16 Advanced Topics in Logic

Key Topics

Fuzzy Logic
Intuitionist Logic
Temporal Logic
Undefined values
Theorem Provers
Logic of partial functions
Logic and AI

16.1 Introduction

In this chapter, we consider some advanced topics in logic including fuzzy logic, temporal logic, intuitionist logic, undefined values, logic and AI and theorem provers. Fuzzy logic is an extension of classical logic that acts as a mathematical model for vagueness, and it handles the concept of partial truth where truth values lie between completely true and completely false. Temporal logic is concerned with the expression of properties that have time dependencies, and it allows temporal properties about the past, present and future to be expressed.

Brouwer and others developed intuitionist logic as the logical foundation for intuitionism, which was a controversial theory of the foundations of mathematics based on a rejection of the law of the excluded middle and an insistence on constructive existence. Martin Löf successfully applied it to type theory in the 1970s.

G. O’Regan, *Guide to Discrete Mathematics*, Texts in Computer Science,
DOI 10.1007/978-3-319-44561-8_16

Partial functions arise naturally in computer science, and such functions may fail to be defined for one or more values in their domain. One approach to dealing with partial functions is to employ a precondition, which restricts the application of the function to values where it is defined. We consider three approaches to deal with undefined values, including the logic of partial functions; Dijkstra's approach with his *cand* and *cor* operators; and Parnas's approach which preserves a classical two-valued logic.

We examine the contribution of logic to the AI field, and the work done by theorem provers to supporting proof.

16.2 Fuzzy Logic

Fuzzy logic is a branch of *many-valued logic* that allows inferences to be made when dealing with vagueness, and it can handle problems with imprecise or incomplete data. It differs from classical two-valued propositional logic, in that it is based on degrees of truth, rather than on the standard binary truth values of "true or false" (1 or 0) of propositional logic. That is, while statements made in propositional logic are either true or false (1 or 0), the truth value of a statement made in fuzzy logic is a value between 0 and 1. Its value expresses the extent to which the statement is true, with a value of 1 expressing absolute truth, and a value of 0 expressing absolute falsity.

Fuzzy logic uses *degrees of truth* as a mathematical model for vagueness, and this is useful since statements made in natural language are often vague and have a certain (rather than an absolute) degree of truth. It is an extension of classical logic to handle the concept of partial truth, where the truth value lies between completely true and completely false. Lofti Zadeh developed fuzzy logic at Berkley in the 1960s, and it has been successfully applied to Expert Systems and other areas of Artificial Intelligence.

For example, consider the statement "John is tall". If John were 6 foot, 4 in. then we would say that this is a true statement (with a truth value of 1) since John is well above average height. However, if John is 5 ft, 9 in. tall (around average height) then this statement has a degree of truth, and this could be indicated by a fuzzy truth valued of 0.6. Similarly, the statement that today is sunny may be assigned a truth value of 1 if there are no clouds, 0.8 if there are a small number of clouds, and 0 if it is raining all day.

Propositions in fuzzy logic may be combined together to form compound propositions. Suppose X and Y are propositions in fuzzy logic, then compound propositions may be formed from the conjunction, disjunction and implication operators. The usual definition in fuzzy logic of the truth values of the compound propositions formed from X and Y is given by

$$\begin{aligned}
&\text{Truth } (\neg X) = 1 - \text{ Truth}(X)\\
&\text{Truth } (X \text{ and } \text{Y}) = \min(\text{Truth}(X),\ \text{Truth}(Y))\\
&\text{Truth } (X \text{ or } \text{Y}) = \max(\text{Truth}(X),\ \text{Truth}(Y))\\
&\text{Truth } (X \rightarrow \text{Y}) = \text{ Truth}(\neg X \text{ or } Y))
\end{aligned}$$

There is another way in which the operators may be defined in terms of multiplication

$$\begin{aligned}
&\text{Truth}(X \text{ and Y}) = \text{Truth}(X)\,{}^*\,\text{Truth}(Y)\\
&\text{Truth}(X \text{ or Y}) = 1 - (1 - \text{Truth}(X))\,{}^*\,(1 - \text{Truth}(Y))\\
&\text{Truth}(X \rightarrow \text{Y}) = \max\{z \mid \text{Truth}(X)\,{}^*\,z \leq \text{Truth}(Y)\} \text{ where } 0 \leq z \leq 1
\end{aligned}$$

Under these definitions, fuzzy logic is an extension of classical two-valued logic, which preserves the usual meaning of the logical connectives of propositional logic when the fuzzy values are just {0, 1}.

Fuzzy logic has been very useful in expert system and artificial intelligence applications. The first fuzzy logic controller was developed in England in the mid-1970s. It has been applied to the aerospace and automotive sectors, and also to the medical, robotics and transport sectors.

