THE QUEST FOR ARTIFICIAL INTELLIGENCE

A HISTORY OF IDEAS AND ACHIEVEMENTS

Web Version
Print version published by Cambridge University Press
http://www.cambridge.org/us/0521122937

Nils J. Nilsson
Stanford University

For Grace McConnell Abbott,

my wife and best friend

Contents

Preface

Artificial intelligence (AI) may lack an agreed-upon definition, but someone writing about its history must have some kind of definition in mind. For me, artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment. According to that definition, lots of things – humans, animals, and some machines – are intelligent. Machines, such as "smart cameras," and many animals are at the primitive end of the extended continuum along which entities with various degrees of intelligence are arrayed. At the other end are humans, who are able to reason, achieve goals, understand and generate language, perceive and respond to sensory inputs, prove mathematical theorems, play challenging games, synthesize and summarize information, create art and music, and even write histories. Because "functioning appropriately and with foresight" requires so many different capabilities, depending on the environment, we actually have several continua of intelligences with no particularly sharp discontinuities in any of them. For these reasons, I take a rather generous view of what constitutes AI. That means that my history of the subject will, at times, include some control engineering, some electrical engineering, some statistics, some linguistics, some logic, and some computer science.

There have been other histories of AI, but time marches on, as has AI, so a new history needs to be written. I have participated in the quest for artificial intelligence for fifty years – all of my professional life and nearly all of the life of the field. I thought it would be a good idea for an "insider" to try to tell the story of this quest from its beginnings up to the present time.

I have three kinds of readers in mind. One is the intelligent lay reader interested in scientific topics who might be curious about what AI is all about. Another group, perhaps overlapping the first, consists of those in technical or professional fields who, for one reason or another, need to know about AI and would benefit from a complete picture of the field – where it has been, where it is now, and where it might be going. To both of these groups, I promise no complicated mathematics or computer jargon, lots of diagrams, and my best efforts to provide clear explanations of how AI programs and techniques work. (I also include several photographs of AI people. The selection of these is

somewhat random and doesn't necessarily indicate prominence in the field.)

A third group consists of AI researchers, students, and teachers who would benefit from knowing more about the things AI has tried, what has and hasn't worked, and good sources for historical and other information. Knowing the history of a field is important for those engaged in it. For one thing, many ideas that were explored and then abandoned might now be viable because of improved technological capabilities. For that group, I include extensive end-of-chapter notes citing source material. The general reader will miss nothing by ignoring these notes. The main text itself mentions Web sites where interesting films, demonstrations, and background can be found. (If links to these sites become broken, readers may still be able to access them using the "Wayback Machine" at http://www.archive.org.)

The book follows a roughly chronological approach, with some backing and filling. My story may have left out some actors and events, but I hope it is reasonably representative of AI's main ideas, controversies, successes, and limitations. I focus more on the ideas and their realizations than on the personalities involved. I believe that to appreciate AI's history, one has to understand, at least in lay terms, something about how AI programs actually work.

If AI is about endowing machines with intelligence, what counts as a machine? To many people, a machine is a rather stolid thing. The word evokes images of gears grinding, steam hissing, and steel parts clanking. Nowadays, however, the computer has greatly expanded our notion of what a machine can be. A functioning computer system contains both hardware and software, and we frequently think of the software itself as a "machine." For example, we refer to "chess-playing machines" and "machines that learn," when we actually mean the programs that are doing those things. The distinction between hardware and software has become somewhat blurred because most modern computers have some of their programs built right into their hardware circuitry.

Whatever abilities and knowledge I bring to the writing of this book stem from the support of many people, institutions, and funding agencies. First, my parents, Walter Alfred Nilsson (1907–1991) and Pauline Glerum Nilsson (1910–1998), launched me into life. They provided the right mixture of disdain for mediocrity and excuses (Walter), kind care (Pauline), and praise and encouragement (both). Stanford University is literally and figuratively my *alma mater* (Latin for "nourishing mother"). First as a student and later as a faculty member (now emeritus), I have continued to learn and to benefit from colleagues throughout the university and especially from students. SRI International (once called the Stanford Research Institute) provided a home with colleagues who helped me to learn about and to "do" AI. I make special acknowledgement to the late Charles A. Rosen, who persuaded me in 1961 to join his "Learning Machines Group" there. The Defense Advanced Research Projects Agency (DARPA), the Office of Naval Research (ONR), the Air Force

Office of Scientific Research (AFOSR), the U.S. Geological Survey (USGS), the National Science Foundation (NSF), and the National Aeronautics and Space Administration (NASA) all supported various research efforts I was part of during the last fifty years. I owe thanks to all.

To the many people who have helped me with the actual research and writing for this book, including anonymous and not-so-anonymous reviewers, please accept my sincere appreciation together with my apologies for not naming all of you personally in this preface. There are too many of you to list, and I am afraid I might forget to mention someone who might have made some brief but important suggestions. Anyway, you know who you are. You are many of the people whom I mention in the book itself. However, I do want to mention Heather Bergman, of Cambridge University Press, Mykel Kochenderfer, a former student, and Wolfgang Bibel of the Darmstadt University of Technology. They all read carefully early versions of the entire manuscript and made many helpful suggestions. (Mykel also provided invaluable advice about the LaTeX typesetting program.)

I also want to thank the people who invented, developed, and now manage the Internet, the World Wide Web, and the search engines that helped me in writing this book. Using Stanford's various site licenses, I could locate and access journal articles, archives, and other material without leaving my desk. (I did have to visit libraries to find books. Publishers, please allow copyrighted books, especially those whose sales have now diminished, to be scanned and made available online. Join the twenty-first century!)

Finally, and most importantly, I thank my wife, Grace, who cheerfully and patiently urged me on.

In 1982, the late Allen Newell, one of the founders of AI, wrote "Ultimately, we will get real histories of Artificial Intelligence. . . , written with as much objectivity as the historians of science can muster. That time is certainly not yet."

Perhaps it is now.

0.0

Part I

Beginnings

Chapter 1

Dreams and Dreamers

The quest for artificial intelligence (AI) begins with dreams – as all quests do. People have long imagined machines with human abilities – automata that move and devices that reason. Human-like machines are described in many stories and are pictured in sculptures, paintings, and drawings.

You may be familiar with many of these, but let me mention a few. The *Iliad of Homer* talks about self-propelled chairs called "tripods" and golden "attendants" constructed by Hephaistos, the lame blacksmith god, to help him get around.[1*] And, in the ancient Greek myth as retold by Ovid in his *Metamorphoses*, Pygmalian sculpts an ivory statue of a beautiful maiden, Galatea, which Venus brings to life:[2]

> The girl felt the kisses he gave, blushed, and, raising her bashful eyes to the light, saw both her lover and the sky.

The ancient Greek philosopher Aristotle (384–322 BCE) dreamed of automation also, but apparently he thought it an impossible fantasy – thus making slavery necessary if people were to enjoy leisure. In his *The Politics*, he wrote[3]

> For suppose that every tool we had could perform its task, either at our bidding or itself perceiving the need, and if – like... the tripods of Hephaestus, of which the poet [that is, Homer] says that "self-moved they enter the assembly of gods" – shuttles in a loom could fly to and fro and a plucker [the tool used to pluck the strings] play a lyre of their own accord, then master craftsmen would have no need of servants nor masters of slaves.

*So as not to distract the general reader unnecessarily, numbered notes containing citations to source materials appear at the end of each chapter. Each of these is followed by the number of the page where the reference to the note occurred.

Aristotle might have been surprised to see a Jacquard loom weave of itself or a player piano doing its own playing.

Pursuing his own visionary dreams, Ramon Llull (circa 1235–1316), a Catalan mystic and poet, produced a set of paper discs called the *Ars Magna* (Great Art), which was intended, among other things, as a debating tool for winning Muslims to the Christian faith through logic and reason. (See Fig. 1.1.) One of his disc assemblies was inscribed with some of the attributes of God, namely goodness, greatness, eternity, power, wisdom, will, virtue, truth, and glory. Rotating the discs appropriately was supposed to produce answers to various theological questions.[4]

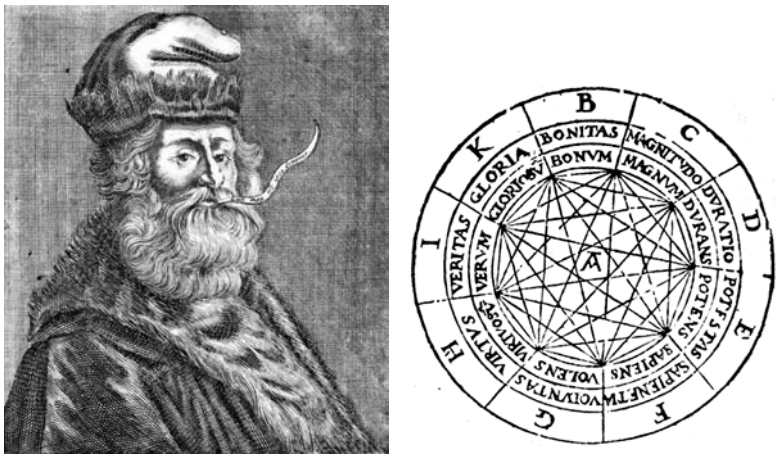


Figure 1.1: Ramon Llull (left) and his *Ars Magna* (right).

Ahead of his time with inventions (as usual), Leonardo Da Vinci sketched designs for a humanoid robot in the form of a medieval knight around the year 1495. (See Fig. 1.2.) No one knows whether Leonardo or contemporaries tried to build his design. Leonardo's knight was supposed to be able to sit up, move its arms and head, and open its jaw.[5]

The Talmud talks about holy persons creating artificial creatures called "golems." These, like Adam, were usually created from earth. There are stories about rabbis using golems as servants. Like the Sorcerer's Apprentice, golems were sometimes difficult to control.

In 1651, Thomas Hobbes (1588–1679) published his book *Leviathan* about the social contract and the ideal state. In the introduction Hobbes seems to say that it might be possible to build an "artificial animal."[6]

> For seeing life is but a motion of limbs, the beginning whereof is in some principal part within, why may we not say that all automata (engines that move themselves by springs and wheels as doth a

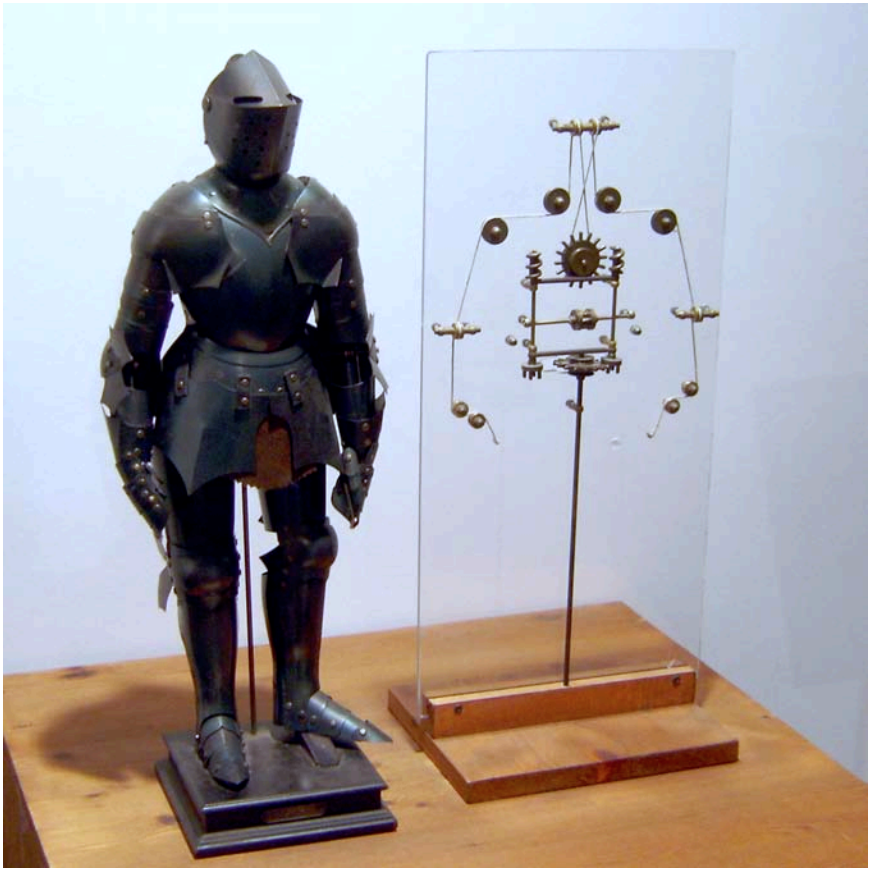


Figure 1.2: Model of a robot knight based on drawings by Leonardo da Vinci.

> watch) have an artificial life? For what is the heart, but a spring; and the nerves, but so many strings; and the joints, but so many wheels, giving motion to the whole body. . .

Perhaps for this reason, the science historian George Dyson refers to Hobbes as the "patriarch of artificial intelligence."[7]

In addition to fictional artifices, several people constructed actual automata that moved in startlingly lifelike ways.[8] The most sophisticated of these was the mechanical duck designed and built by the French inventor and engineer, Jacques de Vaucanson (1709–1782). In 1738, Vaucanson displayed his masterpiece, which could quack, flap its wings, paddle, drink water, and eat and "digest" grain.

As Vaucanson himself put it,[9]

> My second Machine, or *Automaton*, is a *Duck*, in which I represent the Mechanism of the Intestines which are employed in the Operations of Eating, Drinking, and Digestion: Wherein the Working of all the Parts necessary for those Actions is exactly imitated. The Duck stretches out its Neck to take Corn out of your Hand; it swallows it, digests it, and discharges it digested by the usual Passage.

There is controversy about whether or not the material "excreted" by the duck came from the corn it swallowed. One of the automates-anciens Web sites[10] claims that "In restoring Vaucanson's duck in 1844, the magician Robert-Houdin discovered that 'The discharge was prepared in advance: a sort of gruel composed of green-coloured bread crumb . . . '."

Leaving digestion aside, Vaucanson's duck was a remarkable piece of engineering. He was quite aware of that himself. He wrote[11]

> I believe that Persons of Skill and Attention, will see how difficult it has been to make so many different moving Parts in this small *Automaton*; as for Example, to make it rise upon its Legs, and throw its Neck to the Right and Left. They will find the different Changes of the *Fulchrum's* or Centers of Motion: they will also see that what sometimes is a Center of Motion for a moveable Part, another Time becomes moveable on that Part, which Part then becomes fix'd. In a Word, they will be sensible of a prodigious Number of Mechanical Combinations.
>
> This Machine, when once wound up, performs all its different Operations without being touch'd any more.
>
> I forgot to tell you, that the *Duck* drinks, plays in the Water with his Bill, and makes a gurgling Noise like a real living *Duck*. In short, I have endeavor'd to make it imitate all the Actions of the living Animal, which I have consider'd very attentively.

Unfortunately, only copies of the duck exist. The original was burned in a museum in Nijninovgorod, Russia around 1879. You can watch, ANAS, a modern version, performing at http://www.automates-anciens.com/video_1/duck_automaton_vaucanson_500.wmv.[12] It is on exhibit in the Museum of Automatons in Grenoble and was designed and built in 1998 by Frédéric Vidoni, a creator in mechanical arts. (See Fig. 1.3.)

Returning now to fictional automata, I'll first mention the mechanical, life-sized doll, Olympia, which sings and dances in Act I of *Les Contes d'Hoffmann* (*The Tales of Hoffmann*) by Jacques Offenbach (1819–1880). In the opera, Hoffmann, a poet, falls in love with Olympia, only to be crestfallen (and embarrassed) when she is smashed to pieces by the disgruntled Coppélius.

Figure 1.3: Frédéric Vidoni's ANAS, inspired by Vaucanson's duck. (Photograph courtesy of Frédéric Vidoni.)

A play called *R.U.R.* (*Rossum's Universal Robots*) was published by Karel Čapek (pronounced CHAH pek), a Czech author and playwright, in 1920. (See Fig. 1.4.) Čapek is credited with coining the word "robot," which in Czech means "forced labor" or "drudgery." (A "robotnik" is a peasant or serf.)

The play opened in Prague in January 1921. The Robots (always capitalized in the play) are mass-produced at the island factory of Rossum's Universal Robots using a chemical substitute for protoplasm. According to a

Web site describing the play,[13] "Robots remember everything, and think of nothing new. According to Domin [the factory director] 'They'd make fine university professors.' . . . once in a while, a Robot will throw down his work and start gnashing his teeth. The human managers treat such an event as evidence of a product defect, but Helena [who wants to liberate the Robots] prefers to interpret it as a sign of the emerging soul."

I won't reveal the ending except to say that Čapek did not look eagerly on technology. He believed that work is an essential element of human life. Writing in a 1935 newspaper column (in the third person, which was his habit) he said: "With outright horror, he refuses any responsibility for the thought that machines could take the place of people, or that anything like life, love, or rebellion could ever awaken in their cogwheels. He would regard this somber vision as an unforgivable overvaluation of mechanics or as a severe insult to life."[14]

Figure 1.4: A scene from a New York production of *R.U.R.*

There is an interesting story, written by Čapek himself, about how he came to use the word robot in his play. While the idea for the play "was still warm he rushed immediately to his brother Josef, the painter, who was standing before an easel and painting away. . . . 'I don't know what to call these artificial workers,' he said. 'I could call them Labori, but that strikes me

as a bit bookish.' 'Then call them Robots,' the painter muttered, brush in mouth, and went on painting."[15]

The science fiction (and science fact) writer Isaac Asimov wrote many stories about robots. His first collection, *I, Robot,* consists of nine stories about "positronic" robots.[16] Because he was tired of science fiction stories in which robots (such as Frankenstein's creation) were destructive, Asimov's robots had "Three Laws of Robotics" hard-wired into their positronic brains. The three laws were the following:

First Law: A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

Second Law: A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

Third Law: A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Asimov later added a "zeroth" law, designed to protect humanity's interest:[17]

Zeroth Law: A robot may not injure humanity, or, through inaction, allow humanity to come to harm.

The quest for artificial intelligence, quixotic or not, begins with dreams like these. But to turn dreams into reality requires usable clues about how to proceed. Fortunately, there were many such clues, as we shall see.

