11

Knowledge Representation and Reasoning

For a system to be intelligent, it must have knowledge about its world and the means to draw conclusions from, or at least act on, that knowledge. Humans and machines alike therefore must have ways to represent this needed knowledge in internal structures, whether encoded in protein or silicon. Cognitive scientists and AI researchers distinguish between two main ways in which knowledge is represented: procedural and declarative. In animals, the knowledge needed to perform a skilled action, such as hitting a tennis ball, is called procedural because it is encoded directly in the neural circuits that coordinate and control that specific action. Analogously, automatic landing systems in aircraft contain within their control programs procedural knowledge about flight paths, landing speeds, aircraft dynamics, and so on. In contrast, when we respond to a question, such as "How old are you?," we answer with a declarative sentence, such as "I am twenty-four years old." Any knowledge that is most naturally represented by a declarative sentence is called declarative.

In AI research (and in computer science generally), procedural knowledge is represented directly in the programs that use that knowledge, whereas declarative knowledge is represented in symbolic structures that are more-or-less separate from the many different programs that might use the information in those structures. Examples of declarative-knowledge symbol structures are those that encode logical statements (such as those McCarthy advocated for representing world knowledge) and those that encode semantic networks (such as those of Raphael or Quillian). Typically, procedural representations, specialized as they are to particular tasks, are more efficient (when performing those tasks), whereas declarative ones, which can be used by a variety of different programs, are more generally useful. In this chapter, I'll describe some of the ideas put forward during this period for reasoning with and for representing declarative knowledge.

11.1 Deductions in Symbolic Logic

Aristotle got things started in logic with his analysis of syllogisms. In the nineteenth century, George Boole developed the foundations of propositional logic, and Gottlob Frege improved the expressive power of logic by proposing a language that could include internal components (called "terms") as part of propositions. Later developments by various logicians gave us what we call today the predicate calculus – the very language in which McCarthy proposed to represent the knowledge needed by an intelligent system.

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

Here is an instance of one of Aristotle's syllogisms, stated in the language of the predicate calculus:

1. (∀x)[Man(x) ⊃ Mortal(x)]
   (The expression "(∀x)" is a way of writing "for all x"; and the expression "⊃" is a way of writing "implies that." "Man(x)" is a way of writing "x is a man"; and "Mortal(x)" is a way of writing "x is mortal." Thus, the entire expression is a way of writing "for all x, x is a man implies that x is mortal" or, equivalently, "all men are mortal.")
2. Man(Socrates)
   (Socrates is a man.)
3. Therefore, Mortal(Socrates)
   (Socrates is mortal.)

Statement 3, following "Therefore," is an example of a *deduction* in logic. McCarthy proposed that the knowledge that an intelligent agent might need in a specific situation could be deduced from the general knowledge given to it earlier. Thus, for McCarthy-style AI, not only do we need a language (perhaps that of the predicate calculus) but a way to make the necessary deductions from statements in the language.

Logicians have worked out a variety of deduction methods based on what they call "rules of inference." For example, one important inference rule is called *modus ponens* (Latin for "mode that affirms"). It states that if we have the two logical statements P and P ⊃ Q, then we are justified in deducing the statement Q.

By the 1960s programs had been written that could use inference rules to prove theorems in the predicate calculus. Chief among these were those of Paul Gilmore at IBM,[1] Hao Wang at IBM,[2] and Dag Prawitz,[3] now at Stockholm University. Although their programs could prove simple theorems, proving more complex ones would have required too much search.[4]

A Harvard Ph.D. student, Fisher Black (1938–1995), later a co-inventor of the Black–Scholes equation for pricing options,[5] had done early work implementing some of McCarthy's ideas.[6] But it was a Stanford Ph.D. student and SRI researcher, C. Cordell Green, who programmed a system, QA3, that more fully realized McCarthy's recommendation. Although it was not difficult to represent world knowledge as logical statements, what was lacking at the time of Black's work was an efficient mechanical method to deduce conclusions from these statements. Green was able to employ a new method for efficient reasoning developed by John Alan Robinson.