16.3 Temporal Logic

Temporal logic is concerned with the expression of properties that have time dependencies, and the various temporal logics can express facts about the past, present and future. Temporal logic has been applied to specify temporal properties of natural language, artificial intelligence as well as the specification and verification of program and system behaviour. It provides a language to encode temporal knowledge in artificial intelligence applications, and it plays a useful role in the formal specification and verification of temporal properties (e.g. liveness and fairness) in safety critical systems.

The statements made in temporal logic can have a truth value that varies over time. In other words, sometimes the statement is true and sometimes it is false, but it is never true or false at the same time. The two main types of temporal logics are *linear time logics* (reason about a single time line), and *branching time logics* (reason about multiple timelines).

The roots of temporal logic lie in work done by Aristotle in the fourth century B. C., when he considered whether a truth value should be given to a statement about a future event that may or may not occur. For example, what truth value (if any) should be given to the statement that '*There will be a sea battle tomorrow*'. Aristotle argued against assigning a truth value to such statements in the present time.

Newtonian mechanics assumes an absolute concept of time independent of space, and this viewpoint remained dominant until the development of the theory of relativity in the early twentieth century (when space-time became the dominant paradigm).

Arthur Prior began analyzing and formalizing the truth values of statements concerning future events in the 1950s, and he introduced Tense Logic (a temporal logic) in the early 1960s. Tense logic contains four modal operators (strong and weak) that express events in the future or in the past

-*P* (It has at some time been the case that)
-*F* (It will be at some time be the case that)
-*H* (It has always been the case that)
-*G* (It will always be the case that)

The *P* and *F* operators are known as weak tense operators, while the *H* and *G* operators known as strong tense operators. The two pairs of operators are interdefinable via the equivalences

$$P\phi \cong \neg H \neg \phi$$
$$H\phi, \cong \neg P \neg \phi$$
$$F\phi \cong \neg G \neg \phi$$
$$G\phi, \cong \neg F \neg \phi$$

The set of formulae in Prior's temporal logic may be defined recursively, and they include the connectives used in classical logic (e.g. $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$). We can express a property ϕ that is always true as $A\phi \cong H\phi \wedge \phi \wedge G\phi$ and a property that is sometimes true as $E\phi \cong P\phi \vee \phi \vee F\phi$. Various extensions of Prior's tense logic have been proposed to enhance its expressiveness. These include the binary *since* temporal operator '*S*', and the binary *until* temporal operator '*U*'. For example, the meaning of $\phi S \psi$ is that ϕ has been true since a time when ψ was true.

Temporal logics are applicable in the specification of computer systems, and a specification may require *safety*, *fairness* and *liveness properties* to be expressed. For example, a fairness property may state that it will always be the case that a certain property will hold sometime in the future. The specification of temporal properties often involves the use of special temporal operators.

We discuss common temporal operators that are used, including an operator to express properties that will always be true; properties that will eventually be true; and a property that will be true in the next time instance. For example

$\square P$	$-P$ is always true
$\lozenge$	$-P$ will be true sometime in the future
$\bigcirc$	$-P$ is true in the next time instant(*discrete time*)

Linear temporal logic (LTL) was introduced by Pnueli in the late 1970s. This linear time logic is useful in expressing safety and liveness properties. Branching time logics assume a non-deterministic branching future for time (with a deterministic, linear past). Computation tree logic (CTL and CTL*) were introduced in the early 1980s by Emerson and others.

It is also possible to express temporal operations directly in classical mathematics, and the well-known computer scientist, Parnas, prefers this approach. He is critical of computer scientists for introducing unnecessary formalisms when classical mathematics already possesses the ability to do this. For example, the value of a function *f* at a time instance prior to the current time *t* is defined as

$$\mathrm{Prior}(f, t) = \lim_{\varepsilon \to 0} f(t - \varepsilon)$$

For more detailed information on temporal logic the reader is referred to the excellent article on temporal logic in [1].

16.4 Intuitionist Logic

The controversial school of intuitionist mathematics was founded by the Dutch mathematician, L. E. J. Brouwer, who was a famous topologist, and well known for his fixpoint theorem in topology. This constructive approach to mathematics proved to be highly controversial, as its acceptance as a foundation of mathematics would have led to the rejection of many accepted theorems in classical mathematics (including his own fixed point theorem).

Brouwer was deeply interested in the foundations of mathematics, and the problems arising from the paradoxes of set theory. He was determined to provide a secure foundation for mathematics, and his view was that an existence theorem in mathematics that demonstrates the proof of a mathematical object has no validity, unless the proof is constructive and accompanied by a procedure to construct the object. He therefore rejected indirect proof and the law of the excluded middle ($P \vee \neg P$) or equivalently ($\neg\neg P \rightarrow P$), and he insisted on an explicit construction of the mathematical object.

