K. R. Chowdhary

Fundamentals of Artificial Intelligence

Springer

K. R. Chowdhary
Department of Computer Science
and Engineering
Jodhpur Institute of Engineering
and Technology
Jodhpur, Rajasthan, India

ISBN 978-81-322-3970-3 ISBN 978-81-322-3972-7 (eBook)
https://doi.org/10.1007/978-81-322-3972-7

This Springer imprint is published by the registered company Springer Nature India Private Limited
The registered company address is: 7th Floor, Vijaya Building, 17 Barakhamba Road, New Delhi 110 001, India

Contents

About the Author

Prof. K. R. Chowdhary is currently a Professor of Computer Science at JGI, Jodhpur. He is the former Professor and Head, Department of Computer Science & Engineering, M.B.M. Engineering College, Jodhpur, India. He completed his Ph.D. from the same university in 2004. Prof. Chowdhary has over 35 years of teaching and research experience. He had also taught at IIT Jodhpur (2010–2017), and served as the Director at JIET Group of Institutions, Jodhpur from 2014 to 2019. He was honored with senior membership award of Association for Computing Machinery in 2008 and Eminent Computer Engineer's Award from the Institute of Engineers, India in 2011. He has authored several papers published in national and international journals and conference proceedings. His areas of specialization include: Discrete Mathematical Structures, Theory of Computation, AI, Machine Learning, Natural Language and Speech Processing.

Acronyms

ABL	Argument-based Learning
ABOX	Assertion Box
ACL	Agent Communication Language
ADJ	Adjective
ADV	Adverb
AI	Artificial Intelligence
ANNs	Artificial Neural networks
AQE	Automatic Query Expansion
ASR	Automatic Speech Recognition
ATR	Automatic Target Recognition
AUX	Auxiliary verb
BFS	Breadth-first Search
BOW	Bag-of-words
CBR	Case-based Reasoning
CD	Conceptual Dependency
CF	Cluster Feature
CFG	Context-free Grammar
CG	Conceptual Graph
CGT	Combinatorial Game Theory
CLIR	Cross-language IR
CNF	Conjunctive Normal Forms
CSP	Constraint Satisfaction Problem
CYC	enCYClopedia ontology
DAG	Directed Acyclic Graph
DET	Determiner
DFID	Depth-First Iterative Deepening
DFS	Depth-first Search
DL	Description Logic
DNF	Disjunctive Normal Forms
DST	Dempster-Shafer Theory

FIR	Fuzzy Information Retrieval
FOL	First Order Logic
FOPL	First Order Predicate Logic
FST	Finite State Transducer
GAs	Genetic Algorithms
GB	Gödel-Bernays
HMM	Hidden Markov Model
IBL	Instance-based Learning
IE	Information Extraction
IR	Information Retrieval
KB	Knowledge Base
KIF	Knowledge Interchange Format
k-NN	k-Nearest Neighbor
KR	Knowledge Representation
LM	Language Modeling
LOOM	Lexical OWL Ontology Matcher
MAS	Multi-agent System
MDP	Markov Decision Process
NLP	Natural Language Processing
NLTK	Natural Language Tool-Kit
NN	Nearest Neighbor
NP	Nondeterministic Polynomial
OCR	Optical Character Recognition
OWL	Web Ontology Language
PCA	Principle Component Analysis
PD	Prisoner's Dilemma
PSSH	Physical Symbol System Hypothesis
QA	Question Answering
RBS	Rule-based System
RGB	Red Green Blue
RL	Reinforcement Learning
SRC	Search Results Clustering
STRIPS	STanford Research Institute Problem Solver
SVM	Support Vector Machine
SVN	Support Vector Network
TBOX	Terminology Box
TREC	Text REtrieval Conference Series
TSP	Traveling Salesman Problem
WAM	Warren Abstract Machine
WSD	Word Sense Disambiguation
XCON	Expert Configurator
ZF	Zermilo-Fraenkel

Chapter 1
Introducing Artificial Intelligence

Abstract The field of Artificial Intelligence (AI) has interfaces with almost every discipline of engineering, and beyond, and all those can be benefited by the use of AI. This chapter presents the introduction to AI, its roots, sub-domains, Turing test to judge if the given program is intelligent, what are the goals of AI for Engineers and Scientists, what are the basic requirements for AI, symbol system, what are the requirements for knowledge representation, and concludes with a chapter summary, and an exhaustive list of exercises. In addition raises many questions to ponder over, like, consciousness, and whether it is possible to create artificial consciousness.

Keywords Artificial intelligence · Turing test · Imitation game · Philosophy · Logic · Computation · Cognitive science · Evolution · Artificial consciousness · Speech processing · Expert systems · Physical symbol system hypothesis · Formal system · Knowledge representation

1.1 Introduction

The Artificial Intelligence (AI) is a branch of Computer Science, which is mainly concerned with *automation of Intelligent behavior*. This behavior we may consider from all domains—the human, animal world, and vegetation. A compact definition of Intelligence is:

$$Intelligence = Perceive + Analyze + React.$$

The following are often quoted definitions, all expressing this notion of intelligence but with different emphasis in each case:

- "The capacity to learn or to profit by experience."
- "Ability to adapt oneself adequately to relatively new situations in life."
- "A person possesses intelligence insofar as he has learned, or can learn, to adjust himself to his environment."
- "The ability of an organism to solve new problems."

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_1

- "A global concept that involves an individual's ability to act purposefully, think rationally, and deal effectively with the environment."
- "Intelligence is a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience."

The foundation materials of AI comprises—data structures, knowledge representation techniques, algorithms to apply the knowledge and language, and the programming techniques to implement all these.

To get an idea of *Intelligence* it requires answering these and many similar questions:

- Is intelligence due to a *single faculty* or it is a name for a collection of distinct unrelated faculties?
- Is it a priori existence or it can be learned? What does exactly happen when we learn some thing, that is, in terms of information and storage structures.
- What is truly the process of *creativity* and *intuition* in human? These again, in terms of knowledge and its structures.
- Does the intelligence require an internal mechanism, or it can be concluded from the behavior observed?
- What is the mechanism for representing the knowledge in the living cells?
- Are the machines self aware like humans? What are the basic requirements for creating the facility of self-awareness in machines?
- Is it that computer intelligence can be defined only when we know the intelligence in reference to human beings?
- Would it ever be possible to achieve the intelligence in computers? Or, is it true that achievement of intelligence is possible only when or is it that an intelligent entity requires the richness of *sensation* and *experience* which might be found only in a biological existence?

Partly, the aim of Artificial Intelligence (AI) is to find the answer to these questions, through the tools provided by AI. These tools are because, the AI offers the medium, as well as the test-bed for theories of intelligence, which can be expressed in the form of computer programs, and can be tested as well as verified by running these programs on computers.

Unlike Physics and Chemistry, AI is still a premature field, hence, its structure, objectives, and procedures are less clearly defined, and not clear like those in physics and chemistry. The AI has been more concerned about expanding limits of computers, apart from defining itself.

Learning Outcomes of this Chapter[1]:

1. Defining AI. [Familiarity]
2. Describe Turing test thought experiment. [Familiarity]

[1] https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf (pp. 121–129).

3. Differentiate between the concepts of optimal reasoning/behavior and humanlike reasoning/behavior. [Familiarity]
4. Sub-fields of AI. [Familiarity]
5. Determine the characteristics of a given problem that an intelligent system (i.e., AI-based system) must solve. [Assessment]

1.2 The Turing Test

In 1950, in an article "Computing Machinery and Intelligence," *Alan M. Turing* proposed an *empirical test* for machine intelligence, now called *Turing Test* (see Fig. 1.1). It is designed to measure the performance of an intelligent machine against humans, for its intelligent behavior. Turing called it *imitation game*, where machine and human counter-part are put in different rooms, separate from a third person, called *interrogator*. The interrogator is not able to see or speak directly to any of the other two, and does not know which entity is a machine, and communicates to these two solely by textual devices like a dumb terminal [11, 12].

The interrogator is supposed to distinguish the machine from the human solely based on the answers received for the questions asked over the interface device, which is a keyboard (or teletype). Even after having asked the number of questions, if the interrogator is not able to distinguish the machine from the human, then as per the argument of Alan Turing, the machine can be considered intelligent. Interrogator may ask highly computation oriented questions to identify the machine, and other questions related to general awareness, poetry, etc., to identify the human [6].

The game (with the "player machine" omitted) is often in practice under the name of viva-voce to discover whether someone really understands something or has "learned it parrot fashion".

Many researchers argue that the Turing test is not sufficient to establish the presence of intelligence. Some of the arguments *for* and *against* the above test can be as follows:

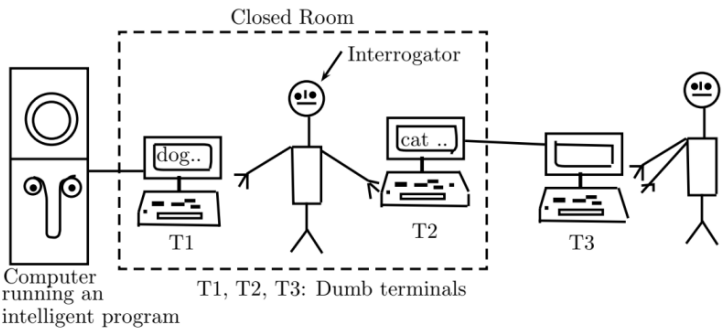


Fig. 1.1 Turing test (imitation game)

1. It takes human being as a reference for intelligent behavior, rather than debating over the true nature of intelligence: *against*.
2. The unmeasurable things are not considered, e.g., whether a computer uses *internal structures*, or for example, whether the machine is *conscious* of its actions, which are currently not answerable: *against*.
3. Eliminates any bias to human oriented interaction mechanisms, since a computer terminal is used as a communication device: *for*.
4. Biased towards only symbolic problem solving: *against*.
5. Perceptual skills or dexterity cannot be checked: *against*.
6. Unnecessarily constrains the machine intelligence to human intelligence: *against*.

Though, the number of counts of *against* are far more than *for*, there is yet no known test which is considered better than the Turing Test.

In the part success of the Turing Test, a powerful computer has deceived humans in the thinking process, where this machine modeled intelligence of a young boy, to become the first machine to pass the Turing test, conducted in June 2014. In this experiment, five machines were tested at the Royal Society in central London to check if these machines could fool the people into thinking. The machines behaved like humans, and the conversation was in the form of text. A computer program, by the name "Eugene Goostman" was developed to simulate a young boy, which came out to convince the one-third of the judges that it was human [5].

1.3 Goals of AI

AI is the area of computer science aiming for the design of intelligent computer systems, i.e., systems that have characteristics of intelligence, like, we observe in human behavior, for example, to understand language(s), and have abilities of learning, reasoning, and problem solving [9].

For many researchers, the goal of AI is to emulate human cognition, while to some researchers, it is the creation of intelligence without considering any human characteristics. To many other researchers, AI is aimed to create useful artifacts for the comforts and needs of human, without any criteria of an abstract notion of intelligence.

The above variation in aims is not necessarily a wrong, as each approach uncovers new ideas and provides a base for pursuing the research in AI. However, there is a convincing argument that due to the absence of a proper definition of AI, it is difficult to establish as what can and what cannot be done through AI.

One of the goals for studying AI is to create intelligence in machines as a general property—not necessarily based on any attribute of humans. When this is the goal, it also includes the objective of creation of artifacts of human comforts and needs, which can be the driving force of technological development. However, this goal also requires a notion of intelligence, to start with.

- *Roger Penrose* in 1995 tried to prove that human mind has non-computable capabilities.

1.4.3 Computation

In the nineteenth and twentieth-century, many scientists defined the formalism of what is *computation*, basic theory of it, and that there are things that are not computable irrespective of whatever are the computing resources and time provided.

In the year 1869, *William Jevon* constructed a *Logic Machine* capable of handling Boolean Algebra and Venn Diagrams, and could solve logical problems faster than human beings.

Alan M. Turing (1912–1954) tried to characterize exactly what are the functions that can be computed. He used, what is now called as Turing Machine. Unfortunately, it is difficult to give the notion of computation as a formal definition, however, the *Church-Turing thesis*, due to Alonzo Church and Turing, states that a Turing machine is capable of computing any computable function, which is now, accepted as a sufficient definition of computability. Turing also showed that there are some functions which no Turing machine can compute (e.g., *Halting Problem*)—these are non-computable functions.

John von Neumann (1903–1957) gave, now what is called as, von Neumann architecture—a description of a logical model of computation and computer, without any physical realization of a computer.

In 1960s, two important concepts emerged—*Intractability* (the solution time of a problem grows at least exponentially) and *Reduction* of complex problems into simpler problems.

1.4.4 Psychology and Cognitive Science

Cognitive Psychology or Cognitive Science is the study about the functioning of the mind, human behavior, and the processing of information about the human brain. An important consequence of human intelligence is human languages. The early work on knowledge representation in AI was about human language, and was produced through research in linguistics.

It is humans' quest to understand as to how our and other animals' brains lead to intelligent behavior, with the aim to ultimately build AI systems. On the other hand, it is also aimed to explore the properties of artificial systems, like, computer models/simulations to test our hypotheses concerning human systems.

Many people working in sub-fields of AI are in the process of building models of how the human system operates, and use artificial systems for solving real-world problems.

1.4.5 Biology and Neuroscience

The field of neuroscience says that human brains, which provide intelligence, are made up of tens of billions of neurons, and each neuron is connected to hundreds or thousands of other neurons. A neuron is an elementary processing unit, performing a function called *firing*, depending on the total amount of activity feeding into it. When a large number of neurons are connected together, it gives rise to a very powerful computational device that can compute, as well as learn how to compute.

The concept of human brains, having the capability to compute, as well as learn, is used to build artificial neurons in the form of electronics circuits, and connect them as circuits (called ANN—artificial neural networks) in large quantities, to build powerful AI systems. In addition, the ANN are used to model various human abilities.

The major difference between the functions of neurons and the process of human reasoning is that neurons work at *sub-symbolic level*, whereas much of conscious human reasoning appears to operate at a symbolic level, for example, we do most of the reasoning in the form of thoughts, which are manipulations of sentences.

The collection of neurons in the form of programs called Artificial Neural Networks (ANN), perform well in executing simple tasks, and provide good models of many human abilities. However, there are many tasks of AI that they are not so good at ANN, and other approaches are more promising in those areas, compared to ANN. For example, for natural language processing (NLP) and reasoning, the symbolic logic, called predicate logic is better suited.

1.4.6 Evolution

Unlike the machines, the humans (intelligence) has a very long history of evolution, of millions of years, compared to less than hundred years for electronic machines and computers. The first exhaustive document of human evolution, the evolution by *natural selection* is due to Charles Darwin (1809–1882). The idea is that fitter individuals will naturally tend to live longer and produce more children (may not be truly valid in the modern world). Hence, after many generations, a population will automatically emerge with good innate properties [3].

Due to this evolution, the structure of the human brain, and even the knowledge, are to a sufficient extent built-in at the time of the birth. This is an advantage over ANNs, which have no pre-stored knowledge, hence they need to acquire the entire knowledge by learning only. However, the present-day computers are powerful enough that even the evolution can be simulated using them, and can evolve the AI systems. It has now become possible to evolve the neural networks to some extent so that they are efficient at learning. But, may still be challenging to recreate the long history of the evolution of humans in the ANNs.

A closely related field to ANNs is *genetic programming*, which is concerned with writing the programs that evolve over time, and do not need to modify them as it is done in usual programs when the system requirement changes [4].

1.5 Artificial Consciousness

Right from the time, automated machines like computers came into existence, it was the quest of researchers to build machines that can compete in intelligence with humans. Looking at the current progress rate of smart machines, like smartphones, it is believed that in the not very far future, it may be possible to build machines which may be, if not more, but have comparable intelligence to that of humans. Using such machines, it may be possible to produce human like consciousness in machines—called as artificial consciousness.

On the contrary, even a far of realization of artificial consciousness gives rise to several philosophical nature of questions:

- Can the computers be made to think or they will just calculate?
- Is consciousness a human prerogative only, or it can be created in machines also?
- Is the consciousness due to the material comprised in the human brain, or it can be created in silicon also (the computer hardware)?

To provide the answers to these questions is difficult as of now, mainly because it requires combining the knowledge from the fields of computer science, neurophysiology, and philosophy.

On the other hand, the very talk of artificial consciousness—a possible product of the human imagination, express human desires, and fears about future technologies—may influence the course of progress.

At a social level, the science fiction stories simulate future scenarios that can help prepare us for crucial transitions by predicting the consequences of such technological advances [1].

1.6 Techniques Used in AI

The AI systems have a lot of variations, for example, the rule-based systems are based on symbolic representations, and work on inferences. There are other extremes, the ANN-based system work on the interface with other neurons, and connection weights. In spite of all these, there are four common features among all of them.

Representation

All AI systems have an important feature of knowledge representation. The rule-based systems, frame-based systems, and semantic networks make use of a sequence of *if-then rules*, while the artificial neural networks make use of connections along with connection weights.

Learning

All AI systems have capability of learning, using which they automatically build up the knowledge from the environment, e.g., acquiring the rules for a rule-based expert system, or determining the appropriate connection weights in an artificial neural network.

Rules

The rules of an AI-based system can be implicit or explicit. When explicit, the rules are created by a knowledge engineer, say, for an expert system, and when implicit, they can be, for example, in the form of connection weights in a neural network.

Search

The search can be in many forms, for example, searching the sequence of states that lead to solution faster, or searching for an optimum set of connection weights in an ANN by minimizing the fitness function.

1.7 Sub-fields of AI

Considering AI as replication of human intelligence may be misleading, and primitive. The later is because the true process of human intelligence and its sources are still under debate. However, if AI is taken to mean the advanced computing, it means more justified. In the past two decades, particularly after the year 2k, the AI applications have evolved and expanded, in the commercial, industrial, medicines and drug decide, medical science, consumer products, manufacturing processes, and even in management, to list only a few of its total domain. The use of AI techniques in every organization has become necessary to maintain competitiveness in the market. Many organizations keep secret of the true AI techniques they use.

AI now consists of many sub-fields, using a variety of techniques, such as the following:

Speech Processing: To understand speech, speech generation, machine dialog, machine user-interface.
Natural Language Processing: Information retrieval, Machine translation, Question/Answering, summarization.
Planning: Scheduling, game playing.
Engineering and Expert Systems: Troubleshooting medical diagnosis, Decision support systems, teaching systems.
Fuzzy Systems: For fuzzy controls.
Models of Brain and Evolutionary: Genetic algorithms, genetic programming, Brain modeling, time series prediction, classification.
Machine Vision and Robotics: Object recognition, image understanding, Intelligent control, autonomous exploration.
Machine Learning: Decision tree learning, version space learning.

Most of these have both engineering and scientific aspects. Many of these are going to be discussed in this text. Following is brief Introduction to some of these areas.

