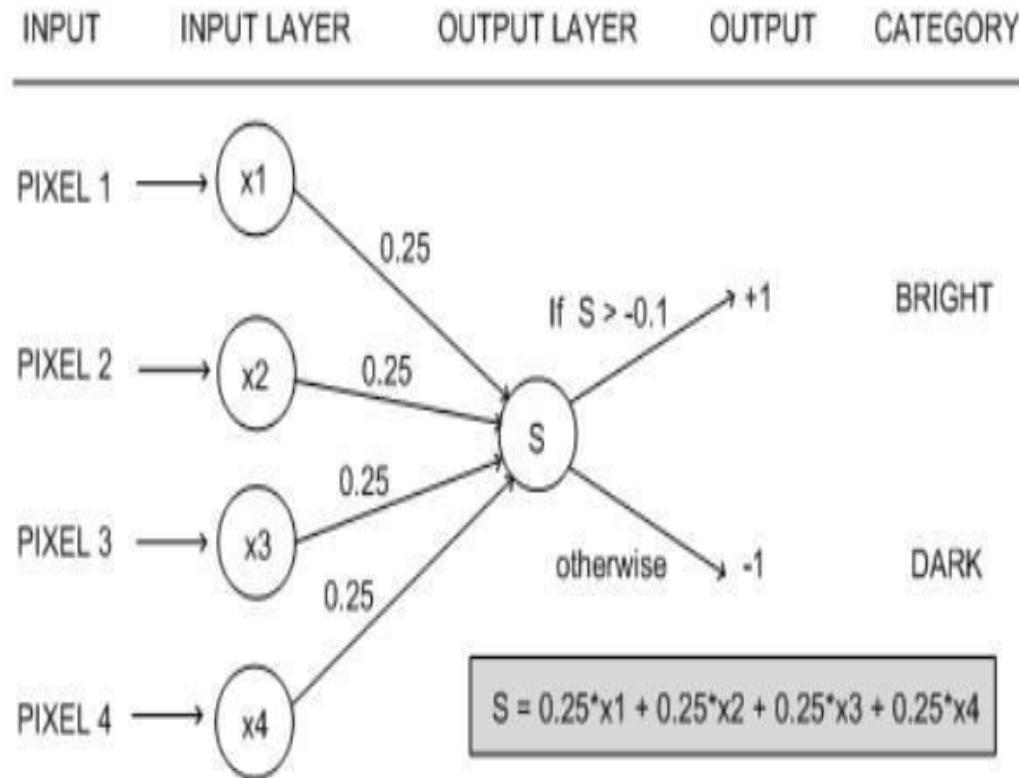


$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

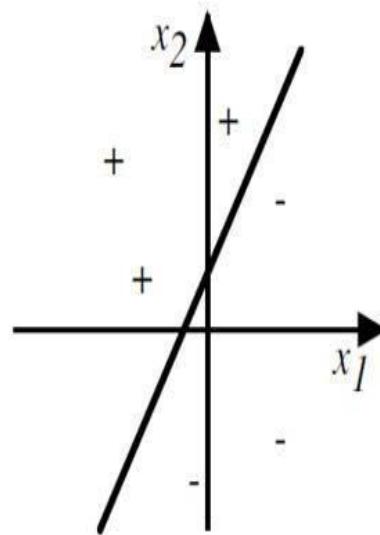
Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

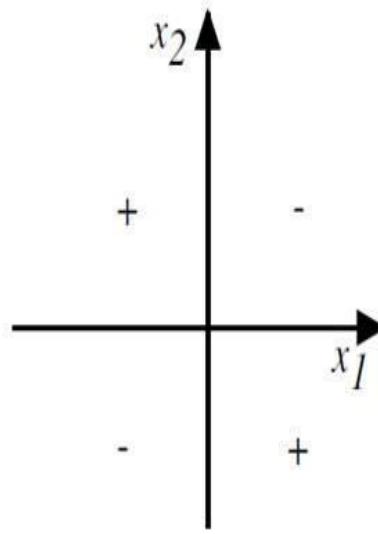
PERCEPTRONS



PERCEPTRONS



(a)



(b)

Figure : The decision surface represented by a two-input perceptron.

(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. (b) A set of training examples that is not linearly separable.

x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

*The perceptron can be viewed as representing a hyperplane decision surface in the n-dimensional space of instances.

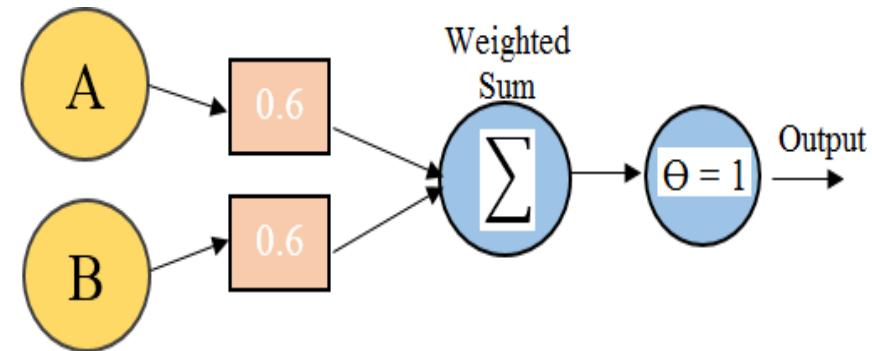
*The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side

Representational Power of
Perceptrons

PERCEPTRONS

A single perceptron can be used to represent many Boolean functions AND function

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1



- If $A=0 \& B=0 \rightarrow 0*0.6 + 0*0.6 = 0$.
This is not greater than the threshold of 1, so the output=0.
- If $A=0 \& B=1 \rightarrow 0*0.6 + 1*0.6 = 0.6$.
This is not greater than the threshold, so the output=0.
- If $A=1 \& B=0 \rightarrow 1*0.6 + 0*0.6 = 0.6$.
This is not greater than the threshold, so the output=0.
- If $A=1 \& B=1 \rightarrow 1*0.6 + 1*0.6 = 1.2$.
This exceeds the threshold, so the output=1.

The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule.

The Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = \eta(t - o)x_i$$

Here,

t is the target output for the current training example

o is the output generated by the perceptron

η is a positive constant called the *learning rate*

- The role of the *learning rate* is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases

Drawback: The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.