The problem with the law of the excluded middle (LEM) arises in dealing with properties of infinite sets. For finite sets, one can decide if all elements of the set possess a certain property *P* by testing each one. However, this procedure is no longer possible for infinite sets. We may know that a certain element of the infinite set does not possess the property, or it may be the actual method of construction of the set allows us to prove that every element has the property. However, the application of the law of the excluded middle is invalid for infinite sets, as we cannot conclude from the situation where not all elements of an infinite set possesses a property *P* that there exists at least one element which does not have the property *P*. In other words, the law of the excluded middle may only be applied in cases where the conclusion can be reached in a finite number of steps.

Consequently, if the Brouwer view of the world was accepted then many of the classical theorems of mathematics (including his own well-known results in topology) could no longer be said to be true. His approach to the foundations of mathematics hardly made him popular with other mathematicians (the differences were so fundamental that it was more like a war), and intuitionism never became mainstream in mathematics. It led to deep and bitter divisions between Hilbert and Brouwer, with Hilbert accusing Brouwer (and Weyl) of trying to overthrow everything that did not suit them in mathematics, and that intuitionism was treason to science. Hilbert argued that a suitable foundation for mathematics should aim to preserve most of mathematics. Brouwer described Hilbert's formalist program as a false theory that would produce nothing of mathematical value. For Brouwer, 'to exist' is synonymous with 'constructive existence', and constructive mathematics is relevant to computer science, as a program may be viewed as the result obtained from a constructive proof of its specification.

Brouwer developed one of the more unusual logics that have been invented (intuitionist logic), in which many of the results of classical mathematics were no longer true. Intuitionist logic may be considered the logical basis of constructive mathematics, and formal systems for intuitionist propositional and predicate logic were developed by Heyting and others [2].

Consider a hypothetical mathematical property $P(x)$ of which there is no known proof (i.e. it is unknown whether $P(x)$ is true or false for arbitrary x where x ranges over the natural numbers). Therefore, the statement $\forall x\ (P(x) \vee \neg\ P(x))$ cannot be asserted with the present state of knowledge, as neither $P(x)$ or $\neg P(x)$ has been proved. That is, unproved statements in intuitionist logic are not given an intermediate truth value, and they remain of an unknown truth value until they have been either proved or disproved.

The intuitionist interpretation of the logical connectives is different from classical propositional logic. A sentence of the form $A \vee B$ asserts that either a proof of A or a proof of B has been constructed, and $A \vee B$ is not equivalent to $\neg\ (\neg A \wedge \neg B)$. Similarly, a proof of $A \wedge B$ is a pair whose first component is a proof of A, and whose second component is a proof of B. The statement $\forall x\ \neg P(x)$ is not equivalent to $\exists x\ P(x)$ in intuitionist logic.

Intuitionist logic was applied to Type Theory by Martin Löf in the 1970s [3]. Intuitionist type theory is based on an analogy between propositions and types, where $A \wedge B$ is identified with $A \times B$, the Cartesian product of A and B. The elements in the set $A \times B$ are of the form $(a,\ b)$ where $a \in A$ and $b \in B$. The expression $A \vee B$ is identified with $A + B$, the disjoint union of A and B. The elements in the set $A + B$ are got from tagging elements from A and B, and they are of the form inl(a) for $a \in A$, and inr(b) for $b \in B$. The left and right injections are denoted by inl and inr.

16.5 Undefined Values

Total functions f: X → Y are functions that are defined for every element in their domain, and total functions are widely used in mathematics. However, there are functions that are undefined for one or more elements in their domain, and one example is the function $y = 1/x$. This function is undefined at $x = 0$.

Partial functions arise naturally in computer science, and such functions may fail to be defined for one or more values in their domain. One approach to dealing with partial functions is to employ a precondition, which restricts the application of the function to where it is defined. This makes it possible to define a new set (a proper subset of the domain of the function) for which the function is total over the new set.

Undefined terms often arise[1] and need to be dealt with. Consider, the example of the square root function $\sqrt{x}$ taken from [4]. The domain of this function is the positive real numbers, and the following expression is undefined

$$\big((x > 0) \wedge (y = \sqrt{x})\big) \vee \big((x \leq 0) \wedge (y = \sqrt{-x})\big)$$

The reason this is undefined is since the usual rules for evaluating such an expression involve evaluating each sub-expression, and then performing the Boolean operations. However, when $x < 0$ the sub-expression $y = \sqrt{x}$ is undefined, whereas when $x > 0$ the sub-expression $y = \sqrt{-x}$ is undefined. Clearly, it is desirable that such expressions be handled, and that for the example above, the expression would evaluate to true.