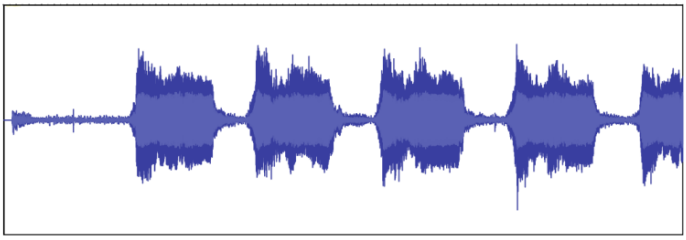


Fig. 1.2 Sound waves for the text "Hello" repeated five times by a five year child

1.7.1 Speech Processing

The speech processing and understanding of human speech has number of applications, some of them we come across quite often, like, speech recognition for dictation systems, speech production for automated announcements, voice-activated control, human–computer interface (HCI), and voice-activated transactions are a few examples.

One of the primary goals is, how do we get from sound waves to text streams and vice-versa? The Fig. 1.2 is an example, showing the sound wave pattern for the text "Hello" repeated five times.

To be precise, how should we go about segmenting the stream into words? How can we distinguish between "Recognize speech" and "Wreck a nice beach"?

1.7.2 Natural Language Processing

Consider the machine understanding and translation of simple sentences given below.

Ram saw the boy in the park with a telescope.
Ram saw the boy in the park with a dog.

In the parse-tree in Fig. 1.3a, the sentence structure is "Ram saw, the boy in the park, with a Telescope." Whereas in Fig. 1.3b, it is "Ram saw, the boy in the park with a dog." In first, it shows the association of verb *saw* and *telescope*, i.e., someone is seeing using telescope. In Fig. 1.3b the association of *boy*, and *dog* is shown, and all are in the park. The further deeper contexts help in resolving this ambiguity.

Though the sentences appear simple, finding out the meaning of each using the machine is difficult, as the parse-tree of each need to be analyzed for the meaning associated in each, in addition, the context knowledge is important.

1.8 Perception, Understanding, and Action

These fields are concerned with vision, speech processing, and robotics. The basic theme is applications that make machine sense (e.g., to see, hear, or touch), then understand and think, and finally take action.

For example, the basic objective of machine vision may be to make the machine "understand" the input consisting of reflected brightness values. Once this understanding is achieved, the results can be used for interpretation of patterns, inspecting parts, action of robots, and so forth. Developing an understanding presents the same difficulty in all areas of AI, including knowledge based systems—people understand what they see by integrating an optical image with complex background knowledge. Such background knowledge has been built over years of experiencing perceptions. Creating this type of information processing in the machine is a challenging task; however, some interesting applications have already appeared as the evidence to support future progress.

The speech processing uses two major technologies. One of these areas focus on input or speech recognition where acoustic input, like optical input in machine vision, is difficult task to automate. People understand what they hear with complex background knowledge. Speech recognition technology includes signal detection, pattern recognition, and possibly semantics—a feature closely related to Natural Language understanding. The other technology concerns to the creation of output or text-to-speech (tts) synthesis. Speech synthesis is easier than recognition, and its commercialization has been well established.

The field of Robotics integrate many techniques of sensing, and, is one of the AI areas in which industrial applications have the longest and widest successful records. The abilities of these robots are relatively limited. For example, only in a limited task such as welding seams and installing windshields, etc [7].

1.9 Physical Symbol System Hypothesis

The symbols are the basic requirements of intelligent activity, e.g., by human the symbols are basic number systems, alphabet of our languages, sign language, etc. Similar is the case with entire computer science, the languages, commands, computations, all have symbols as the base. When the information is processed by computers, on the completion of the task, we measure the progress, as well as the quality of results and efficiency of computations, only based on its symbols' contents in the end results [8].

1.9.1 Formal System

Basic requirement of achieving AI is *formal system*, which is based on *physical symbol system hypothesis*. The term was coined by *Allen Newell and Herbert Simon*, which states that a "physical symbol system" is a necessary requirement for AI to function. As per this, the physical patterns, called symbols, are combined to produce structures (i.e., expressions), and the processes act on these to manipulate to produce new expressions [8].

This symbol system hypothesis claims that human intelligence is due to this symbol system, which comprises all the alphabets, numerals, and other punctuation symbols. Thus, the symbol system is a "necessary" requirement for achieving the intelligence. Based on this argument, one can say that, if the machines are provided with symbol system with symbol manipulation capabilities, it is "sufficient" for achieving the intelligence in the machines.

As per the Physical Symbol System Hypothesis (PSSH), the capabilities of symbol manipulation are the essence for human's, as well as machines' intelligence. Hence, it is a necessary and sufficient tool for achieving intelligence in both the machines and humans. There is also experimental evidence, that, in various problem solving, like, in mathematical puzzles, planning of activities, and execution, the symbol system is the key requirement. By the term "necessary" here means, that the system possessing general intelligence, on analysis, will prove to be based on a physical symbol system. And, the term "sufficient" means that the physical symbol system can be organized to be exhibiting the general intelligence. When the problem-solving process of humans were simulated step by step on computers by the researchers, it was found to be simply the process of symbols' manipulations.

Of course, various researchers have criticized this hypothesis strongly, but still, it forms the central part of AI research. The critics argue that the symbol systems work only for high level processes like chess, games, and puzzles, but not suitable for low level systems like vision and speech recognition. This distinction is based on the fact that high-level symbols directly correspond to objects, like $\langle cat \rangle$, $\langle house \rangle$, $\langle hill \rangle$, etc, but not to the low-level symbols that are present in the machinelike neural networks (or ANN).

1.9.2 Symbols and Physical Symbol Systems

If we look at the entire knowledge of computer science, it is the symbols, which have been used to explain this knowledge at the most fundamental level. The explanation is nothing but the scientific proposition of nature, which is empirically derived over a long period of time, through a graduate development. Hence, the symbols are at the root of artificial intelligence, and are also the primary topic of artificial intelligence.

For all the information processed by computers in the service of finding the end goals, the intelligence of the system (computers) is their ability to reach goals in

the face of difficulties and complexity of the solution, as well as the complexity introduced by the environment. The fundamental requirements to achieve artificial intelligence is to store and manipulate the symbols, however, there is no uniformity and specific requirement of storage structures, as the structures vary from method to method used for implementation of AI, which are mostly the variants of network-based representation and predicate-based representations.

The "physical systems" used have two important characteristics: 1. Operation of systems is governed by the laws of physics, when they are realized as engineered systems, and made of engineered components; and 2. The "symbols" are not limited only to the symbols used by human beings.

1.9.3 Formal Logic

The "physical symbol system" hypothesis has its root to Russel's *formalizing logic*, which states that one need to capture the basic conceptual notion of mathematics in logic and put that notion to proof and deduction as sound base. This notion, with the effort ultimately grew in the form of mathematical logic—the propositional logic, predicate logic, and their variants [13].

1.9.4 The Stored Program Concept

The second generation of computers brought the concept of stored program concept in the mid-forties, after the *Eniac* computer. The arrival of these computers was considered as a milestone in terms of conceptual progress, as well as practical availability of systems. In such systems, the programs are treated as data, which, in turn, was processed by other programs, like a compiler program processing another program as data to generate object code. Interestingly, this capability was already verified and existed in Turing machine, which came as early as 1936. The Turing machine is a model of computing given by Alan M. Turing, where, in a universal Turing machine, an algorithm (another Turing machine) and data are on the very same tape. This idea was realized practically when machines were built with enough memory to make it practicable to store actual programs in some internal place, along with the data on which the program will act, as well as the data which will be produced as a result of the execution of the programs.

1.10 Considerations for Knowledge Representation

As far as AI is concerned, the following are the aspects of knowledge representation:

- What is the meaning of Knowledge?
- How the Knowledge can be represented in the machine?
- What are the requirements of representation of knowledge, e.g., structures, methods, size, etc.
- How the practical and theoretical aspects differ for knowledge representation?
- Can it be? Or, if yes, how to represent the knowledge using Natural Language?
- Can we call the databases as a form of knowledge representation?
- What are the semantic networks, and what are the frames? How the knowledge can be represented using these approaches?
- How the knowledge can be represented using the First-Order Predicate Logic (FOPL)?
- What is a Rule-Based Systems?
- What is an expert system?
- Out of the many techniques, which is the best technique for knowledge representation?

1.10.1 Defining the Knowledge

As per the Webster English language dictionary, the following are the meanings of knowledge:

1. The act or state of knowing; clear perception of fact, truth, or duty; certain apprehension; familiar cognizance; cognition. [1913 Webster]
   Knowledge, which is the highest degree of the speculative faculties, consists in the perception of the truth of affirmative or negative propositions—Locke. [1913 Webster]
2. That which is or may be known; the object of an act of knowing; a cognition—Chiefly used in the plural. [1913 Webster]
3. That which is gained and preserved by knowing; instruction; acquaintance; enlightenment; learning; scholarship; erudition. [1913 Webster]

1.10.2 Objective of Knowledge Representation

The objective of knowledge representation is to express the knowledge in computer so that the AI programs can use it to perform reasoning and inferences using this in an efficient way. The knowledge is represented using certain representation language,

for example, a predicate like language. The language has two important components in it.

Syntax

The system of a language defines the methods using which we or the machine can distinguish the correct structures from incorrect, i.e., it makes possible to identify the structurally valid sentences.

Semantics

The semantics of a language defines the world, or facts in the world of the concerned domain. And, hence defines the meaning of the sentence in reference, to the world.

1.10.3 Requirements of a Knowledge Representation

A good knowledge representation system for any particular domain should possess the following properties.

Adequacy of representation

The representation system should be able to represent all kind of knowledge needed in the concerned AI-based system.

Adequacy of Inference

The representation should be such that all that can be inferable by manipulating the given knowledge structures should be inferred by the system, when needed.

Inference Efficiency

The knowledge structures in the representation are so organized that the attention of the system, in the form of deductions, navigates in such a direction that it can reach the goal quickly.

Efficient acquisition

It should be able to acquire the new information automatically and efficiently, as and when needed, and also to update the knowledge regularly. In addition, there should be a provision that knowledge engineer can update the information in the system.

1.10.4 Practical Aspects of Representations

We are aware of good and bad knowledge representation, when we consider the knowledge representation in English or any other natural language. These are due to factors, like, syntax, semantics, partial versus full knowledge on any subject, depth and breadth of knowledge, etc.

AI has its inter-related goals for scientific, as well as engineering areas. Its roots are in several historical disciplines, which include, philosophy, logic, computation, psychology, cognitive science, neuroscience, biology, and evolution.

The major sub-fields of AI now include: neural networks, machine learning, evolutionary computation, speech recognition, text-to-speech translation, fuzzy logic, genetic algorithms, vision systems and robotics, expert systems, natural language processing, and planning. Many of these domains have dependency and are inter-related, for example, neural network is one of the techniques for machine learning. The common techniques used across these sub-fields are: knowledge representation, search, and information manipulations.

Human brain and evolution are also the areas of AI modeling.

The study of logic and computers have demonstrated that intelligence lies in the *physical symbol system* (PSS)—a collection of patterns and processes. The PSS needs the capability to manipulate the patterns, i.e., it should be able to create the patterns, modify the patterns, and should be able to destroy the patterns. The patterns have important properties, that they can designate objects, processes, and other patterns. When the patterns designate processes, the later can be interpreted, i.e., to perform the process steps. The two significant classes of symbol systems we are familiar with those that are used by human beings, and those by computers. The later uses binary strings or patterns.

The PSSH (physical symbol system hypothesis) says that to achieve the intelligence, it is sufficient to have three things,

i. a representation system, using which anything can be represented,
ii. a manipulation system, using which the symbols can be manipulated, and
iii. search using which the solution can be searched.

For the above, it is in fact, not important as whether the medium of storage is the human brain (neurons) or the electronic memory of computer systems.

Various approaches for knowledge representations (KR) are:

i. Natural languages versus databases.
ii. Frame versus semantic network-based representation.
iii. Propositional and predicate logic-based representation.
iv. Rule-based representation.

Knowledge representation helps to know the object or the concerned concept. Various characteristics of KR are:

i. KR has syntax and semantics.
ii. Requirements for knowledge representation are: adequacy of representation and inferencing, and efficiency of inference and acquisition.
iii. Its practical aspects are: complete, computable, and suppression of irrelevant data.
iv. Components for KR are: lexical, structural, semantic, and procedural.

Exercises

1. Try to analyze your own learning behavior, and list the goals of learning, in the order of their difficulty level of learning.
2. List the living beings—human, dog, cow, camel, elephant, cat, birds, insects, in the order of their intelligence levels. Also, justify your argument.
3. Suggest some method to combine large number of human beings, in a group, and formulate a method/algorithm to perform the pipeline or any fast computing work.
4. Write an essay, describing the various anticipated theories as to how the human processes the information?
5. Consider a task requiring knowledge like baking a cake. Out of your imagination, suggest what are the knowledge requirements to complete this task.
6. Out of your reasoning, explain the distinction between knowledge and belief.
7. What is the basic difference between neural network level processing and processing carried out for human reasoning?
8. What are the major advantages of humans over modern computers?
9. List the examples where PSSH is not sufficient or not the basis of achieving Intelligence. Justify your claims.
10. Write an essay, describing how the things (objects/activities) are memorized by

    a. Plants.
    b. Birds.
    c. Sea animals.
    d. Land animals.
    e. Human.

    What can be the size of memories in each of the above cases?
11. Discuss the potential uses of AI in the following applications:

    a. Word processing systems.
    b. Smartphones.
    c. Web-based auction sites.
    d. Scanner machines.
    e. Facebook.
    f. Twitter.
    g. Linkedin.
    h. Amazon and Flipkart.

12. How the artificial neural networks (ANN) and genetic algorithms differ from each other in respect of leaning to be used for problem solution? (Note: You need to explore the AI resources to answer this question).

13. It is an accepted scientific base that physical characteristics of life are genetically transferred. Do you believe that information and knowledge are also genetically transferred? Justify for yes/no?
14. Are the beliefs of rebirth and reincarnation are also the goals of AI research? How and how not?

References

1. Buttazzo G (2001) Artificial consciousness: utopia or real possibility? IEEE Comput 34(7):24–30
2. Church KW, Lisa FR (1995) Commercial applications of natural language processing. Commun ACM 38(11)
3. Forrest S (1993) Genetic algorithms: principles of natural selection applied to computation. Science 261(13):872–878
4. Goldberg DE (1994) Genetic and evolutionary algorithms coming of age. Commun ACM 37(3):113–119
5. http://www.theguardian.com/technology/2014/jun/08/super-computer-simulates-13-year-old-boy-passes-turing-test. Accessed 19 Dec 2017
6. Ludger GF (2009) Artificial intelligence - structures and strategies for complex problem solving, 5th edn. Pearson, New Delhi
7. Munakata T (1994) Commercial and industrial applications of AI. Commun ACM 37(03)
8. Newell A, Herbert AS (1976) Computer science as an empirical inquiry: symbols and search. Commun ACM 19(3):113–126
9. Russell SJ (1997) Rationality and intelligence. Artif Intell 94:57–77
10. Russell SJ, Norvig P (2005) Artificial intelligence, a modern approach, 2nd edn. Pearson, New Delhi
11. Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/turing-test/. Accessed 19 Dec 2017
12. Turing AM (1950) Computing machinery and intelligence. MIND - Q Rev Psychol Philos Lix(236):433
13. Whitehead AN, Russell B (1910) Principia mathematica, vol 1 (Part I. Mathematical logic). Merchant Books. ISBN 978-1-60386-182-3

Chapter 2
Logic and Reasoning Patterns

Abstract Logic is the foundation of AI, and the majority of AI's principles are based on logical or deductive reasoning. The chapter presents: contributions of pioneers of logic, the argumentation theory, which is based on logic and with its roots in propositional logic, the process of validating the propositional formulas, their syntax and semantics, interpretation of a logical expression through semantic tableau, followed with presents the basic reasoning patterns used by human, and their formal notations. In addition, presents the normal forms of propositional formulas and application of resolution principle on these for inference. The nonmonotonic reasoning and its significance is briefly described. At the end, the chapter presents the axiomatic system due to Hilbert and its limitations, and concludes with chapter summary.

Keywords Logic · Propositional logic · Deductive reasoning · Argumentation theory · Syntax and semantics of propositional formulas · Nonmonotonic reasoning · Hilbert's axiomatic system

2.1 Introduction

The ancient *Greeks* are the source of modern logic, their education system emphasized the competence in rhetoric (proficient in language) and philosophy; the words *axioms* and *theorem* are from Greek. The logic was used to formalize the deductions—the derivation of true *conclusions*—from true *premises*. Later it was formalized as a set theory by the mathematician *George Boole*. Till the arrival of the nineteenth century, the logic remained more of a philosophical nature, rather than a mathematical and scientific tool. Later, since complex things could not be reasoned through logic, the logic became part of mathematics, where mathematical deduction became justifiable through formalizing a system of logic, and resulted in one very important breakthrough. This was, about the set of true statements, stated as "the set of provable statements are only those that are true statements." This is because some proof exists for those due to some other true statements.

At the beginning of nineteenth century, the mathematician *David Hilbert* introduced the logic, as well as theories of the nature of logic-a far more generalization

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_2

of the logic. But, this generalization received a blow when another mathematician *Kurt Gödel* showed in 1931 that there are true statements of arithmetics that are not provable, through his *incompleteness* theorem.

Now, though mathematical logic remains the branch of pure mathematics, it is extensively applied to computer science and artificial intelligence in the form of propositional logic and predicate logic (first-order predicate logic (FOPL)).

As per the Newell's and Simons's Physical Symbol System Hypothesis (PSSH), discussed in the previous chapter, the knowledge representation is the first requirement of achieving intelligence. This chapter presents the knowledge representation using *propositional logic*, introduces first-order predicate logic (FOPL), and drawing of inferences using propositional logic.

Logic is a formal method for reasoning, using its concepts can be translated into symbolic representation, which closely approximate the meaning of these concepts. The symbolic structures can be manipulated using computer programs to deduce facts to carry out the form of automated reasoning [9].

The *aim* of logic is to learn principles of valid reasoning as well as to discern good reasoning from bad reasoning, identifying invalid arguments, distinguishing *inductive* versus *deductive* arguments, identifying *fallacies* as well as avoiding the fallacies.

The *Objective* of logic is to equip oneself with various tools and techniques, i.e., *decision procedures* for validating given arguments, detecting and avoiding fallacies of a given deductive or inductive argument.

We study the logic because of the following reasons:

- Logic deals with what follows from what? For example, Logical consequence, inference pattern, and validating such patterns,
- We want the computer to understand our language and does some intelligent tasks for us (Knowledge representation),
- To engage in debates, solving puzzles, game like situation,
- Identify which one is a fallacious argument and what is a type of fallacy?
- Proving theorems through *deduction*. To find out whether whatever proved is correct, or whatever obviously true has a proof, and
- To solve some problems concerning the foundations of mathematics.

Learning Outcomes of This Chapter:

1. Convert logical statements from informal language to propositional logic expressions. [Usage]
2. Apply formal methods of symbolic propositional such as calculating the validity of formula and computing normal forms. [Usage]
3. Use the rules of inference to construct proofs in propositional. [Usage]
4. Describe how symbolic logic can be used to model real-life situations or applications, including those arising in computing contexts such as software analysis (e.g., program correctness), database queries, and algorithms. [Usage]

The meaning (*semantics*) is associated with each formula by defining its *interpretation*, which assign a value *true* (T) or *false* (F) to every formula. The syntax is also used to define the concept of *proof*—the symbolic manipulations of formulas to deduce the given theorem. The important thing we should note is that provable formulas are only those which are always true.