During the early 1960s, the English (and American) mathematician and logician John Alan Robinson (1930–) developed a deduction method particularly well suited to computer implementation. It was based on an inference rule he called "resolution."[7] Although a full description of resolution would involve too much technical detail, it is a rule (as *modus ponens* is) whose application produces a new statement from two other statements. For example, resolution applied to the two statements ¬P ∨ Q and P produces Q. (The symbol "¬" is a way of writing "not," and the symbol "∨" is a way of writing "or.") Resolution can be thought of as canceling out the P and the ¬P in the two statements. (Resolution is a kind of generalized *modus*

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

ponens as can be seen from the fact that $\neg P \vee Q$ is logically equivalent to $P \supset Q$.) This example was particularly simple because the statements had no internal terms. Robinson's key contribution was to show how resolution could be applied to general expressions in the predicate calculus, expressions such as $\neg P(x) \vee Q(x)$ with internal terms.

The advantage of resolution is that it can be readily implemented in programs to make deductions from a set of logical statements. To do so, the statements must first be converted to a special form consisting of what logicians call "clauses." (Loosely speaking, a clause is a formula that uses only $\vee$'s and $\neg$'s.) Any logical statement can be converted to clause form (although some, such as John McCarthy, complain that conversion might eliminate clues about how statements might best be used in logical deductions).

The first use of resolution was in computer programs to prove mathematical theorems. (Technically, a "theorem" is any logical statement obtained by successively applying a rule of inference, such as resolution, to members of a base set of logical statements, called "axioms," and to statements deduced from the axioms.) Groups at Argonne National Laboratories (under Lawrence Wos), at the University of Texas at Austin (under Woody Bledsoe), and at the University of Edinburgh (under Bernard Meltzer) soon began work developing theorem-proving programs based on resolution. These programs were able to prove theorems that had previously been proved "by hand" and even some new, never-before-proved, mathematical theorems.[8] One of these latter concerned a conjecture by Herbert Robbins that a Robbins algebra was Boolean. The conjecture was proved in 1996 by William McCune, using an automated theorem prover.[9]

Our concern here, though, is with using deduction methods to automate the reasoning needed by intelligent systems. Around 1968, Green (aided by another Stanford student, Robert Yates) programmed, in LISP, a resolution-based deduction system called QA3, which ran on SRI's time-shared SDS 940 computer. (QA1, Green's first effort, guided by Bertram Raphael at SRI, was an attempt to improve on Raphael's earlier SIR system. QA2 was Green's first system based on resolution, and QA3 was a more sophisticated and practical descendant.) "QA" stood for "question answering," one of the motivating applications.

I'll present a short illustrative example of QA3's question-answering ability taken from Green's Stanford Ph.D. thesis.[10] First, two statements are given to the system, namely,

1. $ROBOT(Rob)$
   (Rob is a robot.)
2. $(\forall x)[MACHINE(x) \supset \neg ANIMAL(x)]$
   (x is a machine implies that it is not an animal.)

The system is then asked "Is everything an animal?" by having it attempt to deduce the statement

3. $(\forall x)ANIMAL(x)$

QA3 not only answers "NO," finding that such a deduction is impossible, but it also gives a "counterexample" as an answer to the question:

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

4. x = Rob
   (This indicates that ¬ANIMAL(Rob) contradicts what was to be deduced.)

The use of resolution, like that of any inference rule, to deduce some specific conclusion from a large body of logical statements involves the need to decide to which two statements, among the many possibilities, the rule should be applied. Then a similar decision must be made again and again until, one hopes, finally the desired conclusion is obtained. So just as with programs for playing checkers, solving puzzles, and proving geometry theorems, deduction programs are faced with the need to try many possibilities in their search for a solution. As with those other programs, various heuristic search methods have been developed for deduction programs.

11.2 The Situation Calculus

Green realized that "question answering" was quite a broad topic. One could ask questions about almost anything. For example, one could ask "What is a program for rearranging a list of numbers so that they are in increasing numerical order?" Or one could ask, "What is the sequence of steps a robot should take to assemble a tower of toy blocks?" The key to applying QA3 to answer questions of this sort lay in using McCarthy's "situation calculus."