Classical two-valued logic does not handle this situation adequately, and there have been several proposals to deal with undefined values. Dijkstra's approach is to use the ***cand*** and ***cor*** operators in which the value of the left hand operand determines whether the right-hand operand expression is evaluated or not. Jone's logic of partial functions [5] uses a three-valued logic[2] and Parnas's[3] approach is an extension to the predicate calculus to deal with partial functions that preserves the two-valued logic.

16.5.1 Logic of Partial Functions

Jones [5] has proposed the logic of partial functions (LPFs) as an approach to deal with terms that may be undefined. This is a three-valued logic and a logical term may be true, false or undefined (denoted ⊥). The definition of the truth functional operators used in classical two-valued logic is extended to three-valued logic. The truth tables for conjunction and disjunction are defined in Fig. 16.1.

[1]It is best to avoid undefinedness by taking care of the definitions of terms and expressions.

[2]The above expression would evaluate to true under Jones three-valued logic of partial functions.

[3]The above expression evaluates to true for Parnas logic (a two-valued logic).

P∧Q

∧ P \ Q	T	F	⊥
T	T	F	⊥
F	F	F	F
⊥	⊥	F	⊥

P∨Q

∨ P \ Q	T	F	⊥
T	T	T	T
F	T	F	⊥
⊥	T	⊥	⊥

Fig. 16.1 Conjunction and disjunction operators

P→Q

→ P \ Q	T	F	⊥
T	T	F	⊥
F	T	T	T
⊥	T	⊥	⊥

P↔Q

↔ P \ Q	T	F	⊥
T	T	F	⊥
F	F	T	⊥
⊥	⊥	⊥	⊥

Fig. 16.2 Implication and equivalence operators

The conjunction of *P* and *Q* is true when both *P* and *Q* are true; false if one of *P* or *Q* is false, and undefined otherwise. The operation is commutative. The disjunction of *P* and *Q* ($P \vee Q$) is true if one of *P* or *Q* is true; false if both *P* and *Q* are false; and undefined otherwise. The implication operation ($P \rightarrow Q$) is true when *P* is false or when *Q* is true; false when *P* is true and *Q* is false and undefined otherwise. The equivalence operation ($\mathrm{P} \leftrightarrow Q$) is true when both *P* and *Q* are true or false; it is false when *P* is true and *Q* is false (and vice versa); and it is undefined otherwise (Fig. 16.2).

The not operator ($\neg$) is a unary operator such $\neg A$ is true when *A* is false, false when *A* is true and undefined when *A* is undefined (Fig. 16.3).

The result of an operation may be known immediately after knowing the value of one of the operands (e.g. disjunction is true if *P* is true irrespective of the value of *Q*). The law of the excluded middle: i.e. $A \vee \neg A$ does not hold in the three-valued logic, and Jones [5] argues that this is reasonable as one would not expect the following to be true

$$\left({}^1/_0 = 1\right) \vee \left({}^1/_0 \neq 1\right)$$

A	¬A
T	F
F	T
⊥	⊥

Fig. 16.3 Negation

Table 16.1 Examples of parnas evaluation of undefinedness

Expression	$x < 0$	$x \geq 0$
$y = \sqrt{x}$	*False*	*True if* $y = \sqrt{x}$, *False otherwise*
$y = {}^1/_0$	*False*	*False*
$y = x^2 + \sqrt{x}$	*False*	*True if* $y = x^2 + \sqrt{x}$, *False otherwise*

There are other well-known laws that fail to hold such as

(i) $E \Rightarrow E$
(ii) Deduction Theorem $E_1 \vdash E_2$ does not justify $\vdash E_1 \Rightarrow E_2$ unless it is known that E_1 is defined.

Many of the tautologies of standard logic also fail to hold.

16.5.2 Parnas Logic

Parnas's approach to logic is based on classical two-valued logic, and his philosophy is that truth values should be true or false only,[4] and that there is no third logical value. It is an extension to predicate calculus to deal with partial functions. The evaluation of a logical expression yields the value 'true' or 'false' irrespective of the assignment of values to the variables in the expression. This allows the expression: $(y = \sqrt{x}) \vee (y = \sqrt{-x})$ that is undefined in classical logic to yield the value true.

The advantages of his approach are that no new symbols are introduced into the logic, and that the logical connectives retain their traditional meaning. This makes it easier for engineers and computer scientists to understand, as it is closer to their intuitive understanding of logic.

The meaning of predicate expressions is given by first defining the meaning of the primitive expressions. These are then used as the building blocks for predicate expressions. The evaluation of a primitive expression $R_j(V)$ (where V is a comma separated set of terms with some elements of V involving the application of partial functions) is false if the value of an argument of a function used in one of the terms of V is not in the domain of that function.[5] The following examples (Tables 16.1 and 16.2) should make this clear.