MULTILAYER NEURAL NETWORKS

Multilayer networks learned by the **BACKPROPAGATION** algorithm are capable of expressing a rich variety of nonlinear decision surfaces

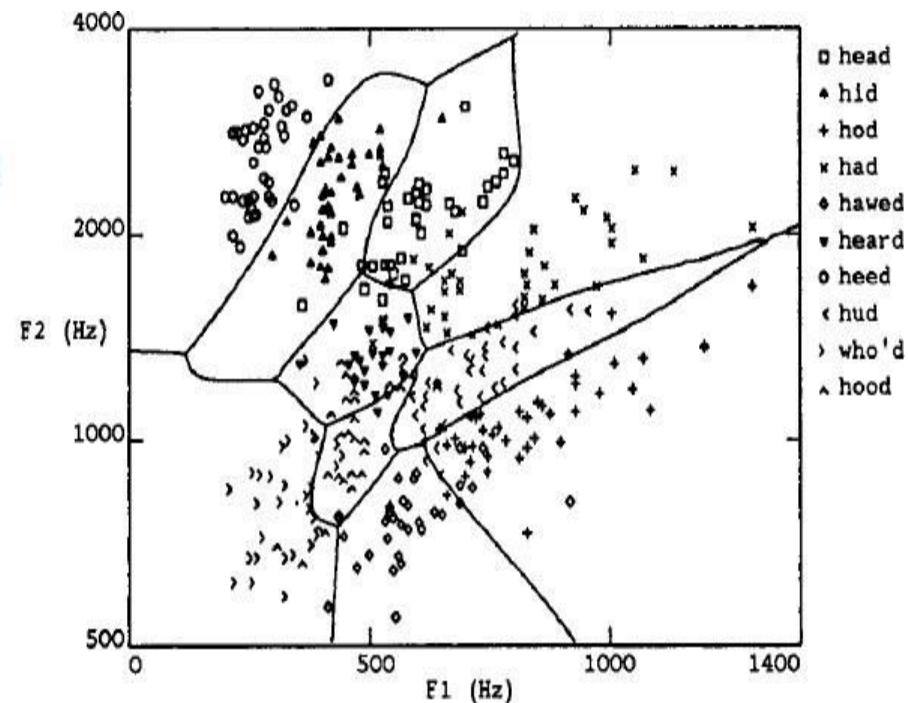
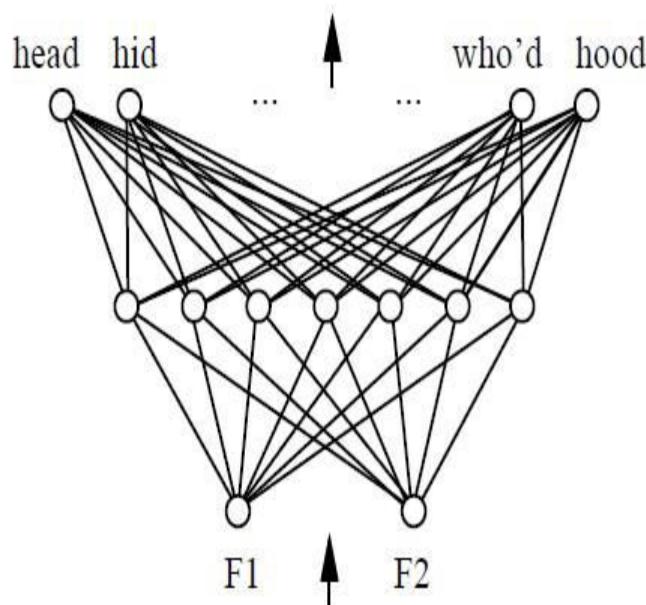


Figure: Decision regions of a multilayer feedforward network.

MULTILAYER NEURAL NETWORKS

- Decision regions of a multilayer feedforward network. The network shown here was trained to recognize 1 of **10** vowel sounds occurring in the context "h_d" (e.g., "had," "hid").
- The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest.
- The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network.

A Differentiable Threshold Unit

- Sigmoid unit-a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.
- The sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result. In the case of the sigmoid unit, however, the threshold output is a continuous function of its input.
- More precisely, the sigmoid unit computes its output \mathbf{o} as

$$\mathbf{o} = \sigma(\vec{w} \cdot \vec{x})$$

Where,

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

σ is the sigmoid
function

A Differentiable Threshold Unit

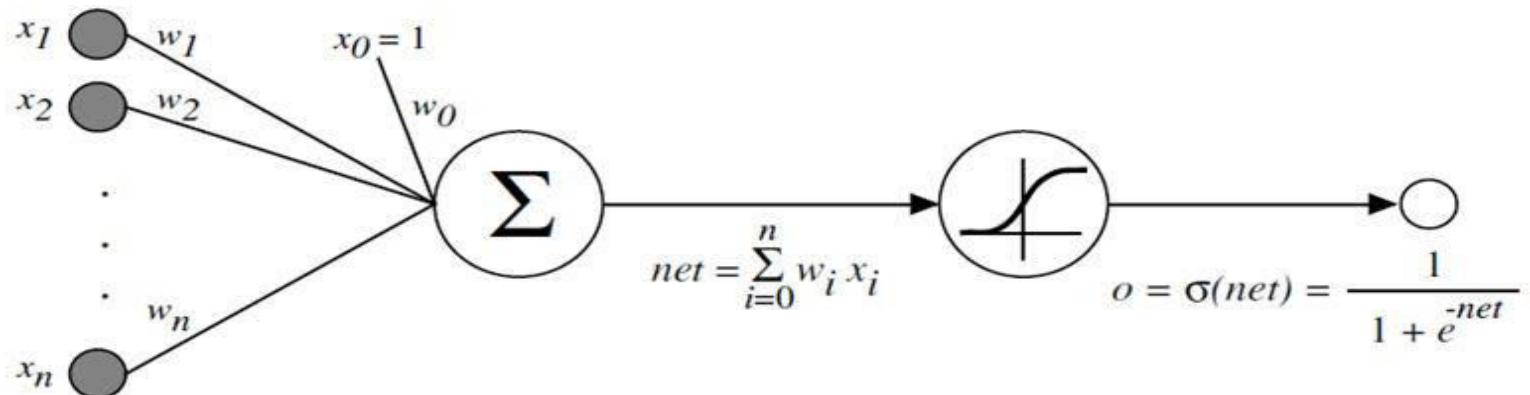


Figure: A Sigmoid Threshold Unit

$\sigma(y)$ is the sigmoid function

$$\frac{1}{1 + e^{-y}}$$

Nice property: $\frac{d\sigma(y)}{dy} = \sigma(y)(1 - \sigma(y))$

The BACKPROPAGATION Algorithm

- The BACKPROPAGATION Algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.
 - In BACKPROPAGATION algorithm, we consider networks with multiple output units rather than single units as before, so we redefine E to sum the errors over all of the network output units.

where,

- **outputs** - is the set of output units in the network
 - t_{kd} and O_{kd} - the target and output values associated with the k^{th} output unit
 - d - training example

The BACKPROPAGATION Algorithm

|BACKPROPAGATION (*training_example*, η , n_{in} , n_{out} , n_{hidden})

Each training example is a pair of the form (\vec{x}, \vec{t}) , where (\vec{x}) is the vector of network input values, (\vec{t}) and is the vector of target network output values.

η is the learning rate (e.g., .05). n_b is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji} and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
 - For each (\vec{x}, \vec{t}) , in training examples, Do

Propagate the input forward through the network:

1. Input the instance \vec{x} , to the network and compute the output o_u of every unit u in the network.

The BACKPROPAGATION Algorithm

Propagate the errors backward through the network:

2. For each network output unit k , calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h , calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Derivation of the BACKPROPAGATION Rule

- Deriving the stochastic gradient descent rule: Stochastic gradient descent involves iterating through the training examples one at a time, for each training example d descending the gradient of the error E_d with respect to this single example
- For each training example d every weight w_{ji} is updated by adding to it Δw_{ji}

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \quad \dots\dots\dots \text{equ. (1)}$$

where, E_d is the error on training example d , summed over all output units in the network

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in \text{output}} (t_k - o_k)^2$$

Here **outputs** is the set of output units in the network, t_k is the target value of unit k for training example d , and o_k is the output of unit k given training example d .