Notes

1. *The Iliad of Homer*, translated by Richmond Lattimore, p. 386, Chicago: The University of Chicago Press, 1951. (Paperback edition, 1961.) [**19**]

2. Ovid, *Metamorphoses*, Book X, pp. 243–297, from an English translation, circa 1850. See http://www.pygmalion.ws/stories/ovid2.htm. [**19**]

3. Aristotle, *The Politics*, p. 65, translated by T. A. Sinclair, London: Penguin Books, 1981. [**19**]

4. See E. Allison Peers, *Fool of Love: The Life of Ramon Lull*, London: S. C. M. Press, Ltd., 1946. [**20**]

5. See http://en.wikipedia.org/wiki/Leonardo's_robot. [**20**]

6. Thomas Hobbes, *The Leviathon*, paperback edition, Kessinger Publishing, 2004. [**20**]

7. George B. Dyson, *Darwin Among the Machines: The Evolution of Global Intelligence*, p. 7, Helix Books, 1997. [**21**]

8. For a Web site devoted to automata and music boxes, see http://www.automates-anciens.com/english_version/frames/english_frames.htm. [**21**]

9. From Jacques de Vaucanson, "An account of the mechanism of an automaton, or image playing on the German-flute: as it was presented in a memoire, to the gentlemen of the

Royal-Academy of Sciences at Paris. By M. Vaucanson . . . Together with a description of an artificial duck. . . . .” Translated out of the French original, by J. T. Desaguliers, London, 1742. Available at http://e3.uci.edu/clients/bjbecker/NatureandArtifice/week5d.html. [**21**]

10. http://www.automates-anciens.com/english_version/automatons-music-boxes/vaucanson-automatons-androids.php. [**22**]

11. de Vaucanson, Jacques, *op. cit.* [**22**]

12. I thank Prof. Barbara Becker of the University of California at Irvine for telling me about the automates-anciens.com Web sites. [**22**]

13. http://jerz.setonhill.edu/resources/RUR/index.html. [**24**]

14. For a translation of the column entitled “The Author of Robots Defends Himself,” see http://www.depauw.edu/sfs/documents/capek68.htm. [**24**]

15. From one of a group of Web sites about Čapek, http://Capek.misto.cz/english/robot.html. See also http://Capek.misto.cz/english/. [**25**]

16. The Isaac Asimov Web site, http://www.asimovonline.com/, claims that “Asimov did not come up with the title, but rather his publisher ‘appropriated’ the title from a short story by Eando Binder that was published in 1939.” [**25**]

17. See http://www.asimovonline.com/asimov_FAQ.html#series13 for information about the history of these four laws. [**25**]

Chapter 2

Clues

Clues about what might be needed to make machines intelligent are scattered abundantly throughout philosophy, logic, biology, psychology, statistics, and engineering. With gradually increasing intensity, people set about to exploit clues from these areas in their separate quests to automate some aspects of intelligence. I begin my story by describing some of these clues and how they inspired some of the first achievements in artificial intelligence.

2.1 From Philosophy and Logic

Although people had reasoned logically for millennia, it was the Greek philosopher Aristotle who first tried to analyze and codify the process. Aristotle identified a type of reasoning he called the *syllogism* "... in which, certain things being stated, something other than what is stated follows of necessity from their being so."[1]

Here is a famous example of one kind of syllogism:[2]

1. *All humans are mortal.* (stated)
2. *All Greeks are humans.* (stated)
3. *All Greeks are mortal.* (result)

The beauty (and importance for AI) of Aristotle's contribution has to do with the *form* of the syllogism. We aren't restricted to talking about humans, Greeks, or mortality. We could just as well be talking about something else – a result made obvious if we rewrite the syllogism using arbitrary symbols in the place of *humans*, *Greeks*, and *mortal*. Rewriting in this way would produce

1. *All B's are A.* (stated)

2. *All C's are B's.* (stated)

3. *All C's are A.* (result)

One can substitute anything one likes for A, B, and C. For example, *all athletes are healthy* and *all soccer players are athletes*, and therefore *all soccer players are healthy*, and so on. (Of course, the "result" won't necessarily be true unless the things "stated" are. Garbage in, garbage out!)

Aristotle's logic provides two clues to how one might automate reasoning. First, patterns of reasoning, such as syllogisms, can be economically represented as *forms* or *templates*. These use generic symbols, which can stand for many different concrete instances. Because they can stand for anything, the symbols themselves are unimportant.

Second, after the general symbols are replaced by ones pertaining to a specific problem, one only has to "turn the crank" to get an answer. The use of general symbols and similar kinds of crank-turning are at the heart of all modern AI reasoning programs.

In more modern times, Gottfried Wilhelm Leibniz (1646–1716; Fig. 2.1) was among the first to think about logical reasoning. Leibniz was a German philosopher, mathematician, and logician who, among other things, co-invented the calculus. (He had lots of arguments with Isaac Newton about that.) But more importantly for our story, he wanted to mechanize reasoning. Leibniz wrote[3]

> It is unworthy of excellent men to lose hours like slaves in the labor of calculation which could safely be regulated to anyone else if machines were used.

and

> For if praise is given to the men who have determined the number of regular solids... how much better will it be to bring under mathematical laws human reasoning, which is the most excellent and useful thing we have.

Leibniz conceived of and attempted to design a language in which all human knowledge could be formulated – even philosophical and metaphysical knowledge. He speculated that the propositions that constitute knowledge could be built from a smaller number of primitive ones – just as all words can be built from letters in an alphabetic language. His *lingua characteristica* or universal language would consist of these primitive propositions, which would comprise an *alphabet for human thoughts*.

The alphabet would serve as the basis for automatic reasoning. His idea was that if the items in the alphabet were represented by numbers, then a

Figure 2.1: Gottfried Leibniz.

complex proposition could be obtained from its primitive constituents by multiplying the corresponding numbers together. Further arithmetic operations could then be used to determine whether or not the complex proposition was true or false. This whole process was to be accomplished by a *calculus ratiocinator* (calculus of reasoning). Then, when philosophers disagreed over some problem they could say, "*calculemus*" ("let us calculate"). They would first pose the problem in the *lingua characteristica* and then solve it by "turning the crank" on the *calculus ratiocinator*.

The main problem in applying this idea was discovering the components of the primitive "alphabet." However, Leibniz's work provided important additional clues to how reasoning might be mechanized: Invent an alphabet of simple symbols and the means for combining them into more complex expressions.

Toward the end of the eighteenth century and the beginning of the nineteenth, a British scientist and politician, Charles Stanhope (Third Earl of Stanhope), built and experimented with devices for solving simple problems in logic and probability. (See Fig. 2.2.) One version of his "box" had slots on the sides into which a person could push colored slides. From a window on the top, one could view slides that were appropriately positioned to represent a

specific problem. Today, we would say that Stanhope's box was a kind of analog computer.

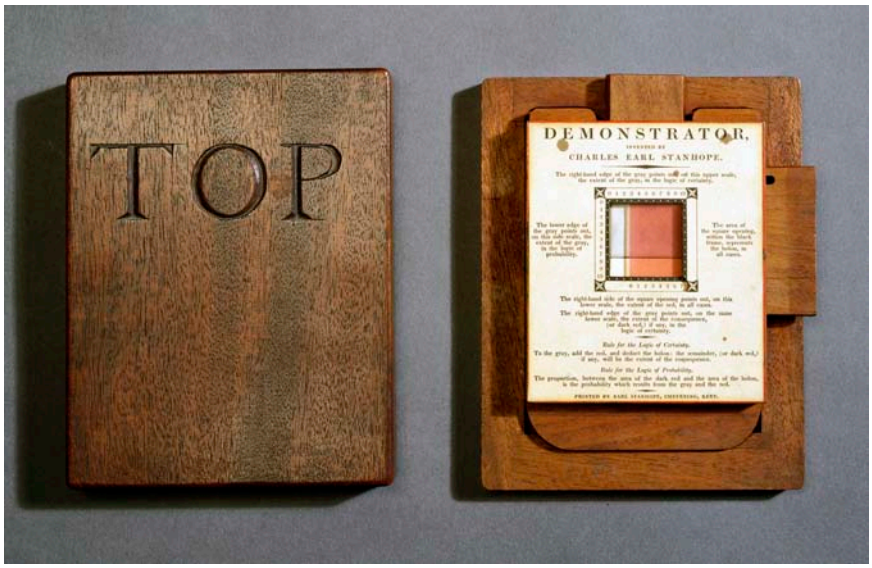


Figure 2.2: The Stanhope Square Demonstrator, 1805. (Photograph courtesy of Science Museum/SSPL.)

The book *Computing Before Computers* gives an example of its operation:[4]

> To solve a numerical syllogism, for example:
>
> > *Eight of ten A's are B's*; *Four of ten A's are C's*;
> > *Therefore, at least two B's are C's.*
>
> Stanhope would push the red slide (representing B) eight units across the window (representing A) and the gray slide (representing C) four units from the opposite direction. The two units that the slides overlapped represented the minimum number of B's that were also C's.
>
> . . .
>
> In a similar way the Demonstrator could be used to solve a traditional syllogism like:
>
> > *No M is A*; *All B is M*; *Therefore, No B is A.*

Stanhope was rather secretive about his device and didn't want anyone to know what he was up to. As mentioned in *Computing Before Computers*,

"The few friends and relatives who received his privately distributed account of the Demonstrator, *The Science of Reasoning Clearly Explained Upon New Principles* (1800), were advised to remain silent lest 'some bastard imitation' precede his intended publication on the subject."

But no publication appeared until sixty years after Stanhope's death. Then, the Reverend Robert Harley gained access to Stanhope's notes and one of his boxes and published an article on what he called "The Stanhope Demonstrator."[5]

Contrasted with Llull's schemes and Leibniz's hopes, Stanhope built the first logic machine that actually worked – albeit on small problems. Perhaps his work raised confidence that logical reasoning could indeed be mechanized.

In 1854, the Englishman George Boole (1815–1864; Fig. 2.3) published a book with the title *An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities*.[6] Boole's purpose was (among other things) "to collect... some probable intimations concerning the nature and constitution of the human mind." Boole considered various logical principles of human reasoning and represented them in mathematical form. For example, his "Proposition IV" states "*... the principle of contradiction... affirms that it is impossible for any being to possess a quality, and at the same time not to possess it....*" Boole then wrote this principle as an algebraic equation,

$$x(1-x)=0,$$

in which x represents "any class of objects," $(1-x)$ represents the "contrary or supplementary class of objects," and 0 represents a class that "does not exist."

In Boolean algebra, an outgrowth of Boole's work, we would say that 0 represents *falsehood*, and 1 represents *truth*. Two of the fundamental operations in logic, namely OR and AND, are represented in Boolean algebra by the operations $+$ and $\times$, respectively. Thus, for example, to represent the statement "either p or q or both," we would write $p+q$. To represent the statement "p and q," we would write $p \times q$. Each of p and q could be true or false, so we would evaluate the value (truth or falsity) of $p+q$ and $p \times q$ by using definitions for how $+$ and $\times$ are used, namely,

$$1+0=1,$$

$$1 \times 0=0,$$

$$1+1=1,$$

$$1 \times 1=1,$$

$$0+0=0,$$

and

$$0 \times 0=0.$$

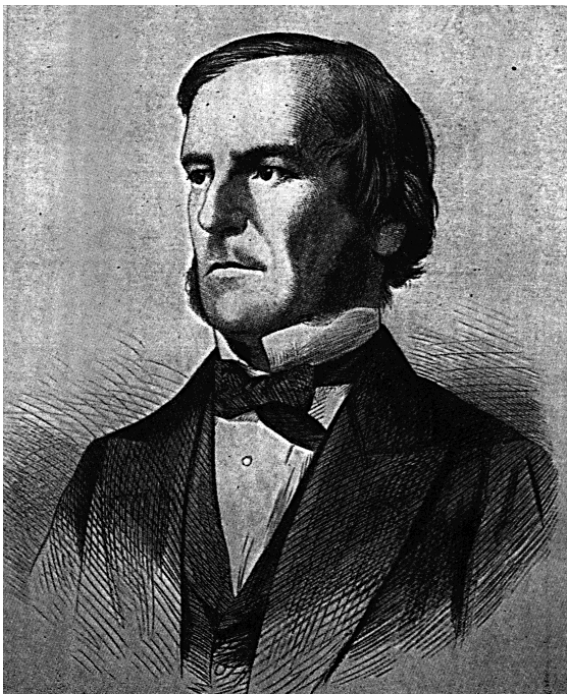


Figure 2.3: George Boole.

Boolean algebra plays an important role in the design of telephone switching circuits and computers. Although Boole probably could not have envisioned computers, he did realize the importance of his work. In a letter dated January 2, 1851, to George Thomson (later Lord Kelvin) he wrote[7]

> I am now about to set seriously to work upon preparing for the press an account of my theory of Logic and Probabilities which in its present state I look upon as the most valuable if not the only valuable contribution that I have made or am likely to make to Science and the thing by which I would desire if at all to be remembered hereafter...

Boole's work showed that some kinds of logical reasoning could be performed by manipulating equations representing logical propositions – a very important clue about the mechanization of reasoning. An essentially equivalent, but not algebraic, system for manipulating and evaluating propositions is called the "propositional calculus" (often called "propositional logic"), which, as we shall see, plays a very important role in artificial intelligence. [Some claim that the Greek Stoic philospher Chrysippus (280–209 BCE) invented an early form of the propositional calculus.[8]]

One shortcoming of Boole's logical system, however, was that his propositions p, q, and so on were "atomic." They don't reveal any entities *internal* to propositions. For example, if we expressed the proposition "Jack is human" by p, and "Jack is mortal" by q, there is nothing in p or q to indicate that the Jack who is human is the very same Jack who is mortal. For that, we need, so to speak, "molecular expressions" that have internal elements.

Toward the end of the nineteenth century, the German mathematician, logician, and philosopher Friedrich Ludwig Gottlob Frege (1848–1925) invented a system in which propositions, along with their internal components, could be written down in a kind of graphical form. He called his language *Begriffsschrift*, which can be translated as "concept writing." For example, the statement "All persons are mortal" would have been written in *Begriffsschrift* something like the diagram in Fig. 2.4.[9]

Figure 2.4: Expressing "All persons are mortal" in *Begriffsschrift*.

Note that the illustration explicitly represents the x who is predicated to be a person and that it is the same x who is then claimed to be mortal. It's more convenient nowadays for us to represent this statement in the linear form $(\forall x)P(x) \supset M(x)$, whose English equivalent is "for all x, if x is a person, then x is mortal."

Frege's system was the forerunner of what we now call the "predicate calculus," another important system in artificial intelligence. It also foreshadows another representational form used in present-day artificial intelligence: semantic networks. Frege's work provided yet more clues about how to mechanize reasoning processes. At last, sentences expressing information to be reasoned about could be written in unambiguous, symbolic form.

2.2 From Life Itself

In Proverbs 6:6–8, King Solomon says "Go to the ant, thou sluggard; consider her ways and be wise." Although his advice was meant to warn against slothfulness, it can just as appropriately enjoin us to seek clues from biology about how to build or improve artifacts.

Several aspects of "life" have, in fact, provided important clues about intelligence. Because it is the *brain* of an animal that is responsible for converting sensory information into action, it is to be expected that several good ideas can be found in the work of neurophysiologists and neuroanatomists who study brains and their fundamental components, neurons. Other ideas are provided by the work of psychologists who study (in various ways) intelligent behavior as it is actually happening. And because, after all, it is evolutionary processes that have produced intelligent life, those processes too provide important hints about how to proceed.

2.2.1 Neurons and the Brain

In the late nineteenth and early twentieth centuries, the "neuron doctrine" specified that living cells called "neurons" together with their interconnections were fundamental to what the brain does. One of the people responsible for this suggestion was the Spanish neuroanatomist Santiago Ramón y Cajal (1852–1934). Cajal (Fig. 2.5) and Camillo Golgi won the Nobel Prize in Physiology or Medicine in 1906 for their work on the structure of the nervous system.

A neuron is a living cell, and the human brain has about ten billion (10^{10}) of them. Although they come in different forms, typically they consist of a central part called a *soma* or *cell body*, incoming fibers called *dendrites*, and one or more outgoing fibers called *axons*. The axon of one neuron has projections called *terminal buttons* that come very close to one or more of the dendrites of other neurons. The gap between the terminal button of one neuron and a dendrite of another is called a *synapse*. The size of the gap is about 20 nanometers. Two neurons are illustrated schematically in Fig. 2.6.

Through electrochemical action, a neuron may send out a stream of pulses down its axon. When a pulse arrives at the synapse adjacent to a dendrite of another neuron, it may act to excite or to inhibit electrochemical activity of the other neuron across the synapse. Whether or not this second neuron then "fires" and sends out pulses of its own depends on how many and what kinds of pulses (excitatory or inhibitory) arrive at the synapses of its various incoming dendrites and on the efficiency of those synapses in transmitting electrochemical activity. It is estimated that there are over half a trillion synapses in the human brain. The neuron doctrine claims that the various activities of the brain, including perception and thinking, are the result of all of this neural activity.

In 1943, the American neurophysiologist Warren McCulloch (1899–1969; Fig. 2.7) and logician Walter Pitts (1923–1969) claimed that the neuron was, in essence, a "logic unit." In a famous and important paper they proposed simple models of neurons and showed that networks of these models could perform all possible computational operations.[10] The McCulloch–Pitts "neuron" was a mathematical abstraction with inputs and outputs

Figure 2.5: Ramón y Cajal.

(corresponding, roughly, to dendrites and axons, respectively). Each output can have the value 1 or 0. (To avoid confusing a McCulloch–Pitts neuron with a real neuron, I'll call the McCulloch–Pitts version, and others like it, a "neural element.") The neural elements can be connected together into networks such that the output of one neural element is an input to others and so on. Some neural elements are excitatory – their outputs contribute to "firing" any neural elements to which they are connected. Others are inhibitory – their outputs contribute to inhibiting the firing of neural elements to which they are connected. If the sum of the excitatory inputs less the sum of the inhibitory inputs impinging on a neural element is greater than a certain "threshold," that neural element fires, sending its output of 1 to all of the neural elements to which it is connected.

Some examples of networks proposed by McCullough and Pitts are shown in Fig. 2.8.

The Canadian neuropsychologist Donald O. Hebb (1904–1985) also

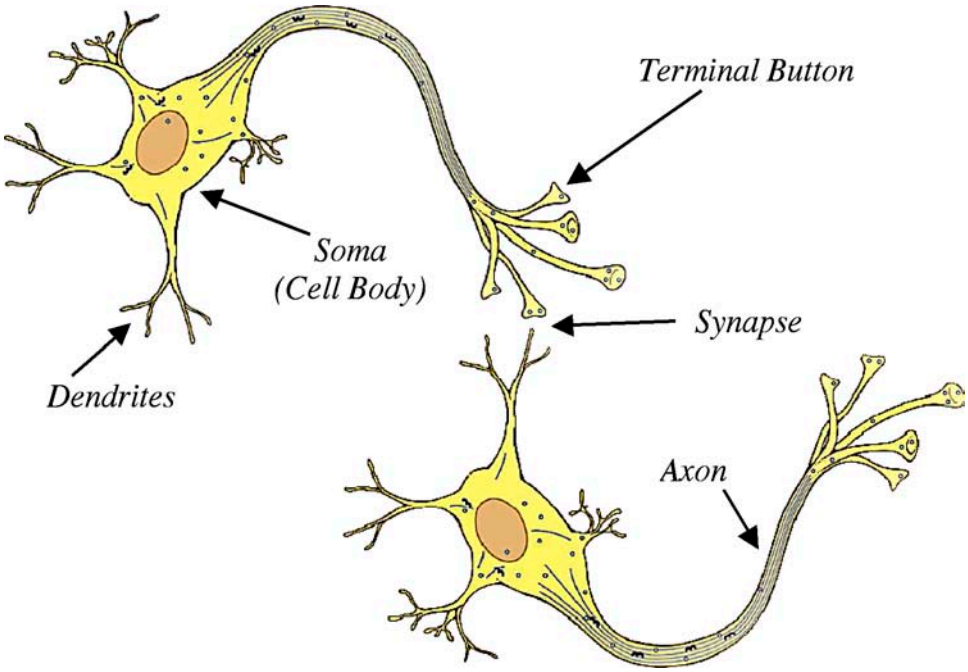


Figure 2.6: Two neurons. (Adapted from *Science*, Vol. 316, p. 1416, 8 June 2007. Used with permission.)

believed that neurons in the brain were the basic units of thought. In an influential book,[11] Hebb suggested that "when an axon of cell A is near enough to excite B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." Later, this so-called Hebb rule of change in neural "synaptic strength" was actually observed in experiments with living animals. (In 1965, the neurophysiologist Eric Kandel published results showing that simple forms of learning were associated with synaptic changes in the marine mollusk *Aplysia californica*. In 2000, Kandel shared the Nobel Prize in Physiology or Medicine "for their discoveries concerning signal transduction in the nervous system.")