McCarthy proposed a version of logic he called the "situation calculus" in which one could write logical statements that explicitly named the situation in which something or other was true. For example, one toy block may be on top of another in one situation but not in another. Green developed a version of McCarthy's logic in which the situation, in which something was true, appeared as one of the terms in an expression stating that something was true. For example, to say that block A is on top of block B in some situation S (allowing for the fact that this might not be the case in other situations), Green would write

$$\mathrm{On}(A, B, S),$$

to say that block A is blue in all situations, Green would write

$$(\forall s)\mathrm{Blue}(A, s),$$

and to say that there exists *some* situation in which block A is on block B, Green would write

$$(\exists s)\mathrm{On}(A, B, s).$$

Here "(∃s)" is a way of writing "there exists some s such that . . . "

Not only was QA3 able to deduce statements, but when it deduced a so-called existential statement (such as the one just mentioned), it was able to compute an instance of what was alleged to "exist." Thus, when it deduced the statement (∃s)On(A, B, s), it also computed for which situation the deduction was valid. Green devised a way in which this value could be expressed in terms of a list of actions for a robot that would change some initial situation into the situation for which the

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

Figure 11.1. Robert Kowalski (left) and Alain Colmerauer (right). (Photographs courtesy of Robert Kowalski and of Alain Colmerauer.)

deduced statement was true. Thus, for example, QA3 could be used to plan courses of action for a robot. Later, we'll see how it was used for this purpose.

11.3 Logic Programming

In the same way that QA3 could be used to make robot plans, it could also construct simple computer programs. In his 1969 paper, Green wrote

> The formalization given here [can] be used to precisely state and solve the problem of automatic generation of programs, including recursive programs, along with concurrent generation of proofs of the correctness of these programs. Thus any programs automatically written by this method have no errors.

Green's work on automatic programming was the first attempt to write programs using logical statements. Around this time, Robert A. Kowalski (1941– ; Fig. 11.1), an American who had just earned a Ph.D. at the University of Edinburgh, and Donald Kuehner developed a more efficient version of Robinson's resolution method, which they called "SL-resolution."[11] In the summer of 1972, Kowalski visited Alain Colmerauer (1941– ; Fig. 11.1), the head of Groupe d'Intelligence Artificielle (GIA), Centre National de la Recherche Scientifique and Université II of Aix-Marseille in Marseille. Kowalski wrote "It was during that second visit that logic programming, as we commonly understand it, was born."[12]

Colmerauer and his Ph.D. student, Philippe Roussel, were the ones who developed, in 1972, the new programming language, PROLOG. (Roussel chose the name as an abbreviation for "PROgrammation en LOGique.") In PROLOG, programs consist of an ordered sequence of logical statements. The exact order in which these statements are written, along with some other constructs, is the key to efficient program execution. PROLOG uses an ordering based on the ordering of deductions

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

in SL-resolution. Kowalski, Colmerauer, and Roussel all share credit for PROLOG, but Kowalski admits ". . . it is probably fair to say that my own contributions were mainly philosophical and Alain's were more practical."[13]

The PROLOG language gradually grew in importance to rival LISP, although it is used mainly by AI people outside of the United States. Some American researchers, especially those at MIT, argued against PROLOG (and other resolution-based deduction systems), claiming (with some justification) that computation based on deduction was not efficient. They advocated computation controlled by embedding knowledge about the problem being solved and how best to solve it directly into programs to reduce search. This "procedural embedding of knowledge" was a feature of the PLANNER languages developed by Carl Hewitt and colleagues at MIT. (Hewitt coined the phrase "procedural embedding of knowledge" in a 1971 paper.)[14]

11.4 Semantic Networks

Semantic networks were (and still are) another important format for representing declarative knowledge. I have already mentioned their use by Ross Quillian as a model of human long-term memory. In the 1970s, Stanford cognitive psychologist Gordon Bower (1932–) and his student John Anderson (1947–) presented a network-based theory of human memory in their book *Human Associative Memory*.[15] According to a biographical sketch of Anderson, the book "immediately attracted the attention of everyone then working in the field. The book played a major role in establishing propositional semantic networks as the basis for representation in memory and spreading activation through the links in such networks as the basis for retrieval of information from memory."[16]

The theory was partially implemented in a computer simulation called HAM (an acronym for Human Associative Memory). HAM could parse simple propositional sentences and store them in a semantic network structure. Using its accumulated memory, HAM could answer simple questions.