Derivation of the BACKPROPAGATION Rule

The derivation of the stochastic gradient descent rule is conceptually straightforward, but requires keeping track of a number of subscripts and variables .

x_{ji} = the i^{th} input to unit j

w_{ji} = the weight associated with the i^{th} input to unit j

$\text{net}_j = \sum_i w_{ji}x_{ji}$ (the weighted sum of inputs for unit j)

o_j = the output computed by unit j

t_j = the target output for unit j

σ = the sigmoid function

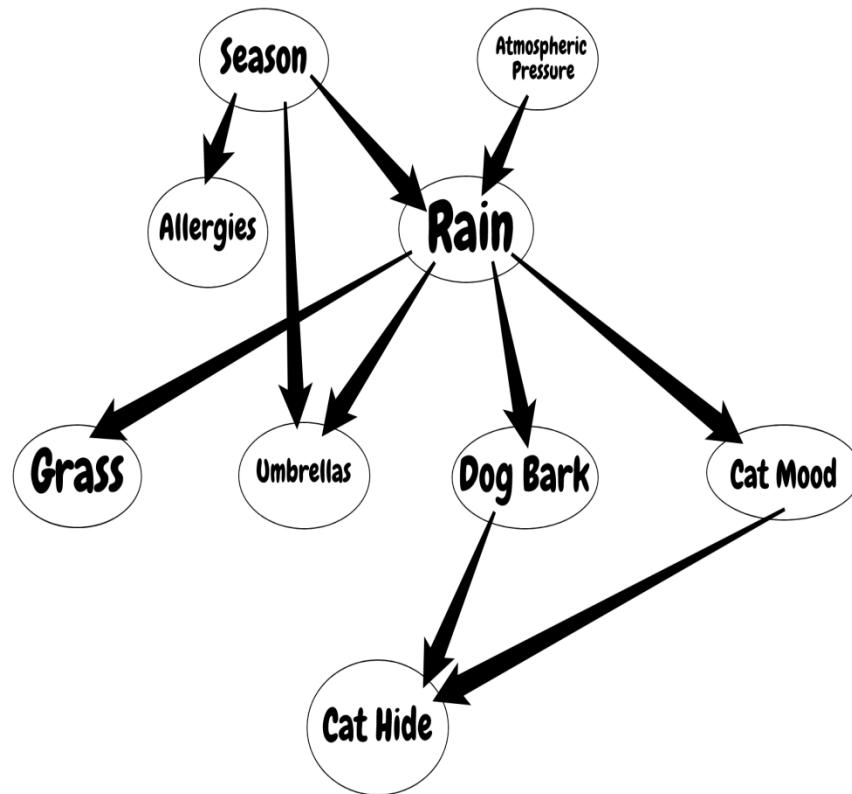
outputs = the set of units in the final layer of the network

Downstream(j) = the set of units whose immediate inputs include the output of unit j

Bayesian Methods for Learning Belief Networks.

Bayesian Belief Network or Bayesian Network or Belief Network is a Probabilistic Graphical Model (PGM) that represents conditional dependencies between variables in a directed acyclic graph (DAG).

An Example Bayesian Belief Network Representation



Belief Networks.

- Bayesian Networks are applied in many fields. For example, disease diagnosis, optimized web search, spam filtering, gene regulatory networks, etc. And this list can be extended.
- The main objective of these networks is trying to understand the structure of causality relations. To clarify this, let's consider a disease diagnosis problem.
- With given symptoms and their resulting disease, we construct our Belief Network and when a new patient comes, we can infer which disease or diseases may have the new patient by providing probabilities for each disease.
- Similarly, these causality relations can be constructed for other problems and inference techniques can be applied to interesting results.

Mathematical Definition of Bayesian Networks

The probabilities are calculated in the belief networks by the following formula

$$P(X_1, \dots, X_N) = \prod_{i=1}^N P(X_i | Parents(X_i))$$

- This formula, to be able to calculate the joint distribution we need to have conditional probabilities indicated by the network.
- But further that if we have the joint distribution, then we can start to ask interesting questions.
- For example, in the first example, we ask for the probability of “RAIN” if “SEASON” is “WINTER” and “DOG BARK” is “TRUE”.

Examples on Bayesian Networks

Bayesian network is a model of some aspect of the world. The network has between the events it models.

Bayesian networks ----- two general purposes:

1. Making future predictions
2. Explaining observations

According to Previous example Each of these nodes has possible states.

For example:

Season: Spring / Summer / Fall / Winter

Grass: Dry / Wet

Cat mood: Sleepy / Excite

Recurrent Neural Networks

Recurrent Neural Network is a type of artificial deep learning neural network designed to process sequential data and recognize patterns in it (that's where the term "recurrent" comes from).

The primary intention behind implementing RNN neural network is to produce an output based on input from a particular perspective.

The core concepts behind RNN are sequences and vectors. Let's look at both:

- Vector is an abstract representation of raw data that reiterates its meaning into a comprehensive form for the machine. It is a kind of text-to-machine translation of data.
- The sequence can be described as a collection of data points with some defined order (usually, it is a time-based, there can also be other specific criteria involved). An example of sequence can be time series stock market data - single point shows the current price while its sequence over a certain period shows the permutations of the cost.

Recurrent Neural Networks

- Unlike other types of neural networks that process data straight, where each element is processed independently of the others, recurrent neural networks keep in mind the relations between different segments of data, in more general terms, context.
- Given the fact that understanding of the context is critical in perception of information of any kind, this makes recurrent neural networks extremely efficient at recognizing and generating data based on patterns put into a specific context.
- In essence, RNN is the network with contextual loops that enable the persistent processing of every element of the sequence with the output building upon the previous computations, which in other words, means Recurrent Neural Network enables making sense of data.

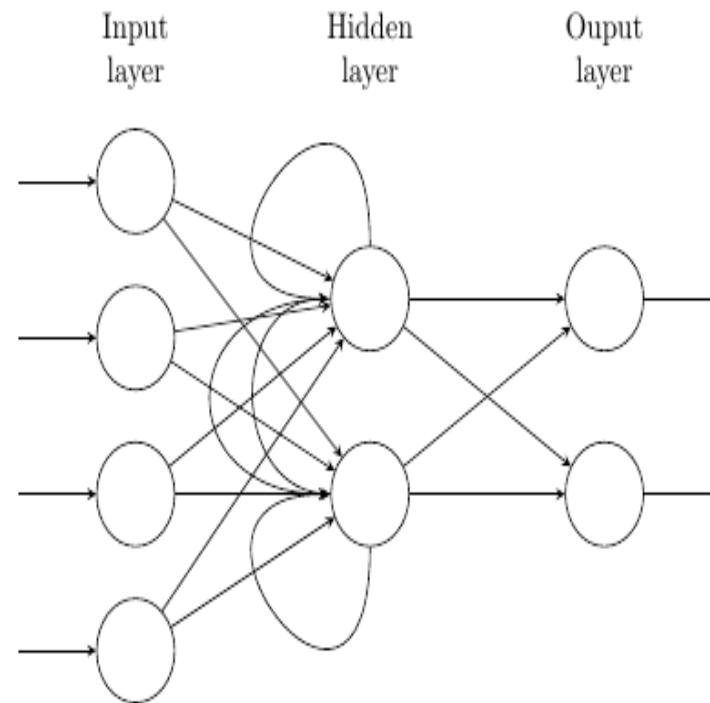
How Does Recurrent Neural Network work?