These primitive expressions are used to build the predicate expressions, and the standard logical connectives are used to yield truth values for the predicate expression. Parnas logic is defined in detail in [4].

The power of Parnas logic may be seen by considering a tabular expressions example [4]. The table below specifies the behaviour of a program that searches the

[4]It seems strange to assign the value false to the primitive predicate calculus expression $y = {}^1/_0$.
[5]The approach avoids the undefined logical value ($\perp$) and preserves the two-valued logic.

Table 16.2 Example of Undefinedness in Array

Expression	$i \in \{1 \ldots N\}$	$i \notin \{1 \ldots N\}$
$B[i] = x$	*True* if $B[i] = x$	*False*
$\exists i, B[i] = x$	*True* if $B[i] = x$ for some i, *False* otherwise	*False*

		$(\exists\, i, B[i]{=}x)$	$\neg(\exists\, i,\ B[i]{=}x)$	H_2
H_1	$j'\vert$	$B[j']{=}x$	<u>*true*</u>	G
	present'=	*true*	*false*	

Fig. 16.4 Finding Index in Array

array B for the value x. It describes the properties of the values of j' and *present'*. There are two cases to consider (Fig. 16.4):

1. There is an element in the array with the value of x.
2. There is no such element in the array with the value of x.

Clearly, from the example above the predicate expressions $\exists i$, B[i] = x and $\neg(\exists\, i, B[i] = x)$ are defined. One disadvantage of the Parnas approach is that some common relational operators (e.g., >, $\geq$, $\leq$, and <) are not primitive in the logic. However, these relational operators are then constructed from primitive operators. Further, the axiom of reflection does not hold in the logic.

16.5.3 Dijkstra and Undefinedness

The ***cand*** and ***cor*** operators were introduced by Dijkstra (Fig. 16.5) to deal with undefined values. They are non-commutative operators and allow the evaluation of predicates that contain undefined values.

Consider the following expression:

$$y = 0 \vee (x/y = 2)$$

Fig. 16.5 Edsger Dijkstra. Courtesy of Brian Randell

Table 16.3 *a cand b*

a	***b***	***a cand b***
T	T	T
T	F	F
T	U	U
F	T	F
F	F	F
F	U	F
U	T	U
U	F	U
U	U	U

Table 16.4 *a cor b*

a	***b***	***a cor b***
T	T	T
T	F	T
T	U	T
F	T	T
F	F	F
F	U	U
U	T	U
U	F	U
U	U	U

Then this expression is undefined when $y = 0$ as x/y is undefined, since the logical disjunction operation is not defined when one of its operands is undefined. However, there is a case for giving meaning to such an expression when $y = 0$, since in that case the first operand of the logical or operation is true. Further, the logical *disjunction* operation is defined to be true if either of its operands is true. This motivates the introduction of the ***cand*** and ***cor*** operators. These operators are associative and their truth tables are defined in Tables 16.3 and 16.4:

The order of the evaluation of the operands for the ***cand*** operation is to *evaluate the first operand*; if the first operand is true then the result of the operation is the second operand; otherwise the result is false. The expression *a* **cand** *b* is equivalent to

$$a\,\boldsymbol{cand}\,b \cong \textbf{if } a \textbf{ then } b \textbf{ else } F$$

The order of the evaluation of the operands for the ***cor*** operation is to evaluate the first operand. If the first operand is true then the result of the operation is true; otherwise the result of the operation is the second operand. The expression *a* ***cor*** *b* is equivalent to

$$a\,\textbf{cor}\,b \cong \textbf{if } a \textbf{ then } T \textbf{ else } b$$

The **cand** and **cor** operators satisfy the following laws:

- *Associativity*
  The cand and cor operators are associative.

$$(\mathrm{A}\ \mathbf{cand}\ \mathrm{B})\ \mathbf{cand}\ \mathrm{C} = \mathrm{A}\ \mathbf{cand}\ (\mathrm{B}\ \mathbf{cand}\ \mathrm{C})$$
$$(\mathrm{A}\ \mathbf{cor}\ \mathrm{B})\ \mathbf{cor}\ \mathrm{C} = \mathrm{A}\ \mathbf{cor}\ (\mathrm{B}\ \mathbf{cor}\ \mathrm{C})$$

- *Distributivity*
  The **cand** operator distributes over the **cor** operator and vice versa.

$$\mathrm{A}\ \mathbf{cand}\ (\mathrm{B}\ \mathbf{cor}\ \mathrm{C}) = (\mathrm{A}\ \mathbf{cand}\ \mathrm{B})\ \mathbf{cor}\ (\mathrm{A}\ \mathbf{cand}\ \mathrm{C})$$
$$\mathrm{A}\ \mathbf{cor}\ (\mathrm{B} \wedge \mathrm{C}) = (\mathrm{A}\ \mathbf{cor}\ \mathrm{B})\ \mathbf{cand}\ (\mathrm{A}\ \mathbf{cor}\ \mathrm{C})$$