We start the propositional logic with the individual propositional variables. These variables themselves are formulas, which cannot be further analyzed. We represent these by English alphabets and subscripted alphabets $p, q, r, s, t, p_1, p_2, q_1, q_2, \ldots,$ etc. These formulas may have smaller constituents but it is not the role of propositional logic to go into the details of their constructions. The use of letters to represent propositions is not in true sense variables, they simply represent the propositions or statements in a symbolic form, and they are not the variables in the sense used in *predicate logic* (to be discussed later), or in *high-level languages* like *C* or *Fortran*, where a variable stands for a domain of values. For example, an integer variable in a Fortran program stands for any integer number as per the specifications of the language.

The other symbols of propositional logic are *operators* as follows:

$\wedge$ conjunction operator,
$\vee$ disjunction operator,
$\neg$ not or inverting operator,
$\rightarrow$ implication, i.e., if ... than ... rule, and
$\perp$ contradiction (false).

Let following be the propositions:

$p =$ Sun is star.
$q =$ Moon is satellite.

We can construct the following formulas using the above propositions:

$p \wedge q =$ Sun is star *and* Moon is satellite.
$p \vee q =$ Sun is star *or* Moon is satellite tennis.
$\neg p \vee q =$ Sun is *not* star or Moon is satellite.
$\neg p \rightarrow q =$ *if* Sun is *not* star *then* Moon is satellite.

A formula in propositional logic can be recursively defined as follows:

(i) Each propositional variable and null are formulas, therefore, p, q, ϕ are formulas,
(ii) If p, q are formulas, then $p \wedge q, p \vee q, \neg p, p \rightarrow q, (p)$, are also formulas,
(iii) A string of symbols is a formula only as determined by finitely many applications of above (i) and (ii), and
(iv) nothing else is propositional formula.

This recursive form of the definition can be expressed using BNF (Backups-Naur Form) notation as follows:

$$
\begin{aligned}
&1.\ formula := atomicformula \mid formula \wedge formula \mid formula \vee formula \\
&\quad \mid formula \rightarrow formula \mid \neg formula \mid (formula) \\
&2.\ atomicformula := \bot \mid p \mid q \mid r \mid p_0 \mid p_1 \mid p_2 \mid \ldots
\end{aligned}
\tag{2.1}
$$

In the above notation, the symbols—$formula$ and $atomicformula$, that appears to the left-hand are called *non-terminals* and represent grammatical classes. The $p, q, r, \bot, p_1$, etc, that appear only to the right-hand side, are called *terminals*, and represent the symbols of the language.

A sentence in the propositional language is obtained through a derivation that starts with a non-terminal, and repeatedly applied the substitution rules from the BNF notations, until the terminals are reached [8].

Example 2.1 Derivation for $p \wedge q \rightarrow r$.

The sequence of substitutions rules to derive this formula, i.e., to establish that it is syntactically correct, are as follows:

$$
\begin{aligned}
formula &\Rightarrow formula \rightarrow formula \\
&\Rightarrow formula \wedge formula \rightarrow formula \\
&\Rightarrow atomic \wedge formula \rightarrow formula \\
&\Rightarrow p \wedge formula \rightarrow formula \\
&\Rightarrow p \wedge atomic \rightarrow formula \\
&\Rightarrow p \wedge q \rightarrow formula \\
&\Rightarrow p \wedge q \rightarrow atomic \\
&\Rightarrow p \wedge q \rightarrow r.
\end{aligned}
$$

The symbol *atomic* stands for atomic formula and the symbol "$\Rightarrow$" stands for "implies", i.e., the expression to right to this is implied by the expression to left of "$\Rightarrow$".

The derivation can also be represented by a *derivation-tree* (*parse-tree*), shown in Fig. 2.2. From the derivation-tree, we can obtain another tree shown in Fig. 2.3, called *syntax-tree* or *formation-tree*, by replacing each non-terminal by the child that is an operator under that. There is always unique syntax-tree for every formula. □

Considering two propositions p, q, the interpretation (semantics) of the formulas constructed when they are joined using binary operators ($\vee, \wedge, \rightarrow$) are shown in the truth-table Table 2.1.

The *Material conditional* '$\rightarrow$' joins two simpler propositions, e.g., $p \rightarrow q$, read as "if p then q". The proposition to the left of the arrow is called the *antecedent* and to the right is *consequent*. There is no such designation for conjunction or disjunction operators because they are commutative operations. The $p \rightarrow q$ expresses that q is true whenever p is true. Thus it is true in every case in Table 2.1, except in row three, because this is the only case when p is true but q is not. Using "if p then q",

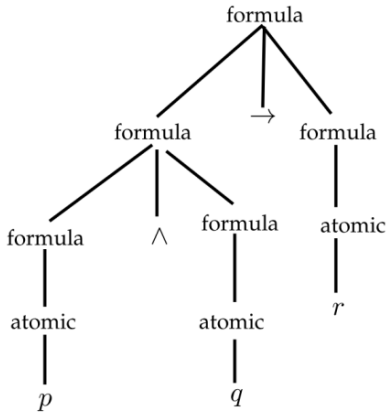


Fig. 2.2 Parse-tree for the expression $p \wedge q \rightarrow r$

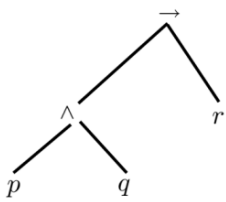


Fig. 2.3 Syntax-tree for the expression $p \wedge q \rightarrow r$

Table 2.1 Interpretation of propositional formulas

p	q	$p \vee q$	$p \wedge q$	$p \rightarrow q$
F	F	F	F	T
F	T	T	F	T
T	F	T	F	F
T	T	T	T	T

we can express that "if it is raining outside then there is a cold over Kashmir". The material conditional is often confused with *physical causation*. The material conditional, however, only relates two propositions by their truth values—which is not the relation of *cause* and *effect*. It is contentious in the literature whether the material implication represents logical causation.

2.4.1 Interpretation of Formulas

The interpretation of formula is assigning truth value to that formula. As discussed earlier, a formula can be atomic or in may be complex, i.e., joining or atomic formulas. The following are some definitions related to the interpretation of formulas [1].

Definition 2.1 (*Satisfied, model, valid, and tautology*) A propositional formula A is *satisfied* iff $\mathscr{I}(A) = True$ for some interpretation $\mathscr{I}$. A *satisfying interpretation* is called model for A. The formula A is called valid, denoted by $\models A$, iff $\mathscr{I}(A) = True$ for all interpretations $\mathscr{I}$. A valid propositional formula is also called tautology.

A propositional formula is *unsatisfiable* (also called *contradiction,* $\perp$), *iff* it is not satisfiable, i.e., $\mathscr{I}(A) = False$, for all interpretations $\mathscr{I}$. If $\mathscr{I}(A) = False$ for some interpretation $\mathscr{I}$, then A is called non-valid or falsifiable, and denoted by $\not\models A$.

Definition 2.2 (*Simultaneously satisfiable*) A set of formulas $S = \{A_1, A_2, \ldots, A_n\}$ is *simultaneously satisfiable iff* there exists an interpretation $\mathscr{I}$ such that $\mathscr{I}(A_i) = True$ for all i. The S is unsatisfiable *iff* for every interpretation $\mathscr{I}$ there exits an i such that $\mathscr{I}(A_i) = False$.

2.4.2 Logical Consequence

The *logical consequence* or *logically follows* is the central concept in the foundations of logic. It is much more interesting to assume that a set of formulas is true and then to investigates the consequences of these assumptions [1].

Assume that θ and ψ are formulas (sentences) of a set $\mathscr{P}$, and $\mathscr{I}$ is an interpretation of $\mathscr{P}$. The sentence θ of propositional logic is true under an interpretation $\mathscr{I}$ iff $\mathscr{I}$ assigns the truth value T to that sentence. The θ is false under an interpretation $\mathscr{I}$ iff θ is not true under $\mathscr{I}$.

Definition 2.3 (*Logical consequence*) A sentence ψ of propositional logic is a *logical consequence* of a sentence (or set of sentences) θ, represented as $\theta \models \psi$, if every interpretation $\mathscr{I}$ that satisfy θ also satisfy ψ.

In fact, ψ need not be true in every possible interpretation, only in those interpretations which satisfy θ, i.e, those interpretations which satisfy every formula in θ. In the formula $((p \rightarrow q) \wedge p) \vdash q$, the q is logical consequence of $((p \rightarrow q) \wedge p)$. The sign '$\vdash$', is sign of *deduction*, and $S \vdash q$ is read as S deduces q, where S is a set of formulas and q is the formula.

A sentence of propositional logic is *consistent iff* it is true under at least one interpretation. It is *inconsistent* if it is not consistent.

Example 2.2 Determine the logical consequence of $\psi = (p \vee r) \wedge (\neg q \vee \neg r)$ from $\theta = \{p, \neg q\}$, i.e., find $\theta \models \psi$, and validity for ψ.

Here ψ is logical consequence of θ, denoted by $\theta \models \psi$, because ψ is true under all the interpretations such that $\mathscr{I}(p) = True$, and $\mathscr{I}(q) = False$, is the interpretation, for which θ is satisfied.

However, ψ is not valid, since it is not true under the interpretation $\mathscr{I}(p) = F$, $\mathscr{I}(q) = T$, $\mathscr{I}(r) = T$.

Further note that $\theta \vdash \psi$ is a valid statement because the expression $\theta \vdash \psi$ is always true. □

2.4.3 Syntax and Semantics of an Expression

Syntax is name given to a correct *structure* of a statement. It is the meaning associated with the expression. It is mapping to the real-world situation is *semantics*. The semantics of a language defines the truth of each sentence with respect to each possible *world*. For example, the usual semantics for interpretation of the statement $(p \vee q) \wedge r$ is true in a *world* where either p or q or both are *true* and r is *true*. Different worlds can be all the possible sets of *truth* values of p, q, r, which is total 8. The truth values are simply the assignment to these variables, and not necessarily the values which are only *true*. For example, $\mathscr{I}(p) = F, \mathscr{I}(q) = F, \mathscr{I}(r) = T$; and $\mathscr{I}(p) = T, \mathscr{I}(q) = F, \mathscr{I}(r) = T$ are the possible worlds for the expression $(p \vee q) \wedge r$.

2.4.4 Semantic Tableau

Semantic tableau is relatively efficient method for deciding satisfiability for the formula of propositional calculus. The method (or algorithm) *systematically* searches for a model for a formula. If it is found, the formula is satisfiable, else not satisfiable. We start with the definition of some terms, and then analyze some formulas to motivate us for the construction of semantic tableau [1].

Definition 2.4 (*Literal and complementary pair*) A literal is an atom or negation of an atom. For any atom p, the set $\{p, \neg p\}$ is called complementary pair of literals. For any formula A, $\{A, \neg A\}$ is complementary pair of formulas.

Example 2.3 Analysis of the satisfiability of a formula.

Consider that a formula $A = p \wedge (\neg q \vee \neg p)$, has an arbitrary interpretation $\mathscr{I}$. Given this, $\mathscr{I}(A) = T$ iff $\mathscr{I}(p) = T$ and $\mathscr{I}(\neg q \vee \neg p) = T$. Hence, $\mathscr{I}(A) = T$ iff either,

1. $\mathscr{I}(p) = T$ and $\mathscr{I}(\neg q) = T$, or
2. $\mathscr{I}(p) = T$ and $\mathscr{I}(\neg p) = T$.

Hence A is satisfiable if either (1) interpretation holds or (2) holds. But (2) is not feasible. So, A is satisfiable when the interpretation of (1) holds true. Note that the satisfiability of a formula is reduced to the satisfiability of literals.

It is clear that a set of literals is satisfied if and only if it does not contain complementary pair of literals. In the above case, the pair of literals $\{p, \neg p\}$ in case (2) is complementary pair, hence the formula is unsatisfied for this interpretation. However, the first set $\{p, \neg q\}$ is not the complementary pair, hence it is satisfiable.

From the above discussion, we have trivially constructed a model for the formula A by assigning *True* to positive literals and *False* to negative literals. Hence, $p =$

Table 2.3 Inference rules

Rule	Formula	Description
Modus ponens	$((p \rightarrow q) \wedge p) \vdash q$	If p then q; p; therefore q
Modus tollens	$((p \rightarrow q) \wedge \neg q) \vdash \neg p$	If p then q; not q; therefore not p
Abduction	$(p \rightarrow q) \wedge q \vdash p$	if p then q; q; therefore q
Hypothetical syllogism	$((p \rightarrow q) \wedge (q \rightarrow r))$	if p then q; $\vdash (p \rightarrow r)$ if q then r; therefore, if p then r
Disjunctive syllogism	$((p \vee q) \wedge \neg p) \vdash q$	Either p or q, or both; not p; therefore, q

conclusion is not necessarily true, because there are other reasons also for lung cancer, which are not due to smoking. When statistics and probability theory are used along with abduction, it may result in most probable inferences out of the many likely inferences. To illustrate how the abduction based reasoning works, we consider a logical system comprising a general rule and one specific proposition.

> All successful enterprising industrialists are rich (general rule). Rajan is a rich person (specific proposition). Therefore, a plausible inference can be that Rajan is a successful, enterprising industrialist.

However, this conclusion can be false also, because there are many other paths to richness, such as a lottery, inherited property, coming across a treasure, and so on. If we have a table of all the riches and how they became rich, we may draw the probability of abduction for richness to be true in this case.

Inductive Reasoning

The inductive reasoning arrives at a conclusion about all members of a class. It is based on examination of only a few members of the class and based on that it generalizes for the entire class. It is broadly reasoning from a *specific* to the *general*. For example, the traffic police comes to know about following situation on a particular day about nature of road accidents:

> 1st accident was due to wrong side drive,
> 2nd accident was due to wrong side drive,
> 3rd accident was due to wrong side drive.

One would logically infer that all the accidents are due to wrong side driving. Another example is about the birds for their flying attribute.

> Crow fly,
> peacock fly,
> pigeon fly.

Thus, we conclude that all the birds fly.
Another example is about the progressive sum of 1st n odd integers:

$$1 = 1^2$$
$$1 + 3 = 2^2$$
$$1 + 3 + 5 = 3^2$$
$$1 + 3 + 5 + 7 = 4^2$$

Thus, by induction we prove that, the sum of n successive odd integers is n^2.

The outcome of the inductive reasoning process will frequently contain some measures of uncertainty because including all possible facts in the premises are usually impossible.

We know that the inference of an accident's example is not always true, and also of "all birds fly" is not true, because, ostrich and penguins do not fly. However, for 1st odd integers sum, it is true.

The deductive or inductive approaches are used in logic, rule-based systems, and in frames.

Analogical Reasoning

The analogical reasoning assumes that when question is asked, the answer can be derived by analogy, as in the case of following example.

Premise: All the 100 m racers get 5% additional in their merit score.
Question: How much one 400 m racer will get additional in academic score?
Conclusion: Because, 400 m is a race, and an sports activity like 100 m, so it will also benefit one with 5% in final scores.

Analogical reasoning is a type of verbalization of an internalized learning process. An individual uses processes that require the ability to recognize previously encountered experiences. This approach is not very common in AI, however, the case-based reasoning, semantic networks, and frames use this analogical reasoning approach.

Formal Reasoning

It uses the process of syntactic manipulation of data structures to deduce new facts. A typical example is the mathematical logic used in proving theorems in geometry. For example, proof by *resolution*.

Procedural and Numeric Reasoning

It uses mathematical models or simulation to solve the problems. The model-based reasoning is an example of this approach.

Generalization and Abstraction

The approaches of generalization and abstraction, both can be used with the logical and semantic representation of knowledge.

Meta-level Reasoning

The meta-level reasoning involves the knowledge about what you, how much you know about so and so. Also, which approach to use, how successful the inference will

be, depends on a great extent on which knowledge representation method is used. For example, reasoning by analogy can be more successful with semantic networks than with frames.

2.5.1 Rule-Based Reasoning

The rule-based reasoning is also called *pattern matching*, and uses forward and backward chaining. The implementation of rule-based system makes use of modus ponens and other approaches. Consider the rule:

Rule 1: If export rises the prosperity increases.

Using the modus ponens, if the premises, e.g., "The export rises" is *true*, the conclusion of the rule is accepted as *true*. We call this accepting the rule as "rule fires". The firing of a rule occurs when all its premises are satisfied, whether all are true or some are false. On firing, the resulting conclusion is stored in the assertion base, to use for further firing of the rules and generate the assertions. When a premise is not available as an assertion, it can be obtained by querying the user, or by firing other rules. Testing of a rule premise or conclusion is as simple as matching a symbol pattern.

Every rule in the knowledge base can be checked to see if its premises or conclusion can be satisfied by previously made assertions. This process of matching, if done using forward chaining, i.e., premises to conclusions. If it is done from conclusions to premises, it is called *backward chaining*.

2.5.2 Model-Based Reasoning

A reasoning within a context is important in any reasoning system. In real-life situations, one often provides a lot of missing contexts or out of context information when answering certain queries. This situation can be correctly modeled by supplementing the existing knowledge about the world, with additional context-specific information. When it is supplemented by context information, reasoning within context becomes a deduction process.

The added information may act as constrain to the existing information in the system, as in the absence of this additional information the deduction process has more paths of freedom in the reasoning process. But, due to the availability of this added context information the reasoning task becomes easier because the domain in which reasoning takes place gets restricted (constrained) due to having lesser flexibility of deduction paths to be navigated. This task can be formalized as a task of varying contexts.

The knowledge that comprises the information for reasoning in the model-based system is in the form of a set of models of the world. These models satisfy the

assignments and examples of the world. This is, in contrast, to the use of only the formulas in the first-order predicate logic to describe the world. The other difference is that the model-based approach is motivated from a cognitive point of view – the forerunners of this approach of reasoning are cognitive psychologists who support the "reasoning by examples." When a model-based reasoning system is presented with a query, the reasoning is performed by evaluating the query on these models.

Let us suppose that model-based knowledge base representation Γ, and a query α are both given, and it is required to find out if Γ implies α (i.e., $\Gamma \models \alpha$)? This we can determine in two steps: 1) evaluate α on all the models in the representation, 2) If there is a model of Γ that does not satisfy α, the Γ does not model the alpha (i.e., $\Gamma \not\models \alpha$), otherwise we conclude that $\Gamma \models \alpha$. This means if the model-based representation contains all the models of Γ, then by definition, this approach verifies the implication correctly, and produces the correct deduction.

However, there is a problem—the representation of Γ, such that it explicitly holds all the models, is not a plausible solution. The model-based approach is feasible only if Γ can be replaced by small model-based representation, and after that also it should correctly support the deduction.

Various topics in reasoning are as follows:

Monotonic versus nonmonotonic reasoning,
Reasoning with uncertainty,
Shallow and deep representation of knowledge,
Semantic networks,
Blackboard approach,
Inheritance approach,
Pattern matching,
Conflict resolution.

These are discussed in current, and the following chapters, in details.