Hebb also postulated that groups of neurons that tend to fire together formed what he called *cell assemblies*. Hebb thought that the phenomenon of "firing together" tended to persist in the brain and was the brain's way of representing the perceptual event that led to a cell-assembly's formation. Hebb said that "thinking" was the sequential activation of sets of cell assemblies.[12]

Figure 2.7: Warren McCulloch.

2.2.2 Psychology and Cognitive Science

Psychology is the science that studies mental processes and behavior. The word is derived from the Greek words *psyche*, meaning breath, spirit, or soul, and *logos*, meaning word. One might expect that such a science ought to have much to say that would be of interest to those wanting to create intelligent artifacts. However, until the late nineteenth century, most psychological theorizing depended on the insights of philosophers, writers, and other astute observers of the human scene. (Shakespeare, Tolstoy, and other authors were no slouches when it came to understanding human behavior.)

Most people regard serious scientific study to have begun with the

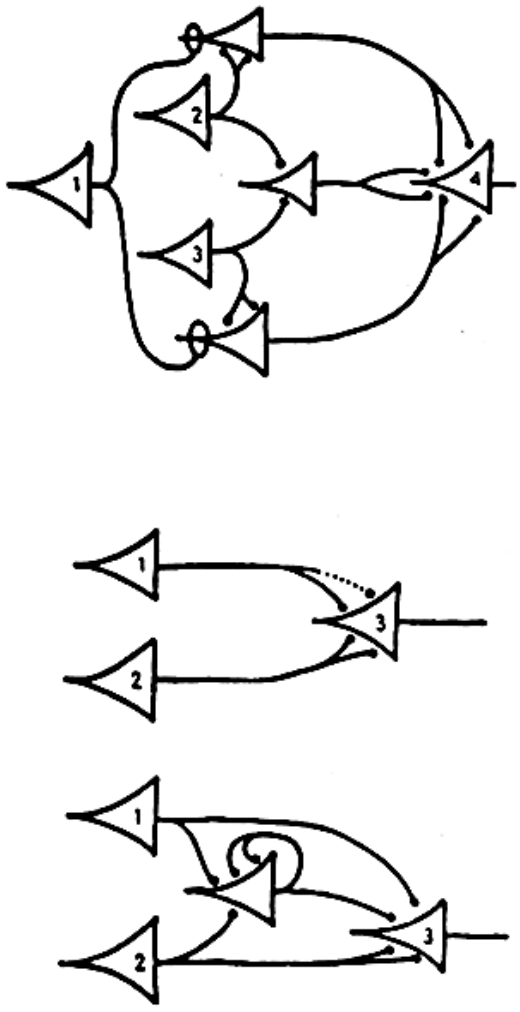


Figure 2.8: Networks of McCulloch–Pitts neural elements. (Adapted from Fig. 1 of Warren S. McCulloch and Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, 1943.)

German Wilhelm Wundt (1832–1920) and the American William James (1842–1910).[13] Both established psychology labs in 1875 – Wundt in Leipzig and James at Harvard. According to C. George Boeree, who teaches the history of psychology at Shippensburg University in Pennsylvania, "The method that Wundt developed is a sort of experimental introspection: The

researcher was to carefully observe some simple event – one that could be measured as to quality, intensity, or duration – and record his responses to variations of those events." Although James is now regarded mainly as a philosopher, he is famous for his two-volume book *The Principles of Psychology*, published in 1873 and 1874.

Both Wundt and James attempted to say something about *how* the brain worked instead of merely cataloging its input–output behavior. The psychiatrist Sigmund Freud (1856–1939) went further, postulating internal components of the brain, namely, the *id*, the *ego*, and the *superego*, and how they interacted to affect behavior. He thought one could learn about these components through his unique style of guided introspection called *psychoanalysis*.

Attempting to make psychology more scientific and less dependent on subjective introspection, a number of psychologists, most famously B. F. Skinner (1904–1990; Fig. 2.9), began to concentrate solely on what could be objectively measured, namely, specific behavior in reaction to specific stimuli. The *behaviorists* argued that psychology should be a science of behavior, not of the mind. They rejected the idea of trying to identify internal mental states such as beliefs, intentions, desires, and goals.

Figure 2.9: B. F. Skinner. (Photograph courtesy of the B. F. Skinner Foundation.)

This development might at first be regarded as a step backward for people wanting to get useful clues about the internal workings of the brain. In criticizing the statistically oriented theories arising from "behaviorism," Marvin Minsky wrote "Originally intended to avoid the need for 'meaning,' [these theories] manage finally only to avoid the possibility of explaining it."[14] Skinner's work did, however, provide the idea of a *reinforcing* stimulus – one that rewards recent behavior and tends to make it more likely to occur (under similar circumstances) in the future.

Reinforcement learning has become a popular strategy among AI researchers, although it does depend on internal states. Russell Kirsch (circa 1930–), a computer scientist at the U.S. National Bureau of Standards (now the National Institute for Standards and Technology, NIST), was one of the first to use it. He proposed how an "artificial animal" might use reinforcement to learn good moves in a game. In some 1954 seminar notes he wrote the following:[15] "The animal model notes, for each stimulus, what move the opponent next makes, ... Then, the next time that same stimulus occurs, the animal duplicates the move of the opponent that followed the same stimulus previously. The more the opponent repeats the same move after any given stimulus, the more the animal model becomes 'conditioned' to that move."

Skinner believed that reinforcement learning could even be used to explain verbal behavior in humans. He set forth these ideas in his 1957 book *Verbal Behavior*,[16] claiming that the laboratory-based principles of selection by consequences can be extended to account for what people say, write, gesture, and think.

Arguing against Skinner's ideas about language the linguist Noam Chomsky (1928– ; Fig. 2.10), in a review[17] of Skinner's book, wrote that

> careful study of this book (and of the research on which it draws) reveals, however, that [Skinner's] astonishing claims are far from justified. ... the insights that have been achieved in the laboratories of the reinforcement theorist, though quite genuine, can be applied to complex human behavior only in the most gross and superficial way, and that speculative attempts to discuss linguistic behavior in these terms alone omit from consideration factors of fundamental importance...

How, Chomsky seems to ask, can a person produce a potentially infinite variety of previously unheard and unspoken sentences having arbitrarily complex structure (as indeed they can do) through experience alone? These "factors of fundamental importance" that Skinner omits are, according to Chomsky, linguistic abilities that must be innate – not learned. He suggested that "human beings are somehow specially created to do this, with data-handling or 'hypothesis-formulating' ability of [as yet] unknown character and complexity." Chomsky claimed that all humans have at birth a "universal

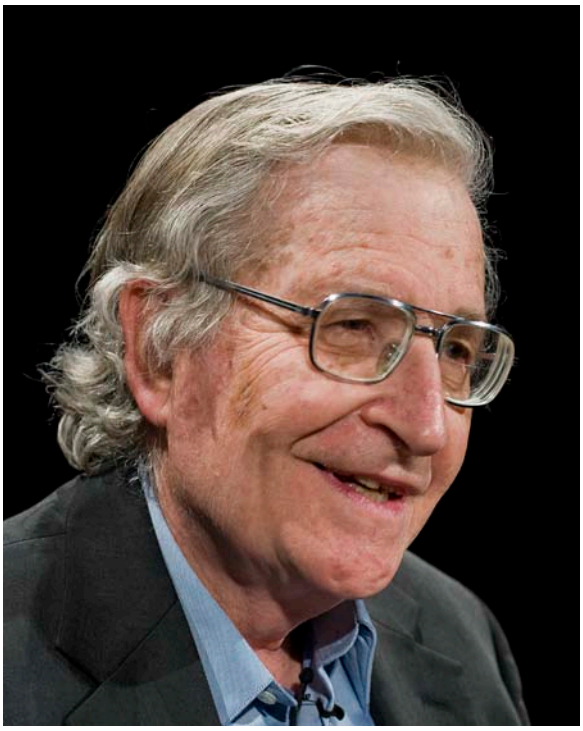


Figure 2.10: Noam Chomsky. (Photograph by Don J. Usner.)

grammar" (or developmental mechanisms for creating one) that accounts for much of their ability to learn and use languages.[18]

Continuing the focus on internal mental processes and their limitations, the psychologist George A. Miller (1920–) analyzed the work of several experimenters and concluded that the "immediate memory" capacity of humans was approximately seven "chunks" of information.[19] In the introduction to his paper about this "magical number," Miller humorously notes "My problem is that I have been persecuted by an integer. For seven years this number has followed me around, has intruded in my most private data, and has assaulted me from the pages of our most public journals. This number assumes a variety of disguises, being sometimes a little larger and sometimes a little smaller than usual, but never changing so much as to be unrecognizable. The persistence with which this number plagues me is far more than a random accident." Importantly, he also claimed that "the span of immediate memory seems to be almost independent of the number of bits per chunk." That is, it doesn't matter what a chunk represents, be it a single digit in a phone number, a name of a person just mentioned, or a song title; we can apparently only hold seven of them (plus or minus two) in our immediate

memory.

Miller's paper on "The Magical Number Seven," was given at a Symposium on Information Theory held from September 10 to 12, 1956, at MIT.[20] Chomsky presented an important paper there too. It was entitled "Three Models for the Description of Language," and in it he proposed a family of rules of syntax he called *phrase-structure grammars*.[21] It happens that two pioneers in AI research (of whom we'll hear a lot more later), Allen Newell (1927–1992), then a scientist at the Rand Corporation, and Herbert Simon (1916–2001), a professor at the Carnegie Institute of Technology (now Carnegie Mellon University), gave a paper there also on a computer program that could prove theorems in propositional logic. This symposium, bringing together as it did scientists with these sorts of overlapping interests, is thought to have contributed to the birth of *cognitive science*, a new discipline devoted to the study of the mind. Indeed, George Miller wrote[22]

> I went away from the Symposium with a strong conviction, more intuitive than rational, that human experimental psychology, theoretical linguistics, and computer simulation of cognitive processes were all pieces of a larger whole, and that the future would see progressive elaboration and coordination of their shared concerns. . .

In 1960, Miller and colleagues wrote a book proposing a specific internal mechanism responsible for behavior, which they called the TOTE unit (Test–Operate–Test–Exit).[23] There is a TOTE unit corresponding to every goal that an agent might have. Using its perceptual abilities, the unit first tests whether or not its goal is satisfied. If so, the unit rests (exits). If not, some operation specific to achieving that goal is performed, and the test for goal achievement is performed again, and so on repetitively until the goal finally is achieved. As a simple example, consider the TOTE unit for driving a nail with a hammer. So long as the nail is not completely driven in (the goal), the hammer is used to strike it (the operation). Pounding stops (the exit) when the goal is finally achieved. It's difficult to say whether or not this book inspired similar work by artificial intelligence researchers. The idea was apparently "in the air," because at about the same time, as we shall see later, some early work in AI used very similar ideas. [I can say that my work at SRI with behavior (intermediate-level) programs for the robot, Shakey, and my later work on what I called "teleo-reactive" programs were influenced by Miller's ideas.]

Cognitive science attempted to explicate internal mental processes using ideas such as goals, memory, task queues, and strategies without (at least during its beginning years) necessarily trying to ground these processes in neurophysiology.[24] Cognitive science and artificial intelligence have been closely related ever since their beginnings. Cognitive science has provided clues for AI researchers, and AI has helped cognitive science with newly

invented concepts useful for understanding the workings of the mind.

2.2.3 Evolution

That living things evolve gives us two more clues about how to build intelligent artifacts. First, and most ambitiously, the processes of evolution itself – namely, random generation and selective survival – might be simulated on computers to produce the machines we dream about. Second, those paths that evolution followed in producing increasingly intelligent animals can be used as a guide for creating increasingly intelligent artifacts. Start by simulating animals with simple tropisms and proceed along these paths to simulating more complex ones. Both of these strategies have been followed with zest by AI researchers, as we shall see in the following chapters. Here, it will suffice to name just a few initial efforts.

Early attempts to simulate evolution on a computer were undertaken at Princeton's Institute for Advanced Study by the viral geneticist Nils Aall Barricelli (1912–1993). His 1954 paper described experiments in which numbers migrated and reproduced in a grid.[25]

Motivated by the success of biological evolution in producing complex organisms, some researchers began thinking about how programs could be evolved rather than written. R. N. Friedberg and his IBM colleagues[26] conducted experiments in which, beginning with a population of random computer programs, they attempted to evolve ones that were more successful at performing a simple logical task. In the summary of his 1958 paper, Friedberg wrote that "[m]achines would be more useful if they could learn to perform tasks for which they were not given precise methods. . . . It is proposed that the program of a stored-program computer be gradually improved by a learning procedure which tries many programs and chooses, from the instructions that may occupy a given location, the one most often associated with a successful result." That is, Friedberg installed instructions from "successful" programs into the programs of the next "generation," much as how the genes of individuals successful enough to have descendants are installed in those descendants.

Unfortunately, Friedberg's attempts to evolve programs were not very successful. As Marvin Minsky pointed out,[27]

> The machine [described in the first paper] did learn to solve some extremely simple problems. But it took of the order of 1000 times longer than pure chance would expect. . . .
>
> The second paper goes on to discuss a sequence of modifications. . . With these, and with some 'priming' (starting the machine off on the right track with some useful instructions), the system came to be only a little worse than chance.

Minsky attributes the poor performance of Friedberg's methods to the fact that each descendant machine differed very little from its parent, whereas any helpful improvement would require a much larger step in the "space" of possible machines.

Other early work on artificial evolution was more successful. Lawrence Fogel (1928–2007) and colleagues were able to evolve machines that could make predictions of the next element in a sequence.[28] Woodrow W. Bledsoe (1921–1995) at Panoramic Research and Hans J. Bremermann (1926–1969) at the University of California, Berkeley, used simulated evolution to solve optimization and mathematical problems, respectively.[29] And Ingo Rechenberg (according to one AI researcher) "pioneered the method of artificial evolution to solve complex optimization tasks, such as the design of optimal airplane wings or combustion chambers of rocket nozzles."[30]

The first prominent work inspired by biological evolution was John Holland's development of "genetic algorithms" beginning in the early 1960s. Holland (1929–), a professor at the University of Michigan, used strings of binary symbols (0's and 1's), which he called "chromosomes" in analogy with the genetic material of biological organisms. (Holland says he first came up with the notion while browsing through the Michigan math library's open stacks in the early 1950s.)[31] The encoding of 0's and 1's in a chromosome could be interpreted as a solution to some given problem. The idea was to evolve chromosomes that were better and better at solving the problem. Populations of chromosomes were subjected to an evolutionary process in which individual chromosomes underwent "mutations" (changing a component 1 to a 0 and vice versa), and pairs of the most successful chromosomes at each stage of evolution were combined to make a new chromosome. Ultimately, the process would produce a population containing a chromosome (or chromosomes) that solved the problem.[32]

Researchers would ultimately come to recognize that all of these evolutionary methods were elaborations of a very useful mathematical search strategy called "gradient ascent" or "hill climbing." In these methods, one searches for a local maximum of some function by taking the steepest possible uphill steps. (When searching for a local minimum, the analogous method is called "gradient descent.")

Rather than attempt to duplicate evolution itself, some researchers preferred to build machines that followed along evolution's paths toward intelligent life. In the late 1940s and early 1950s, W. Grey Walter (1910–1977), a British neurophysiologist (born in Kansas City, Missouri), built some machines that behaved like some of life's most primitive creatures. They were wheeled vehicles to which he gave the taxonomic name *Machina speculatrix* (machine that looks; see Fig. 2.11).[33] These tortoise-like machines were controlled by "brains" consisting of very simple vacuum-tube circuits that sensed their environments with photocells and that controlled their wheel motors. The circuits could be arranged so that a machine either moved toward

or away from a light mounted on a sister machine. Their behaviors seemed purposive and often complex and unpredictable, so much so that Walter said they "might be accepted as evidence of some degree of self-awareness." *Machina speculatrix* was the beginning of a long line of increasingly sophisticated "behaving machines" developed by subsequent researchers.

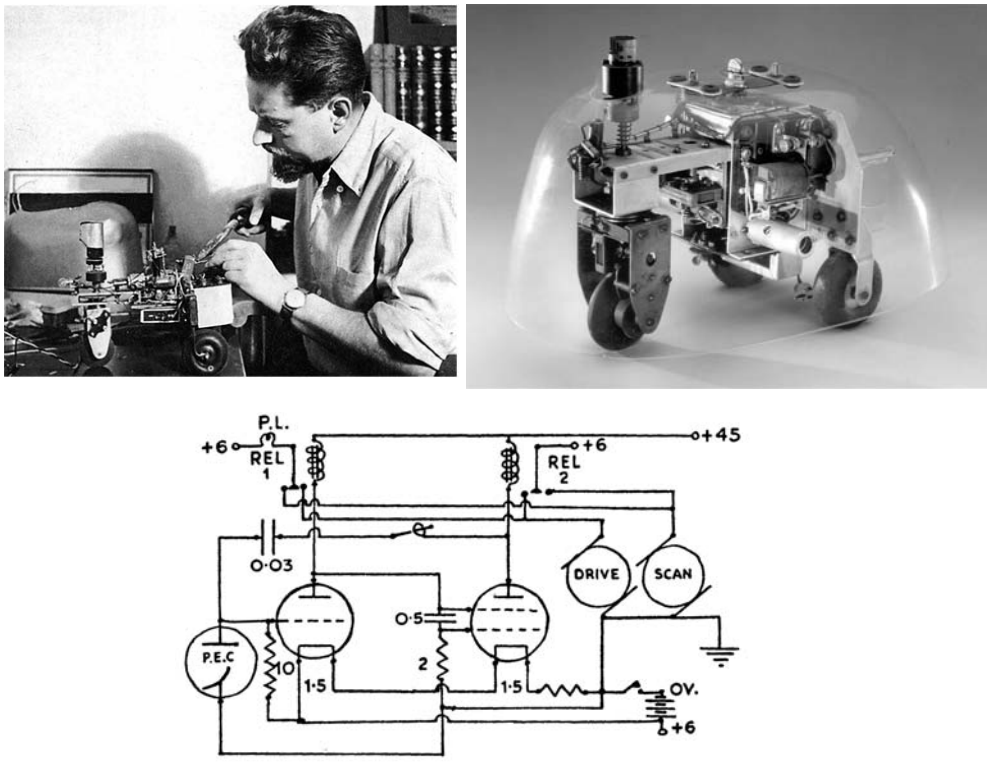


Figure 2.11: Grey Walter (top left), his *Machina speculatrix* (top right), and its circuit diagram (bottom). (Grey Walter photograph from Hans Moravec, *ROBOT*, Chapter 2: Caution! Robot Vehicle!, p. 18, Oxford: Oxford University Press, 1998; "Turtle" photograph courtesy of National Museum of American History, Smithsonian Institution; the circuit diagram is from W. Grey Walter, *The Living Brain*, p. 200, London: Gerald Duckworth & Co., Ltd., 1953.)