De Morgan's law enables logical expressions to be simplified.

$$\neg(\mathrm{A}\ \mathbf{cand}\ \mathrm{B}) = \neg\mathrm{A}\mathbf{cor}\neg\mathrm{B}$$
$$\neg(\mathrm{A}\ \mathbf{cor}\ \mathrm{B}) = \neg\mathrm{A}\ \mathbf{cand}\ \neg\mathrm{B}$$

16.6 Logic and AI

The long-term goal of Artificial Intelligence is to create a thinking machine that is intelligent, has consciousness, has the ability to learn, has free will, and is ethical. Artificial Intelligence is a young field and John McCarthy and others coined the term in 1956. Alan Turing devised the Turing Test in the early 1950s as a way to determine whether a machine was conscious and intelligent. Turing believed that machines would eventually be developed that would stand a good chance of passing the 'Turing Test'.

There are deep philosophical problems in Artificial Intelligence, and some researchers believe that its goals are impossible or incoherent. Even if Artificial Intelligence is possible there are moral issues to consider such as the exploitation of artificial machines by humans and whether it is ethical to do this. Weizenbaum argues that AI is a threat to human dignity, and that AI should not replace humans in positions that require respect and care.

John McCarthy (Fig. 16.6) has long advocated the use of logic in AI, and mathematical logic has been used in the AI field to formalize knowledge, and in guiding the design of mechanized reasoning systems. Logic has been used as an analytic tool, as a knowledge representation formalism, and as a programming language.

Fig. 16.6 John McCarthy. Courtesy of John McCarthy

McCarthy's long-term goal was to formalize common sense reasoning: i.e. the normal reasoning that is employed in problem solving and dealing with normal events in the real world. McCarthy [6] argues that it is reasonable for logic to play a key role in the formalization of common sense knowledgwe, and this includes the formalization of basic facts about actions and their effects; facts about beliefs and desires; and facts about knowledge and how it is obtained. His approach allows common sense problems to be solved by logical reasoning.

Its formalization requires sufficient understanding of the common sense world, and often the relevant facts to solve a particular problem are unknown. It may be that knowledge thought relevant may be irrelevant and vice versa. A computer may have millions of facts stored in its memory, and the problem is how to determine which of these should be chosen from its memory to serve as premises in logical deduction.

McCarthy's influential 1959 paper discusses various common sense problems such as getting home from the airport. Mathematical logic is the standard approach to express premises, and it includes rules of inferences that are used to deduce valid conclusions from a set of premises. Its rigorous deductive reasoning shows how new formulae may be logically deduced from a set or premises.

McCarthy's approach to programs with common sense has been criticized by Bar-Hillel and others on the grounds that common sense is fairly elusive, and the difficulty that a machine would have in determining which facts are relevant to a particular deduction from its known set of facts. However, McCarthy's approach has showed how logical techniques can contribute to the solution of specific AI problems.

Logic programming languages describe what is to be done, rather than how it should be done. These languages are concerned with the statement of the problem to be solved, rather than how the problem will be solved. These languages use mathematical logic as a tool in the statement of the problem definition. Logic is a useful tool in developing a body of knowledge (or theory), and it allows rigorous mathematical deduction to derive further truths from the existing set of truths. The theory is built up from a small set of axioms or postulates and rules of inference derive further truths logically.

The objective of logic programming is to employ mathematical logic to assist with computer programming. Many problems are naturally expressed as a theory, and the statement of a problem to be solved is often equivalent to determining if a new hypothesis is consistent with an existing theory. Logic provides a rigorous way to determine this, as it includes a rigorous process for conducting proof.

Computation in logic programming is essentially logical deduction, and logic programming languages use first-order[6] predicate calculus. They employ theorem proving to derive a desired truth from an initial set of axioms. These proofs are constructive[7] in that more an actual object that satisfies the constraints is produced rather than a pure existence theorem. Logic programming specifies the objects, the relationships between them and the constraints that must be satisfied for the problem.

- The set of objects involved in the computation
- The relationships that hold between the objects
- The constraints of the particular problem.

The language interpreter decides how to satisfy the particular constraints. Artificial Intelligence influenced the development of logic programming, and John McCarthy[8] demonstrated that mathematical logic could be used for expressing knowledge. The first logic programming language was Planner developed by Carl Hewitt at MIT in 1969. It uses a procedural approach for knowledge representation rather than McCarthy's declarative approach.