2.6 Proof Methods

There are two different methods, one is through *model checking* and other is *deduction* based. The first comprises enumeration of truth-tables, and is always exponential in n, where n is the size of the set of propositional symbols. The other, i.e., deduction based approach is repeated application of inference rules. The inference rules are used as operators in the standard search algorithm. In fact, the application of the inference approach to proof is called searching for solution. Proper selection of search directions is important here, as these will eliminate many unnecessary paths that are not likely to result in the goal. Consequently, the proof-based approach for reasoning is considered better and efficient compared to model enumeration/checking based method. The later is exhaustive and exponential in n, where n is the size of the set of propositional symbols.

The property, the logical system follows, is the fundamental property of *monotonicity*. As per this, if $S \vdash \alpha$, and β is additional assertion, then $S \wedge \beta \vdash \alpha$.

Thereby, the application of inference rules is legitimate (*sound*) rule, which helps in the generation of new knowledge from the existing. If a search algorithm like DFS (depth first search) is used, it will always be possible to find the proof, as it will search the goal, whatever the depth may it be. Hence, the inference method in this case is *complete* also [7].

Before the inference rules are applied on the knowledge base, the existing sentences in the knowledge base (KB) needs to be converted into some *normal form*.

2.6.1 Normal Forms

A logical expression can be represented as sum-of-product terms or product-of-sum terms. If a given logical expression is represented as sums of elementary products, then this form is called *disjunctive normal form* (DNF), and if it is represented as product of elementary sums, it is called *conjunctive normal form* (CNF). In DNF, the elementary product terms are called *minterms*, while in a CNF elementary sum terms are called *maxterms*. For a given formula, an equivalent disjunctive normal form with only disjunctions of minterms is called *principle disjunctive normal form* or *sum-of-products canonical form*. Similarly, an equivalent CNF with only conjunctions of maxterms is called *principle conjunctive normal form* or *product-of-sums canonical form* [2].

One technique to get a CNF expression for a given DNF expression, say, $\neg a \neg bc + \neg ab \neg c + \neg abc + a \neg bc$ is given in steps as follows:

1. Considering a DNF expression of three variable a, b, c, write down all the minterms: $\neg a \neg b \neg c$, $\neg a \neg bc$, $\neg ab \neg c$, $\neg abc$, $a \neg b \neg c$, $a \neg bc$, $ab \neg c$, abc.
2. Cross out all combinations in the original DNF. We are left with $\neg a \neg b \neg c$, $a \neg b \neg c$, $ab \neg c$, abc.
3. Next, write the expression in CNF by inverting each subset of three variables and ORing as $(a + b + c)(\neg a + b + c)(\neg a + \neg b + c)(\neg a + \neg b + \neg c)$ in the form of CNF.

Obtaining DNF from CNF is just the reverse process.

2.6.2 Resolution

The *resolution rule* is an inference which uses *deduction* approach. It is used in theorem proving. If two disjunctions have complementary literals, then a resultant inference of these is disjunction of these expressions, with complementary terms removed. If $p = p_1 \vee p_2 \vee c$ and $q = q_1 \vee \neg c$ are two formulas, then resolution of

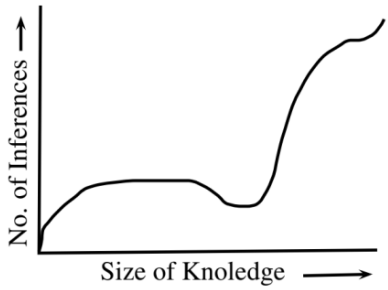


Fig. 2.5 Nonmonotonic reasoning

based on classical logic. In classical logic, if a conclusion is warranted on the basis of certain premises (knowledge), no additional premises will ever invalidate the conclusion.

In everyday life, however, it seems clear that we humans draw sensible conclusions from what we know and that, on the face of new information we often have to take back previous conclusions. This happens even when the new information we gathered in no compel us to take back our previous assumptions (see Fig. 2.5).

For example, we may hold the assumption that "most birds fly", but that "penguins are birds that do not fly". On learning that "Tweety is a bird", we infer that "Tweety flies." However, on learning that "Tweety is a penguin," will in no way make us change our mind about the fact that most birds fly, and also that penguins are birds that do not fly or the fact that Tweety is a bird. However, it should make us abandon our conclusion about Tweety's flying capabilities. It is desirable that intelligent automated systems will have to do the same kind of (nonmonotonic) inferences.

Considering that Γ is a set of sentences of propositional logic, and α is inferred from it, i.e $\Gamma \vdash \alpha$. For any new propositional sentences β, if $\Gamma \cup \{\beta\} \vdash \alpha$ then it is *monotonic* reasoning. If it is not necessary that $\Gamma \cup \{\beta\} \vdash \alpha$, then it is *nonmonotonic* reasoning. We note from Fig. 2.5, that some times, even when we add into knowledge base, the number of inferences decreases instead of increasing; and, this is property of nonmonotonic reasoning.

Some of the systems that perform such nonmonotonic inferences are—*negation as failure*, *circumscription*, *modal system*, *default logic*, *autoepistemic logic*, and *inheritance systems*.

2.8 Hilbert and the Axiomatic Approach

An axiomatic system comprises a set of *axioms* and a set of *primitives*, where the primitives are object names but, these objects are left undefined. The axioms are the sentences that make assertions about the primitives. Further, these assertions are not provided with any justifications, so they are neither true nor false. The subsequent or new assertions about the primitives are called *theorems*, are rigorous logical consequences of axioms and previously proved theorems.

In 1899 the mathematician David Hilbert published his ground-breaking research in the form of a book. He provided a complex deductive system based on five *groups of axioms*, namely:

1. Axioms of *incidence*,
2. Axioms of *order*,
3. Axioms of *congruence*,
4. Axioms of *continuity*, and
5. an axiom of *parallels*.

As per Hilbert's approach, the basic concepts of geometry comprises *points*, *lines* and *planes* of Euclidean geometry. However, these concepts are never explicitly defined. Instead, they are implicitly defined by the axioms such that, points, lines, and planes are any family of *mathematical objects* that satisfy the given axioms of geometry.

Twenty years later Hilbert was considered as the chief promoter of a program intended to provide solid foundations to arithmetic, based on purely axiomatic methods—the mathematics that model all the computations. It was called *formalist* program, and Hilbert was identified as the champion of the formalist approach to mathematics as a whole [6].

2.8.1 Roots and Early Stages

The *formal definitions* in an axiomatic system serves the purpose to simplify the things as they can be used to create new objects made of complex combinations of primitives and previously defined terms (objects and theorems). If a definite meaning is assigned to a primitive of an axiomatic system, called as an *interpretation*, the theorems become meaningful assertions.

Following are some definitions of the axiomatic system.

Definition 2.11 (*Model (for axiomatic system.)*) If all the axioms are true for a given interpretation, then everything asserted by the theorem is also true. Such an interpretation is called a model for the axiomatic system.

Definition 2.12 (*Inconsistent (axiomatic system.)*) Since a contradiction can never be true, an axiomatic system using a contradiction can arrive at a logical deduction that it has no model. An axiomatic system with this property is called *inconsistent*.

Definition 2.13 (*Consistent (axiomatic system.)*) If an abstract axiom system does have a model, then such system is consistent.

Definition 2.14 (*Isomorphic*) If two models of the same axiom system can be proved as structurally equivalent, then they are isomorphic to each other.

An axiomatic system can have more than one model.

Definition 2.15 (*Categorical Axioms*) If all models of an axiom system are *isomorphic* then the axiom system is *categorical*.

Thus, for a categorical axiom system, there exists a model—the one and only interpretation in which its theorems are all true.

The qualities—*truth, logical necessity, consistency*, and *uniqueness* were considered as the base of classical Euclidean geometry. Till recently, it was accepted that Euclidean geometry is the only way to think about space. Now, the axiomatic systems are taken as the basis of geometry, and later all of the mathematics including the computational mathematics and algorithms.

Hilbert's definition of an axiomatic system lays the foundation of *theory* and verifies that this system satisfies three main properties: *independence*, *consistency*, and *completeness*. He proposed that just as in geometry, this kind of axiomatic analysis should be applied to other fields of knowledge, and in particular to physical theories. When we study any system of axioms as per Hilbert's perspectives, the focus of interest remained always on the *disciplines* themselves rather than on the axioms. The axioms are just a means to improve our understanding of the discipline, and not aimed to turn mathematics into a formally axiomatized game. For example, in the case of geometry, a set of axioms were selected in such a way that they reflected the basic manifestations of the intuition of space [4].

2.8.2 Axiomatics and Formalism

To understand the role of axioms, we will discuss the axioms of the set, as they are useful in reasoning and inferences. By analyzing the mathematical arguments, logicians become convinced that the notion of "set" is the most fundamental concept of mathematics. For example, it can be shown that the notion of an integer can be derived from the abstract notion of a set. Thus, in our world all the objects are sets, and we do not postulate the existence of any more primitive objects. To support this intuition, we can think our universe as all sets which can be built by successive collecting processes, starting from the empty set, and we allow the formation of infinite sets.

The first set of axioms for a general set theory was given by E. Zermelo in 1908, and later developed by A. Fraenkel, hence usually referred to as Zermelo-Fraenkel (ZF) set theory, the one we are most concerned. Another systems of axioms, which has only finitely many axioms, but is less natural, was developed by von Neumann, Bernays, and Gödel. The later is usually referred to as Gödel-Bernays (GB) set theory.

Following are some of the important axioms of ZF set theory [3, 8].

1. *Axioms of Extensibility.*

$$\forall x \forall y (\forall z (z \in x \leftrightarrow z \in y) \rightarrow x = y) \tag{2.7}$$

The above says that set is determined by its members. We can define the subsets as follows:

$$x \subseteq y \leftrightarrow \forall z(z \in x \rightarrow z \in y). \tag{2.8}$$

Also,

$$x \subset y \leftrightarrow x \subseteq y \wedge \neg x = y. \tag{2.9}$$

2. *Axiom of the Null set.*

$$\exists x \forall y(\neg y \in x). \tag{2.10}$$

The set defined by this axiom is the empty or null set and we denote it by ϕ.

3. *Axiom of Unordered Pairs.*

$$\forall x \forall y \exists z \forall w(w \in z \leftrightarrow w = x \vee w = y). \tag{2.11}$$

We represent the set z by $\{x, y\}$. Also, $\{x\}$ is $\{x, x\}$ and we put $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$. The set $\langle x, y\rangle$ is called *ordered pair* of x and y.
Using the above we can define a function as follows: a function is a set f of ordered pairs such that $\langle x, y\rangle, \langle x, z\rangle \in f \rightarrow y = z$. The set of x such that $\langle x, y\rangle \in f$ is called *domain*, and set of y is called *range*. We say, f maps in set u if the range of f is in u.

4. *Axiom of set Union.* It can be expressed as:

$$\forall x \exists y \forall z(z \in y \leftrightarrow \exists t(z \in t \wedge t \in x)). \tag{2.12}$$

The above says that y is union of all sets in x. Using the axiom Eq. 2.12, we can deduce that given x and y, there exists z, such that $z = x \cup y$, that is, $t \in z \leftrightarrow t \in x \vee t \in y$.
To motivate for the next axiom being described, if x is an integer, the successor of x will be defined as $x \cup \{x\}$. Then the "axiom of infinity" generates a set that contains all the integers and thus infinite.
5. *Axiom of Infinity*. It can be expressed as follows, and we understand that it is the principle of Induction.

$$\exists x(\phi \in x \wedge \forall(y \in x \rightarrow y \cup \{y\} \in x). \tag{2.13}$$

6. *Axiom of Power Set.* This axiom states that there exists for each x the set y for all the subsets of x.

$$\forall x \exists y \forall z(z \in y \leftrightarrow z \subseteq x). \tag{2.14}$$

If the axiom of extensionality is dropped, the resulting system may contain atoms, i.e., sets x such that $\forall y(\neg y \in x)$ yet the sets x are different. Indeed, one possible view is that integers are atoms and should not be taken as sets.

The first interesting axiom is the Axiom of Infinity. If we drop it, then we can take a model for ZF set of all finite sets which can be built from ϕ.

The axioms discussed above can be used to prove theorems, like, *mathematical induction*, *invertible functions*, and in fact another theorem of set theory, as well as the corollaries, but the same are not appropriate to cover here, and a curious reader is encouraged to refer the literature given in the bibliography.

2.9 Summary

Logic is used for valid deductions, and it avoids fallacy reasoning. Logic is also useful in *argumentation theory*—a study of how conclusions can be reached through logical reasoning, that is, whether the claims are soundly based on premises or not. Argumentation includes debate and negotiation, that are concerned with reaching mutually acceptable conclusions. The logic is used in proofs, games, and puzzles solutions. The arguments have the internal structure: comprising of premises, reasoning process, and consequence.

The most commonly used, Propositional logic, represents sentences using single symbols, called *atoms*, which are joined using the operators $\vee, \wedge, \neg, \rightarrow$ to create compound sentences. The sign of "$\rightarrow$" in $p \rightarrow q$ is *material implication*, also called *conditional join, if p then q*. Propositional logic expressions are called sentences/statements; these are interpreted as *true* or *false*. The sentences are called *wff*, and are defined recursively. A formula is a *syntactic* concept, which means whether or not a string of symbols is a formula.

The meaning (*semantics*) is associated with each formula by defining its *interpretation*, which assign a value *true* (*T*) or *false* (*F*) to every formula. Interpretation of a statement means the assignment of true values to its atoms. A set of truth values assigned to the atoms in a statement is called its *world*. Assignment of truth values to the atoms in a statement, which makes the statement true is called *model* of the statement.

The model checking is the process of truth-table enumeration, and is exponential on n, the number of atoms in a statement. The derivation can also be represented by a *derivation-tree* (*parse-tree*).

A propositional formula A is *satisfied iff* $v(A) = True$ for some interpretation v. A satisfying interpretation is called *model* for A. The formula A is called *valid*, denoted by $\models A$, iff $v(A) = True$ for all interpretations v. A sentence is logically true (valid) iff it is true under every interpretation. $\models \theta$ means that θ is valid.

A reasoning, in which addition of new knowledge may produce inconsistency in the knowledge base, is called *nonmonotonic reasoning*. As per the property of *monotonicity*, if $S \vdash \alpha$, and β is additional assertion, then $S \wedge \beta \vdash \alpha$. The *Nonmonotonic logic* is the study of those systems that do not satisfy the *monotonicity* property satisfied by all methods based on classical logic.

The reasoning pattern comprises *inference methods*: *modus ponens, modus tollens, syllogism*; and *Proof methods: resolution theorem, model checking, model checking,*

7. Kaushik S (2002) Logic and prolog programming. New Age International, New Delhi
8. Puppes P (1972) Axiomatic set theory. Dover, New York
9. Shankar N (2009) Automated deduction for verification. ACM Comput Surv 41(4):20:1. https://doi.org/10.1145/1592434.1592437

Chapter 3
First Order Predicate Logic

Abstract The first order predicate logic (FOPL) is backbone of AI, as well a method of formal representation of Natural Language (NL) text. The Prolog language for AI programming has its foundations in FOPL. The chapter demonstrates how to translate NL to FOPL in the form of facts and rules, use of quantifiers and variables, syntax and semantics of FOPL, and conversion of predicate expressions to clause forms. This is followed with unification of predicate expressions using instantiations and substitutions, compositions of substitutions, unification algorithm and its analysis. The resolution principle is extended to FOPL, a simple algorithm of resolution is presented, and use of resolution is demonstrated for theorem proving. The interpretation and inferences of FOPL expressions are briefly discussed, along with the use of Herbrand's universe and Herbrand's theorem. At the end, the most general unifier (mgu) and its algorithms are presented, and chapter is concluded with summary.

Keywords First Order Predicate Logic (FOPL) · Natural language · Quantifiers · Syntax and semantics of FOPL · Unification · Most general unifier · Resolution theorem · Theorem proving · Herbrand's universe · Herbrand's theorem

3.1 Introduction

This chapter presents a formulation of first-order logic which is best suited as a basic theoretical instrument—a computer based theorem proving program. As per the requirements of theory, an inference method should be *sound*—allows only logical consequences of premises deducible from the premises. In addition, it should be *effective*—algorithmically decidable whether a claimed application of the inference principle is really an application of it. When the inference principle is performed by computer, the complexity of the inference principle is not an issue. However, for more powerful principles, usage of combinatorial information processing for single application may become dominant.

The system described in the following is an inference principle—the *resolution principle*, is a machine-oriented rather than human-oriented system. Resolution principle is quite powerful in psychological sense also, as it obeys a single type of

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_3

inference, which is often beyond the ability of the human to grasp. In theoretical sense, it is a single inference principle that forms the complete system of first-order logic. However, this latter property is not of much significance, but it is interesting in the sense that no any other *complete system* of first-order logic is based on just one inference principle, if ever one tries to realize a device of introducing a logical axioms, or by a schema as an inference principle. The principle advantage of using the resolution is due to its ability that allows us to avoid any major combinatorial obstacles to efficiency, which used to be a serious problem in earlier theorem-proving procedures.

Learning Outcomes of this Chapter:

1. Translate a natural language (e.g., English) sentence into predicate logic statement. [Usage]
2. Apply formal methods of symbolic predicate logic, such as calculating validity of formula and computing normal forms. [Usage]
3. Use the rules of inference to construct proofs in predicate logic. [Usage]
4. Convert a logic statement into clause form. [Usage]
5. Describe the strengths and limitations of predicate logic. [Familiarity]
6. Apply resolution to a set of logic statements to answer a query. [Usage]
7. Implement a unification-based type-inference algorithm for a simple language. [Usage]
8. Precisely specify the invariants preserved by a sound type system. [Familiarity]

3.2 Representation in Predicate Logic

The first Order Predicate Logic (FOPL) offers formal approach to reasoning that has sound theoretical foundations. This aspect is important to mechanize the automated reasoning process where inferences should be correct and *logically sound*.

The statements of FOPL are flexible enough to permit the accurate representation of *natural languages*. The words—*sentence* or *well formed formula* will be indicative of predicate statements. Following are some of the translations of English sentences into predicate logic:

- English sentence: Ram is man and Sita is women.
  Predicate form: $man(Ram) \wedge woman(Sita)$
- English sentence: Ram is married to Sita.
  Predicate form: $married\,(Ram, Sita)$
- English sentence: Every person has a mother.
  The above can be reorganized as: For all x, there exists a y, such that if x is person then x's mother is y.
  Predicate form: $\forall x \exists y[person(x) \Rightarrow hasmother(x, y)]$

- English sentence: If x and y are parents of a child z, and x is man, then y is not man.
  $\forall x \forall y[[parents(x, z) \wedge parents(y, z) \wedge man(x)] \Rightarrow \neg man(y)]$

We note that predicate language comprises constants {Ram, Sita}, variables $\{x, y\}$, operators $\{\Rightarrow, \wedge, \vee, \neg\}$, quantifiers $\{\exists, \forall\}$ and functions/ predicates {married(x, y), person(x)}. Unless specifically mentioned, the letters $a, b, c, \ldots$ at the beginning of English alphabets shall be treated as constants to indicate names of *objects* and *entities*, and those at the end, i.e., $u, v, w, \ldots$ shall be used as variables or identifiers for objects and entities.