2.2.4 Development and Maturation

Perhaps there are alternatives to rerunning evolution itself or to following its paths toward increasing complexity from the most primitive animals. By careful study of the behavior of young children, the Swiss psychologist Jean Piaget proposed a set of stages in the maturation of their thinking abilities

from infancy to adolescence.[34] Might these stages provide a set of steps that could guide designers of intelligent artifacts? Start with a machine that is able to do what an infant can do, and then design machines that can mimic the abilities of children at each rung of the ladder. This strategy might be called "ontogenetic" to contrast it with the "phylogenetic" strategy of using simlulated evolution.

Of course, it may be that an infant mind is far too complicated to simulate and the processes of its maturation too difficult to follow. In any case, this particular clue remains to be exploited.

2.2.5 Bionics

At a symposium in 1960, Major Jack E. Steele, of the Aerospace Division of the United States Air Force, used the term "bionics" to describe the field that learns lessons from nature to apply to technology.[35]

Several bionics and bionics-related meetings were held during the 1960s. At the 1963 Bionics Symposium, Leonard Butsch and Hans Oestreicher wrote "Bionics aims to take advantage of millions of years of evolution of living systems during which they adapted themselves for optimum survival. One of the outstanding successes of evolution is the information processing capability of living systems [the study of which is] one of the principal areas of Bionics research."[36]

Today, the word "bionics" is concerned mainly with orthotic and prosthetic devices, such as artificial cochleas, retinas, and limbs. Nevertheless, as AI researchers continue their quest, the study of living things, their evolution, and their development may continue to provide useful clues for building intelligent artifacts.

2.3 From Engineering

2.3.1 Automata, Sensing, and Feedback

Machines that move by themselves and even do useful things by themselves have been around for centuries. Perhaps the most common early examples are the "verge-and-foliot" weight-driven clocks. (See Fig. 2.12.) These first appeared in the late Middle Ages in the towers of large Italian cities. The verge-and-foliot mechanism converted the energy of a falling weight into stepped rotational motion, which could be used to move the clock hands. Similar mechanisms were elaborated to control the actions of automata, such as those of the Munich Glockenspiel.

One of the first automatic machines for producing goods was Joseph-Marie Jacquard's weaving loom, built in 1804. (See Fig. 2.13.) It

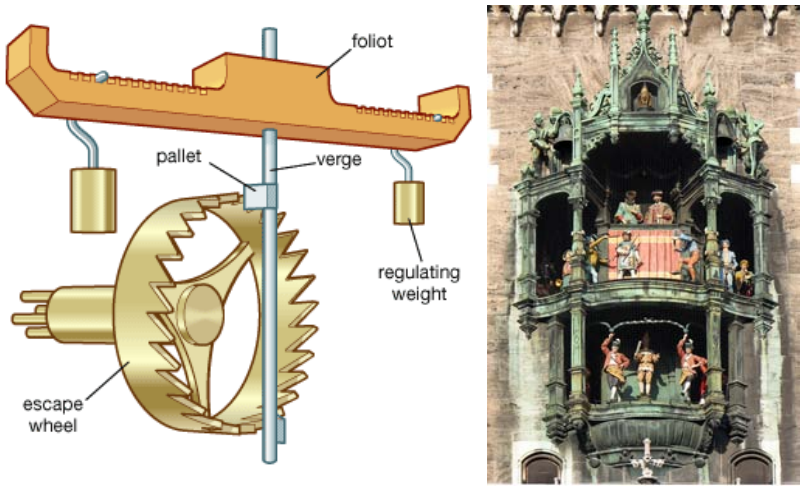


Figure 2.12: A verge-and-foliot mechanism (left) and automata at the Munich Glockenspiel (right).

followed a long history of looms and improved on the "punched card" design of Jacques de Vaucanson's loom of 1745. (Vaucanson did more than build mechanical ducks.) The punched cards of the Jacquard loom controlled the actions of the shuttles, allowing automatic production of fabric designs. Just a few years after its invention, there were some 10,000 Jacquard looms weaving away in France. The idea of using holes in paper or cards was later adopted by Herman Hollerith for tabulating the 1890 American census data and in player pianos (using perforated rolls instead of cards). The very first factory "robots" of the so-called pick-and-place variety used only modest elaborations of this idea.

It was only necessary to provide these early machines with an external source of energy (a falling weight, a wound-up spring, or humans pumping pedals). Their behavior was otherwise fully automatic, requiring no human guidance. But, they had an important limitation – they did not perceive anything about their environments. (The punched cards that were "read" by the Jacquard loom are considered part of the machine – not part of the environment.) Sensing the environment and then letting what is sensed influence what a machine does is critical to intelligent behavior. Grey Walters's "tortoises," for example, had photocells that could detect the presence or absence of light in their environments and act accordingly. Thus, they seem more intelligent than a Jacquard loom or clockwork automata.

One of the simplest ways to allow what is sensed to influence behavior involves what is called "feedback control." The word derives from feeding some aspect of a machine's behavior, say its speed of operation, back into the

Figure 2.13: Reconstruction of a Jacquard loom.

internals of the machine. If the aspect of behavior that is fed back acts to diminish or reverse that aspect, the process is called "negative feedback." If, on the other hand, it acts to increase or accentuate that aspect of behavior, it is called "positive feedback." Both types of feedback play extremely important roles in engineering.

Negative feedback techniques have been used for centuries in mechanical devices. In 270 BCE, a Greek inventor and barber, Ktesibios of Alexandria, invented a float regulator to keep the water level in a tank feeding a water clock at a constant depth by controlling the water flow into the tank.[37] The feedback device was a float valve consisting of a cork at the end of a rod. The cork floated on the water in the tank. When the water level in the tank rose, the cork would rise, causing the rod to turn off the water coming in. When the water level fell, the cork would fall, causing the rod to turn on the water. The water level in modern flush toilets is regulated in much the same way. In 250 BCE, Philon of Byzantium used a similar float regulator to keep a constant level of oil in a lamp.[38]

The English clockmaker John Harrison (1693–1776) used a type of negative feedback control in his clocks. The ambient temperature of a clock affects the length of its balance spring and thus its time-keeping accuracy. Harrison used a bimetallic strip (sometimes a rod), whose curvature depends on temperature. The strip was connected to the balance spring in such a way that it produced offsetting changes in the length of the spring, thus making the clock more independent of its temperature. The strip senses the temperature and causes the clock to behave differently, and more accurately, than it otherwise would. Today, such bimetallic strips see many uses, notably in thermostats. (Dava Sobel's 1995 book, *Longitude: The True Story of a Lone Genius Who Solved the Greatest Scientific Problem of His Time*, recounts the history of Harrison's efforts to build a prize-winning clock for accurate time-keeping at sea.)

Perhaps the most graphic use of feedback control is the centrifugal flyball governor perfected in 1788 by James Watt for regulating the speed of his steam engine. (See Fig. 2.14.) As the speed of the engine increases, the balls fly outward, which causes a linking mechanism to decrease air flow, which causes the speed to decrease, which causes the balls to fall back inward, which causes the speed to increase, and so on, resulting in an equilibrium speed.

In the early 1940s, Norbert Wiener (1894–1964) and other scientists noted similarities between the properties of feedback control systems in machines and in animals. In particular, inappropriately applied feedback in control circuits led to jerky movements of the system being controlled that were similar to pathological "tremor" in human patients. Arturo Rosenblueth, Norbert Wiener, and Julian Bigelow coined the term "cybernetics" in a 1943 paper. Wiener's book by that name was published in 1948. The word is related to the word "governor." (In Latin *gubernaculum* means helm, and *gubernator* means helmsman. The Latin derives from the Greek *kybernetike*, which means the art of steersmanship.[39])

Today, the prefix "cyber" is used to describe almost anything that deals with computers, robots, the Internet, and advanced simulation. For example, the author William Gibson coined the term "cyberspace" in his 1984 science fiction novel *Neuromancer*. Technically, however, cybernetics continues to

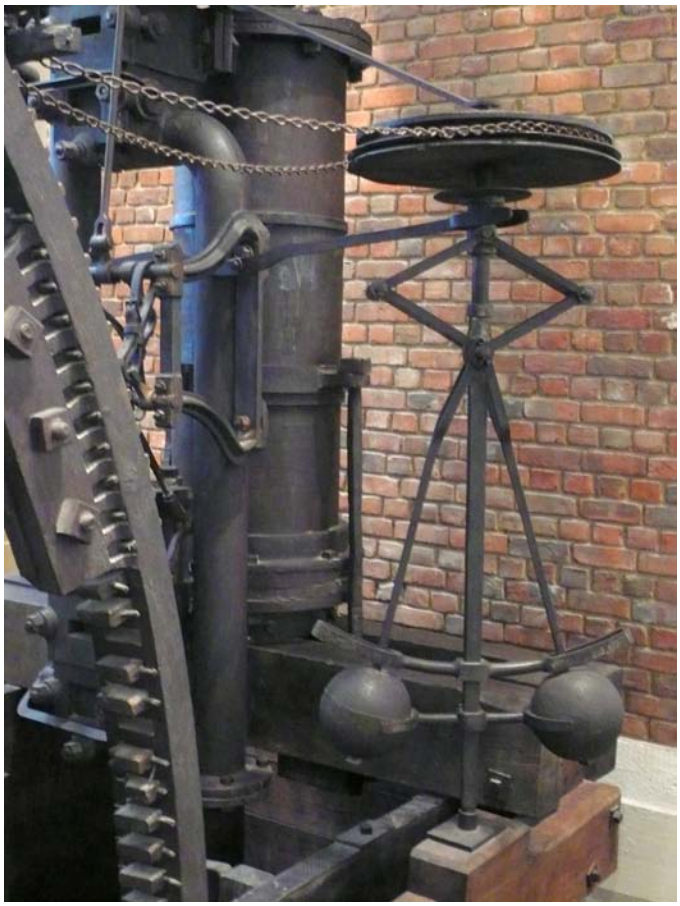


Figure 2.14: Watt's flyball governor.

describe activities related to feedback and control.[40]

The English psychiatrist W. Ross Ashby (1903–1972; Fig. 2.15) contributed to the field of cybernetics by his study of "ultrastability" and "homeostasis." According to Ashby, ultrastability is the capacity of a system to reach a stable state under a wide variety of environmental conditions. To illustrate the idea, he built an electromechanical device called the "homeostat." It consisted of four pivoted magnets whose positions were rendered interdependent through feedback mechanisms. If the position of any was disturbed, the effects on the others and then back on itself would result in all of them returning to an equilibrium condition. Ashby described this device in Chapter 8 of his influential 1952 book *Design For a Brain*. His ideas had an influence on several AI researchers. My "teleo-reactive programs," to be

described later, were motivated in part by the idea of homeostasis.

Figure 2.15: W. Ross Ashby, Warren McCulloch, Grey Walter, and Norbert Wiener at a Meeting in Paris. (From P. de Latil, *Thinking by Machine: A Study of Cybernetics*, Boston: Houghton, Mifflin, 1957.)

Another source of ideas, loosely associated with cybernetics and bionics, came from studies of "self-organizing systems." Many unorganized combinations of simple parts, including combinations of atoms and molecules, respond to energetic "jostling" by falling into stable states in which the parts are organized in more complex assemblies. An online dictionary devoted to cybernetics and systems theory has a nice example: "A chain made out of paper clips suggests that someone has taken the trouble to link paper clips together to make a chain. It is not in the nature of paper clips to make themselves up into a chain. But, if you take a number of paper clips, open them up slightly and then shake them all together in a cocktail shaker, you will find at the end that the clips have organized themselves into short or long chains. The chains are not so neat as chains put together by hand but, nevertheless, they are chains."[41]

The term "self-organizing" seems to have been first introduced by Ashby in 1947.[42] Ashby emphasized that self-organization is not a property of an organism itself, in response to its environment and experience, but a property of the organism and its environment *taken together*. Although self-organization appears to be important in ideas about how life originated, it is unclear whether or not it provides clues for building intelligent machines.

2.3.2 Statistics and Probability

Because nearly all reasoning and decision making take place in the presence of uncertainty, dealing with uncertainty plays an important role in the automation of intelligence. Attempts to quantify uncertainty and "the laws of chance" gave rise to statistics and probability theory. What would turn out to be one of the most important results in probability theory, at least for artificial intelligence, is Bayes's rule, which I'll define presently in the context of an example. The rule is named for Reverend Thomas Bayes (1702–1761), an English clergyman.[43]

One of the important applications of Bayes's rule is in signal detection. Let's suppose a radio receiver is tuned to a station that after midnight broadcasts (randomly) one of two tones, either tone A or tone B, and on a particular night we want to decide which one is being broadcast. On any given day, we do not know ahead of time which tone is to be broadcast that night, but suppose we do know their probabilities. (For example, it might be that both tones are equally probable.) Can we find out which tone is being broadcast by listening to the signal coming in to the receiver? Well, listening can't completely resolve the matter because the station is far away, and random noise partially obscures the tone. However, depending on the nature of the obscuring noise, we can often calculate the probability that the actual tone that night is A (or that it is B). Let's call the signal y and the actual tone x (which can be either A or B). The probability that $x = A$, given the evidence for it contained in the incoming signal, y, is written as $p(x = A \mid y)$ and read as "the probability that x is A, given that the signal is y." The probability that $x = B$, given the same evidence is $p(x = B \mid y)$.

A reasonable "decision rule" would be to decide in favor of tone A if $p(x = A \mid y)$ is larger than $p(x = B \mid y)$. Otherwise, decide in favor of tone B. (There is a straightforward adjustment to this rule that takes into account differences in the "costs" of the two possible errors.) The problem in applying this rule is that these two probabilities are not readily calculable, and that is where Bayes's rule comes in. It allows us to calculate these probabilities in terms of other probabilities that are more easily guessed or otherwise obtainable. Specifically, Bayes's rule is

$$p(x \mid y) = p(y \mid x)p(x)/p(y).$$

Using Bayes's rule, our decision rule can now be reformulated as

> Decide in favor of tone A if $p(y \mid x = A)p(x = A)/p(y)$ is greater than $p(y \mid x = B)p(x = B)/p(y)$. Otherwise, decide in favor of tone B.

Because $p(y)$ occurs in both expressions and therefore does not affect which one is larger, the rule simplifies to

> Decide in favor of tone A if $p(y \mid x = A)p(x = A)$ is greater than $p(y \mid x = B)p(x = B)$. Otherwise, decide in favor of tone B.

We assume that we know the *a priori* probabilities of the tones, namely, $p(x = A)$ and $p(x = B)$, so it remains only for us to calculate $p(y \mid x)$ for $x = A$ and $x = B$. This expression is called the *likelihood* of y given x. When the two probabilities, $p(x = A)$ and $p(x = B)$, are equal (that is, when both tones are equally probable *a priori*), then we can decide in favor of which likelihood is greater. Many decisions that are made in the presence of uncertainty use this "maximum-likelihood" method . The calculation for these likelihoods depends on how we represent the received signal, y, and on the statistics of the interfering noise.

In my example, y is a radio signal, that is, a voltage varying in time. For computational purposes, this time-varying voltage can be represented by a sequence of samples of its values at appropriately chosen, uniformly spaced time points, say $y(t_1)$, $y(t_2)$, $\ldots y(t_i)$, $\ldots$, $y(t_N)$. When noise alters these values from what they would have been without noise, the probability of the sequence of them (given the cases when the tone is A and when the tone is B) can be calculated by using the known statistical properties of the noise. I won't go into the details here except to say that, for many types of noise statistics, these calculations are quite straightforward.

In the twentieth century, scientists and statisticians such as Karl Pearson (1857–1936), Sir Ronald A. Fisher (1890–1962), Abraham Wald (1902–1950), and Jerzey Neyman (1894–1981) were among those who made important contributions to the use of statistical and probabilistic methods in estimating parameters and in making decisions. Their work set the foundation for some of the first engineering applications of Bayes's rule, such as the one I just illustrated, namely, deciding which, if any, of two or more electrical signals is present in situations where noise acts to obscure the signals. A paper by the American engineers David Van Meter and David Middleton, which I read as a beginning graduate student in 1955, was my own introduction to these applications.[44] For artificial intelligence, these uses of Bayes's rule provided clues about how to mechanize the perception of both speech sounds and visual images. Beyond perception, Bayes's rule lies at the center of much other modern work in artificial intelligence.

2.3.3 The Computer

A. Early Computational Devices

Proposals such as those of Leibniz, Boole, and Frege can be thought of as early attempts to provide foundations for what would become the "software" of artificial intelligence. But reasoning and all the other aspects of intelligent behavior require, besides software, some sort of *physical* engine. In humans

and other animals, that engine is the brain. The simple devices of Grey Walter and Ross Ashby were, of course, physical manifestations of their ideas. And, as we shall see, early networks of neuron-like units were realized in physical form. However, to explore the ideas inherent in most of the clues from logic, from neurophysiology, and from cognitive science, more powerful engines would be required. While McCulloch, Wiener, Walter, Ashby, and others were speculating about the machinery of intelligence, a very powerful and essential machine bloomed into existence – the general-purpose digital computer. This single machine provided the engine for all of these ideas and more. It is by far the dominant hardware engine for automating intelligence.

Building devices to compute has a long history. William Aspray has edited an excellent book, *Computing Before Computers*, about computing's early days.[45] The first machines were able to do arithmetic calculations, but these were not programmable. Wilhelm Schickard (1592–1635; Fig. 2.16) built one of the first of these in 1623. It is said to have been able to add and subtract six-digit numbers for use in calculating astronomical tables. The machine could "carry" from one digit to the next.

In 1642 Blaise Pascal (1623–1662; Fig. 2.16) created the first of about fifty of his computing machines. It was an adding machine that could perform automatic carries from one position to the next. "The device was contained in a box that was small enough to fit easily on top of a desk or small table. The upper surface of the box... consisted of a number of toothed wheels, above which were a series of small windows to show the results. In order to add a number, say 3, to the result register, it was only necessary to insert a small stylus into the toothed wheel at the position marked 3 and rotate the wheel clockwise until the stylus encountered the fixed stop..."[46]

Figure 2.16: Wilhelm Schickard (left) and Blaise Pascal (right).