Just like traditional Artificial Neural Networks, RNN consists of nodes with three distinct layers representing different stages of the operation.

- The nodes represent the “Neurons” of the network.
- The neurons are spread over the temporal scale (i.e., sequence) separated into three layers.

How Does Recurrent Neural Network work?

The layers
are:



How Does Recurrent Neural Network work?

1. Input layer represents information to be processed;
 2. A hidden layer represents the algorithms at work;
 3. Output layer shows the result of the operation;
-
- Hidden layer contains a temporal loop that enables the algorithm not only to produce an output but to feed it back to itself.
 - This means the neurons have a feature that can be compared to short-term memory. The presence of the sequence makes them to “remember” the state (i.e., context) of the previous neuron and pass that information to themselves in the “future” to further analyze data.
 - Overall, the RNN neural network operation can be one of the three types:

How Does Recurrent Neural Network work?

1. One input to multiple outputs - as in image recognition, image described with words;
2. Several contributions to one output - as in sentiment analysis, where the text is interpreted as positive or negative;
3. Many to many - as in machine translation, where the word of the text is translated according to the context they represent as a whole;

Applications of RNN

- Speech recognition
 - Machine translation
 - Music composition
 - Handwriting recognition
 - Grammar learning
 - Sentiment Classification
- etc...

Applications of RNN

Sentiment Classification – This can be a task of simply classifying tweets into positive and negative sentiment. So here the input would be a tweet of varying lengths, while output is of a fixed type and size.



Reinforcement Learning

Agenda

Why What and How of Reinforcement learning?

Elements of Reinforcement learning

Reinforcement learning problem & example

Major differences from Supervised learning

Q&A

Why What and How of Reinforcement learning?

- Supervised Learning- Task Driven
- Unsupervised Learning- Data Driven
- Reinforcement Learning- It's type of machine learning where an agent learns to behave in an environment by performing actions and seeing results

Why What and How of Reinforcement learning?



Supervised Learning



Unsupervised Learning



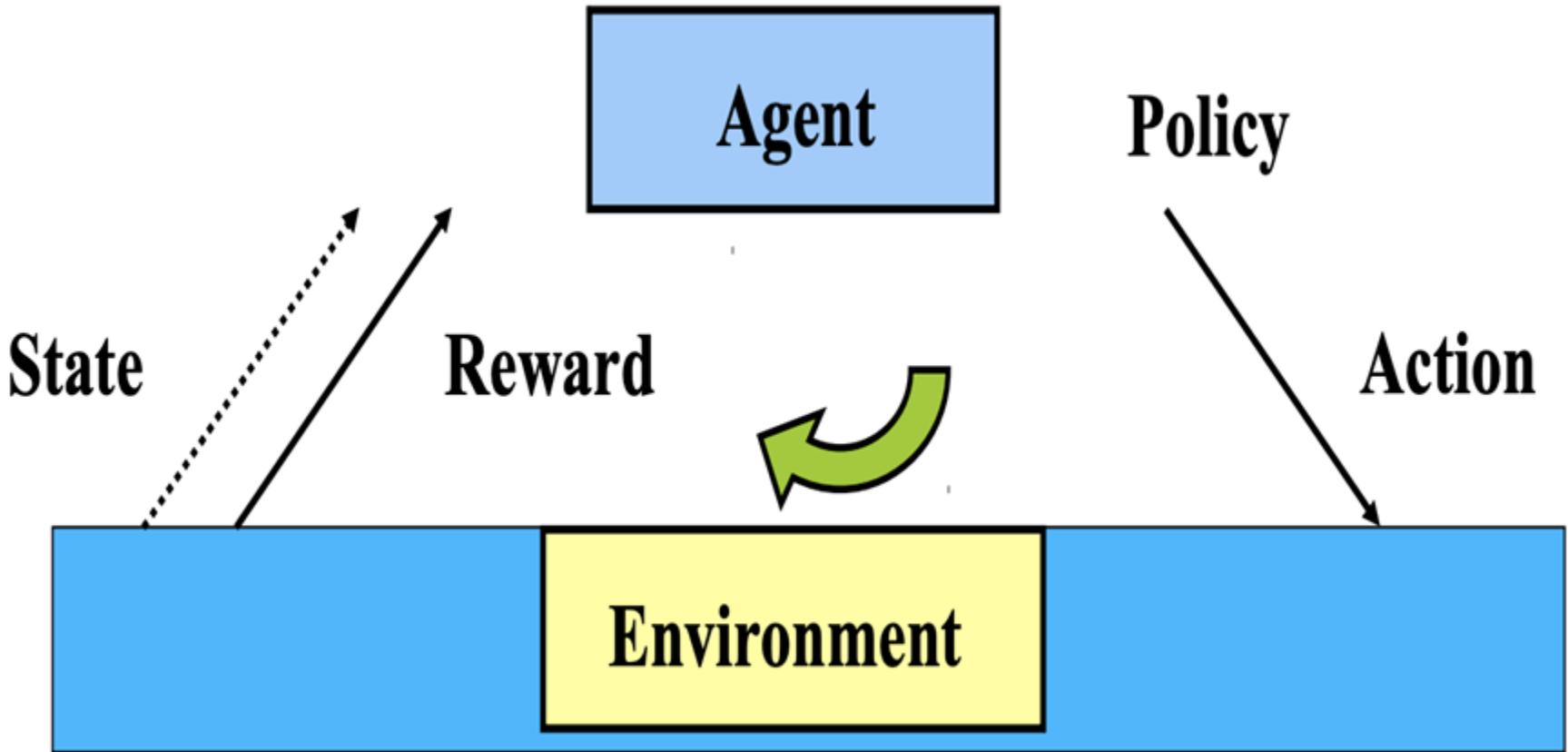
Reinforcement Learning

Why What and How of Reinforcement learning?

Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment.

- Reinforcement Learning is learning how to act in order to maximize a numerical reward.
- Reinforcement learning is not a type of neural network, nor is it an alternative to neural networks. Rather, it is an orthogonal approach for Learning Machine.