To indicate that an expression is universally true, we use the *universal quantifier* symbol $\forall$, meaning 'for all'. Consider the sentence "any object that has a feathers is a bird." Its predicate formula is: $\forall x[hasfeathers(x) \Rightarrow isbird(x)]$. Then certainly, $hasfeathers(parrot) \Rightarrow isbird(parrot)$ is true. Some expressions, although not always *True*, are *True* at least for some objects: in logic, this is indicted by 'there exists', and the *existential quantifier* symbol $\exists$ is used for this. For example, $\exists x[bird(x)]$, when *True*, this expression means that there is at least one possible object, that when substituted in the position of x, makes the expression inside the parenthesis as *True* [1].

Following are some examples of representations of knowledge FOPL.

Example 3.1 Kinship Relations.

$mother(namrata, priti)$.(That is, Namrata is mother of Preeti.)
$mother(namrata, bharat)$.
$father(rajan, priti)$.
$father(rajan, bharat)$.
$\forall x \forall y \forall z[father(y, x) \wedge mother(z, x) \Rightarrow spouse(y, z)]$.
$\forall x \forall y \forall z[father(y, x) \wedge mother(z, x) \Rightarrow spouse(z, y)]$.
$\forall x \forall y \forall z[mother(z, x) \wedge mother(z, y) \Rightarrow sibling(x, y)]$.

In above, the predicate $father(x, y)$ means x is father of y; $spouse(y, z)$ means y is spouse of z, and $sibling(x, z)$ means x is sibling of y. □

Example 3.2 Family tree.

Suppose that we represent "Sam is Bill's father" by *father(sam, bill)* and "Harry is one of Bill's ancestors" by *ancestor(harry, bill)*. Write a wff to represent "Every ancestor of Bill is either his father, his mother, or one of their ancestors".

$$\begin{aligned}\forall x \forall y[ancester(y, bill) \Rightarrow & (father(y, bill) \vee mother(y, bill)) \\ & \vee ((father(x, bill) \wedge ancester(y, x)) \\ & \vee ((mother(x, bill) \wedge (ancester(y, x))].\end{aligned}$$

Example 3.3 Represent the following sentences by predicate calculus wffs.

1. A computer system is intelligent if it can perform a task which, if performed by a human, requires intelligence.

$$\exists x[[(perform(human, x) \rightarrow requires(human, intelligence)) \\ \wedge (perform(computer, x)] \rightarrow intelligent(computer))]$$

2. A formula whose main connective is $\Rightarrow$ is equivalent to a formula whose main connective is $\vee$.

$$\forall x \forall y[(formula(x) \wedge mainconnective(x, '\Rightarrow')) \\ \wedge (formula(y) \wedge mainconnective(y, \vee)) \\ \rightarrow x \equiv y].$$

3. If a program cannot be told a fact, then it cannot learn that fact. $\forall x[(program(x) \wedge \neg told(x, fact)) \rightarrow \neg learn(x, fact)]$ □

Example 3.4 Blocks World.

Consider that there are physical objects, like—*cuboid*, *cone*, *cylinder* placed on the table-top, with some relative positions, as shown in Fig. 3.1. There are four blocks on the table: a, c, d are cuboid, and b is a cone. Along with these there is a robot arm, to lift one of the object having clear top.

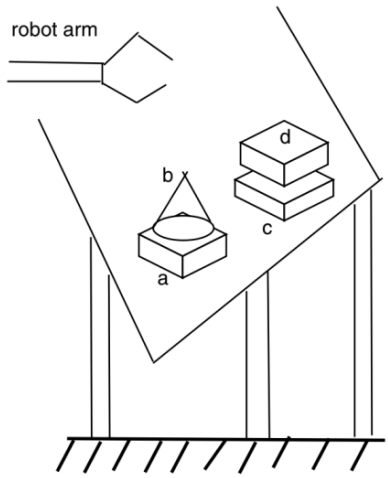


Fig. 3.1 Blocks world

than B. If A and B have same length, then A has the alphabetically earlier symbol in the first symbol position, at which A and B have distinct symbols.

- *Herbrand Universe*. It is set of ground terms associated with any set of S of clauses. Let F be the set of all function symbols which occur in clause set S. If F contains any function symbols of degree 0, then the functional vocabulary of S is F, otherwise, it is $\{a\} \cup F$, where a is a ground term. In this case, the Herbrand universe of S is set of all ground terms with only symbols of the functional vocabulary of S.
- *Models*. It is a set of ground literals having no complementary pair. If M is a Model and S is a set of *ground clauses*, then M is a model of S if, for all $C \in S$, C contains a member of M. In general, if S is any set of clauses, and H is the Herbrand Universe of S, then M is model of $H(S)$.
- *Satisfiability*. A set S is Satisfiable if there is a model of S, otherwise S is Unsatisfiable.

For every sentence S_1 of first order predicate logic there is always a sentence S_2 in *Clausal Form* which is satisfiable if and only if S_1 is also satisfiable. In other words, for every non-clause form sentence there is a logically equivalent clause form sentence. Due to this, all questions concerning to the *validity* or *satisfiability* of sentences in FOPL can be addressed to sentences in clausal form.

Procedure for obtaining clausal-form for any well-formed formula (*wff*) are discussed later in this chapter. In the above we have defined part of the syntax of predicate logic, which is concerned with the specification of well-formed formulas. The formalism we are going to use in the next section is based on the notions of *unsatisfiability* and *refutation* rather than upon the notions of *validity* and *proof*.

To work on the criteria of refutation and unsatisfirability, it is necessary to convert the given *wff* into clausal form.

To determine whether a finite set of sentences (S) of first-order predicate is satisfiable, it is sufficient to assume that each sentence in S is in clause form, and there is no existential quantifiers as the prefix to S. In addition, the matrix of each sentence in S is assumed to be a disjunction of formulas, each of which is either atomic formula or the negation of an atomic formula. Therefore, the syntax of S is designed such that the syntactical unit is a finite set of sentences in this special form, called *clause form*. Towards the end of conversion process, the quantifier prefix is omitted from each sentence, since it is necessary that universal quantifiers bind each variable in the sentence. The matrix of each sentence is simply a set of disjuncts and the order and multiplicity of the disjuncts are not important.

3.4 Conversion to Clausal Form

Following are the steps to convert a predicate formula into clausal-form [2].

1. Eliminate all the implications symbols using the logical equivalence: $p \rightarrow q \equiv \neg p \vee q$.

2. Move the outer negative symbol into the atom, for example, replace $\neg\forall x\ p(x)$ by $\exists x \neg p(x)$.
3. In an expression of nested quantifiers, existentially quantified variables not in the scope of universal quantifiers are replaced by constants. Replace $\exists x \forall y(f(x) \rightarrow f(y))$ by $\forall y(f(a) \rightarrow f(y))$.
4. Rename the variables if necessary. For example, in $\forall x(p(x)) \rightarrow q(x)$, rename second free variable x, as $\forall x(p(x) \rightarrow q(y))$.
5. Replace existentially quantified variables with *Skolem* functions; then eliminate corresponding quantifiers. For example, for $\forall x \exists y[\neg p(x) \vee q(y)]$, we obtain $\forall x[\neg p(x) \vee q(f(x))$. These newly created functions are celled *Skolem functions*, and the process is called *Skolemization*.
6. Move the universal quantifiers to the left of the equation. For example, substitute $\exists x[\neg p(x) \vee \forall y\ q(y)]$ by $\exists x \forall y[\neg p(x) \vee q(y)]$
7. Move the disjunctions down to the Literals, i.e., terms should be connected by conjunctions only, vertically.
8. Eliminate the conjunctions.
9. Rename the variables, if necessary.
10. Drop all the universal quantifiers, and write each term in a separate line.

The resulting sentence is a CNF, and suitable for inferencing using resolution.

Example 3.5 Convert the expression $\exists x \forall y[\forall z\ \ p(f(x), y, z) \Rightarrow (\exists u\ \ q(x, u) \wedge \exists v\ r(y, v))]$ to clausal form.

The steps discussed above are applied precisely, to get the clausal form of the predicate formula.

1. Eliminate implication.

$$\exists x \forall y[\neg \forall z\ p(f(x), y, z) \vee (\exists u\ q(x, u) \wedge \exists v\ r(y, v))]$$

2. Move negative symbols to the atom.

$$\exists x \forall y[\exists z \neg p(f(x), y, z) \vee (\exists u\ q(x, u) \wedge \exists v\ r(y, v))]$$

3. Replace existentially quantified variables not in the scope of universal quantifier to constants.

$$\forall y[\exists z \neg p(f(a), y, z) \vee (\exists u\ q(a, u) \wedge \exists v\ r(y, v))]$$

4. Rename variables (not required in this example.)
5. Replace existentially quantified variables that are functions of universal quantified variables, by Skolem functions:

$$\forall y[\neg p(f(a), y, g(y) \vee (q(a, h(y) \wedge r(y, l(y))]$$

6. Move ∀ to left is not required in this example.
7. Move disjunctions down to Literals.

$$\forall y[(\neg p(f(a), y, g(y)) \vee (q(a, h(y))) \wedge (\neg p(f(a), y, g(y)) \vee r(y, l(y)))]$$

8. Eliminate conjunctions.

$$\forall y[\neg p(f(a), y, g(y)) \vee (q(a, h(y)), (\neg p(f(a), y, g(y)) \vee r(y, l(y))]$$

9. Renaming variable is not required in this example.
10. Drop all universal quantifiers and write each term on separate line.

$$\neg p(f(a), y, g(y)) \vee (q(a, h(y)),$$
$$\neg p(f(a), y, g(y)) \vee r(y, l(y)).$$

Example 3.6 Convert the following wff to clause form.

$$(\forall x)(\exists y)\{[p(x, y) \Rightarrow q(y, x)] \wedge [q(y, x) \Rightarrow s(x, y)]\}$$
$$\Rightarrow (\exists x)(\forall y)[p(x, y) \Rightarrow s(x, y)]$$

For $[p(x, y) \Rightarrow q(y, x)] \wedge [q(y, x) \Rightarrow s(x, y)]$ by application of syllogism, it can be reduced to $[p(x, y) \Rightarrow s(x, y)]$. Thus, original expression reduces to:

$$= (\forall x)(\exists y)[p(x, y) \Rightarrow s(x, y)] \Rightarrow (\exists x)(\forall y)[p(x, y) \Rightarrow s(x, y)]$$
$$= \neg(\forall x)(\exists y)[p(x, y) \Rightarrow s(x, y)] \vee (\exists x)(\forall y)[p(x, y) \Rightarrow s(x, y)]$$
$$= (\exists x)\neg(\exists y)[p(x, y) \Rightarrow s(x, y)] \vee (\exists x)(\forall y)[p(x, y) \Rightarrow s(x, y)]$$
$$= (\exists x)(\forall y)\neg[p(x, y) \Rightarrow s(x, y)] \vee (\exists x)(\forall y)[p(x, y) \Rightarrow s(x, y)]$$
$$= (\forall y)\neg[p(a, y) \Rightarrow s(a, y)] \vee [p(a, y) \Rightarrow s(a, y)]$$
$$= [p(a, y) \wedge \neg s(a, y)] \vee [\neg p(a, y) \vee s(a, y)]$$
$$= T$$

3.5 Substitutions and Unification

The following definitions are concerned with the operation of *instantiation*, i.e, substitutions of terms for variables in the well-formed expressions and in sets of well-formed expressions [8].

Substitution Components

A substitution component is any expression of the form T/V, where V is any variable and T is any term different from V. The T can be any constant, variable, function, predicate, or expression.

Substitutions

A substitution is any finite set (possibly empty) of substitution components, none of the variables of which are same. If P is any set of terms, and the terms of the components of the substitution θ are all in P, we say that θ is a substitution over P. We write the substitution where components are $T_1/V_1, \ldots, T_k/V_k$ as $\theta = \{T_1/V_1, \ldots, T_k/V_k\}$, with the understanding that order of components is immaterial. We will use lowercase Greek letters θ, λ, μ denote substitutions.

Instantiations

If E is any function string of symbols, and $\theta = \{T_1/V_1, \ldots, T_k/V_k\}$ is any substitution, then the instantiation of E by θ is the operation of replacing each occurrence of variable V_i, $1 \leq i \leq k$, in E by term T_i. The resulting string denoted by $E\theta$ is called an instance of E by θ. That is, if E is the string $E_0 V_{i_1} E_1 \ldots V_{i_n} E_n$, $n \geq 0$, then $E\theta$ is the string $E_0 T_{i_1} E_1 \ldots T_{i_n} E_n$. Here, none of the substrings E_j of E contain occurrences of variables $V_1, \ldots, V_k$ after substitution. Some of E_j are possibly *null*, and each V_{i_j} is an occurrence of one of the variables $V_1, \ldots, V_k$.

3.5.1 Composition of Substitutions

If $\theta = \{T_1/V_1, \ldots, T_k/V_k\}$ and λ are any two substitutions, then the composition of θ and λ denoted by $\theta\lambda$ is union $\theta' \cup \lambda'$, defined as follows:

The θ' is set of all components $T_i\lambda/V_i$, $1 \leq i \leq k$, such that $T_i\lambda$ (λ substituted in θ) is different from V_i, and λ' is set of all components of λ whose variables are not among $V_1, \ldots, V_k$.

Within a given scope, once a variable is bound, it may not be given a new binding in future unifications and inferences. If θ and λ are two substitution sets, then the composition of θ and λ, i.e., $\theta\lambda$, is obtained by applying λ to the elements of θ and adding the result to λ.

Following examples illustrate two different scenario of composition of substitutions.

Example 3.7 Find out the composition of $\{x/y, w/z\}$, $\{v/x\}$, and $\{A/v, f(B)/w\}$.

Let us assume that $\theta = \{x/y, w/z\}$, $\lambda = \{v/x\}$ and $\mu = \{A/v, f(B)/w\}$. Following are the steps:

1. To find the composition $\lambda\mu$, A is is substituted for v, and v is then substituted for x. Thus, $\lambda\mu = \{A/x, f(B)/w\}$.
2. When result of $\lambda\mu$ is substituted in θ, we get composition $\theta\lambda\mu = \{A/y, f(B)/z\}$. □

Example 3.8 Find out the composition of $\theta = \{g(x, y)/z\}$, and $\lambda = \{A/x, B/y, C/w, D/z\}$.

By composition,

$$\begin{aligned}\theta\lambda &= \{g(x, y)/z\} \circ \{A/x, B/y\} \\ &= \{g(A, B)/z, A/x, B/y, C/w\}\end{aligned}$$

The $\{D/z\}$ has not been included in the resultant substitution set, because otherwise, there will be two terms for the variable z, one $g(A, B)$ and other D. □

One of the important property of substitution is that, if E is any string, and $\sigma = \theta\lambda$, then $E\sigma = E\theta\lambda$. It is straight forward to verify that $\varepsilon\theta = \theta\varepsilon = \theta$ for any substitution θ. Also, composition enjoys the associative property $(\theta\lambda)\mu = \theta(\lambda\mu)$, so we may omit the parentheses in writing multiple compositions of substitutions. The substitutions are not in general commutative; i.e., it is generally not the case that $\theta\lambda = \lambda\theta$, because for this $E\theta\lambda$ has to be equal to $E\lambda\theta$, which is not guaranteed. However, the composition has distributive property.

The point of the composition operation on substitution is that, when E is any string, and $\sigma = \theta\lambda$, the string $E\sigma$ is just the string $E\theta\lambda$, i.e., the instance of $E\theta$ by λ.

3.5.2 *Unification*

If E is any set of well-formed expressions and θ is a substitution, then θ is said to unify E, or to be a unifier of E, if $E\theta$ is a singleton. Any set of well-formed expressions which has a unifier is said to be unifiable [6].

In proving theorems using quantified variables, it is often necessary to "match" certain subexpressions. For example, to apply the combination of modus ponens and universal instantiation (Eq. 3.5) to produce "*mortal*(*socrates*)", it was necessary to find substitution $\{socrtaes/x\}$ for x that makes *man(x)* and *man(socrates)* equal (singleton).

Unification *algorithm* determines the substitutions needed to make two predicate expressions match. For this, all the necessary condition is that variables must be universally quantified. Unless the variables in an expression are existentially quantified, they are assumed to be universally quantified. This criteria allows us full freedom choosing the substitutions. The existentially quantified variables can be eliminated by substituting them with constants or with Skolem functions that makes the sentence true. For example, in sentence,

$$\exists x\ mother(x, jill),$$

we can replace x with a constant designating jill's mother, *susan*, to get:

$$mother(susan, jill);$$

and write unifier as $\{susan/x\}$.

3.6.1 Theorem Proving Formalism

It is a syntactic inference procedure, when applied to clauses, determines, if the satisfied set is unsatisfiable. Proof is similar to proof by *contradiction* and deduce [] (i.e., null). If for example, we have set of clauses (axioms) $C_1, C_2, \ldots, C_n$, and we want to deduce D, i.e., D is logical consequence of of $C_1, C_2, \ldots, C_n$. For this we add $\neg D$ to the set $\{C_1, C_2, \ldots, C_n\}$, then we show that set is unsatisfiable by deducing contradiction [7].

The process of deduction using resolution is given in Algorithm 3.1. Given two clauses C_1, C_2 with no variables in common, and if l_1 is a literal in C_1 and its complement literal l_2 is a literal in C_2, then l_1, l_2 can be dropped and disjunction C is obtained from the remaining part of C_1, C_2. The C is called resolvent of C_1, C_2.

Let $C_1 = \neg P \vee Q$, and $C_2 = \neg Q \vee R$, then following can be deduced through resolution,

$$\frac{P \Rightarrow Q, Q \Rightarrow R}{P \Rightarrow R} \tag{3.2}$$

equivalently,

$$\frac{(\neg P \vee Q), (\neg Q \vee R)}{\therefore (\neg P \vee R)}. \tag{3.3}$$

It can be easily verified that $(\neg P \vee Q) \wedge (\neg Q \vee R) \models (\neg P \vee R)$, hence $(\neg P \vee Q) \wedge (\neg Q \vee R) \Rightarrow (\neg P \vee R)$ is a valid statement. Thus, $\neg P \vee R$ is inference or the resolvent. Arriving to a proof by above is called proof by *refutation.*

Resolution says that if there are axioms of the form $\neg P \vee Q$ and there is another axiom of the form $\neg Q \vee R$, then $\neg P \vee R$ logically follows; called the *resolvent.* Let us see why it is so? When $\neg P \vee Q$ is True, then either $\neg P$ is True or Q is True. For other expression, when $\neg Q \vee R$ is True, then either $\neg Q$ is True or R is True. Then we can say that $\neg P \vee R$ is certainly True. This can be generalized to two expressions, when we have any number of expressions, but two must be of opposite signs.

3.6.2 Proof by Resolution

To prove a theorem, one obvious strategy is to search forward from the axioms, using sound rules of inference. We try to prove a theorem by refutation. It requires to show that negation of a theorem cannot be True. The steps for a proof by resolution are:

1. Assume that negation of the theorem is True.
2. Try to show that axioms and assumed negation of theorem, together are True, which cannot be True.
3. Conclude that above leads to contradiction.
4. Conclude that theorem is True because its negation cannot be True.