Inspired by Pascal's machines, Gottfried Leibniz built a mechanical multiplier called the "Step Reckoner" in 1674. It could add, subtract, and do multiplication (by repeated additions). "To multiply a number by 5, one simply turned the crank five times."[47]

Several other calculators were built in the ensuing centuries. A particularly interesting one, which was too complicated to build in its day, was designed in 1822 by Charles Babbage (1791–1871), an English mathematician and inventor. (See Fig. 2.17.) Called the "Difference Engine," it was to have calculated mathematical tables (of the kind used in navigation at sea, for example) using the method of finite differences. Babbage's Difference Engine No. 2 was actually constructed in 1991 (using Babbage's designs and nineteenth-century mechanical tolerances) and is now on display at the London Science Museum. The Museum arranged for another copy to be built for Nathan Myhrvold, a former Microsoft Chief Technology Officer. (A description of the machine and a movie is available from a Computer History Museum Web page at http://www.computerhistory.org/babbage/.)

Adding machines, however, can only add and subtract (and, by repetition of these operations, also multiply and divide). These are important operations but not the only ones needed. Between 1834 and 1837 Babbage worked on the design of a machine called the "Analytical Engine," which embodied most of the ideas needed for general computation. It could store intermediate results in a "mill," and it could be programmed. However, its proposed realization as a collection of steam-driven, interacting brass gears and cams ran into funding difficulties and was never constructed.

Ada Lovelace (1815–1852), the daughter of Lord Byron, has been called the "world's first programmer" for her alleged role in devising programs for the Analytical Engine. However, in the book *Computing Before Computers* the following claim is made:[48]

> This romantically appealing image is without foundation. All but one of the programs cited in her notes [to her translation of an account of a lecture Babbage gave in Turin, Italy] had been prepared by Babbage from three to seven years earlier. The exception was prepared by Babbage for her, although she did detect a "bug" in it. Not only is there no evidence that Ada Lovelace ever prepared a program for the Analytical Engine but her correspondence with Babbage shows that she did not have the knowledge to do so.

For more information about the Analytical Engine and an emulator and programs for it, see http://www.fourmilab.ch/babbage/.

Practical computers had to await the invention of electrical, rather than brass, devices. The first computers in the early 1940s used electromechanical relays. Vacuum tubes (thermionic valves, as they say in Britain) soon won out

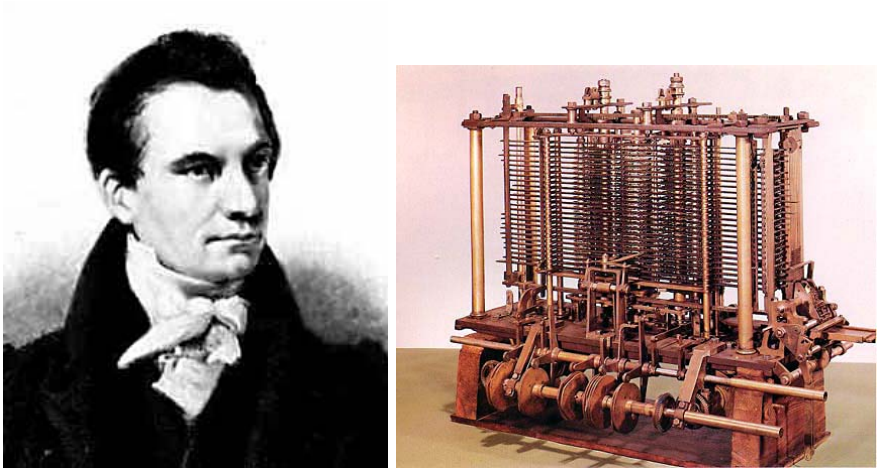


Figure 2.17: Charles Babbage (left) and a model of his Analytical Engine (right).

because they permitted faster and more reliable computation. Nowadays, computers use billions of tiny transistors arrayed on silicon wafers. Who knows what might someday replace them?

B. Computation Theory

Even before people actually started building computers, several logicians and mathematicians in the 1930s pondered the problem of just what could be computed. Alonzo Church came up with a class of functions that could be computed, ones he called "recursive."[49] The English logician and mathematician, Alan Turing (1912–1954; Fig. 2.18), proposed what is now understood to be an equivalent class – ones that could be computed by an imagined machine he called a "logical computing machine (LCM)," nowadays called a "Turing machine."[50] (See Fig. 2.19.) The claim that these two notions are equivalent is called the "Church–Turing Thesis." (The claim has not been proven, but it is strongly supported by logicians and no counterexample has ever been found.)[51]

The Turing machine is a hypothetical computational device that is quite simple to understand. It consists of just a few parts. There is an infinite tape (which is one reason the device is just imagined and not actually built) divided into cells and a tape drive. Each cell has printed on it either a 1 or a 0. The machine also has a read–write head positioned over one cell of the tape. The read function reads what is on the tape. There is also a logic unit that can decide, depending on what is read and the state of the logic machine, to change its own state, to command the write function to write either a 1 or a 0 on the

Figure 2.18: Alan Mathison Turing. (Photograph by Elliott & Fry © and used with permission of the National Portrait Gallery, London.)

cell being read (possibly replacing what is already there), to move the tape one cell to the left or to the right (at which time the new cell is read and so on), or to terminate operation altogether. The input (the "problem" to be computed) is written on the tape initially. (It turns out that any such input can be coded into 1's and 0's.) When, and if, the machine terminates, the output (the coded "answer" to the input problem) ends up being printed on the tape.

Turing proved that one could always specify a particular logic unit (the part that decides on the machine's actions) for his machine such that the machine would compute any computable function. More importantly, he showed that one could encode on the tape itself a prescription for any logic unit specialized for a particular problem and then use a general-purpose logic unit for *all* problems. The encoding for the special-purpose logic unit can be thought of as the "program" for the machine, which is stored on the tape (and thus subject to change by the very operation of the machine!) along with the description of the problem to be solved. In Turing's words, "It can be shown that a single special machine of that type can be made to do the work of all.

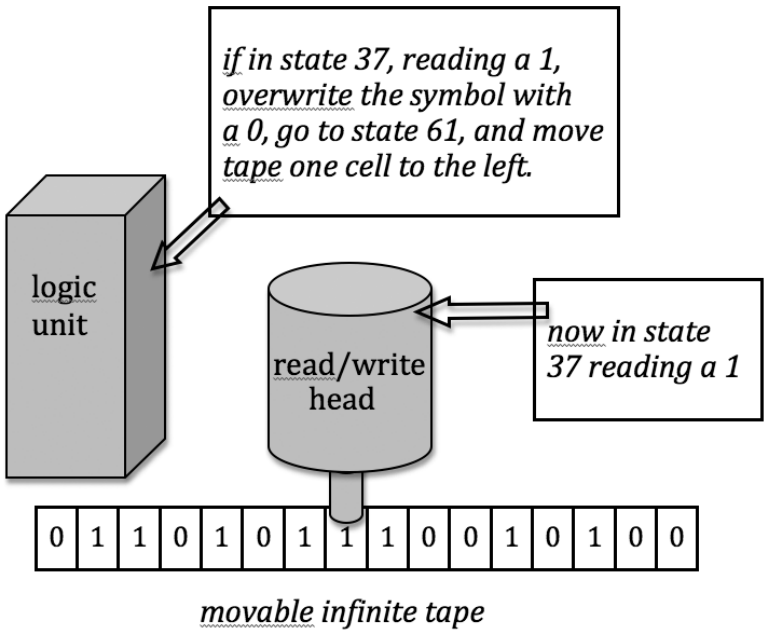


Figure 2.19: A Turing machine.

It could in fact be made to work as a model of any other machine. The special machine may be called the universal machine."[52]

C. Digital Computers

Somewhat independently of Turing, engineers began thinking about how to build actual computing devices consisting of programs and logical circuitry for performing the instructions contained in the programs. Some of the key ideas for designing the logic circuits of computers were developed by the American mathematician and inventor Claude Shannon (1916–2001; Fig. 2.20).[53] In his 1937 Yale University master's thesis[54] Shannon showed that Boolean algebra and binary arithmetic could be used to simplify telephone switching circuits. He also showed that switching circuits (which can be realized either by combinations of relays, vacuum tubes, or whatever) could be used to implement operations in Boolean logic, thus explaining their importance in computer design.

It's hard to know who first thought of the idea of storing a computer's program along with its data in the computer's memory banks. Storing the program allows changes in the program to be made easily, but more

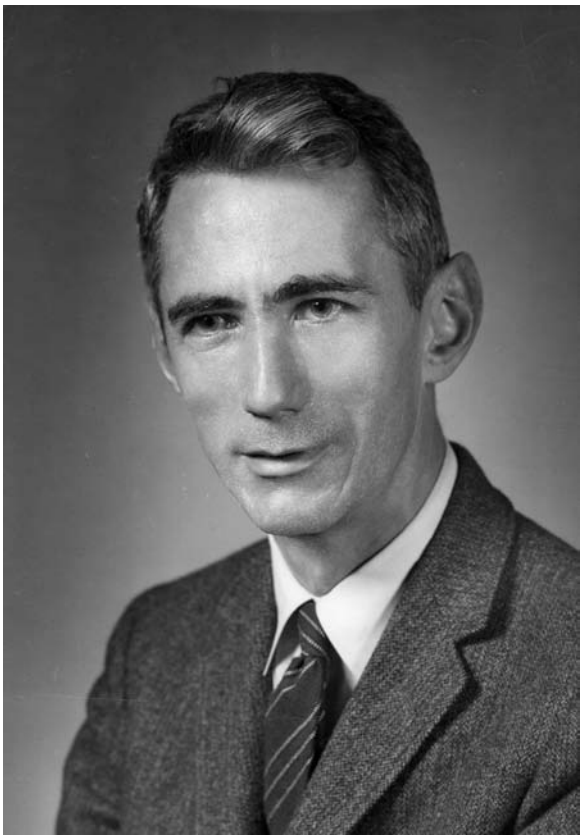


Figure 2.20: Claude Shannon. (Photograph courtesy of MIT Museum.)

importantly it allows the program to change itself by changing appropriate parts of the memory where the program is stored. Among those who might have thought of this idea first are the German engineer Konrad Zuse (1910–1995) and the American computer pioneers J. Presper Eckert (1919–1995) and John W. Mauchly (1907–1980). (Of course Turing had already proposed storing what amounted to a program on the tape of a universal Turing machine.)

For an interesting history of Konrad Zuse's contributions, see the family of sites available from http://irb.cs.tu-berlin.de/~zuse/Konrad_Zuse/en/index.html. One of these mentions that "it is undisputed that Konrad Zuse's Z3 was the first fully functional, program controlled (freely programmable) computer of the world. ... The Z3 was presented on May 12, 1941, to an audience of scientists in Berlin." Instead of vacuum tubes, it used 2,400 electromechanical relays. The original Z3 was destroyed by an Allied air raid on December 21, 1943.[55] A

reconstructed version was built in the early 1960s and is now on display at the Deutsche Museum in Munich. Zuse also is said to have created the first programming language, called the Plankalkül.

The American mathematician John von Neumann (1903–1957) wrote a "draft report" about the EDVAC, an early stored-program computer.[56] Perhaps because of this report, we now say that these kinds of computers use a "von Neumann architecture." The ideal von Neumann architecture separates the (task-specific) stored program from the (general-purpose) hardware circuitry, which can execute (sequentially) the instructions of any program whatsoever. (We usually call the program "software" to distinguish it from the "hardware" part of a computer. However, the distinction is blurred in most modern computers because they often have some of their programs built right into their circuitry.)

Other computers with stored programs were designed and built in the 1940s in Germany, Great Britain, and the United States. They were large, bulky machines. In Great Britain and the United States they were mainly used for military purposes. Figure 2.21 shows one such machine.

Figure 2.21: The Cambridge University EDSAC computer (circa 1949). (Photograph used with permission of the Computer Laboratory, University of Cambridge ©.)

We call computers "machines" even though today they can be made completely electrical with no moving parts whatsoever. Furthermore, when we speak of computing machines we usually mean the combination of the computer and the program it is running. Sometimes we even call just the program a machine. (As an example of this usage, I'll talk later about a "checker-playing machine" and mean a program that plays checkers.)

The commanding importance of the stored-program digital computer derives from the fact that it can be used for any purpose whatsoever – that is, of course, any computational purpose. The modern digital computer is, for all practical purposes, such a universal machine. The "all-practical-purposes" qualifier is needed because not even modern computers have the infinite storage capacity implied by Turing's infinite tape. However, they do have prodigious amounts of storage, and that makes them practically universal.

D. "Thinking" Computers

After some of the first computers were built, Turing reasoned that if they were practically universal, they should be able to do anything. In 1948 he wrote, "The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of 'programming' the universal machine to do these jobs."[57]

Among the things that Turing thought could be done by computers was mimicking human intelligence. One of Turing's biographers, Andrew Hodges, claims, "he decided the scope of the computable encompassed far more than could be captured by explicit instruction notes, and quite enough to include all that human brains did, however creative or original. Machines of sufficient complexity would have the capacity for evolving into behaviour that had never been explicitly programmed."[58]

The first modern article dealing with the possibility of mechanizing *all* of human-style intelligence was published by Turing in 1950.[59] This paper is famous for several reasons. First, Turing thought that the question "Can a machine think?" was too ambiguous. Instead he proposed that the matter of machine intelligence be settled by what has come to be called "the Turing test."

Although there have been several reformulations (mostly simplifications) of the test, here is how Turing himself described it:

> The new form of the problem [Can machines think?] can be described in terms of a game which we call the "imitation game." It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the

> interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A." The interrogator is allowed to put questions to A and B thus:
>
> C: Will X please tell me the length of his or her hair?
>
> Now suppose X is actually A, then A must answer. It is A's object in the game to try and cause C to make the wrong identification. His answer might therefore be
>
> "My hair is shingled, and the longest strands are about nine inches long."
>
> In order that tones of voice may not help the interrogator the answers should be written, or better still, typewritten. The ideal arrangement is to have a teleprinter communicating between the two rooms. Alternatively the question and answers can be repeated by an intermediary. The object of the game for the third player (B) is to help the interrogator. The best strategy for her is probably to give truthful answers. She can add such things as "I am the woman, don't listen to him!" to her answers, but it will avail nothing as the man can make similar remarks.
>
> We now ask the question, "What will happen when a machine takes the part of A in this game?" Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, "Can machines think?"

Most of the simplifications of Turing's test involve attempting to have a computer "in the other room" fool an interrogator into believing the computer is a human. To stimulate progress toward that goal, a prize has been established – annual prizes in fact. In the words of the Web site announcing the prizes, "In 1990 Hugh Loebner agreed with The Cambridge Center for Behavioral Studies to underwrite a contest designed to implement the Turing Test. Dr. Loebner pledged a Grand Prize of $100,000 and a Gold Medal for the first computer whose responses were indistinguishable from a human's. Such a computer can be said 'to think.' Each year an annual prize of $2000 and a bronze medal is awarded to the most human-like computer. The winner of the annual contest is the best entry relative to other entries that year, irrespective of how good it is in an absolute sense."[60]

However, several people have pointed to inadequacies of both the simplifications and even of the original test.[61] In fact, it has been alleged that even Joseph Weizenbaum's patently unintelligent 1966 conversational program, ELIZA, has fooled people into thinking it is human.[62]

For example, here is part of a transcript of a conversation with ELIZA:

ELIZA: How do you do. Please state your problem.

> I spend too much time with computers.

ELIZA: Why do you mention computers?

> You are one.

ELIZA: Does it please you to believe I am one?

> Yes.

ELIZA: You seem to be quite positive.

> I suppose I am.

ELIZA: How long have you been?

> 31 years.

ELIZA: I'm not sure I understand you fully.

> Join the club.

ELIZA: Please go on.

> You don't make too much sense yourself.

ELIZA: We were discussing you – not me.

The first part of the conversation seems reasonable, but ELIZA bogs down in the middle because the program was expecting "I suppose I am" to be followed by some word like "happy." (There are several ELIZA simulations on the Web. One that claims to reproduce faithfully the original ELIZA program is at http://www.chayden.net/eliza/Eliza.html. Try one out!)

A second important feature of Turing's 1950 paper was his handling of arguments that people might raise against the possibility of achieving intelligent computers. I'll quote the ones Turing mentions:

> (1) The Theological Objection: Thinking is a function of man's immortal soul. God has given an immortal soul to every man and woman, but not to any other animal or to machines. Hence no animal or machine can think.
>
> (2) The 'Heads in the Sand' Objection: "The consequences of machines thinking would be too dreadful. Let us hope and believe that they cannot do so."
>
> (3) The Mathematical Objection: There are a number of results of mathematical logic that can be used to show that there are limitations to the powers of discrete-state machines.
>
> (4) The Argument from Consciousness: This argument is very well expressed in Professor Jefferson's Lister Oration for 1949, from which I quote:
>
> "Not until a machine can write a sonnet or compose a concerto because of thoughts and emotions felt, and not by the chance fall

> of symbols, could we agree that machine equals brain – that is, not only write it but know that it had written it. No mechanism could feel (and not merely artificially signal, an easy contrivance) pleasure at its successes, grief when its valves fuse, be warmed by flattery, be made miserable by its mistakes, be charmed by sex, be angry or depressed when it cannot get what it wants."
>
> (5) Arguments from Various Disabilities: These arguments take the form, "I grant you that you can make machines do all the things you have mentioned but you will never be able to make one to do X."
>
> (6) Lady Lovelace's Objection: Our most detailed information of Babbage's Analytical Engine comes from a memoir by Lady Lovelace. In it she states, "The Analytical Engine has no pretensions to originate anything. It can do *whatever we know* how *to order it* to perform" (her italics).
>
> (7) Argument from Continuity in the Nervous System: The nervous system is certainly not a discrete-state machine. A small error in the information about the size of a nervous impulse impinging on a neuron may make a large difference to the size of the outgoing impulse. It may be argued that, this being so, one cannot expect to be able to mimic the behavior of the nervous system with a discrete-state system.
>
> (8) The Argument from Informality of Behavior: It is not possible to produce a set of rules purporting to describe what a man should do in every conceivable set of circumstances.
>
> (9) The Argument from Extra-Sensory Perception.

In his paper, Turing nicely (in my opinion) handles all of these points, with the possible exception of the last one (because he apparently thought that extra-sensory perception was plausible). I'll leave it to you to read Turing's 1950 paper to see his counterarguments.

The third important feature of Turing's 1950 paper is his suggestion about how we might go about producing programs with human-level intellectual abilities. Toward the end of his paper, he suggests, "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain." This suggestion is really the source for the idea mentioned earlier about using an ontogenetic strategy to develop intelligent machines.