Elements of Reinforcement learning



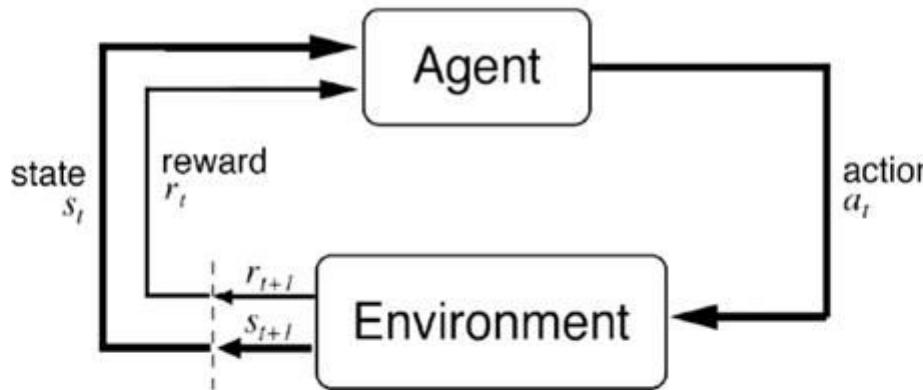
Elements of Reinforcement learning

- Agent: Intelligent programs
- Environment: External condition
- Policy:
 - Defines the agent's behavior at a given time
 - A mapping from states to actions
 - Lookup tables or simple function
- Reward function :
 - Defines the goal in an RL problem
 - Policy is altered to achieve this goal

Elements of Reinforcement learning

- Value function:
 - Reward function indicates what is good in an immediate sense while a value function specifies what is good in the long run.
 - Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- Model of the environment :
 - Predict mimic behavior of environment
 - Used for planning & if Know current state and action then predict the resultant next state and next reward.

Elements of Reinforcement learning



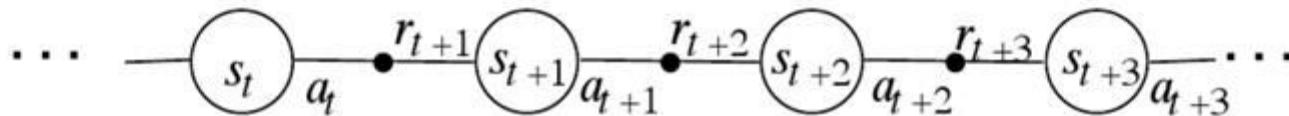
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \dots$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



Elements of Reinforcement learning

1. The agent observes an input state
2. An action is determined by a decision making function (policy)
3. The action is performed
4. The agent receives a scalar reward or reinforcement from the environment
5. Information about the reward given for that state /action pair is recorded

Reinforcement Learning algorithms

There are three approaches to implement a Reinforcement Learning algorithm.

- Value-Based:
 - In a value-based Reinforcement Learning method, you should try to maximize a value function $V(s)$. In this method, the agent is expecting a long-term return of the current states under policy π .
- Policy-based:
 - In a policy-based RL method, you try to come up with such a policy that the action performed in every state helps you to gain maximum reward in the future.

Two types of policy-based methods are there:

- Deterministic: For any state, the same action is produced by the policy π .
- Stochastic: Every action has a certain probability, which is determined by the following equation.
Stochastic Policy

$$n(a|s) = P[A, = a | S, = s]$$

- Model-Based:
 - In this Reinforcement Learning method, you need to create a virtual model for each environment. The agent learns to perform in that specific environment.

Types of RL techniques

There are mainly two types

- **Passive Learning**- The agent's policy is fixed which means that it's told what to do. Always the goal of passive agent is to execute a fixed set of policy(sequential actions) and evaluate.
- **Active Learning**- The agent needs to decide what it needs to do as there is no fixed policy to act on. So the goal of active agent will be to always act and learn an optimal policy.

Types of RL techniques

- Passive Learning
 - Direct Utility Estimation
 - Adaptive Dynamic Programming (ADP)
 - Temporal Difference Learning (TD)
- Active Learning
 - ADP with exploration function
 - Q Learning

Direct Utility Estimation - Here the agent executes sequences of trials. Each trial gives a sample value and agent estimates the utility based on sample values.

Pros & Cons - Slow to converge

Types of RL techniques

Adaptive Dynamic Programming (ADP) - It's smarter than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^\pi(s')$$

Where $R(s)$ = reward for being in state s , $P(s'|s, \pi(s))$ = transition model, γ = discount factor and $U^\pi(s)$ = utility of being in state s' .

Pros & Cons - converges fast but costly to compute

Types of RL techniques

Temporal Difference Learning (TD) - ADP is a model based approach and requires the transition model of the environment where in TD is model free.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^\pi(s')$$

TD doesn't require the agent to learn the transition model. The update occurs between successive states and agent only updates states that are directly affected.

Pros & Cons - Slower to converge, simpler to compute

Types of RL techniques

ADP with exploration function - As the goal of an active agent is to learn an optimal policy, the agent needs to learn the expected utility of each state and update its policy.

Can be done using a passive ADP agent and then using value or policy iteration it can learn optimal actions. But this approach results into a greedy agent.

Hence, we use an approach that gives higher weights to unexplored actions and lower weights to actions with lower utilities.

Types of RL techniques

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^\pi(s')$$

Where $f(u, n)$ is the exploration function that increases with expected value u and decreases with number of tries a

$$f(u, n) = \begin{cases} R^+, & \text{if } n < N_e \\ u, & \text{otherwise} \end{cases}$$

R^+ is an optimistic reward and N_e is the number of times we want an agent to be forced to pick an action in every state. The exploration function *converts a passive agent into an active one*

Types of RL techniques

Q Learning - It's TD learning method which does not require the agent to learn the transitional model, instead learns Q-value functions $Q(s, a)$.