To apply the resolution rule,

1. Find two sentences that contain the same literal, one in its positive form and one in its negative form, like,

$$CNF : summer \vee winter, \neg winter \vee cold,$$

2. use the *resolution* rule to eliminate the complement literals from both sentences to get,

$$CNF : summer \vee cold.$$

The Algorithm 3.1 is an algorithm for theorem proving through resolution-refutation, where α is the theorem to be proved, and β is set of axioms, both of these are input to the algorithm. All the inputs to algorithm are in the clause form. The algorithm returns "true" if the theorem is true, else returns "False".

Algorithm 3.1 Algorithm-Resolve(Input: α, β)

```
Γ = β ∪ {¬α}
while there is a resolvable pair of clauses C_i, C_j ∈ Γ do
    C = resolve(C_i, C_j)
    if C = NIL then
        return "Theorem α is true"
    end if
    Γ = Γ ∪ {C}
end while
Report that theorem is False
```

3.7 Complexity of Resolution Proof

The question is, how you can be so clever to pickup the right clauses to resolve? The answer is that you take advantage of two ideas:

1. You can be sure that every resolution involves the *negated theorem*, directly or indirectly.
2. You know where you are and where you are going, hence you can compute the difference to help you proceed with your intuition for selection of clauses.

Consider there are total n clauses, $c_1 \ldots c_n$. We can try to match c_1 with $c_2 \ldots c_n$, and in next level c_2 is matched with $c_3 \ldots c_n$, and so on. This results to breadth first search (BFS). Consider that resolvents generated due to this matching are $c'_1 \ldots c'_m$. Next all the newly generated clauses are matched with the original, and then they are merged into the original. This process is repeated until contradiction is reached, showing that theorem is proved. Since, the entire set of clauses are compared, the

proof is bound to result, if it exists, at all. This gives completeness to the resolution proof.

The other alternative is, nodes which are farther and farther away are matched before those which are closer to the root. The c_1 is matched with first child c_2 out of $c_2 \ldots c_n$. Then c_2 is matched with its first child generated, and so on, resulting to the search process called DFS (depth first search).

However, the above both are brute-force algorithms, and are complex. The other methods are heuristic based. In fact, there is difficulty to express your concepts required in pure logic. One of the approaches is to use the clauses having smallest number of literals. Another, to use negated clauses.

The resolution search strategies are subject to the exponential-explosion problem. Due to this, those proofs which require long *chains of inferences*, will be exponentially expensive in time.

All resolution search strategies are subject to a version of *halting problem*, for search is not guaranteed to terminate unless there actually is a proof. In fact, all complete proof procedures for the first order predicate calculus are subject to halting problem. Complete proof procedures are said to be *semi-decidable*, because they are generated to tell you whether an expression is a theorem, only if the expression is indeed a theorem.

Theorem proving is suitable for certain problems, but not for all problems, due to the following reasons:

1. Complete theorem proving requires search, and search is inherently exponential,
2. Theorem provers may not help you to solve practical problems, even if they do their work instantaneously.

3.8 Interpretation and Inferences

A FOPL statement is made of predicates, arguments (constants or variables), functions, operators, and quantifiers. Interpretation is process of assignment of truth values (True/False) to subexpressions and atomic expressions, and computing the resultant value of any expression/statement. A statement or expression in predicate logic is also called *wwf* (well formed formula).

Consider the interpretation of predicate formula:

$$\forall x[bird(x) \rightarrow flies(x)]. \tag{3.4}$$

To find out the *satisfiability* of the formula (3.4), we need to substitute (*instantiate*) a value for x (an instance of x) and check if $flies(x)$ is true. Until, that x is found, it may require instantiation with large number of values. Similarly, to check if the Eq. (3.4) is valid, it may require infinitely large number of values in the domain of

x to be verified. If any one of that makes the formula false, the formula is not valid. Thus, checking of satisfiability as well as validity of a formula is predicate logic are complex process. The approach of *truth table* and *tableau* method we discussed in the previous chapter are applicable here also.

Given a predicate sentences of m number of predicates each having one argument, and domain size of all the arguments is n, in the worst case it will require total n^m substitutions to test for satisfiability, as well as for validity checking. However, a sentence of m propositions will require in the worst case only 2^m substitutions. Hence, satisfiability checking in predicate sentences is much more complex than that in proposition logic. It equally applies with expressions having existential quantifiers, like, $\exists x[bird(x) \rightarrow flies(x)]$.

Thus, it is only the proof methods, using which *logical deductions* can be carried out in realistic times.

Example 3.11 Given, "All men are mortal" and "Socrates is man", infer using predicate logic, that "Socrates is mortal".

The above statement can be written in predicate logic as:

$$\begin{aligned} &\forall x[man(x) \Rightarrow mortal(x)], \\ &man(socrates). \end{aligned} \tag{3.5}$$

Using a rule called *universal instantiation*, a variable can be instantiated by a constant and universal quantifier can be dropped. Hence, from (3.5) we have,

$$\begin{aligned} &man(socrates) \Rightarrow mortal(socrates), \\ &man(socrates). \end{aligned} \tag{3.6}$$

Using the rule of *modus ponens* on (3.6) we deduce "$mortal(socrates)$". It is also *logical consequence*. If $\Gamma = \{[man(socrates) \Rightarrow mortal(socrates)] \wedge man(socrates)\}$, and $\alpha = mortal(socrates)$, then we can say that $\Gamma \vdash \alpha$.

The set of formulas Γ is called knowledge base. To find out the result for the query "Who is man?", we must give the query

$$?man(X).$$

in Prolog (to be discussed later), which will match (called *unify* or *substitute*) $man(X)$ with $man(socrates)$ with a unification set, say, $\theta = \{socrates/X\}$. The substitution which returns $man(socrates)$ is represented by $man(X)\theta$. □

Example 3.12 Prolog Program.

The sentence in Eq. (3.5) will appear in prolog as,

$$mortal(socrates) \text{ :- } man(socrates).$$
$$man(socrates).$$

Here, the sign ':-'is read as 'if'. The subexpression before the sign ':-'is called 'head'or *procedure name* and the part after ':-'is called *body* of the *rule*. The sentence (3.6) can also be written in a *clause form* (to be precise, in *Horn clause* form) as,

$$mortal(socrates) \vee \neg man(socrates).$$
$$man(socrates). \tag{3.7}$$

3.8.1 Herbrand's Universe

Defining an operational semantics for a programming language is nothing but to define an implementation independent interpreter for it. In case of predicate logic, the proof procedure itself behaves like an interpreter. The Herbrand's Universe and Herbrand's base play an important role in interpretation of predicate language. In the following, we define the Herbrand's Universe and Herbrand's Base.

Definition 3.1 (*Herbrand's Universe*) In a predicate logic program, a Herbrand Universe **H**, is a set of ground terms that use only function symbols and constants.

Definition 3.2 (*Herbrand's Base*) A set of atomic formulas formed by predicate symbols in a program, is called Herbrand's base. The additional condition is that, arguments of these predicate symbols are in the Herbrand Universe.

For a predicate program, the Herbrand universe and Herbrand base are countably infinite if the predicate program contains a function symbol of positive arity. If the arity of function symbols is zero, then both the haerbrand's universe and base are finite [3].

In special cases, when resolution proof is used on FOPL, it reduces the expressions to propositional form. If the set of clauses is **A**, its Harbrand's universe is set of all the ground terms formed using only the function symbols and constants in **A**. For example, if **A** has constants a, b, and a unary function symbol f, then the Herbrand universe is the infinite set:

$$\{a, b, f(a), f(b), f(f(a)), f(f(b)), f(f(f(a)), \ldots\}.$$

The Herbrand's base of **A** is the set of all ground clauses $c\theta$ where $c \in \mathbf{A}$ and θ is a substitution that assigns the variables in c to terms in the Herbrand's universe.

3.8.2 Herbrand's Theorem

Herbrands theorem is a fundamental theorem based on mathematical logic, that permits a certain type in reduction from FOPL to propositional logic [4].

In its simplest form, the Herbrand's theorem states that a formula of first-order predicate logic $\exists x\, A$, where A is quantifier free, is provable if and only if there exist ground terms $M_1, \ldots, M_n$ such that,

$$\models A[x := M_1] \vee \cdots \vee A[x := M_n]. \tag{3.10}$$

When using the classical formulation, the Herbrand's theorem relates the validity of a first-order formula in *Skolem prenex form*[1] to the validity of one of its Herbrand extensions. That means, the formula $\forall x_1 \ldots \forall x_n \psi(x_1 \ldots, x_n)$ is valid if, and only if, $\bigwedge_i^m \psi(t_{i,1}, \ldots, t_{i,n})$ is valid for some $m \geq 1$ and some collection of ground Herbrand terms $t_{i,j}$.

Since it is possible that every classical first-order formula can be reduced to this Skolem prenex form through the Skolemization while preserving its satisfiability, the Herbrand's theorem provides a way to reduce the question of validity of first-order formulas to propositional logic formula.

However, the required Herbrand's extension and the terms $t_{i,j}$ cannot be computed recursively (for otherwise first-order logic would be decidable), this result is highly useful for the automated reasoning as it gives a way to some highly efficient proof methods such as *resolution* and the *resolution refutation*.

Theorem 3.1 *A closed formula F in Skolem form is satisfiable if and only if it has a Herbrand model.*

Proof If the formula has a Herbrand model then it is satisfiable. For the other direction let $\mathscr{A} = (U_{\mathscr{A}}, I_{\mathscr{A}})$ be an arbitrary model of F. We define a Herbrand structure $\mathscr{B} = (U_{\mathscr{B}}, I_{\mathscr{B}})$ as follows:

Universe: $U_{\mathscr{B}} = D(F)$

Functional Symbols: $f^{\mathscr{B}}(t_1, t_2, \ldots, t_n) = f(t_1, t_2, \ldots, t_n)$

Predicate Symbols: $(t_1, \ldots, t_n) \in P^{\mathscr{B}}$ iff $\mathscr{A}(t_1), \ldots, \mathscr{A}(t_n) \in P^{\mathscr{A}}$.

Claim: $\mathscr{B}$ is also a model of F.

We prove a stronger assertion: For every closed form G in Skolem form such that G^* only contains atomic formulas of F^* : *if* $\mathscr{A} \models G$ then $\mathscr{B} \models G$.

By induction on the number n of universal quantifiers of G.

Basis ($n = 0$). Then G has no quantifiers at all.

It follows $\mathscr{A}(G) = \mathscr{B}(G)$, this proves the theorem. □

To perform reasoning with the Herbrand base, the unifiers are not required, and we have a *sound* and *complete* reasoning procedure, which is guaranteed to terminate. The idea used in this approach is: Herbrand's base will typically be an infinite set of propositional clauses, but it will be finite when Herbrand's universe is finite (there

[1] A string of quantifiers followed by a quantifier-free part, e.g., $\forall x_1 \ldots \forall x_n \psi(x_1 \ldots, x_n)$.

is no function symbols and only finitely many constants appear in it). Sometimes we can keep the universe finite by considering the type of the arguments (say t) and values of functions (f), and include a term like $f(t)$ in the universe only if the type of t is appropriate for the function f. For example, $f(t)$ may be, $birthday(john)$, which produces a date.

3.8.3 The Procedural Interpretation

It is easy to procedurally interpret the sets of clauses, say, **A**, which contain at most one positive literal per clause. However, along with this any number of negative literals can also exist. Such sets of clauses are called *Horn sentences* or *Horn Clauses* or simply clauses. We distinguish three kinds of *Horn clauses* [3].

1. '[]'the *empty clause*, containing no literals and denoting the truth value *false*, is interpreted as a *halt statement*.
2. $\bar{B}_1 \vee \cdots \vee \bar{B}_n$, a clause consisting of no positive literals and $n \geq 1$ negative literals, is interpreted as a *goal statement*. Note that goal statement is negated and added into the knowledge base to obtain the proof through *resolution refutation*.
3. $A \vee \bar{B}_1 \vee \cdots \vee \bar{B}_n$, a clause consisting of exactly one positive literal and $n \geq 0$ negative literals is interpreted as a *procedure declaration* (i.e., rule in Prolog program). The positive literal A is the *procedure name* and the collective negative literals are the *procedure body*. Each negative literal B_i, in the procedure body is interpreted as a *procedure call*. When $n = 0$ the procedure declaration has an empty body and interpreted as an unqualified assertion of fact.

In the procedural interpretation, a set of procedure declarations is a program. Computation is initiated by an *initial goal* statement, which proceeds by using declared procedures to derive new goal statements (*subgoals*) B_is from old goal statements, and terminates on the derivation of the halt statement. Such derivation of goal statements is accomplished by *resolution*, which is interpreted as *procedural invocation*.

Consider that, a selected procedure call $\bar{A}_1$ inside the body of a goal statement as,

$$\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{A}_i \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n \tag{3.11}$$

and a procedure declaration is given as,

$$A' \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m, m \geq 0. \tag{3.12}$$

Suppose, the name of procedure A' matches with the procedure call A_i, i,e., some substitution θ of terms for variables makes A_i and A' identical. In such a case, the resolution derives a new goal statement by disjunction formulas (3.11) and (3.12) as given below, subject to substitution θ.

$$(\bar{A}_1 \vee \cdots \vee \bar{A}_{i-1} \vee \bar{B}_1 \vee \cdots \vee \bar{B}_m \vee \bar{A}_{i+1} \vee \cdots \vee \bar{A}_n)\theta. \tag{3.13}$$

In general, any *derivation* can be regarded as a computation, and any *refutation* (i.e. derivation of []) can be regarded as a successfully terminating computation. It is to be noted that, only goal oriented resolution derivations correspond to the standard notion of computation.

Thus, a goal-oriented derivation, from an initial set of Horn clauses **A** and from an initial goal statement (computation) $C_1 \in \mathbf{A}$, is a sequence of goal statements $C_1, \ldots, C_n$. So that each C_i contains a single selected procedure call and C_{i+1}, obtained from C_i by procedure invocation relative to the selected procedure call in C_i, using a procedure declaration in **A**.

For the implementation of above, one method is *model elimination*. Using this, the selection of procedure calls is governed by the last-in/first-out rule: a goal statement is treated as a stack of procedure calls. The selected procedure call must be at the top of the stack. The new procedure calls which by procedure invocation replace the selected procedure call are inserted at the top of the stack. This would result to a *depth-first search* procedure.

The Predicate logic is a *nondeterministic* programming language. Consequently, given a single goal statement, several procedure declarations can have a name which matches the selected procedure call. Each declaration gives rise to a new subgoal statement. A *proof procedure* which sequences the generation of derivations in the search for a refutation behaves as an *interpreter* for the program incorporated in the initial set of clauses.

The following example explains how to use procedural interpretation to append two given lists.

Example 3.13 Appending two lists [3].

Let a term $cons(x, y)$ is interpreted as a list whose first element, the *head*, is x and whose *tail* y is the rest of the list. The constant *nil* denotes the empty list. The terms u, x, y, and z are variables. The predicate *append(x,y,z)* denotes the relationship: z is obtained by appending y to x.

The following two clauses constitute a program for appending two lists.

$$append\,(nil, x, x). \tag{3.14}$$

$$append\,(cons(x, y), z, cons(x, u)) \vee \overline{append}\,(y, z, u). \tag{3.15}$$

The clause in statement (3.14) represents halt statement. In (3.15) there is a positive literal for procedure name, and negative literal(s) for the procedure body, both together it is procedure declaration. The positive literal means, if $cons(x, y)$ is appended with z, it results to x appended with u such that u is, y appended with z. The later part is indicated by the complementary (negative) term. Note that clausal expression (3.15) is logically equivalent to the expression $append\,(y, z, u) \rightarrow append\,(cons(x, y), z, cons(x, u))$.

Suppose it is required to compute the result of appending list $cons(b, nil)$ to the list $cons(a, nil)$. Therefore, the goal statement is,

$$append(cons(a, nil), cons(b, nil), v), \tag{3.16}$$

where v (a variable) and a, b (constants), are the "atoms" of the lists. To prove using resolution, we add the negation of the goal,

$$\overline{append}(cons(a, nil), cons(b, nil), v), \tag{3.17}$$

into the set of clauses. The program is activated by this *goal statement* to carry out the append operation. With this goal statement the program is deterministic, because only one choice is available for matching. The following computation follows with a goal directed theorem prover as interpreter: The goal statement,

$$C_1 = \overline{append}(cons(a, nil), cons(b, nil), v). \tag{3.18}$$

matches with the clause statement (3.15) with matchings: $x = a, y = nil, z = cons(b, nil)$. Also, $v = cons(x, u) = cons(a, u)$, i.e., there exists a unifier $\theta_1 = \{cons(a, w)/v\}$. The variable u has been renamed as w. On unifying clauses (3.18) and (3.15), the next computation C_2 is:

$$C_2 = \overline{append}(nil, cons(b, nil), w)\theta_1. \tag{3.19}$$

Keeping θ_1 accompanying the predicate in above is for the purpose that if C_2 is to be unified with some other predicate, the matching of the two shall be subject to the same unifier θ_1.

As next matching, C_2 can be unified with (3.14) using a new unifier $\theta_2 = \{cons(b, nil)/w\}$ to get next computation,

$$C_3 = []\theta_2. \tag{3.20}$$

The result of the computation is value of v in the substitution, i.e.,

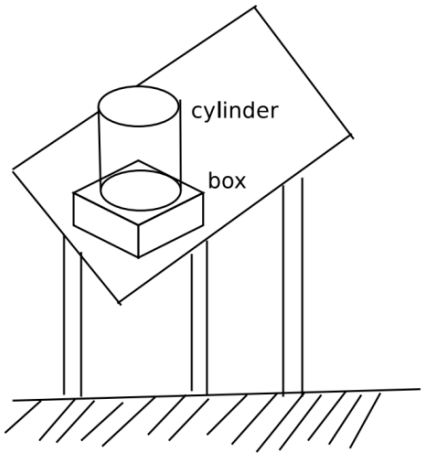
$$\begin{aligned} v &= cons(a, u) \\ &= cons(a, w) \\ &= cons(a, cons(b, nil)). \end{aligned}$$

The above result is equal to goal: $append(cons(a, nil), cons(b, nil), v)$. □

Example 3.14 Theorem proving using resolution-refutation.

Following axioms are about the observed block relationship shown in Fig. 3.3, which are already in clausal form.

Fig. 3.3 Objects on table



$$on(cylinder, box).$$

$$on(box, table).$$

It is required to be shown that object *cylinder* is above table, i.e., $above(cylinder, table)$, given the the following rules:

$\forall x \forall y[on(x, y) \rightarrow above(x, y)]$, and
$\forall x \forall y \forall z[above(x, y) \wedge above(y, z) \rightarrow above(x, z)]$.

After we have gone through the procedure for conversion to clausal form, the above axioms are transformed into clause forms.

$\neg on(u, v) \vee above(u, v)$.
$\neg above(x, y) \vee \neg above(y, z) \vee above(x, z)$.

The expression to be proved is "above(cylinder, table)"; its negation is $\neg above(cylinder, table)$. Let us list all the clauses systematically.