The best-known logic programming languages is Prolog, which was developed in the early 1970s by Alain Colmerauer and Robert Kowalski. It stands for ***pro***gramming in ***log***ic. It is a goal-oriented language that is based on predicate logic. Prolog became an ISO standard in 1995. The language attempts to solve a goal by tackling the sub-goals that the goal consists of

$$\text{goal} : -\text{subgoal}_1, \ldots, \text{subgoal}_n.$$

That is, in order to prove a particular goal it is sufficient to prove sub-goal$_1$ through sub-goal$_n$. Each line of a Prolog program consists of a rule or a fact, and the language specifies what exists rather than how. The following program fragment has one rule and two facts

[6]First-order logic allows quantification over objects but not functions or relations. Higher order logics allow quantification of functions and relations.

[7]For example, the statement $\exists x$ such that $x = \sqrt{4}$ states that there is an x such that x is the square root of 4, and the constructive existence yields that the answer is that $x = 2$ or $(x - (-2))$ i.e. constructive existence provides more the truth of the statement of existence, and an actual object satisfying the existence criteria is explicitly produced.

[8]John McCarthy received the Turing Award in 1971 for his contributions to Artificial Intelligence. He also developed the programming language LISP.

```
grandmother(G, S) :-parent(P, S), mother(G, P).
mother(sarah, isaac).
parent(isaac, jacob).
```

The first line in the program fragment is a rule that states that G is the grandmother of S if there is a parent P of S and G is the mother of P. The next two statements are facts stating that Isaac is a parent of Jacob, and that Sarah is the mother of Isaac. A particular goal clause is true if all of its subclauses are true

$$\text{goalclause}\left(V_\text{g}\right) : - \text{ clause}_1(V_1), \ldots, \text{clause}_\text{m}(V_\text{m})$$

A Horn clause consists of a goal clause and a set of clauses that must be proven separately. Prolog finds solutions by *unification*: i.e. by binding a variable to a value. For an implication to succeed, all goal variables V*g* on the left side of: must find a solution by binding variables from the clauses which are activated on the right side. When all clauses are examined and all variables in V*g* are bound, the goal succeeds. But if a variable cannot be bound for a given clause, then that clause fails. Following the failure, Prolog *backtracks*, and this involves going back to the left to previous clauses to continue trying to unify with alternative bindings. Backtracking gives Prolog the ability to find multiple solutions to a given query or goal.

Logic programming languages generally use a simple searching strategy to consider alternatives

- If a goal succeeds and there are more goals to achieve, then remember any untried alternatives and go on to the next goal.
- If a goal is achieved and there are no more goals to achieve then stop with success.
- If a goal fails and there are alternative ways to solve it then try the next one.
- If a goal fails and there are no alternate ways to solve it, and there is a previous goal, then go back to the previous goal.
- If a goal fails and there are no alternate ways to solve it, and no previous goal, then stop with failure.

Constraint programming is a programming paradigm where relations between variables can be stated in the form of constraints. Constraints specify the properties of the solution, and differ from the imperative programming languages in that they do not specify the sequence of steps to execute.

16.7 Theorem Provers for Logic

The word "*proof*" is generally interpreted as facts or evidence that support a particular proposition or belief, and such proofs are conducted in natural language. The proof of a theorem in mathematics requires additional rigour, and such proofs consist of a mixture of natural language and mathematical argument. It is common to skip over the trivial steps in a mathematical proof, and independent mathematicians conduct peer reviews to provide additional confidence in the correctness of the proof, and to ensure that no unwarranted assumptions or errors in reasoning have been made. Proofs conducted in logic are extremely rigorous with every step in the proof is explicit.[9]

Herbert Simon and Alan Newell developed the first theorem prover with their work on a program called 'Logic Theorist' or 'LT' [7]. This program could independently provide proofs of various theorems in Russell's and Whitehead's Principia Mathematica[10] [8]. Russell and Whitehead had attempted to derive all mathematics from axioms and the inference rules of logic, and the LT program conducted proof from a small set of propositional axioms and deduction rules. The LT program succeeded in proving 38 of the 52 theorems in Chap. 2 of Principia Mathematica. Its approach was to start with the theorem to be proved, and to then search for relevant axioms and operators to prove the theorem.

LT was demonstrated at the Dartmouth conference in 1956 (the conference that led to the birth of the Artificial Intelligence field), and it showed that computers had the ability to encode knowledge and information, and to perform intelligent operations such as solving theorems in mathematics. The heuristic approach of the LT program tried to emulate human mathematicians, and could not guarantee that a proof could be found for every valid theorem.

The proof of theorems in formal verification of computer system often involves several million formulae and manual proof is error prone. There are several tools available to support theorem proving, and these include the Boyer–Moore theorem prover (known as NQTHM); the Isabelle theorem prover; and the HOL system.

B.S. Boyer and J.S. Moore developed the Boyer–Moore theorem prover in the early 1970s [9]. It has been improved since then and it is currently known as NQTHM (it has been superseded by ACL2 available from the University of Texas).