$$U(s) = \max_a Q(s, a)$$

Q-values can be updated using the following equation,

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

Next action can be selected using the following policy,

$$a_{next} = \arg \max_{a'} f(Q(s', a'), N(s', a'))$$

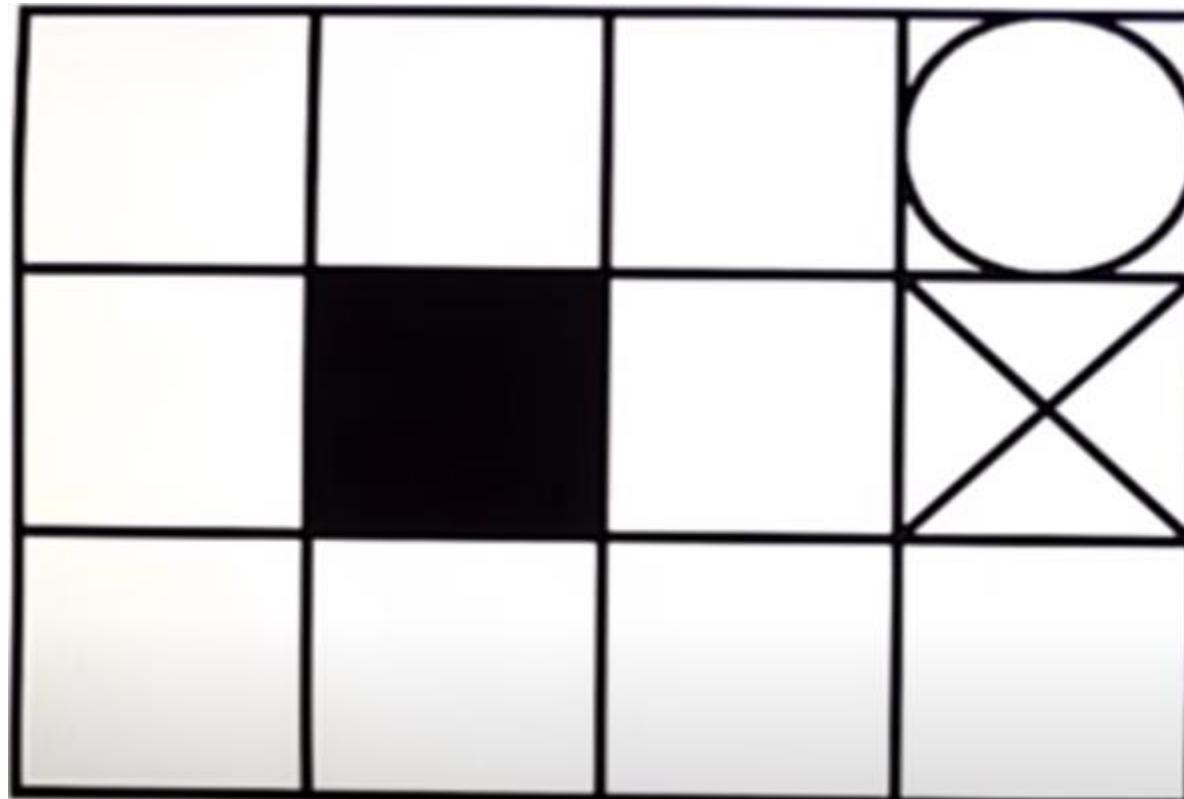
Comparision b/w Active & Passive

	Fixed Policy (Active)	Policy not fixed (Passive)
Model-free (real world)	Temporal Difference Learning (TD)	Q-learning
Model-based (simulation)	Adaptive Dynamic Programming(ADP)	ADP with proper exploration function

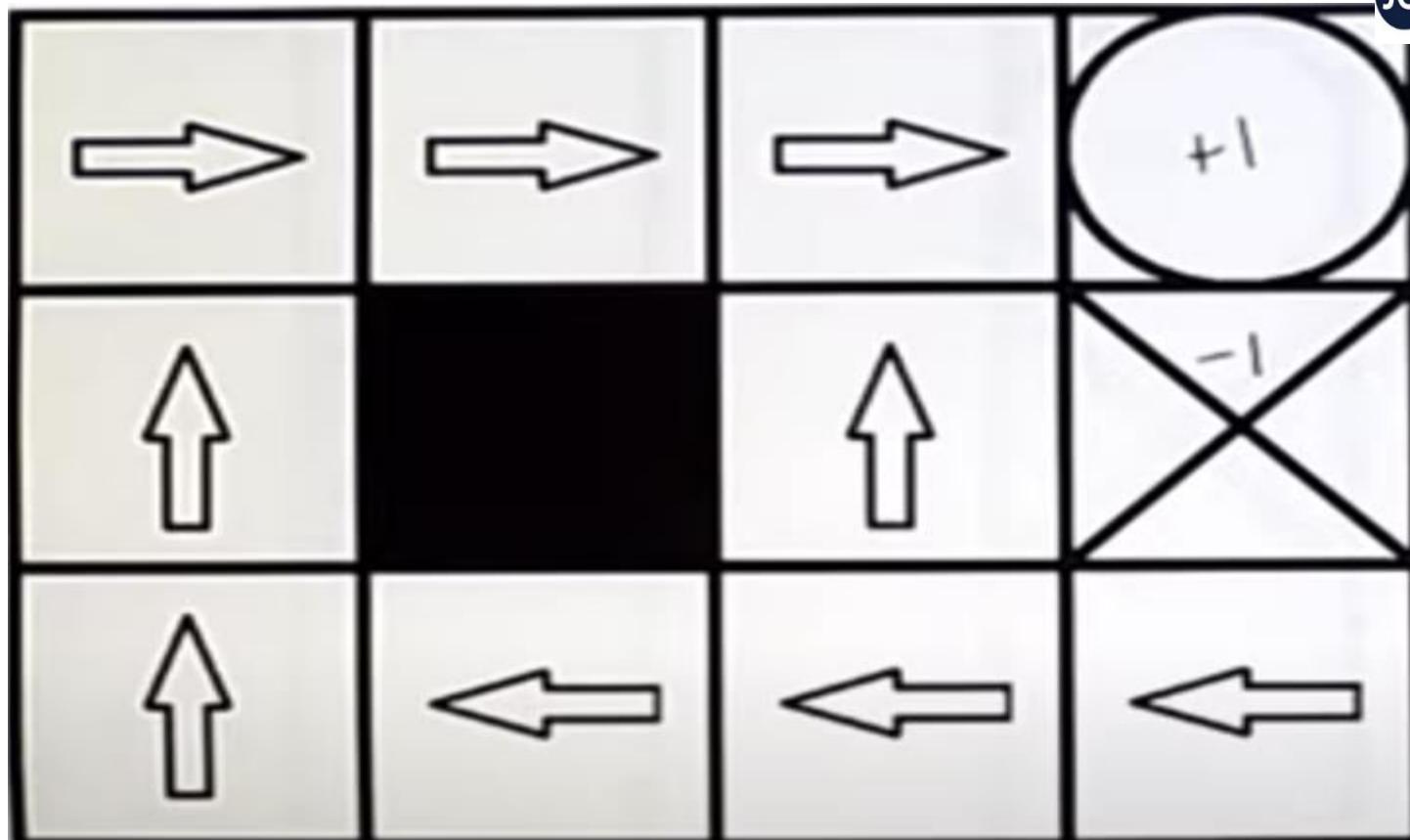
Reinforcement learning problem & example

Passive Learning in known environment:

Grid World Problem with reward of +1 and penalty of -1 and moment cost 0.04

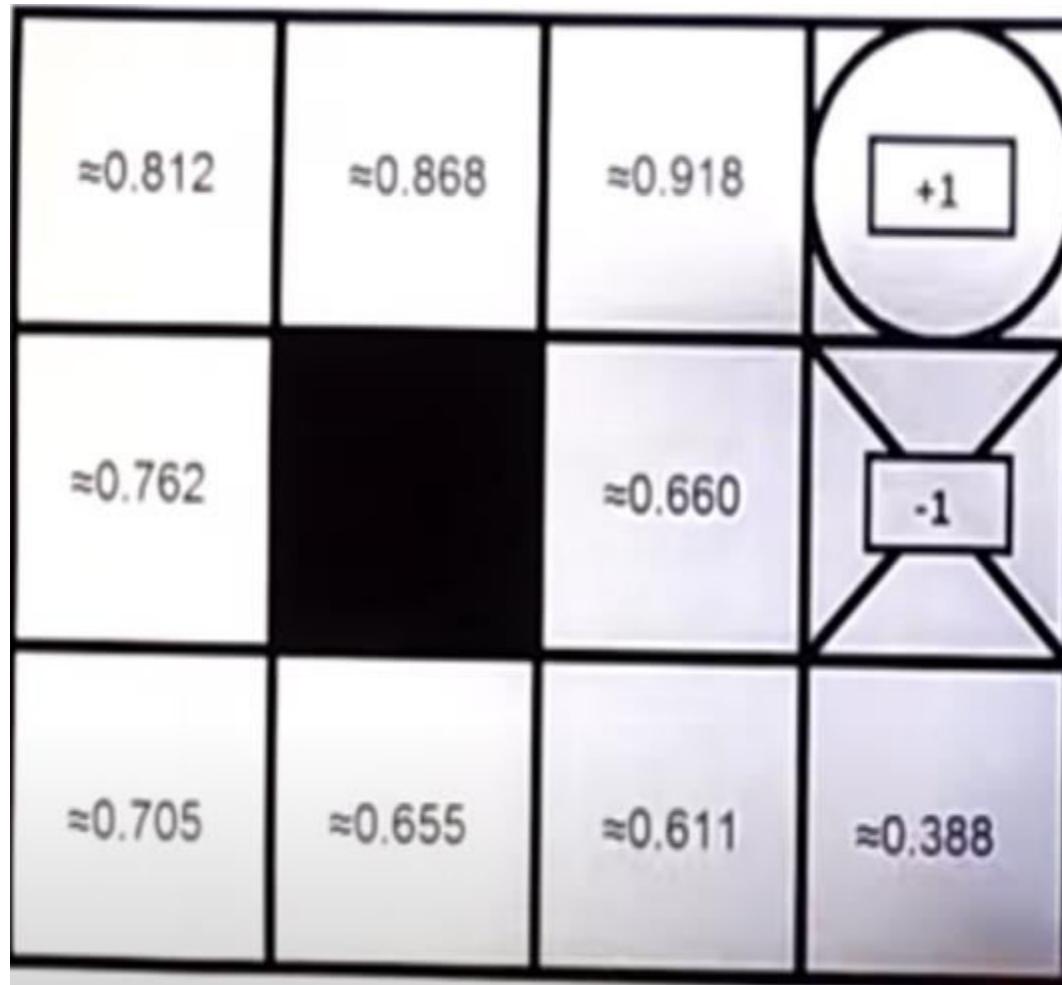


Reinforcement learning problem & example



The optimal path solution for the grid world path looks like this. And we can calculate the traversal costs from any point in the grid to the destination like below

Reinforcement learning problem & example

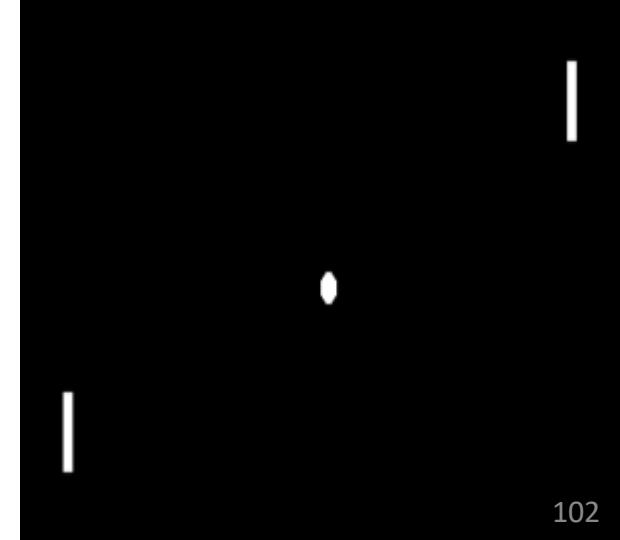


Reinforcement learning problem & example

- We will see simple pong game as an example of RL
- Pong is a special case of a

Markov Decision Process (MDP)

- Rules here are simple
 - You play as one of the paddles and you have to bounce
 - the ball past the other player



Reinforcement learning problem & example

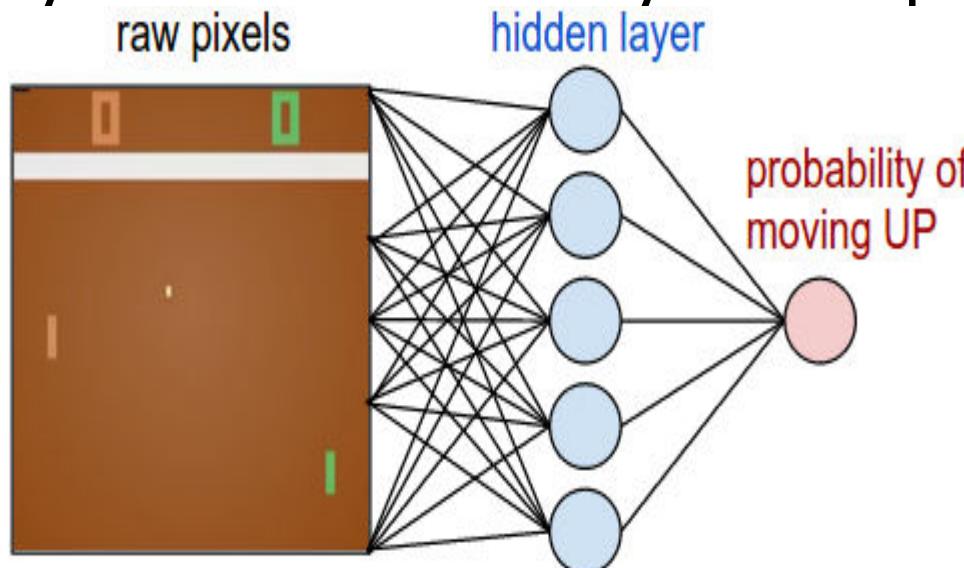
- On the low level the game works as follows:
 - We receive an image frame (a 210x160x3 byte array (0 to 255 pixel values))
 - We get to decide if we want to move the paddle UP or DOWN (i.e. a binary choice)
 - After every single choice the game simulator executes the action and gives us a reward
 - a +1 reward if the ball went past the opponent
 - a -1 reward if we missed the ball, or 0 otherwise
 - our goal is to move the paddle so that we get lots of reward.