Several other network-based representations were explored during the late 1960s and early 1970s. Robert F. Simmons, after moving from SDC to the University of Texas in Austin, began using semantic networks as a computational linguistic theory of structures and processing operations required for computer understanding of natural language. He wrote "Semantic nets are simple – even elegant – structures for representing aspects of meaning of English strings in a convenient computational form that supports useful language-processing operations on computers."[17]

In 1971, Stuart C. Shapiro (1944–), then at the University of Wisconsin in Madison, introduced a network structure called MENS (MEmory Net Structure) for storing semantic information.[18] An auxiliary system called MENTAL (MEmory Net That Answers and Learns) interacted with a user and with MEMS. MENTAL aided MEMS in deducing new information from that already stored. Shapiro envisioned that MENTAL would be able to answer users' questions using information stored in MEMS.

Shapiro later moved to the State University of New York at Buffalo where he and colleagues are continuing to develop a series of systems called SNePS (Semantic NEtwork Processing System).[19] SNePS combines features of logical representations

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

Figure 11.2. Roger Schank. (Photograph courtesy of Roger Schank.)

with those of network representations and has been used for natural language understanding and generation and other applications.[20]

In his Ph.D. research in linguistics at the University of Texas at Austin, Roger C. Schank (1946– ; Fig. 11.2) began developing what he called "conceptual dependency representations for natural language sentences."[21] Subsequently, as a Professor at Stanford and at Yale, he and colleagues continued to develop these ideas. The basis of Schank's work was his belief that people transform natural language sentences into "conceptual structures" that are independent of the particular language in which the sentences were originally expressed. These conceptual structures, he claimed, were how the information in sentences is understood and remembered. So, for example, when one translates a sentence from one language into another, one first represents its information content as a conceptual structure and then uses that structure to reason about what was said or to regenerate the information as a sentence in another language. As he put it in one of his papers, ". . . any two utterances that can be said to mean the same thing, whether they are in the same or different languages, should be characterized in only one way by the conceptual structures."[22]

The notation Schank used for his conceptual structures (sometimes called "conceptual dependency graphs") evolved somewhat during the 1970s.[23] As an example, Fig. 11.3, taken from one of his papers, shows how he would represent the sentence "John threw the pencil to Sam." This structure uses three of the "primitive actions" Schank has defined for these representations. These are ATRANS, which means a transfer of possession; PTRANS, which means a transfer of physical location; and PROPEL, which means an application of force to an object. Schank defined several other primitive actions to represent movement, attending to, speaking, transferring of ideas, and so on.

An expanded literal reading of what this structure represents would be "John applied physical force to a pencil, which caused it to go through the air from John's location to Sam's location, which caused Sam to possess it" or something like that. Schank, like many others who are interested in meaning representation languages, notes that these representations can be used directly to perform deductions and answer questions. For example, answers to questions such as "How did Sam get the pencil?" and "Who owned the pencil after John threw it?" are easily extracted.

Although network structures are illustrated graphically in papers about them, they were encoded using LISP for computer processing.