(1) $\neg on(u, v) \vee above(u, v)$.
(2) $\neg above(x, y) \vee \neg above(y, z) \vee above(x, z)$.
(3) $on(cylinder, box)$.
(4) $on(box, table)$.
(5) $\neg above(cylinder, table)$.

Now, we manually run the Algorithm 3.1 on the clauses (1)–(5), as well as those which would created new, to unify them according to unification Algorithm 3.2, until we reach to a null resolvent.

First we resolve clauses (2) and (5) and bind x to '*cylinder*'and z to '*table*'. Applying the resolution, we get resolvent (6). Unifier for this is $\{cylinder/x, table/z\}$.

$$\text{Then } ES = \{p(fred), p(fred)\}$$
$$\text{and } GS^{'} = \{fred/x, fred/y\}$$
$$\text{and therefore } EGS^{'} = \{p(fred), p(fred)\} = ES.$$

So, given a unifier, you can always create a more general unifier. When both of these unifiers are composed and instantiate the original expression E, you get the same instance as it was obtained with the earlier unifier.

3.9.1 Lifting

It is necessary to show that the general resolution principle is *sound* and *complete*. However, a technical difficulty is the completeness of the proof. Using the Herbrand's theorem and semantic trees, we can prove that there is a ground *resolution refutation* of an unsatisfiable set of clauses. But, this cannot be generalized as a proof for general resolution, because the concept of semantic trees cannot be generalized. Why it cannot be generalized, is due to the variables, which give rise to potentially infinite number of elements in the Herbrand's base, as we will show it shortly.

Fortunately, there is a technique, called, "Lifting", to prove completeness of a theorem. Following are the steps for lifting:

1. first prove the *completeness* of the system for a set of *ground* classes, then,
2. as a second step, lift the proof to non-ground case.

Example 3.16 Infinite inferences.

Let us assume that there are two non-ground clauses: 1. $p(u, a) \vee q_1(u)$ and, 2. $\neg p(v, w) \vee q_2(v, w)$. If the signature pattern contains function symbols, then these clauses have infinite *set* of instances, as follows:

$\{p(r, a) \vee q_1(r) \mid r \text{ is ground}\}$.
$\{\neg p(s, t) \vee q_2(s, t) \mid s, t \text{ are ground}\}$.

We can resolve above instances if and only if $r = s$ and $t = a$. Then we can apply the resolution refutation and obtain the inference given in the denominator of Eq. (3.21), which are infinite, due to variable s.

$$\frac{p(s, a) \vee q_1(s), \neg p(s, a) \vee q_2(s, a)}{q_1(s) \vee q_2(s, a)} \tag{3.21}$$

□

The above difficulty can be overcome by taking a ground resolution refutation and "lifting" it to a more abstract general form.

The lifting is an idea to represent infinite number of ground inferences of the form given in Eq. (3.21) by a single non-ground inferences:

$$\frac{p(u, a) \vee q_1(u), \neg p(v, w) \vee q_2(v, w)}{q_1(v) \vee q_2(v, a)}$$

This lifting can be done using most general unifier, we will be discussing shortly.

Example 3.17 Find out the Lifting for following clauses:

$C_1 = p(u) \vee p(f(v)) \vee p(f(w)) \vee q(u)$
$C_2 = \neg p(f(x)) \vee \neg p(z) \vee r(x)$

Using the substitution $\theta = \{f(a)/u, a/v, a/w, a/x, f(a)/z\}$, the above clauses become $C_1' = p(f(a)) \vee q(f(a))$, and $C_2' = \neg p(f(a)) \vee r(a)$. Using C_1' and C_2', it resolves to $C' = q(f(a)) \vee r(a)$. The lifting claims that there is a clause $C = q(f(x)) \vee r(x)$ which is resolvent for clauses C_1 and C_2, such that clause C' is ground instance of C. This can be realized using the *unification algorithm* to obtain a most general unifier (mgu) of clauses C_1 and C_2, the latter two clauses resolves to C, as $\{f(x)/u, x/v, x/w, f(x)/z\}$.

3.9.2 Unification Algorithm

A unification algorithm is central to most of the theorem-proving systems. This algorithm receives as input a pair of expressions, and returns as output a set of substitutions (assignments) that make the two expressions look identical.

The unification algorithm recursively compares the structures of the clauses to be matched, working across element by element. The criteria is that,

1. the matching individuals, functions, and predicates must have the same names,
2. the matching functions and predicates must have the same number of arguments, and
3. all bindings of variables to values must be consistent throughout the whole match.

To unify two atomic formulas in an expression **A**, we need to understand the *disagreement* set.

Definition 3.8 Disagreement Set.

If **A** is any set of well-formed expressions, we call the set D the disagreement set of **A**, whenever D is the set of all well-formed subexpressions of the well-formed expressions in **A**, which begin at the first symbol position at which not all well-formed expressions in **A** have the same symbol. □

Example 3.18 Find out the disagreement set for given set of atoms.

Let the string is, $\mathbf{A} = \{p(x, h(x, y), y), p(x, k(y), y), p(x, a, b)\}$, having three predicate expressions. The disagreement set for **A** is,

$$D = \{h(x, y), k(y), a\}. \tag{3.22}$$

Once the disagreement is resolved through unification for this this symbol position, there is no disagreement at this position. The process is repeated for the new first symbol position at which all wffs in **A** do not have same symbol, and so on, until **A** becomes a singleton.

Evidently, if **A** is nonempty and is not a *singleton* (a set with exactly one element), then the disagreement set of **A** is not a singleton and nonempty. Also, if θ unifies **A**, and **A** is not singleton, the θ unifies the disagreement set **A**. □

For **A** to be a finite nonempty set of well-formed expressions for which the substitution Algorithm 3.2 terminates with "return $\sigma_{\mathbf{A}}$", the substitution $\sigma_{\mathbf{A}}$ available as output of the unification algorithm is called the most general unifier (mgu) of **A**, and **A** is said to be most generally unifiable [8, 9].

Algorithm 3.2 Unification-Algorithm (Input: **A**, Output: $\sigma_{\mathbf{A}}$)

1: Set $\sigma_0 = \varepsilon, k = 0$
2: **while** true **do**
3: **if** $A\sigma_k$ is a singleton **then**
4: Set $\sigma_{\mathbf{A}} = \sigma_k$
5: terminate
6: **end if**
7: Let U_k be the earliest and V_k be the next earliest element in the disagreement set D_k of $\mathbf{A}\sigma_k$ (see Eq. 3.22)
8: **if** V_k is a variable, and does not occur in U_k **then**
9: set $\sigma_{k+1} = \sigma_k\{U_k/V_k\}$,
10: $k = k + 1$
11: **else**
12: (**A** is not unifiable)
13: exit.
14: **end if**
15: **end while**

Through manually running the Algorithm 3.2 for the disagreement set in (3.22), stepwise computation for σ_k is as follows:

For $k = 0$, and $\sigma_0 = \varepsilon$,

$$\begin{aligned} \sigma_{k+1} &= \sigma_k\{k(y)/h(x, y)\} \\ \Rightarrow \sigma_1 &= \{k(y)/h(x, y)\}. \end{aligned}$$

which, in the next iteration becomes,

$$\begin{aligned} \sigma_2 &= \sigma_1\{a/k(y)\} \\ &= \{k(y)/h(x, y)\}\{a/k(y)\}. \end{aligned}$$

The same process is repeated for the disagreement set of 3rd argument in **A**, which results to substitution set as $\{b/y\}$.

$$\begin{aligned}\sigma_3 &= \sigma_2\{b/y\} \\ &= \{k(y)/k(x,y)\}\{a/k(y)\}\{b/y\}.\end{aligned}$$

On substituting these, we have,

$$\mathbf{A} = \{p(x,a,b), p(x,a,b), p(x,a,b\}.$$

which is a singleton, and σ_3 is mgu.

For obtaining the unifier σ_k, the necessary relation required between U_k and V_k is, V_k has to be a variable, and U_k can be a constant, variable, function, or predicate. V_k may even be a predicate or function with variable.

The Algorithm 3.2 always terminates for finite nonempty set of well-formed expressions, otherwise it would generate an infinite sequence of $A, A\sigma_1, A\sigma_2, \ldots$, each of which is a finite nonempty sets of well-formed expressions, with the property that each successive set contains one less variable than its predecessor. However, this is impossible because A contains only finitely many distinct variables.

The Algorithm 3.2 runs in $O(n^2)$ time on the length of the terms, and an even better. However, there exists more complex, but linear time algorithms for same. Because, most general unifiers (mgus) greatly reduce the search, and can be calculated efficiently, almost all Resolution-based systems implementations are based on the concept of mgus.

3.10 Unfounded Sets

In the well-founded semantics, the unfounded sets provide the basis for negative conclusions. Let there is a program **P** (set of rules and facts in FOPL), its associated Herbrand base is H, and suppose its partial interpretation is I. Then, some $A \subseteq H$ is called an *unfounded set* of **P** with respect to the interpretation I, with following condition: for each instantiated rule $R \in \mathbf{P}$, at least one of the following holds: (In the rules **P** , we assume that p is a head, and q_i are the corresponding subgoals.)

1. Some positive / negative subgoal q_i of the body of the rule is false in the interpretation I,
2. Some positive subgoals q_i of the body occurs in the unfounded set A.

For rule R with respect to I, a literal that makes conditions 1 or 2 above true is called *witness of unusability*.

Intuitively, the interpretation I is intended model of **P**. The rules that satisfy condition 1 cannot be used for further derivations because their hypotheses are already known to be false.

The condition 2 in above, called *unfoundedness condition*, states that all the rules which might still be usable to derive something in A, should have an atom (i.e., a fact) in A as true. In other words, there is no single atom in A, that can be established to be true by the rules of $\mathbf{P}$ (as per the knowledge of interpretation I). Therefore, if we infer that some or all atoms in A are false, there is no way available later, using that we could infer that an atom is true [4].

Hence, the well-founded semantics uses conditions 1 and 2 to draw negative conclusions, and simultaneously infers all atoms in A to be false. The following example demonstrates the construction of unfounded set from the set of rules and facts.

Example 3.19 Unfounded set.

Assume that we have a program in predicate logic with instantiated atoms.

$$
\begin{aligned}
&p(c).\\
&q(a) \leftarrow p(d).\\
&p(a) \leftarrow p(c), \neg\, p(b).\\
&q(b) \leftarrow q(a).\\
&p(b) \leftarrow \neg\, p(a).\\
&p(d) \leftarrow q(a), \neg\, q(b).\\
&p(d) \leftarrow q(b), \neg\, q(c).\\
&p(e) \leftarrow \neg\, p(d).
\end{aligned}
$$

From above rules, we see that $A = \{p(d), q(a), q(b), q(c)\}$ is an unfounded set with respect to ϕ (null set). Since A is unfounded, its subsets are also unfounded. The component, $\{q(c)\}$ is unfounded due to condition (1), because there is no rule available to establish its truth. The set $\{p(d), q(a), q(b)\}$ is unfounded due to condition (2) (their subgoals or body appear in unfounded set.

There is no way available to establish $p(d)$ without first establishing $q(a)$ or $q(b)$. In other words, whether we can establish $\neg q(b)$ to support the first rule for $p(d)$ is irrelevant as far as determination of unfoundedness is the concern.

Interestingly, there is no way available to establish $q(a)$ in the absence of first establishing $p(d)$, and also there is no way available to establish $q(b)$ without first establishing $q(a)$. Further, $q(c)$ can never be proven. We note that among $p(d)$, $q(a)$, and $q(b)$ as goals, none can be proved without the other two or their negation as subgoals.

The pair $p(a), p(b)$, even though they depend on each other, but does not form an unfounded set due to the reason that the only dependence is through negation. Hence, it can be concluded that the any attempt for proof of $p(a)$ and $p(b)$ will fail, but this claim is faulty.

The difference between sets $\{p(d), q(a), q(b)\}$ and $\{p(a), p(b)\}$ is as follows: declaring any of $p(d)$, $q(a)$, or $q(b)$ false (unfounded), does not create a proof that any other element of the set is true.

8. Consider a set of statements of FOPL that uses two 1-place predicates: $Large$ and $Small$. The set of object constants are a, b. Find out all possible models for this program. For each of the following sentences find out the models in which each of the sentence becomes true.

   a. $\forall x\ Large(x)$.
   b. $\forall x\ \neg Large(x)$.
   c. $\exists x\ Large(x)$.
   d. $\exists x\ \neg Large(x)$.
   e. $Large(a) \wedge Large(b)$.
   f. $Large(a) \vee Large(b)$.
   g. $\forall x\ [Large(x) \wedge Small(x)]$.
   h. $\forall x\ [Large(x) \vee Small(x)]$.
   i. $\forall x\ [Large(x) \Rightarrow \neg Small(x)]$.

9. Find out the clauses for the following FOPL formulas.

   a. $\exists x \forall y \exists z (P(x) \Rightarrow (Q(y) \Rightarrow R(z)))$.
   b. $\forall x \forall y ((P(x) \wedge Q(y)) \Rightarrow \exists z R(x, y, z))$.

10. Define the required predicates and represent the following sentences in FOPL.

    a. Some students opted Sanskrit in fall 2015.
    b. Every student who opts Sanskrit passes it.
    c. Only one student opted Tamil in fall 2015.
    d. The best score in Sanskrit is always higher than the best score in Tamil.
    e. There is a barber in a village who shaves every one in the village who does not shave himself / herself.
    f. A person born in country X, each of whose parents is a citizen of X or a resident of X, is also a resident of X.

11. Determine whether the expression p and q unify with each other in each of the following cases. If so, give the mgu, if not justify it. The lowercase letters are variables, and upper are predicate, functions, and literals.

    a. $p = f(x_1, g(x_2, x_3), x_2, b);\ q = f(g(h(a, x_5), x_2), x_1, h(a, x_4), x_4)$.
    b. $p = f(x, f(u, x));\ q = f(f(y, a), f(z, f(b, z)))$.
    c. $p = f(g(v), h(u, v));\ q = f(w, j(x, y))$.

12. What can be the strategies for combination of clauses in resolution proof? For example, if there are N clauses, in how many ways they can be combined?
13. Why resolution based inference is more efficient compared modus-ponens?
14. Let Γ is knowledge base and α is inference from Γ. Give a comparison among the following inferences, in terms of their performances:

    a. Proof by Resolution, i.e., $\Gamma \vdash \alpha$,
    b. Proof by Modus poenes, i.e., $\Gamma \vdash \alpha$,
    c. Proof by Resolution Refutation, i.e., $\Gamma \cup \{\neg\alpha\} \vdash \phi$.

15. Given n number of clauses, draw a resolution proof tree to demonstrate combining them. Suggest any two strategies.
16. Given the knowledge base in clausal form, is it possible to extract answers from that making use of resolution principle? For example, finding an answer like, "Where is Tajmahal located?"
17. Represent the following set of statements in predicate logic, convert them to clause from, then apply the resolution proof to answer the question : Did Ranjana kill Lekhi?
    "Rajan owns a pat. Every pat owner is an animal lover. No animal lover ever kills an animal. Either Rajan or Ranjana killed a pat, called Lekhi."
18. Explain:
    a. Unification
    b. Skolemization
    c. Resolution principle versus resolution theorem proving.
19. Use resolution to show that the following set of clauses is unsatisfiable.

$$\{p(a, z), \neg p(f(f(a)), a), \neg p(x, g(y)) \vee p(f(x), y)\}.$$

20. Derive $\perp$ from the following set of clauses using the resolution principle.

$$\{p(a) \vee p(b), \neg p(a) \vee p(b), p(a) \vee \neg p(b), \neg p(a) \vee \neg p(b)\}.$$

21. Give resolution proofs for the inconsistency $\forall x\, shaves(Barber, x) \rightarrow \neg shaves(x, x)$, where *Barber* is a constant.
22. Consider ab locks-world described by facts and rules:

    Facts:

    $ontable(a), ontable(c), on(d, c), on(b, a), heavy(b),$
    $cleartop(e), cleartop(d), heavy(d), wooden(b), on(e, b).$

    Rules:

    All blocks with clear top are black.
    All wooden blocks are black.
    Every heavy and wooden block is big.
    Every big and black block is on a green block.

    Making use of resolution theorem find out the block that is on the green block.
23. Given the following knowledge base:

    If x is on top of y then y supports x.
    If x is above y and they are touching each other then x is on top of y.
    A phone is above a book.
    A phone is touching a book.

Translate the above knowledge base into clause form, and use resolution to show that the predicate "supports(book, phone)" is true.

24. How resolution can be used to show that a sentence is:
    a. Valid?
    b. Unsatisfiable?
25. "The application of resolution principle for theorem proving is a non-deterministic approach." justify this statement.
26. a. Use Herbrand's method to show that formula,

$$\forall x \; shaves(barber, x) \rightarrow \neg shaves(x, x)$$

    is unsatisfiable?
    b. What is Herband's universe for $S = \{P(a), \neg P(f(x)) \vee P(g(x))\}$?
27. Prove that $\forall x \neg p(x)$ and $\neg \exists x \; p(x)$ are equivalent statements.
28. Let S and T be unification problems. Also, let σ be a most general unifier for S and θ be a most general unifier for $\sigma(T)$. Show that $\theta\sigma$ is a most general unifier for $S \cup T$.
29. Write the axioms describing predicates: *grandchild, grandfather, grandmother, soninlaw, fatherinlaw, brother, daughter, aunt, uncle, brotherinlaw*, and *first-cousin*.
30. For each pair of atomic sentences in the following, find out the most general unifier.
    a. $knows(father(y), y)$ and $knows(x, x)$.
    b. $\{f(x, g(x)) = y, h(y) = h(v), v = f(g(z), w)\}$.
    c. $p(a, b, b)$ and $p(x, y, z)$.
    d. $q(y, g(a, b))$ and $q(g(x, x), y)$.
    e. $older(father(y), y)$ and $older(father(x), ram)$.
31. Explain what is wrong with the below given definition of set membership predicate $\in$:

$$\forall x, s : x \in \{x \mid s\}$$
$$\forall x, s : x \in s \Rightarrow \forall y : x \in \{y \mid s\}.$$

32. Consider the following riddle: "Brothers and sisters have I none, but that man's father is my father's son". Use the rules of kinship relations to show who that man is?
33. Let the following be a set of facts and rules:
    Rita, Sat, Bill, and Eden are the only members of a club.
    Rita is married to Sat.
    Bill is Eden's brother.
    Spouse of every married person in the club is also in the club.

a. Represent the above facts and rules using predicate logic.
b. Show that they do not conclude "Eden is not married."
c. Add some some more facts, and show that now the augmented set conclude that Eden is not married.