Reinforcement learning problem & example

The example is based on Active learning in known environment

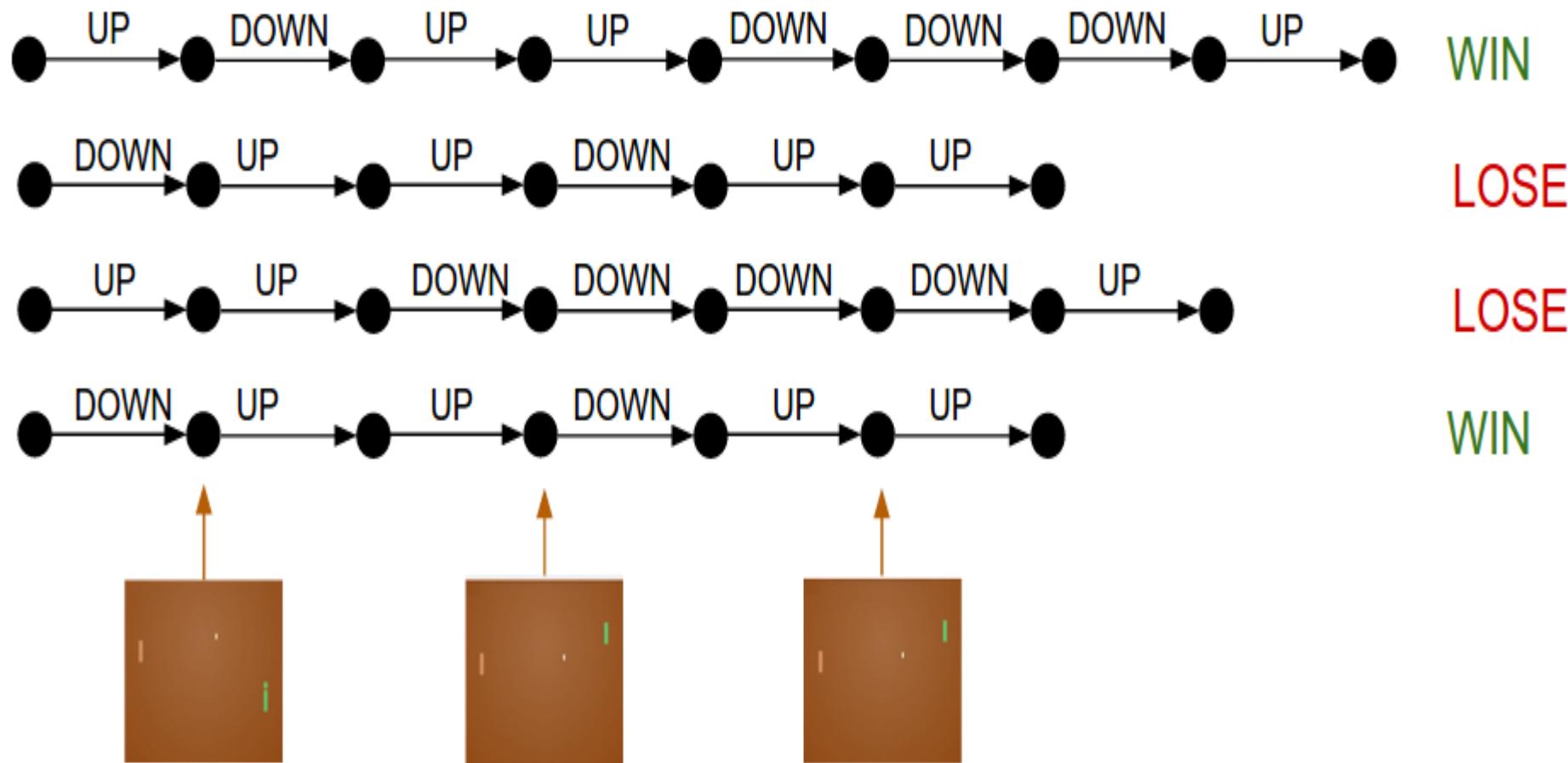
- The environment is fixed and the rules are set.
- The game rules are fixed and the execution is bound by choices

We use policy network to analyse the probability of decision.



Reinforcement learning problem & example

The policy gradient yields a reward based on actions as shown below



Reinforcement learning problem & example



Multi Armed Bandit problem Demo

Reinforcement learning problem & example

Generalization in Reinforcement Learning: It's quite difficult for even the trained algorithms.

- Although trained agents can solve complex tasks, they struggle to transfer their experience to new environments.
- RL agents tend to overfit- they latch onto the specifics of their environment rather than learn generalizable skills.
- Agents are still benchmarked by evaluating on the environments they trained on.

Salient Features of Reinforcement Learning

- Set of problems rather than a set of techniques without specifying how the task is to be achieved.
- “RL as a tool” point of view:
 - RL is training by rewards and punishments.
 - Train tool for the computer learning.
- The learning agent’s point of view:
 - RL is learning from trial and error with the world.
 - Eg. how much reward I much get if I get this.

Salient Features of Reinforcement Learning

- Reinforcement Learning uses Evaluative Feedback.
- Purely Evaluative Feedback
 - Indicates how good the action taken is.
 - Not tell if it is the best or the worst action possible.
 - Basis of methods for function optimization.
- Purely Instructive Feedback
 - Indicates the correct action to take, independently of the action actually taken.
 - Eg: Supervised Learning
- Associative and Non Associative:

Salient Features of Reinforcement Learning

- Associative :
 - Situation Dependent
 - Mapping from situation to the actions that are best in that situation.
- Non Associative:
 - Situation independent
 - No need for associating different action with different situations.
 - Learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is non stationary.
- Exploration and exploitation
 - **Greedy action:** Action chosen with greatest estimated value.
 - Greedy action: a case of Exploitation.
 - **Non greedy action:** a case of Exploration, as it enables us to improve estimate the non-greedy actions value.

Major differences from Supervised learning

REINFORCEMENT LEARNING

SUPERVISED LEARNING

Reinforcement learning is all about making decisions sequentially. In simple words we can say that the output depends on the state of the current input and the next input depends on the output of the previous input

In Supervised learning the decision is made on the initial input or the input given at the start

In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions

Supervised learning the decisions are independent of each other so labels are given to each decision.

Example: Chess game

Example: Object recognition

Video Reference links

S. NO	Description	Video link
1	This video gives the knowledge of Candidate Elimination Algorithm	https://www.youtube.com/watch?v=cW03t3aZkmE
2	This video describes the Decision Tree method with an example Solved	https://www.youtube.com/watch?v=UdTKxGQvYdc
3	Neural Networks 1 – Perceptrons	https://www.youtube.com/watch?v=aiDv1NPdXvU&pbjreload=10
4	Neural Networks 2 - Multi-Layer Perceptrons	https://www.youtube.com/watch?v=s8pDf2Pt9sc
5	Perceptron (single layer) learning with solved Example	https://www.youtube.com/watch?v=x3joYu5VI38
6	Single layer Perceptron neural network	https://www.youtube.com/watch?v=4QuiYBNHGYo

Video Reference links

S. NO	Description	Video link
7	Back Propagation in Neural Network with an Example	https://www.youtube.com/watch?v=GJXKOrqZauk
8	Neural network	https://wwwcomputing.dcu.ie/~humphrys/Notes/Neural/single_neural.html
9	Learning with Bayesian Network with solved examples.	https://www.youtube.com/watch?v=88Rg58t9114
10	Bayes' theorem explained with examples and implications for life.	https://www.youtube.com/watch?v=R13BD8qKeTg
11	Reinforcement Learning: Introduction	https://www.youtube.com/watch?v=8iuO4zr-Hxc
12	Reinforcement Learning 1 - Expected Values	https://www.youtube.com/watch?v=3T5eCou2erg
13	Reinforcement Learning 2 - Grid World	https://www.youtube.com/watch?v=bHeeaXgqVig
14	Reinforcement Learning 3 - Q Learning	https://www.youtube.com/watch?v=1XRahNzA5bE

Thank you