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

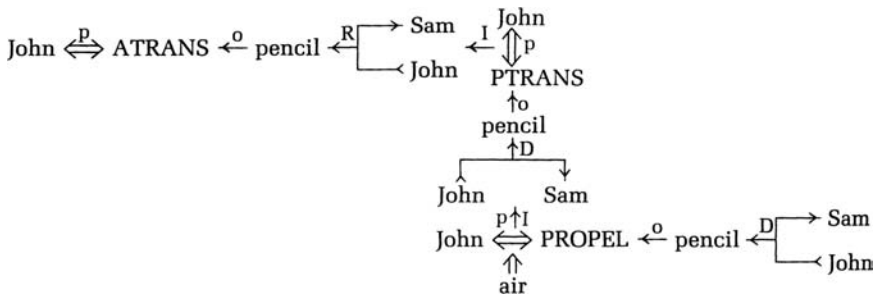

Figure 11.3. Conceptual structure for "John threw the pencil to Sam." (From Roger C. Schank, "Identification of Conceptualizations Underlying Natural Langauge," in Roger Schank and Kenneth Colby (eds.), *Computer Models of Thought and Language*, p. 226, San Francisco: W. H. Freeman and Co., 1973.)

11.5 Scripts and Frames

Graphical knowledge representations, such as semantic networks and conceptual structures, connect related entities together in groups. Such groupings are efficient computationally because things that are related often participate in the same chain of reasoning. When accessing one such entity it is easy to access close-by ones also. Roger Schank and Robert Abelson expanded on this idea by introducing the concept of "scripts."[24] A script is a way of representing what they call "specific knowledge," that is, detailed knowledge about a situation or event that "we have been through many times." They contrast specific knowledge with "general knowledge," the latter of which is the large body of background or commonsense knowledge that is useful in many situations.

Their "restaurant" script ("Coffee Shop version") became their most famous illustrative example. The script consists of four "scenes," namely, Entering, Ordering, Eating, and Exiting. Its "Props" are Tables, Menu, F-Food, Check, and Money. Its "Roles" are S-Customer, W-Waiter, C-Cook, M-Cashier, and O-Owner. Its "Entry conditions" are S is hungry and S has money. Its "Results" are S has less money, O has more money, S is not hungry, and S is pleased (optional). Figures 11.4 shows their script for the "Ordering" scene.

Besides the actions PTRANS (transfer of location) and ATRANS (transfer of possession), this script uses two more of their primitive actions, namely, MTRANS (transfer of information) and MBUILD (creating or combining thoughts). CP(S) stands for S's "conceptual processor" where thought takes place, and DO stands for a "dummy action" defined by what follows. The lines in the diagram show possible alternative paths through the script. So, for example, if the menu is already on the table, the script begins at the upper left-hand corner; otherwise it begins at the upper right-hand corner. I believe most of the script is self-explanatory, but I'll help out by explaining what goes on in the middle. S brings the "food list" into its central processor where it is able to mentally decide (build) a choice of food. S then transfers information to the waiter to come to the table, which the waiter does. Then, S transfers the information about his or her choice of food to the waiter. This continues