It has been effective in proving well-known theorems such as Goedel's Incompleteness Theorem, the insolvability of the Halting problem, a formalization of the Motorola MC 68020 Microprocessor, and many more.

[9]Perhaps a good analogy might be that a mathematical proof is like a program written in a high-level language such as C, whereas a formal proof in logic is like a program written in assembly language.

[10]Russell is said to have remarked that he was delighted to see that the Principia Mathematica could be done by machine, and that if he and Whitehead had known this in advance that they would not have wasted 10 years doing this work by hand in the early twentieth century.

Computational Logic Inc. was a company founded by Boyer and Moore in 1983 to share the benefits of a formal approach to software development with the wider computing community. It was based in Austin, Texas, and provided services in the mathematical modelling of hardware and software systems. This involved the use of mathematics and logic to formally specify microprocessors and other systems. The use of its theorem prover was to formally verify that the implementation meets its specification: i.e. to prove that the microprocessor or other system satisfies its specification.

Isabelle is a theorem proving environment developed at Cambridge University by Larry Paulson and Tobias Nipkow of the Technical University of Munich. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas. The main application is the formalization of mathematical proofs, and proving the correctness of computer hardware or software with respect to its specification, and proving properties of computer languages and protocols.

Isabelle is a generic theorem prover in the sense that it has the capacity to accept a variety of formal calculi, whereas most other theorem provers are specific to a specific formal calculus. Isabelle is available free of charge under an open source license.

The HOL system is an environment for interactive theorem proving in a higher order logic. The HOL system has been applied to the formalization of mathematics and the verification of hardware. It was originally developed at Cambridge University in the United Kingdom in the early 1980s, and HOL 4 is the latest version and is an open source project. It is used by academia and industry.

There is a steep learning curve with the theorem provers above and it generally takes a couple of months for users to become familiar with them. However, automated theorem proving has become a useful tool in the verification of integrated circuit design. Several semiconductor companies use automated theorem proving to demonstrate the correctness of division and other operators on their processors.

16.8 Review Questions

1. What is fuzzy logic?
2. What is intuitionist logic and how is it different from classical logic?
3. Discuss the problem of undefinedness and the advantages and disadvantages of three-valued logics. Describe the approaches of Parnas, Dijkstra and Jones.
4. What is temporal logic?

5. Show how the temporal operators may be expressed in classical mathematics. Discuss the merits of temporal operators.
6. Investigate the Isabelle (or another) theorem proving environment and determine the extent to which it may assist with proof.
7. Discuss the applications of logic to AI.

16.9 Summary

We discussed some advanced topics in logic in this chapter, including fuzzy logic, temporal logic, intuitionist logic, undefined values, logic and AI and theorem provers. Fuzzy logic is an extension of classical logic that acts as a mathematical model for vagueness, whereas temporal logic is concerned with the expression of properties that have time dependencies

Intuitionism was a controversial school of mathematics that aimed to provide a solid foundation for mathematics. Its adherents rejected the law of the excluded middle, and insisted that for an entity to exist that there must be a constructive proof of its existence. Martin Löf applied intuitionistic logic to type theory in the 1970s.

Partial functions arise naturally in computer science, and such functions may fail to be defined for one or more values in their domain. There are a number of approaches to deal with undefined values, including the logic of partial functions; Dijkstra's approach with his cand and cor operators; and Parnas's approach which preserves a classical two-valued logic.

We discussed temporal logic and its applications to the safety critical field, including the specification of properties with time dependencies. We discussed the application of logic to the AI field, and logic has been used to formalize knowledge in an AI systems. Finally, we discussed some of the existing theorem provers, and their applications in providing a rigorous proof of a theorem, and in avoiding errors or jumps in reasoning.

References

1. Temporal Logic. Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/logic-temporal/
2. Intuitionist Logic. An Introduction. A. Heyting. North-Holland Publishing. 1966.
3. Intuitionist Type Theory. Per Martin Löf. Notes by Giovanni Savin of lectures given in Padua, June, 1980. Bibliopolis. Napoli. 1984.
4. Predicate Calculus for Software Engineering. David L.Parnas. IEEE.
5. Systematic Software Development using VDM. Cliff Jones. Prentice Hall International. 1986.

6. Programs with Common Sense. John McCarthy. Proceedings of the Teddington Conference on the Mechanization of Thought Processes. 1959.
7. A. Newell and H. Simon. The Logic Theory Machine. IRE Transactions on Information Theory, 2, 61–79. 1956.
8. Principia Mathematica. B.Russell and A.N. Whitehead. Cambridge University Press. Cambridge. 1910.
9. A Computational Logic. The Boyer Moore Theorem Prover. Robert Boyer and J.S. Moore. Academic Press. 1979.