References

1. Chowdhary KR (2015) Fundamentals of discrete mathematical structures, 3rd edn. EEE, PHI India
2. Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7(3):201–215. https://doi.org/10.1145/321033.321034
3. Emden V, Kowalki RA (1976) The semantics of predicate logic as a programming language. J ACM 23(4):733–742
4. Av Gelder et al (1991) The well-founded semantics for general logic programs. J ACM 38(3):620–650
5. Luckham D, Nilsson NJ (1971) Extracting information from resolution trees. Artif Intell 2:27–54
6. Nilsson NJ (1980) Principles of artificial intelligence, 3rd edn. Narosa, New Delhi
7. Robinson JA (1963) Theorem-proving on the computer. J ACM 10(2):163–174
8. Robinson JA (1965) A machine-oriented logic, based on the resolution principle. J ACM 12(1):23–41
9. Stickel ME (1981) A unification algorithm for associative-commutative functions. J ACM 28(3):423–434

Chapter 4
Rule Based Reasoning

Abstract The popularity of rules-based systems (RBSs) is due to their naturalness. This chapter presents the potential applications of RBSs, the working of RBS, forward and backward chaining RBSs, their Algorithms, and inferencing using these systems. The analysis of complexity of preconditions, and efficiency of rule selection are introduced to sufficient depth, as well the cofmparison between the two types of RBSs are presented. A typical RBS, and other methods—model-based and case-based approaches are also discussed. In addition, number of solved, as well exhaustive list of exercises are provided at the end of the chapter for practice. The chapter concludes with its summary.

Keywords Rule-based systems (RBSs) · Forward chaining · Backward chaining · Forward chaining Algorithm · Backward chaining Algorithm · Model-based reasoning · Case-based reasoning · Conflict resolution

4.1 Introduction

Symbolic rules are popular for knowledge representation and reasoning. Their popularity stems mainly from their naturalness, which facilitates comprehension of the represented knowledge. The basic form of a rule is,

$$if <conditions> then <conclusion>$$

where $<conditions>$ represent the *conditions* or *premises* of a rule, and the $<conclusion>$ represent its conclusion or consequence. The conditions of a rule are connected between each other with logical connectives such as *AND/OR* thus forming a logical function. When sufficient conditions of a rule are satisfied, the conclusion is derived and the rule is said to *fire* (or *trigger*). Rules represent general knowledge regarding a domain.

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_4

In a rule based system, each *if* pattern may match to one or more of assertions in a collections of assertions. The collections of assertions is called *working-memory* (Fig. 4.1). The assertions collectively may match to premises of one or more rules in the knowledge base. This is done by "match" block of the inference-engine. All the rules matching with the assertions in the working memory are put in the *conflict set*, from where one of the rule is "selected" based on some conflict resolving criteria set for, and then the selected rule is executed. The resulting consequences/conclusions (the then patterns) are put in the working memory to form new assertions, and the process continues till desired result (goal) is not reached.

A system like this is called *deduction system*, as it deduces new inferences from the rules and assertions. A realistic example of a rule is:

```
R1: if 1. stiff neck,
       2. high temperature, and
       3. impairment of conciousness occur togetehr,
    then
          meningitis is suspected.
```

In the above, meningitis is a disease related to"Inflation of membrane of spinal chord and brain" will be suspected by this rule if the "if" patterns 1, 2, 3 are found true.

RBS can be applied in judiciary systems also. Following is an example of a more complex a rule.

```
If the plaintiff received an eye injury
 and it was one eye injured
 and treatment for eye required surgery
 and recovery from the injury was almost complete
 and visual acuity was slightly reduced by the injury
 and there is fixed condition,
then increase the injury trauma factor by $5,000.
```

Facts is the kind of data in a knowledge base that express assertions about properties, relations, propositions, etc. In contrast to rules, which the RBS interprets as imperatives, facts are usually static and inactive implicitly. Also, a fact is silent regarding the pragmatic value and dynamic utilization of its knowledge. Although in many contexts facts and rules are logically interchangeable, in the RBSs they are quite distinct.

In addition to its static memory for facts and rules, an RBS uses a *working-memory* to store temporary assertions. These assertions record earlier rule-based inferences. We can describe the contents of working memory as problem-solving state information. Ordinarily, the data in working memory adhere to the syntactic conventions of facts. Temporary assertions thus correspond to dynamic facts.

The basic function of an RBS is to produce results. The primary output may be—a problem solution, an answer to a question, or result of an analysis of some data. Whatever the case, an RBS employs several key processing determining its overall

activity. A *world* manager maintains information in working memory, and a built-in control procedure defines the basic high-level loop; if the built-in control provides for programmable specialized control, an additional process manages branches to and returns from special control blocks.

Some times, the patterns specify the *actions* rather than *assertions*, e.g., "to put them on the table". In such case the rule based system is called *reaction system*.

In both the deduction systems and reaction systems, forward chaining is the process of moving from the *if* patterns to *then* patterns, where *if* patterns identifies the appropriate situation for deductions of new assertion, and performance of an action in the case of *reaction system*.

4.3 Forward Chaining

In a forward chaining system, we start with the initial facts, and use the rules to draw new conclusions (or take certain actions), given those facts. Forward chaining systems are primarily *data-driven*. Whenever an if pattern is observed to match an assertion, the antecedent is *satisfied*. When all the *if* patterns of a rule are satisfied, the rule is *triggered*. When a triggered rule establishes a new assertion or performs an action, it is *fired.* This procedure is repeated until no more rules are applicable (Fig. 4.1).

The selection process is carried out in two steps:

1. *Pre-selection*: Determining the set of all the matching rules, also called the *conflict-set*.
2. *Selection*: Selection of a rule from the conflict set by means of a conflict resolving strategy.

4.3.1 Forward Chaining Algorithm

A simple forward chaining Algorithm 4.1 starts from the known facts in the knowledge base, and triggers all the rules whose premises are the known facts, then adds the consequent of each into the knowledge base. This process is repeated until the query is answered or until there is no conclusion generated to be added into the knowledge base. We will use symbols θ, λ, γ to represent substitutions. The *unify* is a function unifies the newly generated assertion q' and the query α, and returns a unifying substitution λ if they are unified, else returns null.

Let α be the goal. The forward-chaining Algorithm 4.1 picks up any sentence $s \in \Gamma$, where Γ is knowledge base, and checks all possible substitutions θ for s. Let, the predicate form of s is $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow q$. On substituting θ, say, it results to $(p_1 \wedge p_2 \wedge \ldots p_n)\theta = (p'_1 \wedge p'_2 \wedge \ldots p'_n)$, such that p'_i's $\in \Gamma$. Let $(p'_1 \wedge p'_2 \wedge \ldots p'_n) \rightarrow q'$, such that $q' = q\theta$. This q' is added into *new* inference. If the

inference q' and goal α unify, then their unifier λ is returned as the solution, else the Algorithm continues.

Algorithm 4.1 Forward-chaining(Input: Γ, α) // α is a query, Γ is knowledge base

```
1: while True do
2:   new = { }
3:   for each sentence s ∈ Γ do
4:     Convert s into the format p1 ∧ p2 ∧ ··· ∧ pn → q
5:     for each substitution θ such that (p'1 ∧ p'2 ∧ ..., ∧ p'n) ← (p1 ∧ p2 ∧ ..., ∧ pn)θ for
       some p'i s ∈ Γ do
6:       q' ← qθ
7:       new ← new ∪ {q'}
8:       λ ← unify(q', α)
9:       if λ is not null then
10:        return λ
11:      end if
12:    end for
13:    Γ ← Γ ∪ new // add new inferences in knowledge base
14:  end for
15: end while
16: Return Fail
```

The Algorithm may fail to terminate in the case when the raised query has no answer. For example, every natural number can be generated by recursively applying successor operation on a natural number, and assuming that 0 is natural number. This will lead to indefinite loop for very large numbers.

$$naturalnum(0).$$
$$\forall x[naturalnum(x) \rightarrow naturalnum(succ(x)).]$$

The following example demonstrates the manual run of forward chaining Algorithm.

Example 4.1 Produce the inference, given the knowledge base $\Gamma = \{man(x) \rightarrow mortal(x), man(socrates)\}$ and query $\alpha = mortal(w)$, i.e., "Who is mortal?"

We follow the Algorithm 4.1 manually and note that $p_1 \wedge \cdots \wedge p_n = p_1 = man(x)$; $p'_1 = man(socrates)$, substitution $\theta = \{socrates/x\}$, and $q = mortal(x)$. Also,

$$\begin{aligned} q' &= q\theta \\ &= mortal(x)\{socrates/x\} \\ &= mortal(socrates). \end{aligned}$$

Also, $new = \{\} \cup \{q'\} = mortal(socrates)$.
On unification of α (i.e, $mortal(w)$), and q', the unifier λ obtained is,

$$\begin{aligned}\lambda &= unify(q', \alpha) \\ &= unify(mortal(socrates), mortal(w)) \\ &= \{socrates/w\}.\end{aligned}$$

Hence, the answer for query is $w = socrates$. □

Complexity Issues

The complexity of the forward chaining Algorithm is determined by the inner loop in the Algorithm 4.1. It finds all the possible premises such that the premises unify with certain set of facts in the knowledge base Γ. The process is called *pattern matching*, and tried for every rule, for every substitution θ. Thus, if set of rules are P, there are n^k substitutions for each rule, assuming that in worst case k arguments and n number of literals' assignments for each argument in the P. This makes the complexity of above Algorithm as,

$$|P|n^k \tag{4.1}$$

which is exponential. However, since the size and arities are constant in the real-world, the complexity of expression (4.1) is a polynomial in nature. To search the predicates for matching, they are indexed and a hash function is generated for quick search.

4.3.2 Conflict Resolution

When we are doing data-directed reasoning, we may not like to fire all of the rules in case more than one rule is applicable. In cases where we want to eliminate some applicable rules, some kind of conflict resolution is necessary for arriving at the most appropriate rule(s) to fire. In a deduction system all rules generally fire, but in a reaction system, when more than one rule are triggered at the same time, only one of the possible actions is desired [4].

The conflict resolving strategy is used to avoid superfluous rules. The most obvious strategy is to choose a rule at random, out of those that are applicable. Following are some common approaches for deciding the preferences for the rule to fire.

Selection by order

Selection by order may either on the order in which the rule appears in the knowledge base, or the order may be based on how recent the rule was used.

Order

Pick the first applicable rule in the order they have been presented. This is the type of strategy used by *Prolog*, and is one of the most common ones. Production system programmers would take this strategy into account when formulating rule sets.

Recency

Select an applicable rule based on how recently it has been used. There are different versions of this strategy, ranging from firing the rule that matches on the most recently created (or modified) wff, to firing the rule that has been least recently used. The former could be used to make sure that a problem solver stays focused on what it was just doing (typical of depth-first search); the latter would ensure that every rule gets a fair chance to influence the outcome (typical of breadth-first search).
Selection according to syntactic structure of the rule
There are many ways one can consider the syntax, e.g., it may be conditions and their specificity, or the size of the rule: largest, smallest, or in increasing order of size.

Specificity Criteria

Select the applicable rule whose conditions are most specific. A set of conditions is taken as more specific than other if the set of wffs that satisfy it is a subset of those that satisfy the other. For example, consider the following three rules:

i) *if* $bird(x)$ *then* $add(canfly(x))$
ii) *if* $bird(x) \wedge weight(y, x) \wedge \text{gt}(y, 5kg)$ *then* $add(cannotfly(x))$
iii) *if* $bird(x) \wedge penguin(x)$ *then* $add(cannotfly(x))$

Here, the second and third rules are both more specific than the first, because the condition in (ii) and (iii) are more specific than in (i) (condition of (i) is very general). If we have a bird that has weight greater than 5 kg, or it is a penguin, then the first rule applies, because the condition of "bird" is satisfied. However, in this case the other two rules should take precedence, as they are more specific. Note that if the bird is a penguin and heavy, neither second nor third is applicable, and another conflict resolution criteria must be applied to decide between the second and third rules.

Largest Rule First

Apply the syntactically largest rule, i.e., the rule which contains the most propositions.

Incremental in Size

Solutions to subproblems are constructed incrementally, and are cached to avoid the recompilation.

Selection by Means of Supplementary Knowledge

The supplementary knowledge about the rules may be in the form of priority of the rules, or as meta-knowledge.

Highest Priority First

For this purpose each rule must be given a priority, which may be represented, e.g., by a numeric value.

For this, the goal expression is initially placed in the working memory, followed with this, the system performs two operations:

1. matches the rule's "consequent" with the goal. For example, q in $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow q$,
2. selects the matched rule $p_1 \wedge p_2 \wedge \ldots p_n \rightarrow q$ and places its premises $(p_1 \wedge \cdots \wedge p_n)$ in the working memory.

The second step in above corresponds to the transformation of the problem into subgoals. The process continues in the next iteration, with these premises becoming the new goals to match against the consequent of other rules. Thus, the backward chaining system, in the human sense are *hypothesis testing*.

Backward chaining uses *stack*. First, the goal is put in the stack, then all the rules which results to this goal are searched and put on the stack. These becomes the subgoals, and the process goes on till all the facts are proved or are available as literals (ground clauses). Every time the goal and premises are decided, the unification is performed.

4.4.1 Backward Chaining Algorithm

The Algorithm for backward chaining returns the set of substitutions (*unifier*) which makes the goal true. These are initially empty. The input to the Algorithm is knowledge base Γ, goals α, and current substitution θ (initially empty). The Algorithm returns the substitution set λ for which the goal is inferred. The Algorithm 4.2 is the backward-chaining Algorithm.

Algorithm 4.2 Backward-chaining(Input: Γ, α, θ) // α is a query, Γ is knowledge base, θ current substitution (initially empty), λ represent substitution set for the query to be satisfied (initially empty)

1: $\theta = \{\}, \lambda = \{\}$
2: $q' \leftarrow \alpha\theta$
3: **for** each sentence $s \in \Gamma$, where $s = p_1 \wedge p_2 \wedge \ldots, \wedge p_n \rightarrow q$ and $\gamma \leftarrow unify(q, q') \neq null$ **do**
4: $\quad \alpha_{new} \leftarrow (p_1 \wedge p_2 \wedge \cdots \wedge p_n)$
5: $\quad \theta \leftarrow \theta\gamma$
6: $\quad \lambda \leftarrow$backward-chaining$(\Gamma, \alpha_{new}, \theta) \cup \lambda$
7: **end for**
8: return λ

Example 4.2 Apply the backward-chaining Algorithm for inferencing from a given knowledge base.

Let $\Gamma = \{man(x) \rightarrow mortal(x), man(socrates)\}$. Assume that it is required to infer answer for "Who is mortal?" That is, goal $\alpha = mortal(w)$, find w. The loop iterations in the backward-chaining Algorithm 4.2are as follows:

1st Iteration: Initially θ is empty, hence, $q' = \alpha\theta = mortal(w)$. From Algorithm and knowledge base Γ, the sentence is, $s = (man(x) \rightarrow mortal(x))$. Next, $\gamma = unify(mortal(x), mortal(w)) = \{w/x\}$. Also, the new goal, $\alpha_{new} = man(x)$. The new value of current substitution is, $\theta \leftarrow \theta\gamma = \{\}\{w/x\} = \{w/x\}$. Next, compute $\lambda = backward\text{-}chaining(\Gamma, man(x), \{w/x\}) \cup \lambda$, as a recursive call.

Recursive call: We apply the Algorithm in a recursive mode, and get $q' = man(x)$ $\{w/x\} = man(w)$. Next, the subgoal $man(w)$ (i.e., q') matches with other subgoal (it is a *fact*) $man(socrates) \in \Gamma$. Hence, $\gamma = unify(man(socrates), man(w)) = \{socrates/w\}$. Next, the new goal is, $\alpha_{new} = man(socrates)$ and new substitution is:

$$\begin{aligned} \theta &= \theta\gamma \\ &= \{w/x\}\{socrtaes/w\} \\ &= \{socrates/x\}. \end{aligned}$$

In the next call of recursion at step 5, $q' = \alpha\theta = man(socrates)$ $\{socrates/x\}$, the substitution fails, as there is no x where "socrates" can be substituted. Hence, $q' = null$. Since γ is null, the *for loop* at step 3 terminates, and returned value of λ is $\{socrates/w\}$, i.e., $w = socrates$, or $mortal(socrtaes)$. □

4.4.2 Goal Determination

If the goal is not known in the knowledge base, the rule interpreter first decides whether it can be derived or must be requested from user. For the derivation of goal, all the rules which includes the goal in their consequent part are processed. These rules can be identified efficiently if they are indexed according to their consequent parts.

Backward chaining therefore contains implicitly a dialogue control, where order of the questions decides the order of the rules for deriving a parameter. This dependency on the order reduces the modularity of the rule-based system. The efficiency of the backward-chained rule interpreter is determined by the formulation of the goal: the more precise the goal, the smaller is the search tree of rules to be checked and questions to be asked. For example, in medicine, a query may be: "What is name of disease?" as against an alternate query of "Is X the name of disease?".

The examples of back-ward chained rule interpreter are MYCIN, and PROLOG.

4.5 Forward Versus Backward Chaining

Whenever the rules are such that typical set of facts relate to large number of conclusions, i.e., the *fan-out* is larger than *fan-in*, it is better to use backward chaining. This is to stop in explosive growth in search space. On the other hand, when the rules

Table 4.2 Knowledge base for Animal-Kingdom

S. no.	Rule
1	$sponge(x) \rightarrow animal(x)$
2	$arthopod(x) \rightarrow animal(x)$
3	$vertebrate(x) \rightarrow animal(x)$
4	$fish(x) \rightarrow vertebrate(x)$
5	$mammal(x) \rightarrow vertebrate(x)$
6	$carnivore(x) \rightarrow mammal(x)$
7	$dog(x) \rightarrow carnivore(x)$
8	$cat(x) \rightarrow carnivore(x)$

are such that a typical hypothesis can lead to many facts, i.e., *fan-in* is larger than *fan-out*, it is better to use forward chaining. The reason being that if we go backward in a case where fan-in is high, it would lead to exponentially growing search space, so we prefer forward chaining in such knowledge.

If there is a situation that all the required facts are known, and we want to get all the possible conclusions from these, the forward chaining is better. However, if the rules are such that facts are incompletely known, we cannot proceed from the facts to conclusions, and thus goal driven strategy should be used.

Forward chaining is often preferable in cases when there are many rules with the same conclusions, because we choose the satisfying premises. In backward chaining it may lead to non-satisfying premises. The following examples demonstrate this.

Example 4.3 Given a knowledge base for an animal kingdom, infer *animal(bruno)* after adding of *dog(bruno)*. The taxonomy of the animal kingdom includes such rules as shown in Table 4.2.

It is required to show that, $KB + dog(bruno) \rightarrow animal(bruno)$.

1. *Forward Chaining.* We start with rule 7, and unify $dog(bruno)$ with $dog(x)$. Next, we will successively infer and add *carnivore(bruno)*, *mammal(bruno)*, *vertebrate(bruno)*, and *animal(bruno)*. The query will then succeed immediately. The total work is proportional to the height of the hierarchy of this taxonomy, which is 4.
2. *Backward-chaining.* Alternatively, if we use backward chaining, the query $animal(bruno)$ will unify with the first rule above and generate the sub-query $sponge\ (bruno)$, which will initiate a search for $bruno$ through all the subdivisions of sponges. Not finding, it tries with 2nd rule, but orthopod is not in consequent. Next, with rule 3, the goal driven chain is: "animal $\rightarrow$ vertebrate $\rightarrow$ fish", which fails. The successful invocation rule sequence is "3 $\rightarrow$ 5 $\rightarrow$ 6 $\rightarrow$ 7." Thus, it searches the entire taxonomy of animals looking for $dog(bruno)$. We note that work done in the background chaining much more than forward chaining. However, this is not necessarily true always. □