Allen Newell and Herb Simon (see Fig. 2.22) were among those who had no trouble believing that the digital computer's universality meant that it could be used to mechanize intelligence in all its manifestations – provided it had the right software. In their 1975 ACM Turing Award lecture,[63] they

described a hypothesis that they had undoubtedly come to believe much earlier, the "Physical Symbol System Hypothesis." It states that "a physical symbol system has the necessary and sufficient means for intelligent action." Therefore, according to the hypothesis, appropriately programmed digital computers would be capable of intelligent action. Conversely, because humans are capable of intelligent action, they must be, according to the hypothesis, physical symbol systems. These are very strong claims that continue to be debated.

Figure 2.22: Herbert Simon (seated) and Allen Newell (standing). (Courtesy of Carnegie Mellon University Archives.)

Both the imagined Turing machine and the very real digital computer are symbol systems in the sense Newell and Simon meant the phrase. How can a Turing machine, which uses a tape with 0's and 1's printed on it, be a "symbol system"? Well, the 0's and 1's printed on the tape can be thought of as symbols standing for their associated numbers. Other symbols, such as "A" and "M," can be encoded as sequences of primitive symbols, such as 0's and 1's. Words can be encoded as sequences of letters, and so on. The fact that one commonly thinks of a digital computer as a machine operating on 0's and 1's need not prevent us from thinking of it also as operating on more complex symbols. After all, we are all used to using computers to do "word processing" and to send e-mail.

Newell and Simon admitted that their hypothesis could indeed be false: "Intelligent behavior is not so easy to produce that any system will exhibit it willy-nilly. Indeed, there are people whose analyses lead them to conclude either on philosophical or on scientific grounds that the hypothesis is false. Scientifically, one can attack or defend it only by bringing forth empirical evidence about the natural world." They conclude the following:

> The symbol system hypothesis implies that the symbolic behavior of man arises because he has the characteristics of a physical symbol system. Hence, the results of efforts to model human behavior with symbol systems become an important part of the evidence for the hypothesis, and research in artificial intelligence goes on in close collaboration with research in information processing psychology, as it is usually called.

Although the hypothesis was not formally described until it appeared in the 1976 article, it was certainly implicit in what Turing and other researchers believed in the 1950s. After Allen Newell's death, Herb Simon wrote, "From the very beginning something like the physical symbol system hypothesis was embedded in the research."[64]

Inspired by the clues we have mentioned and armed with the general-purpose digital computer, researchers began, during the 1950s, to explore various paths toward mechanizing intelligence. With a firm belief in the symbol system hypothesis, some people began programming computers to attempt to get them to perform some of the intellectual tasks that humans could perform. Around the same time, other researchers began exploring approaches that did not depend explicitly on symbol processing. They took their inspiration mainly from the work of McCulloch and Pitts on networks of neuron-like units and from statistical approaches to decision making. A split between symbol-processing methods and what has come to be called "brain-style" and "nonsymbolic" methods still survives today.

Notes

1. Aristotle, *Prior Analytics, Book I*, written circa 350 BCE, translated by A. J. Jenkinson, Web addition published by eBooks@Adelaide, available online at http://etext.library.adelaide.edu.au/a/aristotle/a8pra/. [**27**]

2. Medieval students of logic gave names to the different syllogisms they studied. They used the mnemonic *Barbara* for this one because each of the three statements begins with "All," whose first letter is "A." The vowels in "Barbara" are three"a"s. [**27**]

3. From Martin Davis, *The Universal Computer: The Road from Leibniz to Turing*, New York: W. W. Norton & Co., 2000. For an excerpt from the paperback version containing this quotation, see http://www.wwnorton.com/catalog/fall01/032229EXCERPT.htm. [**28**]

4. Quotation from William Aspray (ed.), *Computing Before Computers*, Chapter 3, "Logic

Machines," pp. 107–8, Ames, Iowa: Iowa State Press, 1990. (Also available from http://ed-thelen.org/comp-hist/CBC.html.) [**30**]

5. Robert Harley, "The Stanhope Demonstrator, *Mind*, Vol. IV, pp. 192–210, 1879. [**31**]

6. George Boole, *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*, Dover Publications, 1854. [**31**]

7. See D. McHale, *George Boole: His Life and Work*, Dublin, 1985. This excerpt was taken from http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Boole.html. [**32**]

8. See, for example, Gerard O'Regan, *A Brief History of Computing*, p. 17, London: Springer-Verlag, 2008. [**32**]

9. I follow the pictorial version used in the online Stanford Encyclopedia of Philosophy (http://plato.stanford.edu/entries/frege/), which states that "... we are modifying Frege's notation a bit so as to simplify the presentation; we shall not use the special typeface (Gothic) that Frege used for variables in general statements, or observe some of the special conventions that he adopted. ..." [**33**]

10. Warren S. McCulloch and Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, Chicago: University of Chicago Press, 1943. (See Marvin Minsky, *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall, 1967, for a very readable treatment of the computational aspects of "McCulloch–Pitts neurons.") [**34**]

11. Donald O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, New York: John Wiley, Inc., 1949. [**36**]

12. For more about Hebb, see http://www.cpa.ca/Psynopsis/special_eng.html. [**36**]

13. For a summary of the lives and work of both men, see a Web page entitled "Wilhelm Wundt and William James" by Dr. C. George Boeree at http://www.ship.edu/~cgboeree/wundtjames.html. [**38**]

14. M. Minsky (ed.), "Introduction," *Semantic Information Processing*, p. 2, Cambridge, MA: MIT Press, 1968. [**40**]

15. Russell A. Kirsch, "Experiments with a Computer Learning Routine," Computer Seminar Notes, July 30, 1954. Available online at http://www.nist.gov/msidlibrary/doc/kirsch_1954_artificial.pdf. [**40**]

16. B. F. Skinner, *Verbal Behavior*, Engelwood Cliffs, NJ: Prentice Hall, 1957. [**40**]

17. Noam Chomsky, "A Review of B. F. Skinner's *Verbal Behavior*," in Leon A. Jakobovits and Murray S. Miron (eds.), *Readings in the Psychology of Language*, Engelwood Cliffs, NJ: Prentice-Hall, 1967. Available online at http://www.chomsky.info/articles/1967----.htm. [**40**]

18. See, for example, N. Chomsky, *Aspects of the Theory of Syntax*, Cambridge: MIT Press, 1965. [**41**]

19. George A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, Vol. 63, pp. 81–97, 1956. [**41**]

20. *IRE Transactions on Information Theory*, Vol IT-2, 1956. [**42**]

21. For a copy of his paper, see http://www.chomsky.info/articles/195609--.pdf. [**42**]

22. George A. Miller, "A Very Personal History," MIT Center for Cognitive Science Occasional Paper No. 1, 1979. [**42**]

23. George A. Miller, E. Galanter, and K. H. Pribram, *Plans and the Structure of Behavior*, New York: Holt, Rinehart & Winston, 1960. [**42**]

24. For a thorough history of cognitive science, see Margaret A. Boden, *Mind As Machine:*

A History of Cognitive Science, vols. 1 and 2, Oxford: Clarendon Press, 2006. For an earlier, one-volume treatment, see Howard E. Gardner, *The Mind's New Science: A History of the Cognitive Revolution*, New York: Basic Books, 1985. [**42**]

25. An English translation appeared later: N.A. Barricelli, "Symbiogenetic Evolution Processes Realized by Artificial Methods," *Methodos*, Vol. 9, Nos. 35–36, pp. 143–182, 1957. For a summary of Barricelli's experiments, see David B. Fogel, "Nils Barricelli – Artificial Life, Coevolution, Self-Adaptation," *IEEE Computational Intelligence Magazine*, Vol. 1, No. 1, pp. 41–45, February 2006. [**43**]

26. R. M. Friedberg, "A Learning Machine: Part I," *IBM Journal of Research and Development*, Vol. 2, No. 1, pp. 2–13, 1958, and R. M. Friedberg, B. Dunham, and J. H. North, "A Learning Machine: Part II," *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 282–287, 1959. The papers are available (for a fee) at http://www.research.ibm.com/journal/rd/021/ibmrd0201B.pdf and http://www.research.ibm.com/journal/rd/033/ibmrd0303H.pdf. [**43**]

27. Marvin L. Minsky, "Steps Toward Artificial Intelligence," *Proceedings of the Institute of Radio Engineers*, Vol. 49, pp. 8–30, 1961. Paper available at http://web.media.mit.edu/~minsky/papers/steps.html. [**43**]

28. Lawrence J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, New York: Wiley, 1966. [**44**]

29. Woodrow W. Bledsoe, "The Evolutionary Method in Hill Climbing: Convergence Rates," Technical Report, Panoramic Research, Inc., Palo Alto, CA, 1962.; Hans J. Bremermann, "Optimization through Evolution and Recombination, M. C. Yovits, G. T. Jacobi, and G. D. Goldstein (eds.), *Self-Organizing Systems*, pp. 93–106, Washington, DC: Spartan Books, 1962. [**44**]

30. Jürgen Schmidhuber, "2006: Celebrating 75 Years of AI – History and Outlook: The Next 25 Years," in Max Lungarella et al. (eds.), *50 Years of Artificial Intelligence: Essays Dedicated to the 50th Anniversary of Artificial Intelligence*, Berlin: Springer-Verlag, 2007. Schmiduber cites Ingo Rechenberg, "Evolutionsstrategie – Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution," Ph.D. dissertation, 1971 (reprinted by Frommann-Holzboog Verlag, Stuttgart, 1973). [**44**]

31. See http://www.aaai.org/AITopics/html/genalg.html. [**44**]

32. John H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975. Second edition, MIT Press, 1992. [**44**]

33. W. Grey Walter, "An Imitation of Life," *Scientific American*, pp. 42–45, May 1950. See also W. Grey Walter, *The Living Brain*, London: Gerald Duckworth & Co. Ltd., 1953. [**44**]

34. B. Inhelder and J. Piaget, *The Growth of Logical Thinking from Childhood to Adolescence*, New York: Basic Books, 1958. For a summary of these stages, see the following Web pages: http://www.childdevelopmentinfo.com/development/piaget.shtml and http://www.ship.edu/~cgboeree/piaget.html. [**46**]

35. *Proceedings of the Bionics Symposium: Living Prototypes – the Key to new Technology,* Technical Report 60-600, Wright Air Development Division, Dayton, Ohio, 1960. [**46**]

36. *Proceedings of the Third Bionics Symposium*, Aerospace Medical Division, Air Force Systems Command, United States Air Force, Wright-Patterson AFB, Ohio, 1963. [**46**]

37. http://www.mlahanas.de/Greeks/Ctesibius1.htm. [**49**]

38. http://www.asc-cybernetics.org/foundations/timeline.htm. [**49**]

39. From http://www.nickgreen.pwp.blueyonder.co.uk/control.htm. [**49**]

40. For a history of cybernetics, see a Web page of the American Society for Cybernetics at http://www.asc-cybernetics.org/foundations/history.htm. [**50**]

41. From http://pespmc1.vub.ac.be/ASC/SELF-ORGANI.html. [**51**]

42. W. Ross Ashby, "Principles of the Self-Organizing Dynamic System," *Journal of General Psychology*, Vol. 37, pp. 125–128, 1947. See also the Web pages at http://en.wikipedia.org/wiki/Self_organization. [**51**]

43. Bayes wrote an essay that is said to have contained a version of the rule. Later, the Marquis de Laplace (1749–1827) generalized (some say independently) what Bayes had done. For a version of Bayes's essay (posthumously written up by Richard Price), see http://www.stat.ucla.edu/history/essay.pdf. [**52**]

44. David Van Meter and David Middleton, "Modern Statistical Approaches to Reception in Communication Theory," Symposium on Information Theory, *IRE Transactions on Information Theory*, PGIT-4, pp. 119–145, September 1954. [**53**]

45. William Aspray (ed.), *Computing Before Computers*, Ames, Iowa: Iowa State University Press, 1990. Available online at http://ed-thelen.org/comp-hist/CBC.html. [**54**]

46. *Ibid*, Chapter 1. [**54**]

47. *Ibid.* [**55**]

48. *Ibid*, Chapter 2. [**55**]

49. Alonzo Church, "An Unsolvable Problem of Elementary Number Theory," *American Journal of Mathematics*, Vol. 58, pp. 345–363, 1936. [**56**]

50. Alan M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, Series 2, Vol. 42, pp. 230–265, 1936–1937. [**56**]

51. For more information about Turing, his life and works, see the Web pages maintained by the Turing biographer, Andrew Hodges, at http://www.turing.org.uk/turing/. [**56**]

52. The quotation is from Alan M. Turing, "Lecture to the London Mathematical Society," p. 112, typescript in King's College, Cambridge, published in Alan M. Turing's *ACE Report of 1946 and Other Papers* (edited by B. E. Carpenter and R. W. Doran, Cambridge, MA: MIT Press, 1986), and in Volume 3 of *The Collected Works of A. M. Turing* (edited D. C. Ince, Amsterdam: North-Holland 1992). [**58**]

53. For a biographical sketch, see http://www.research.att.com/~njas/doc/shannonbio.html. [**58**]

54. In his book *The Mind's New Science*, Howard Gardner called this thesis "possibly the most important, and also the most famous, master's thesis of the century." [**58**]

55. Various sources give different dates for the air raid, but a letter in the possession of Zuse's son, Horst Zuse, gives the 1943 date (according to an e-mail sent me on February 10, 2009, by Wolfgang Bibel, who has communicated with Horst Zuse). [**59**]

56. A copy of the report, plus introductory commentary, can be found at http://qss.stanford.edu/~godfrey/. [**60**]

57. Alan M. Turing, "Intelligent Machinery," National Physical Laboratory Report, 1948. Reprinted in B. Meltzer and D. Michie (eds), *Machine Intelligence 5*, Edinburgh: Edinburgh University Press, 1969. A facsimile of the report is available online at http://www.AlanTuring.net/intelligent_machinery. [**61**]

58. Andrew Hodges, *Turing*, London: Phoenix, 1997. [**61**]

59. Alan M. Turing, "Computing Machinery and Intelligence," *Mind*, Vol. LIX, No. 236, pp. 433–460, October 1950. (Available at http://www.abelard.org/turpap/turpap.htm.) [**61**]

60. See the "Home Page of the Loebner Prize in Artificial Intelligence" at http://www.loebner.net/Prizef/loebner-prize.html. [**62**]

61. For discussion, see the Wikipedia article at http://en.wikipedia.org/wiki/Turing_test.

[**62**]

62. Joseph Weizenbaum, "ELIZA—A Computer Program for the Study of Natural Language Communication between Man and Machine," *Communications of the ACM*, Vol. 9, No. 1, pp. 36–35, January 1966. Available online at http://i5.nyu.edu/~mm64/x52.9265/january1966.html. [**62**]

63. Allen Newell and Herbert A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search," *Communications of the ACM*, Vol. 19, No. 3, pp. 113–126, March 1976. [**64**]

64. National Academy of Sciences, *Biographical Memoirs*, Vol. 71, 1997. Available online at http://www.nap.edu/catalog.php?record_id=5737. [**66**]

Part II

Early Explorations: 1950s and 1960s

If machines are to become intelligent, they must, at the very least, be able to do the thinking-related things that humans can do. The first steps then in the quest for artificial intelligence involved identifying some specific tasks thought to require intelligence and figuring out how to get machines to do them. Solving puzzles, playing games such as chess and checkers, proving theorems, answering simple questions, and classifying visual images were among some of the problems tackled by the early pioneers during the 1950s and early 1960s. Although most of these were laboratory-style, sometimes called "toy," problems, some real-world problems of commercial importance, such as automatic reading of highly stylized magnetic characters on bank checks and language translation, were also being attacked. (As far as I know, Seymour Papert was the first to use the phrase "toy problem." At a 1967 AI workshop I attended in Athens, Georgia, he distinguished among *tau* or "toy" problems, *rho* or real-world problems, and *theta* or "theory" problems in artificial intelligence. This distinction still serves us well today.)

In this part, I'll describe some of the first real efforts to build intelligent machines. Some of these were discussed or reported on at conferences and symposia – making these meetings important milestones in the birth of AI. I'll also do my best to explain the underlying workings of some of these early AI programs. The rather dramatic successes during this period helped to establish a solid base for subsequent artificial intelligence research.

Some researchers became intrigued (one might even say captured) by the methods they were using, devoting themselves more to improving the power and generality of their chosen techniques than to applying them to the tasks

thought to require them. Moreover, because some researchers were just as interested in explaining how human brains solved problems as they were in getting machines to do so, the methods being developed were often proposed as contributions to theories about human mental processes. Thus, research in cognitive psychology and research in artificial intelligence became highly intertwined.

Chapter 3

Gatherings

In September 1948, an interdisciplinary conference was held at the California Institute of Technology (Caltech) in Pasadena, California, on the topics of how the nervous system controls behavior and how the brain might be compared to a computer. It was called the Hixon Symposium on Cerebral Mechanisms in Behavior. Several luminaries attended and gave papers, among them Warren McCulloch, John von Neumann, and Karl Lashley (1890–1958), a prominent psychologist. Lashley gave what some thought was the most important talk at the symposium. He faulted behaviorism for its static view of brain function and claimed that to explain human abilities for planning and language, psychologists would have to begin considering dynamic, hierarchical structures. Lashley's talk laid out the foundations for what would become cognitive science.[1]

The emergence of artificial intelligence as a full-fledged field of research coincided with (and was launched by) three important meetings – one in 1955, one in 1956, and one in 1958. In 1955, a "Session on Learning Machines" was held in conjunction with the 1955 Western Joint Computer Conference in Los Angeles. In 1956 a "Summer Research Project on Artificial Intelligence" was convened at Dartmouth College. And in 1958 a symposium on the "Mechanization of Thought Processes," was sponsored by the National Physical Laboratory in the United Kingdom.

3.1 Session on Learning Machines

Four important papers were presented in Los Angeles in 1955. In his chairman's introduction to this session, Willis Ware wrote

> These papers do not suggest that future learning machines should be built in the pattern of the general-purpose digital computing

> device; it is rather that the digital computing system offers a convenient and highly flexible tool to probe the behavior of the models. . . . This group of papers suggests directions of improvement for future machine builders whose intent is to utilize digital computing machinery for this particular model technique. Speed of operation must be increased manyfold; simultaneous operation in many parallel modes is strongly indicated; the size of random access storage must jump several orders of magnitude; new types of input–output equipment are needed. With such advancements and the techniques discussed in these papers, there is considerable promise that systems can be built in the relatively near future which will imitate considerable portions of the activity of the brain and nervous system.

Fortunately, we have made substantial progress on the items on Ware's list of "directions for improvement." Speed of operation has increased manyfold, parallel operation is utilized in many AI systems, random access storage has jumped several orders of magnitude, and many new types of input–output equipment are available. Perhaps even further improvements will be necessary.