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

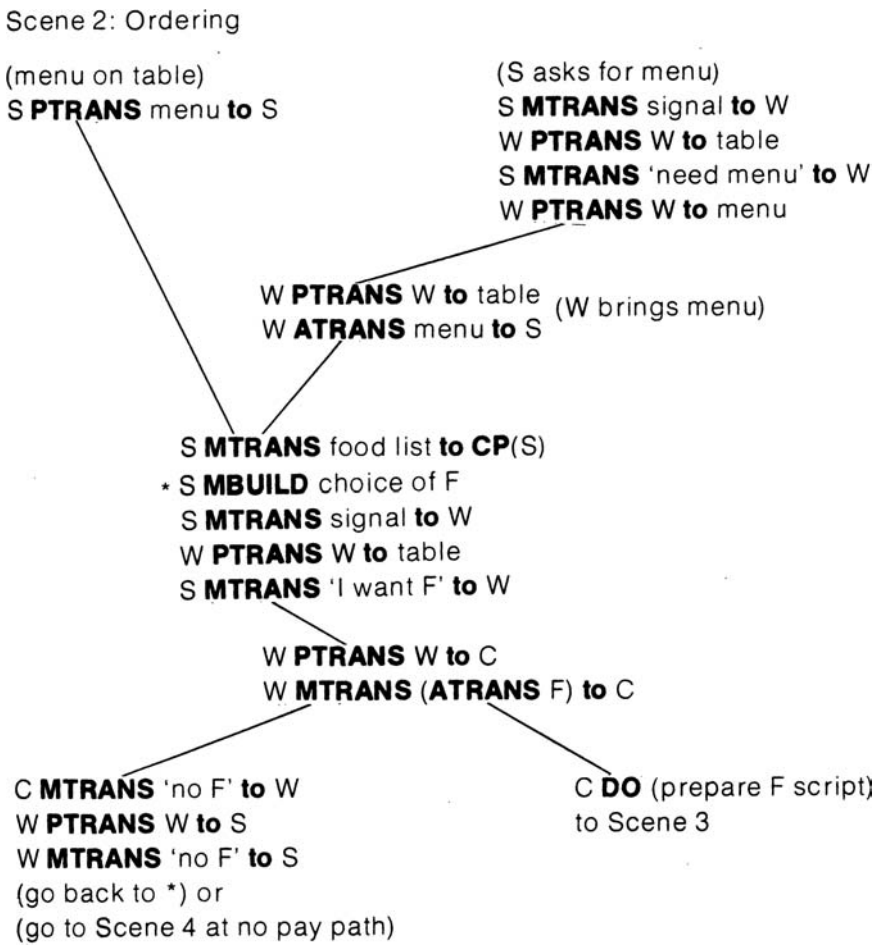

Figure 11.4. A scene in the restaurant script. (From Roger C. Schank and Robert P. Abelson, *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*, p. 43, Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.)

until either the cook tells the waiter that he does not have the food that is ordered or the cook prepares the food. The three other scenes in the restaurant script are similarly illustrated in Schank and Abelson's book.[25] Several other variations of the restaurant script (for different kinds of restaurants, and so on) are possible.

Scripts help explain some of the reasoning we do automatically when we hear a story. For example, if we hear that John went to a coffee shop and ordered lasagne, we can reasonably assume that lasagne was on the menu. If we later learn that John had to order something else instead, we can assume that the coffee shop was out of lasagne. Schank and Abelson give anecdotal evidence that even small children build such scripts and that people must have a great number of them to enable them to navigate through and reason about situations they encounter.

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

Schank later expanded on scripts and related ideas in another book, in which he introduced the idea of "memory organization packets" (MOPS) that describe situations in a more distributed and dynamic way than scripts do.[26] He later "revisited" some of these ideas in a book about their application to education, a field to which he has made significant contributions.[27]

Schank and his claims generated a good deal of controversy among AI researchers. For example, I remember arguing with him in 1983 in a restaurant somewhere (while waiting for the menu?) about the comparative performance of his programs for natural language understanding and that of our programs at SRI. As I recall, he was eager to make more grandiose claims about what his programs could do than I was prepared to believe or to claim about ours. Tufts University philosopher Daniel Dennett is quoted as having said "I've always relished Schank's role as a gadfly and as a naysayer, a guerrilla in the realm of cognitive science, always asking big questions, always willing to discard his own earlier efforts and say they were radically incomplete for interesting reasons. He's a gadfly and a good one."[28] I think his basic idea about scripts was prescient. Also, he has produced a great bunch of students. The "AI Genealogy" Web site[29] lists almost four dozen Schank students, many of whom have gone on to distinguished careers.

Around the time of Schank's work, Marvin Minsky proposed that knowledge about situations be represented in structures he called "frames."[30] He mentioned Schank's ideas (among others) as exemplary of a movement away from "trying to represent knowledge as collections of separate, simple fragments" such as sentences in a logical language. As he defined them,

> A *frame* is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.
> . . .
> Collections of related frames are linked together into *frame-systems*. The effects of important actions are mirrored by transformations between the frames of a system. These are used to make certain kinds of calculations economical, to represent changes of emphasis and attention, and to account for the effectiveness of "imagery."