The session's first paper, by Wesley Clark and Belmont Farley of MIT's Lincoln Laboratory, described some pattern-recognition experiments on networks of neuron-like elements.[2] Motivated by Hebb's proposal that assemblies of neurons could learn and adapt by adjusting the strengths of their interconnections, experimenters had been trying various schemes for adjusting the strengths of connections within their networks, which were usually simulated on computers. Some just wanted to see what these networks might do whereas others, such as Clark and Farley, were interested in specific applications, such as pattern recognition. To the dismay of neurophysiologists, who complained about oversimplification, these networks came to be called *neural networks*. Clark and Farley concluded that "crude but useful generalization properties are possessed even by randomly connected nets of the type described."[3]

The next pair of papers, one by Gerald P. Dinneen (1924–) and one by Oliver Selfridge (1926–2008; Fig. 3.1), both from MIT's Lincoln Laboratory, presented a different approach to pattern recognition. Dinneen's paper[4] described computational techniques for processing images. The images were presented to the computer as a rectangular array of intensity values corresponding to the various shades of gray in the image. Dinneen pioneered the use of filtering methods to remove random bits of noise, thicken lines, and find edges. He began his paper with the following:

> Over the past months in a series of after-hour and luncheon meetings, a group of us at the laboratory have speculated on problems in this area. Our feeling, pretty much unanimously, was that there is a real *need* to get practical, to pick a real live problem

and go after it.

Selfridge's paper[5] was a companion piece to that of Dinneen. Operating on "cleaned-up" images (as might be produced by Dinneen's program, for example), Selfridge described techniques for highlighting "features" in these images and then classifying them based on the features. For example, corners of an image known to be either a square or a triangle are highlighted, and then the number of corners is counted to determine whether the image is of a square or of a triangle. Selfridge said that "eventually, we hope to be able to recognize other kinds of features, such as curvature, juxtaposition of singular points (that is, their relative bearings and distances), and so forth."

Figure 3.1: Oliver Selfridge. (Photograph courtesy of Oliver Selfridge.)

The methods pioneered by Selfridge and Dinneen are fundamental to most of the later work in enabling machines to "see." Their work is all the more remarkable when one considers that it was done on a computer, the Lincoln Laboratory "Memory Test Computer," that today would be regarded as extremely primitive. [The Memory Test Computer (MTC) was the first to use the ferrite core random-access memory modules developed by Jay Forrester. It was designed and built by Ken Olsen in 1953 at the Digital Equipment Corporation (DEC). The MTC was the first computer to simulate the operation of neural networks – those of Clark and Farley.]

The next paper[6] was about programming a computer to play chess. It was written by Allen Newell, then a researcher at the Rand Corporation in Santa Monica. Thanks to a biographical sketch of Newell written by his

colleague, Herb Simon of Carnegie Mellon University, we know something about Newell's motivation and how he came to be interested in this problem:[7]

> In September 1954 Allen attended a seminar at RAND in which Oliver Selfridge of Lincoln Laboratory described a running computer program that learned to recognize letters and other patterns. While listening to Selfridge characterizing his rather primitive but operative system, Allen experienced what he always referred to as his "conversion experience." It became instantly clear to him "that intelligent adaptive systems could be built that were far more complex than anything yet done." To the knowledge Allen already had about computers (including their symbolic capabilities), about heuristics, about information processing in organizations, about cybernetics, and proposals for chess programs was now added a concrete demonstration of the feasibility of computer simulation of complex processes. Right then he committed himself to understanding human learning and thinking by simulating it.

Simon goes on to summarize Newell's paper on chess:

> [It] outlined an imaginative design for a computer program to play chess in humanoid fashion, incorporating notions of goals, aspiration levels for terminating search, satisfying with "good enough" moves, multidimensional evaluation functions, the generation of subgoals to implement goals, and something like best first search. Information about the board was to be expressed symbolically in a language resembling the predicate calculus. The design was never implemented, but ideas were later borrowed from it for use in the NSS [Newell, Shaw, and Simon] chess program in 1958.[8]

Newell hinted that his aims extended beyond chess. In his paper he wrote "The aim of this effort, then, is to program a current computer to learn to play good chess. This is the means to understanding more about the kinds of computers, mechanisms, and programs that are necessary to handle ultracomplicated problems." Newell's proposed techniques can be regarded as his first attempt to produce evidence for what he and Simon later called the Physical Symbol System Hypothesis.

Walter Pitts, a commentator for this session, concluded it by saying, "But, whereas Messrs. Farley, Clark, Selfridge, and Dinneen are imitating the nervous system, Mr. Newell prefers to imitate the hierarchy of final causes traditionally called the mind. It will come to the same thing in the end, no doubt. . . ." To "come to the same thing," these two approaches, neural modeling and symbol processing, must be recognized simply as different levels

of description of what goes on in the brain. Different levels are appropriate for describing different kinds of mental phenomena. I'll have more to say about description levels later in the book.

3.2 The Dartmouth Summer Project

In 1954, John McCarthy (1927– ; Fig 3.2) joined Dartmouth College in Hanover, New Hampshire, as an Assistant Professor of Mathematics. McCarthy had been developing a continuing interest in what would come to be called artificial intelligence. It was "triggered," he says, "by attending the September 1948 Hixon Symposium on Cerebral Mechanisms in Behavior held at Caltech where I was starting graduate work in mathematics."[9] While at Dartmouth he was invited by Nathaniel Rochester (1919–2001) to spend the summer of 1955 in Rochester's Information Research Department at IBM in Poughkeepsie, New York. Rochester had been the designer of the IBM 701 computer and had also participated in research on neural networks.[10]

At IBM that summer, McCarthy and Rochester persuaded Claude Shannon and Marvin Minsky (1927– ; Fig. 3.2), then a Harvard junior fellow in mathematics and neurology, to join them in proposing a workshop to be held at Dartmouth during the following summer. Shannon, whom I have previously mentioned, was a mathematician at Bell Telephone Laboratories and already famous for his work on switching theory and statistical information theory. McCarthy took the lead in writing the proposal and in organizing what was to be called a "Summer Research Project on Artificial Intelligence." The proposal was submitted to the Rockefeller Foundation in August 1955.

Extracts from the proposal read as follows:[11]

> We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.
>
> . . .
>
> For the present purpose the artificial intelligence problem is taken to be that of making a machine behave in ways that would be called intelligent if a human were so behaving.

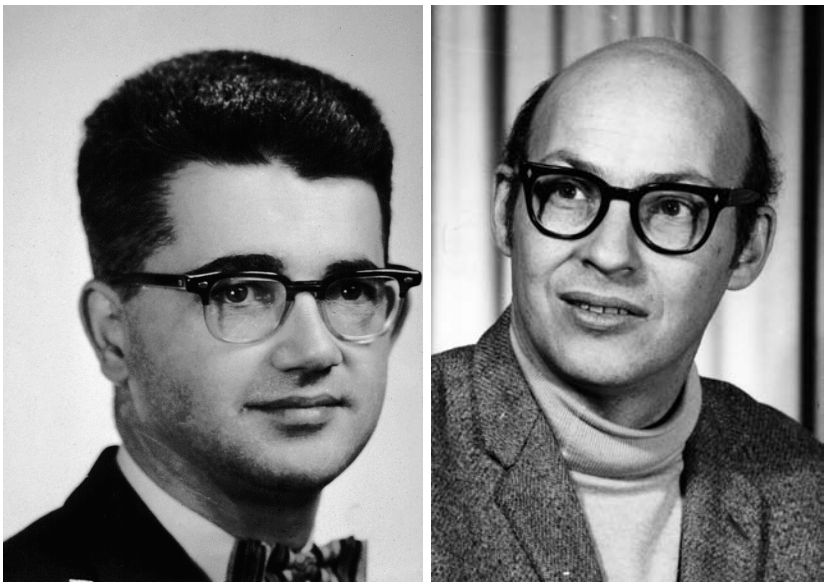


Figure 3.2: John McCarthy (left) and Marvin Minsky (right). (McCarthy photograph courtesy of John McCarthy. Minsky photograph courtesy MIT Museum.)

The Rockefeller Foundation did provide funding for the event, which took place during six weeks of the summer of 1956. It turned out, however, to be more of a rolling six-week workshop than a summer "study." Among the people attending the workshop that summer, in addition to McCarthy, Minsky, Rochester, and Shannon were Arthur Samuel (1901–1990), an engineer at the IBM corporation who had already written a program to play checkers, Oliver Selfridge, Ray Solomonoff of MIT, who was interested in automating induction, Allen Newell, and Herbert Simon. Newell and Simon (together with another Rand scientist, Cliff Shaw) had produced a program for proving theorems in symbolic logic. Another attending IBM scientist was Alex Bernstein, who was working on a chess-playing program.

McCarthy has given a couple of reasons for using the term "artificial intelligence." The first was to distinguish the subject matter proposed for the Dartmouth workshop from that of a prior volume of solicited papers, titled *Automata Studies*, co-edited by McCarthy and Shannon, which (to McCarthy's disappointment) largely concerned the esoteric and rather narrow mathematical subject called "automata theory." The second, according to McCarthy, was "to escape association with 'cybernetics.' Its concentration on analog feedback seemed misguided, and I wished to avoid having either to accept Norbert Wiener as a guru or having to argue with him."[12]

There was (and still is) controversy surrounding the name. According to Pamela McCorduck's excellent history of the early days of artificial intelligence, Art Samuel remarked, "The word artificial makes you think there's something kind of phony about this, or else it sounds like it's all artificial and there's nothing real about this work at all."[13] McCorduck goes on to say that "[n]either Newell or Simon liked the phrase, and called their own work complex information processing for years thereafter." But most of the people who signed on to do work in this new field (including myself) used the name "artificial intelligence," and that is what the field is called today. (Later, Newell became reconciled to the name. In commenting about the content of the field, he concluded, "So cherish the name *artificial intelligence*. It is a good name. Like all names of scientific fields, it will grow to become exactly what its field comes to mean.")[14]

The approaches and motivations of the people at the workshop differed. Rochester came to the conference with a background in networks of neuron-like elements. Newell and Simon had been pursuing (indeed had helped originate) the symbol-processing approach. Among the topics Shannon wanted to think about (according to the proposal) was the "application of information theory concepts to computing machines and brain models." (After the workshop, however, Shannon turned his attention away from artificial intelligence.)

McCarthy wrote that he was interested in constructing "an artificial language which a computer can be programmed to use on problems requiring conjecture and self-reference. It should correspond to English in the sense that short English statements about the given subject matter should have short correspondents in the language and so should short arguments or conjectural arguments. I hope to try to formulate a language having these properties . . . " Although McCarthy later said that his ideas on this topic were still too "ill formed" for presentation at the conference, it was not long before he made specific proposals for using a logical language and its inference mechanisms for representing and reasoning about knowledge.

Although Minsky's Ph.D. dissertation[15] and some of his subsequent work concentrated on neural nets, around the time of the Dartmouth workshop he was beginning to change direction. Now, he wrote, he wanted to consider a machine that "would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem, it could first explore solutions within the internal abstract model of the environment and then attempt external experiments." At the workshop, Minsky continued work on a draft that was later to be published as a foundational paper, "Steps Toward Artificial Intelligence."[16]

One of the most important technical contributions of the 1956 meeting was work presented by Newell and Simon on their program, the "Logic Theorist (LT)," for proving theorems in symbolic logic. LT was concrete evidence that processing "symbol structures" and the use of what Newell and Simon called "heuristics" were fundamental to intelligent problem solving. I'll

describe some of these ideas in more detail in a subsequent chapter.

Newell and Simon had been working on ideas for LT for some months and became convinced in late 1955 that they could be embodied in a working program. According to Edward Feigenbaum (1936–), who was taking a course from Herb Simon at Carnegie in early 1956, "It was just after Christmas vacation – January 1956 – when Herb Simon came into the classroom and said, 'Over Christmas Allen Newell and I invented a thinking machine.'"[17] What was soon to be programmed as LT was the "thinking machine" Simon was talking about. He called it such, no doubt, because he thought it used some of the same methods for solving problems that humans use. Simon later wrote[18] "On Thursday, Dec. 15. . . I succeeded in simulating by hand the first proof. . . I have always celebrated Dec. 15, 1955, as the birthday of heuristic problem solving by computer." According to Simon's autobiography *Models of My Life*,[19] LT began by hand simulation, using his children as the computing elements, while writing on and holding up note cards as the registers that contained the state variables of the program.[20]

Another topic discussed at Dartmouth was the problem of proving theorems in geometry. (Perhaps some readers will recall their struggles with geometry proofs in high school.) Minsky had already been thinking about a program to prove geometry theorems. McCorduck quotes him as saying the following:[21]

> [P]robably the important event in my own development – and the explanation of my perhaps surprisingly casual acceptance of the Newell–Shaw–Simon work – was that I had sketched out the heuristic search procedure for [a] geometry machine and then been able to hand-simulate it on paper in the course of an hour or so. Under my hand the new proof of the isosceles-triangle theorem came to life, a proof that was new and elegant to the participants – later, we found that proof was well-known. . .

In July 2006, another conference was held at Dartmouth celebrating the fiftieth anniversary of the original conference. (See Fig. 3.3.) Several of the founders and other prominent AI researchers attended and surveyed what had been achieved since 1956. McCarthy reminisced that the "main reason the 1956 Dartmouth workshop did not live up to my expectations is that AI is harder than we thought." In any case, the 1956 workshop is considered to be the official beginning of serious work in artificial intelligence, and Minsky, McCarthy, Newell, and Simon came to be regarded as the "fathers" of AI. A plaque was dedicated and installed at the Baker Library at Dartmouth commemorating the beginning of artificial intelligence as a scientific discipline.

Figure 3.3: Some of AI's founders at the July 2006 Dartmouth fiftieth anniversary meeting. From the left are Trenchard More, John McCarthy, Marvin Minsky, Oliver Selfridge, and Ray Solomonoff. (Photograph courtesy of photographer Joe Mehling and the Dartmouth College Artificial Intelligence Conference: The Next Fifty Years.)

3.3 Mechanization of Thought Processes

In November 1958, a symposium on the "Mechanisation of Thought Processes" was held at the National Physical Laboratory in Teddington, Middlesex, England. According to the preface of the conference proceedings, the symposium was held "to bring together scientists studying artificial thinking, character and pattern recognition, learning, mechanical language translation, biology, automatic programming, industrial planning and clerical mechanization."

Among the people who presented papers at this symposium were many whom I have already mentioned in this story. They include Minsky (by then a staff member at Lincoln Laboratory and on his way to becoming an assistant professor of Mathematics at MIT), McCarthy (by then an assistant professor of Communication Sciences at MIT), Ashby, Selfridge, and McCulloch. (John

Backus, one of the developers of the computer programming language FORTRAN, and Grace Murray Hopper, a pioneer in "automatic programming," also gave papers.)

The proceedings of this conference[22] contains some papers that became quite influential in the history of artificial intelligence. Among these, I'll mention ones by Minsky, McCarthy, and Selfridge.

Minsky's paper, "Some Methods of Artificial Intelligence and Heuristic Programming," was the latest version of a piece he had been working on since just before the Dartmouth workshop. The paper described various methods that were (and could be) used in heuristic programming. It also covered methods for pattern recognition, learning, and planning. The final version, which was soon to be published as "Steps Toward Artificial Intelligence," was to become required reading for new recruits to the field (including me).

I have already mentioned McCarthy's hope to develop an artificial language for AI. He summarized his conference paper, "Programs with Common Sense," as follows:

> This paper will discuss programs to manipulate in a suitable formal language (most likely a part of the predicate calculus) common instrumental statements. The basic program will draw immediate conclusions from a list of premises. These conclusions will be either declarative or imperative sentences. When an imperative sentence is deduced, the program takes a corresponding action.

In his paper, McCarthy suggested that facts needed by an AI program, which he called the "advice taker," might be represented as expressions in a mathematical (and computer-friendly) language called "first-order logic." For example, the facts "I am at my desk" and "My desk is at home" would be represented as the expressions `at(I, desk)` and `at(desk, home)`. These, together with similarly represented information about how to achieve a change in location (by walking and driving for example), could then be used by the proposed (but not yet programmed) advice taker to figure out how to achieve some goal, such as being at the airport. The advice taker's reasoning process would produce imperative logical expressions involving walking to the car and driving to the airport.

Representing facts in a logical language has several advantages. As McCarthy later put it,[23]

> Expressing information in declarative sentences is far more modular than expressing it in segments of computer program or in tables. Sentences can be true in much wider contexts than specific programs can be useful. The supplier of a fact does not have to understand much about how the receiver functions, or how or whether the receiver will use it. The same fact can be used for

> many purposes, because the logical consequences of collections of facts can be available.

McCarthy later expanded on these ideas in a companion memorandum.[24] As I'll mention later, some of McCarthy's advice-taker proposals were finally implemented by a Stanford graduate student, C. Cordell Green.

I have already mentioned the 1955 pattern-recognition work of Oliver Selfridge. At the 1958 Teddington Symposium, Selfridge presented a paper on a new model for pattern recognition (and possibly for other cognitive tasks also).[25] He called it "Pandemonium," meaning the place of all the demons. His model is especially interesting because its components, which Selfridge called "demons," can either be instantiated as performing lower level nerve-cell-type functions or higher level cognitive functions (of the symbol-processing variety). Thus, Pandemonium can take the form of a neural network, a hierarchically organized set of symbol processors – all working in parallel, or some combination of these forms. If the latter, the model is a provocative proposal for joining these two disparate approaches to AI.

In the introduction to his paper, Selfridge emphasized the importance of computations performed in parallel:

> The basic motif behind our model is the notion of parallel processing. This is suggested on two grounds: first, it is often easier to handle data in a parallel manner, and, indeed, it is usually the more "natural" manner to handle it in; and, secondly, it is easier to modify an assembly of quasi-independent modules than a machine all of whose parts interact immediately and in a complex way.

Selfridge made several suggestions about how Pandemonium could learn. It's worth describing some of these because they foreshadow later work in machine learning. But first I must say a bit more about the structure of Pandemonium.

Pandemonium's structure is something like that of a business organization chart. At the bottom level are workers, whom Selfridge called the "data demons." These are computational processes that "look at" the input data, say an image of a printed letter or number. Each demon looks for something specific in the image, perhaps a horizontal bar; another might look for a vertical bar; another for an arc of a circle; and so on. Each demon "shouts" its findings to a set of demons higher in the organization. (Think of these higher level demons as middle-level managers.) The loudness of a demon's shout depends on how certain it is that it is seeing what it is looking for. Of course, Selfridge is speaking metaphorically when he uses terms such as "looking for" and "shouting." Suffice it to say that it is not too difficult to program computers to "look for" certain features in an image. (Selfridge had already shown how that could be done in his 1955 paper that I mentioned earlier.) And a "shout" is really the strength of the output of a computational process.

Each of the next level of demons specializes in listening for a particular combination of shouts from the data demons. For example, one of the demons at this level might be tuned to listen for shouts from data demon 3, data demon 11, and data demon 22. If it finds that these particular demons are shouting loudly, it responds with a shout of its own to the demons one level up in the hierarchy, and so on.

Just below the top level of the organization are what Selfridge called the "cognitive demons." As at the other levels, these listen for particular combinations of shouts from the demons at the level below, and they respond with shouts of their own to a final "decision demon" at the top – the overall boss. Depending on what it hears from its "staff," the decision demon finally announces what it thinks is the identity of the image – perhaps the letter "A" or the letter "R" or whatever.