Minsky's paper described how frame systems could be applied to vision and imagery, linguistic and other kinds of understanding, memory acquisition, retrieval of knowledge, and control. Although his paper was rich in ideas, Minsky did not actually implement any frame systems. A couple of years later, some of his students and former students did implement some framelike systems. One, called FRL (for Frame Representation Language), was developed by R. Bruce Roberts and Ira P. Goldstein.[31] Daniel Bobrow and Terry Winograd (the latter being one of Papert's students), implemented a more ambitious system called KRL (for Knowledge Representation Language).[32]

Frame systems accommodated a style of reasoning in which details "not specifically warranted" could be assumed, thus "bypassing "logic," as Minsky would have it. This style was already used earlier in Raphael's SIR system (see p. 98), and

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

researchers advocating the use of logical languages for knowledge representation would later extend logic in various ways to accommodate this style also. Even so, the last section (titled "Criticism of the Logistic Approach") of Minsky's paper about frames gives many reasons why one might doubt (along with Minsky) "the feasibility of representing ordinary knowledge effectively in the form of many small, independently 'true' propositions."

Notes

1. Paul C. Gilmore, "A Proof Method for Quantification Theory: Its Justification and Realization," *IBM Journal of Research and Development*, Vol. 4, pp. 28–35, 1960. [150]
2. Hao Wang, "Proving Theorems by Pattern Recognition," *Communications of the ACM*, Vol. 4, No. 3, pp. 229–243, 1960, and Hao Wang, "Toward Mechanical Mathematics," *IBM Journal of Research and Development*, Vol. 4, pp. 2–21, 1960. [150]
3. D. Prawitz, H. Prawitz, and N. Voghera, "A Mechanical Proof Procedure and Its Realization in an Electronic Computer," *Journal of the Association for Computing Machinery*, Vol. 7, pp. 102–128, 1960. [150]
4. For additional background and history about automated deduction, see Wolfgang Bibel, "Early History and Perspectives of Automated Deduction," in J. Hertzberg, M. Beetz, and R. Englert (eds.), *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI-2007)*, Lecture Notes on Artificial Intelligence, pp. 2–18, Berlin: Springer-Verlag, 2007. [150]
5. In 1997, Myron Scholes and Robert C. Merton were awarded a Nobel Prize in economics for their option-pricing work. Black died of cancer in 1995. The Nobel Prize is not given posthumously; however, in its announcement of the award, the Nobel committee prominently mentioned Black's key role. [150]
6. Fischer Black, "A Deductive Question-Answering System," Ph.D. dissertation, Harvard University, June 1964. Reprinted in Marvin Minsky (ed.), *Semantic Information Processing*, pp. 354–402, Cambridge, MA: MIT Press, 1968. [150]
7. John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, Vol. 12, No. 1, pp. 23–41, 1965. [150]
8. For a description of some of this work, see Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle, *Automated Reasoning: Introduction and Applications*, second edition, New York: McGraw-Hill, 1992. For more recent work, visit Larry Wos's Web page at http://www.mcs.anl.gov/~wos/ [151]
9. William McCune, "Solution of the Robbins Problem," *Journal of Automated Reasoning*, Vol. 19, No. 3, pp. 263–276, 1997. [151]
10. Available as an SRI Technical Note: C. Green, "Application of Theorem Proving to Problem Solving," Technical Note 4, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, March 1969. Online version available at http://www.ai.sri.com/pubs/files/tn004-green69.pdf. See also C. Green, "Theorem Proving by Resolution as a Basis for Question-Answering Systems," in B. Meltzer and D. Michie, *Machine Intelligence 4*, pp. 183ff, Edinburgh: Edinburgh University Press, 1969, and C. Green, "Applications of Theorem Proving to Problem Solving," reprinted from a 1969 IJCAI conference article in B. L. Webber and N. J. Nilsson (eds.), *Readings in Artificial Intelligence*, pp. 202–222, San Francisco: Morgan Kaufmann, 1981. [151]
11. Robert A. Kowalski and Donald Kuehner, "Linear Resolution with Selection Function," *Artificial Intelligence*, Vol. 2, Nos. 3–4, pp. 227–260, 1971. [153]
12. From one of Kowalski's Web pages: http://www.doc.ic.ac.uk/~rak/history.html. [153]

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

13. http://www.doc.ic.ac.uk/~rak/history.html. For Colmerauer and Roussel's account of the birth of PROLOG see Alain Colmerauer and Philippe Roussel, "The Birth of PROLOG, in Thomas J. Bergin and Richard G. Gibson (eds.), *Programming Languages*, New York: ACM Press, Addison-Wesley, 1996. Available online at http://alain.colmerauer.free.fr/ArchivesPublications/HistoirePrologue/19november92.pdf. [154]
14. See Carl Hewitt, "Procedural Embedding of Knowledge in PLANNER," *Proceedings of the Second International Joint Conference on Artificial Intelligence*, pp. 167–182, Los Altos, CA: Morgan Kaufmann Publishing Co., 1971. [154]
15. John R. Anderson and Gordon H. Bower, *Human Associative Memory*, Washington, DC: Winston and Sons, 1973. [154]
16. From the Web site http://rumelhartprize.org/john.htm. [154]
17. Robert F. Simmons, "Semantic Networks: Computation and Use for Understanding English Sentences," in Roger Schank and Kenneth Colby (eds.), *Computer Models of Thought and Language*, pp. 63–113, San Francisco: W. H. Freeman and Co., 1973. [154]
18. Stuart C. Shapiro, "A Net Structure for Semantic Information Storage, Deduction and Retrieval," *Proceedings of the Second International Joint Conference on Artificial Intelligence*, pp. 512–523, Los Altos, CA: Morgan Kaufmann Publishing Co., 1971. [154]
19. An early paper is Stuart C. Shapiro, "The SNePS Semantic Network Processing System," in Nicholas V. Findler (ed.), *Associative Networks: The Representation and Use of Knowledge by Computers*, pp. 179–203, New York: Academic Press, 1979. [154]
20. The SNePS Web page is at http://www.cse.buffalo.edu/sneps/. [155]
21. Roger C. Schank, "A Conceptual Dependency Representation for a Computer-Oriented Semantics," Ph.D. thesis, University of Texas at Austin, 1969. Available as Stanford AI Memo 83 or Computer Science Technical Note 130, Computer Science Department, Stanford University, Stanford, CA, 1969. [155]
22. Roger C. Schank, "Identification of Conceptualizations Underlying Natural Language," in Roger Schank and Kenneth Colby (eds.), *Computer Models of Thought and Language*, pp. 187–247, San Francisco: W. H. Freeman and Co., 1973. [155]
23. Interested readers might refer to various of his books and papers – for example, Roger C. Schank, "Conceptual Dependency: A Theory of Natural Language Understanding" *Cognitive Psychology*, Vol. 3, pp. 552–631, 1972, and Roger C. Schank, *Conceptual Information Processing*, New York: Elsevier, 1975. [155]
24. Roger C. Schank and Robert P. Abelson, *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1977. [156]
25. *Ibid*. [157]
26. Roger C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge: Cambridge University Press, 1982. [158]
27. Roger C. Schank, *Dynamic Memory Revisited*, Cambridge: Cambridge University Press, 1999. [158]
28. See http://www.edge.org/3rd_culture/bios/schank.html. [158]
29. See http://aigp.csres.utexas.edu/~aigp/researcher/show/192. [158]
30. Marvin Minsky, "A Framework for Representing Knowledge," MIT AI Laboratory Memo 306, June 1974. Reprinted in Patrick Winston (ed.), *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975. Available online at http://web.media.mit.edu/~minsky/papers/Frames/frames.html. [158]
31. R. Bruce Roberts and Ira P. Goldstein, *The FRL Primer*, Massachusetts Institute of Technology AI Laboratory Technical Report AIM-408, July 1977; available online at ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-408.pdf. [158]

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press

32. Daniel G. Bobrow and Terry A. Winograd, "An overview of KRL, a Knowledge Representation Language," Report Number CS-TR-76-581, Department of Computer Science, Stanford University, November 1976. Available online at ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/76/581/CS-TR-76-581.pdf. Appeared later as Daniel Bobrow and Terry Winograd, "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, Vol. 1, No. 1, pp. 3–46, January 1977. [158]

https://doi.org/10.1017/CBO9780511819346.014 Published online by Cambridge University Press