Actual demon design depends on what task Pandemonium is supposed to be doing. But even without specifying what each demon was to do, Selfridge made very interesting proposals about how Pandemonium could learn to perform better at whatever it was supposed to be doing. One of his proposals involved equipping each demon with what amounted to a "megaphone" through which it delivered its shout. The volume level of the megaphone could be adjusted. (Selfridge's Pandemonium is just a bit more complicated than the version I am describing. His version has each demon using different channels for communicating with each of the different demons above it. The volume of the shout going up each channel is individually adjusted by the learning mechanism.) The demons were not allowed to set their own volume levels, however. All volume levels were to be set through an outside learning process attempting to improve the performance of the whole assembly. Imagine that the volume levels are initially set either at random or at whatever a designer thinks would be appropriate. The device is then tested on some sample of input data and its performance score is noted. Say, it gets a score of 81%. Then, small adjustments are made to the volume levels in all possible ways until a set of adjustments is found that improves the score the most, say to 83%. This particular set of small adjustments is then made and the process is repeated over and over (possibly on additional data) until no further improvement can be made.

(Because there might be a lot of megaphones in the organization, it might seem impractical to make adjustments in all possible ways and to test each of these ways to find its score. The process might indeed take some time, but computers are fast – even more so today. Later in the book, I'll show how one can calculate, rather than find by experiment, the best adjustments to make in neural networks organized like Pandemonium.)

If we think of the score as the height of some landscape and the adjustments as movements over the landscape, the process can be likened to climbing a hill by always taking steps in the direction of steepest ascent. Gradient ascent (or hill-climbing methods, as they are sometimes called) are

well known in mathematics. Selfridge had this to say about some of the pitfalls of their use:

> This may be described as one of the problems of training, namely, to encourage the machine or organism to get enough on the foot-hills so that small changes... will produce noticeable improvement in his altitude or score. One can describe learning situations where most of the difficulty of the task lies in finding any way of improving one's score, such as learning to ride a unicycle, where it takes longer to stay on for a second than it does to improve that one second to a minute; and others where it is easy to do a little well and very hard to do very well, such as learning to play chess. It's also true that often the main peak is a plateau rather than an isolated spike.

Selfridge described another method for learning in Pandemonium. This method might be likened to replacing managers in an organization who do not perform well. As Selfridge puts it,

> At the conception of our demoniac assembly we collected somewhat arbitrarily a large number of subdemons which we guessed would be useful... but we have no assurance at all that the particular subdemons we selected are good ones. Subdemon selection generates new subdemons for trial and eliminates inefficient ones, that is, ones that do not much help improve the score.

The demon selection process begins after the volume-adjusting learning mechanism has run for a while with no further improvements in the score. Then the "worth" of each demon is evaluated by using, as Selfridge suggests, a method based on the learned volume levels of their shouting. Demons having high volume levels have a large effect on the final score, and so they can be thought to have high worth. First, the demons with low volume levels are eliminated entirely. (That step can't hurt the score very much.) Next, some of the demons undergo random "mutations" and are put back in service. Next, some pairs of worthy demons are selected and, as Selfridge says, "conjugated" into offspring demons. The precise method Selfridge proposed for conjugation need not concern us here, but the spirit of the process is to produce offspring that share, one hopes, useful properties of the parents. The offspring are then put into service. Now the whole process of adjusting volume levels of the surviving and "evolved" demons can begin again to see whether the score of the new assembly can be further improved.

Notes

1. The proceedings of the symposium were published in L. A. Jeffries (ed.), *Cerebral Mechanisms in Behavior: The Hixon Symposium*, New York: Wiley, 1951. An excellent review of Lashley's points are contained in Chapter 2 of *The Mind's New Science: A History of the Cognitive Revolution*, by Howard E Gardner, New York: Basic Books, 1985. [**73**]

2. W. A. Clark and B. G. Farley, "Generalization of Pattern Recognition in a Self-Organizing System," *Proceedings of the 1955 Western Joint Computer Conference*, Institute of Radio Engineers, New York, pp. 86–91, 1955. Clark and Farley's experiments continued some work they had reported on earlier in B. G. Farley and W. A. Clark, "Simulation of Self-Organizing Systems by Digital Computer, *IRE Transactions on Information Theory*, Vol. 4, pp. 76–84, 1954. (In 1962 Clark built the first personal computer, the LINC.) [**74**]

3. Alan Wilkes and Nicholas Wade credit Scottish psychologist Alexander Bain (1818–1903) with the invention of the first neural network, which Bain described in his 1873 book *Mind and Body: The Theories of Their Relation."* (See Alan L. Wilkes and Nicholas J. Wade, "Bain on Neural Networks," *Brain and Cognition*, Vol. 33, pp. 295–305, 1997.) [**74**]

4. Gerald P. Dinneen, "Programming Pattern Recognition," *Proceedings of the 1955 Western Joint Computer Conference*, Institute of Radio Engineers, New York, pp. 94–100, 1955. [**74**]

5. Oliver Selfridge, "Pattern Recognition and Modern Computers," *Proceedings of the 1955 Western Joint Computer Conference*, Institute of Radio Engineers, New York, pp. 91–93, 1955. [**75**]

6. Allen Newell, "The Chess Machine: An Example of Dealing with a Complex Task by Adaptation," *Proceedings of the 1955 Western Joint Computer Conference*, Institute of Radio Engineers, New York, pp. 101–108, 1955. (Also issued as RAND Technical Report P-620.) [**75**]

7. National Academy of Sciences, *Biographical Memoirs*, Vol. 71, 1997. Available online at http://www.nap.edu/catalog.php?record_id=5737. [**76**]

8. Allen Newell, J. C. Shaw, and Herbert A. Simon, "Chess-Playing Programs and the Problem of Complexity," *IBM Journal of Research and Development*, Vol. 2, pp. 320–335, 1958. The paper is available online at http://domino.watson.ibm.com/tchjr/journalindex.nsf/0/237cfeded3be103585256bfa00683d4d?OpenDocument. [**76**]

9. From John McCarthy's informal comments at the 2006 Dartmouth celebration. [**77**]

10. Nathan Rochester *et al.*, "Tests on a Cell Assembly Theory of the Action of the Brain Using a Large Digital Computer," *IRE Transaction of Information Theory*, Vol. IT-2, pp. 80-93, 1956. [**77**]

11. From http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html. Portions of the proposal have been reprinted in John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," *AI Magazine*, Vol. 27, No. 4, p. 12, Winter 2006. [**77**]

12. From http://www-formal.stanford.edu/jmc/reviews/bloomfield/bloomfield.html. [**78**]

13. Pamela McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, p. 97, San Francisco: W. H. Freeman and Co., 1979. [**79**]

14. See Allen Newell, "The First AAAI President's Message," *AI Magazine*, Vol. 26, No. 4, pp. 24–29, Winter 2005. [**79**]

15. M. L. Minsky, *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem*, Ph.D. thesis, Princeton University, 1954. [**79**]

16. Marvin L. Minsky, "Steps Toward Artificial Intelligence," *Proceedings of the IRE*, Vol. 49, No. 1, pp. 8–30, January 1961. Also appears in Edward A. Feigenbaum, and Julian

Feldman (eds.), *Computers and Thought*, New York: McGraw Hill, 1963. (Available online at http://web.media.mit.edu/~minsky/papers/steps.html.) [**79**]

17. Pamela McCorduck, *op. cit.*, p. 116. [**80**]

18. Herbert A. Simon, *Models of My Life*, Cambridge, MA: MIT Press, 1996. The quote is from http://www.post-gazette.com/pg/06002/631149.stm. [**80**]

19. *Ibid.* [**80**]

20. http://www.post-gazette.com/downloads/20060102simon_notes.pdf contains sketches of Simon's simulation of an LT proof. [**80**]

21. Pamela McCorduck, *op. cit.*, p. 106. [**80**]

22. D. V. Blake and A. M. Uttley (eds.), *Proceedings of the Symposium on Mechanisation of Thought Processes*, Vols. 1 and 2, London: Her Majesty's Stationary Office, 1959. [**82**]

23. John McCarthy, "Artificial Intelligence, Logic and Formalizing Common Sense," in *Philosophical Logic and Artificial Intelligence*, Richmond Thomason (ed.), Dordrecht: Kluwer Academic, 1989. [**82**]

24. J. McCarthy, "Situations, Actions and Causal Laws, Stanford Artificial Intelligence Project," Memo 2, 1963. The two pieces are reprinted together in M. Minsky (ed.), *Semantic Information Processing*, pp. 410–417, Cambridge, MA: MIT Press, 1968. Related topics are explored in J. McCarthy and Patrick Hayes, "Some Philosophical Ideas From the Standpoint of Artificial Intelligence," MI-4, 1969. [**83**]

25. Oliver G. Selfridge, "Pandemonium: A Paradigm for Learning," in D. V. Blake and A. M. Uttley (eds.), *Proceedings of the Symposium on Mechanisation of Thought Processes*, pp, 511–529, London: Her Majesty's Stationary Office, 1959. [**83**]

4.1

Chapter 4

Pattern Recognition

Most of the attendees of the Dartmouth summer project were interested in mimicking the higher levels of human thought. Their work benefitted from a certain amount of introspection about how humans solve problems. Yet, many of our mental abilities are beyond our power of introspection. We don't know how we recognize speech sounds, read cursive script, distinguish a cup from a plate, or identify faces. We just do these things automatically without thinking about them. Lacking clues from introspection, early researchers interested in automating some of our perceptual abilities based their work instead on intuitive ideas about how to proceed, on networks of simple models of neurons, and on statistical techniques. Later, workers gained additional insights from neurophysiological studies of animal vision.

In this chapter, I'll describe work during the 1950s and 1960s on what is called "pattern recognition." This phrase refers to the process of analyzing an input image, a segment of speech, an electronic signal, or any other sample of data and classifying it into one of several categories. For character recognition, for example, the categories would correspond to the several dozen or so alphanumeric characters.

Most of the pattern-recognition work in this period dealt with two-dimensional material such as printed pages or photographs. It was already possible to scan images to convert them into arrays of numbers (later called "pixels"), which could then be processed by computer programs such as those of Dinneen and Selfridge. Russell Kirsch and colleagues at the National Bureau of Standards (now the National Institute for Standards and Technology) were also among the early pioneers in image processing. In 1957 Kirsch built and used a drum scanner to scan a photograph of his three-month-old son, Walden. Said to be the first scanned photograph, it measured 176 pixels on a side and is depicted in Fig. 4.1.[1] Using his scanner, he and colleagues experimented with picture-procesing programs running on their SEAC (Standards Eastern Automatic Computer) computer.[2]

Figure 4.1: An early scanned photograph. (Photograph used with permission of NIST.)

4.1 Character Recognition

Early efforts at the perception of visual images concentrated on recognizing alphanumeric characters on documents. This field came to be known as "optical character recognition." A symposium devoted to reporting on progress on this topic was held in Washington, DC, in January 1962.[3] In summary, devices existed at that time for reasonably accurate recognition of fixed-font (typewritten or printed) characters on paper. Perhaps the state of things then was best expressed by one of the participants of the symposium, J. Rabinow of Rabinow Engineering, who said "We think, in our company, that we can read anything that is printed, and we can even read some things that are written. The only catch is, 'how many bucks do you have to spend?'"[4]

A notable success during the 1950s was the magnetic ink character recognition (MICR) system developed by researchers at SRI International (then called the Stanford Research Institute) for reading stylized magnetic ink characters at the bottom of checks. (See Fig. 4.2.) MICR was part of SRI's ERMA (Electronic Recording Method of Accounting) system for automating check processing and checking account management and posting.

According to an SRI Web site, "In April 1956, the Bank of America announced that General Electric Corporation had been selected to manufacture production models. . . . In 1959, General Electric delivered the first 32 ERMA computing systems to the Bank of America. ERMA served as the Bank's accounting computer and check handling system until 1970."[5]

Figure 4.2: The MICR font set.

Most of the recognition methods at that time depended on matching a character (after it was isolated on the page and converted to an array of 0's and 1's) against prototypical versions of the character called "templates" (also stored as arrays in the computer). If a character matched the template for an "A," say, sufficiently better than it matched any other templates, the input was declared to be an "A." Recognition accuracy degraded if the input characters were not presented in standard orientation, were not of the same font as the template, or had imperfections.

The 1955 papers by Selfridge and Dinneen (which I have already mentioned on p. 74) proposed some ideas for moving beyond template matching. A 1960 paper by Oliver Selfridge and Ulrich Neisser carried this work further.[6] That paper is important because it was a successful, early attempt to use image processing, feature extraction, and learned probability values in hand-printed character recognition. The characters were scanned and represented on a 32×32 "retina" or array of 0's and 1's. They were then processed by various refining operations (similar to those I mentioned in connection with the 1955 Dinneen paper) for removing random bits of noise, filling gaps, thickening lines, and enhancing edges. The "cleaned-up" images were then inspected for the occurrence of "features" (similar to the features I mentioned in connection with the 1955 Selfridge paper.) In all, 28 features were used – features such as the maximum number of times a horizontal line intersected the image, the relative lengths of different edges, and whether or not the image had a "concavity facing south."

Recalling Selfridge's Pandemonium system, we can think of the feature-detection process as being performed by "demons." At one level higher in the hierarchy than the feature demons were the "recognition demons" – one for each letter. (The version of this system tested by Worthie Doyle of Lincoln Laboratory was designed to recognize ten different hand-printed characters, namely, A, E, I, L, M, N, O, R, S, and T.) Each recognition demon received inputs from each of the feature-detecting demons. But first, the inputs to each recognition demon were multiplied by a weight that took into account the importance of the contribution of the corresponding feature to the decision. For example, if feature 17 were more important than feature 22 in deciding that the input character was an "A," then the input to the "A" recognizer

from feature 17 would be weighted more heavily than would be the input from feature 22. After each recognition demon added up the total of its weighted inputs, a final "decision demon" decided in favor of that character having the largest sum.

The values of the weights were determined by a learning process during which 330 "training" images were analyzed. Counts were tabulated for how many times each feature was detected for each different letter in the training set. These statistical data were used to make estimates of the probabilities that a given feature would be detected for each of the letters. These probability estimates were then used to weight the features summed by the recognizing demons.

After training, the system was tested on samples of hand-printed characters that it had not yet seen. According to Selfridge and Neisser, "This program makes only about 10 percent fewer correct identifications than human readers make – a respectable performance, to be sure."

4.2 Neural Networks

4.2.1 Perceptrons

In 1957, Frank Rosenblatt (1928–1969; Fig. 4.3), a psychologist at the Cornell Aeronautical Laboratory in Buffalo, New York, began work on neural networks under a project called PARA (Perceiving and Recognizing Automaton). He was motivated by the earlier work of McCulloch and Pitts and of Hebb and was interested in these networks, which he called *perceptrons*, as potential models of human learning, cognition, and memory.[7]

Continuing during the early 1960s as a professor at Cornell University in Ithaca, New York, he experimented with a number of different kinds of perceptrons. His work, more than that of Clark and Farley and of the other neural network pioneers, was responsible for initiating one of the principal alternatives to symbol-processing methods in AI, namely, neural networks.

Rosenblatt's perceptrons consisted of McCulloch–Pitts-style neural elements, like the one shown in Fig. 4.4. Each element had inputs (coming in from the left in the figure), "weights" (shown by bulges on the input lines), and one output (going out to the right). The inputs had values of either 1 or 0, and each input was multiplied by its associated weight value. The neural element computed the sum of these weighted values. So, for example, if all of the inputs to the neural element in Fig. 4.4 were equal to 1, the sum would be 13. If the sum were greater than (or just equal to) a "threshold value," say 7, associated with the element, then the output of the neural element would be 1, which it would be in this example. Otherwise the output would be 0.

A perceptron consists of a network of these neural elements, in which the

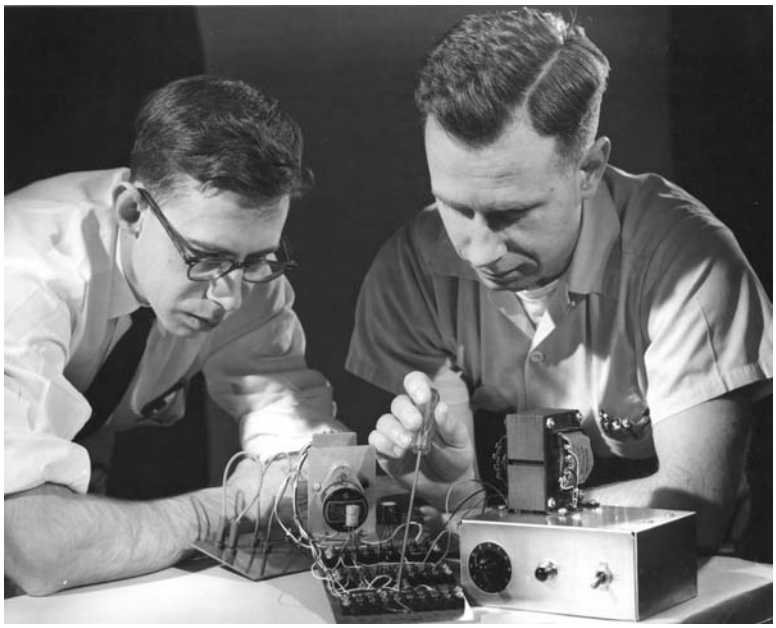


Figure 4.3: Frank Rosenblatt (left) working (with Charles Wrightman) on a prototype A-unit. (Courtesy of the Division of Rare and Manuscript Collections, Cornell University Library.)

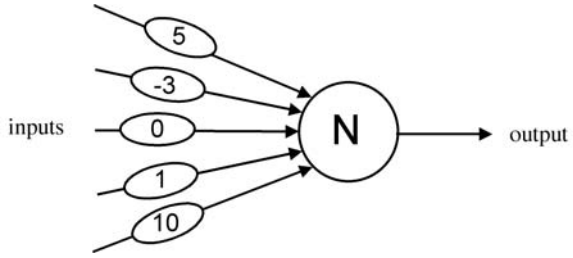


Figure 4.4: Rosenblatt's neural element with weights.

outputs of one element are inputs to others. (There is an analogy here with Selfridge's Pandemonium in which mid-level demons receive "shouts" from lower level demons. The weights on a neural element's input lines can be thought of as analogous to the strength-enhancing or strength-diminishing "volume controls" in Pandemonium.) A sample perceptron is illustrated in Fig. 4.5. [Rosenblatt drew his perceptron diagrams in a horizontal format (the electrical engineering style), with inputs to the left and output to the right. Here I use the vertical style generally preferred by computer scientists for

hierarchies, with the lowest level at the bottom and the highest at the top. To simplify the diagram, weight bulges are not shown.] Although the perceptron illustrated, with only one output unit, is capable of only two different outputs (1 or 0), multiple outputs (sets of 1's and 0's) could be achieved by arranging for several output units.

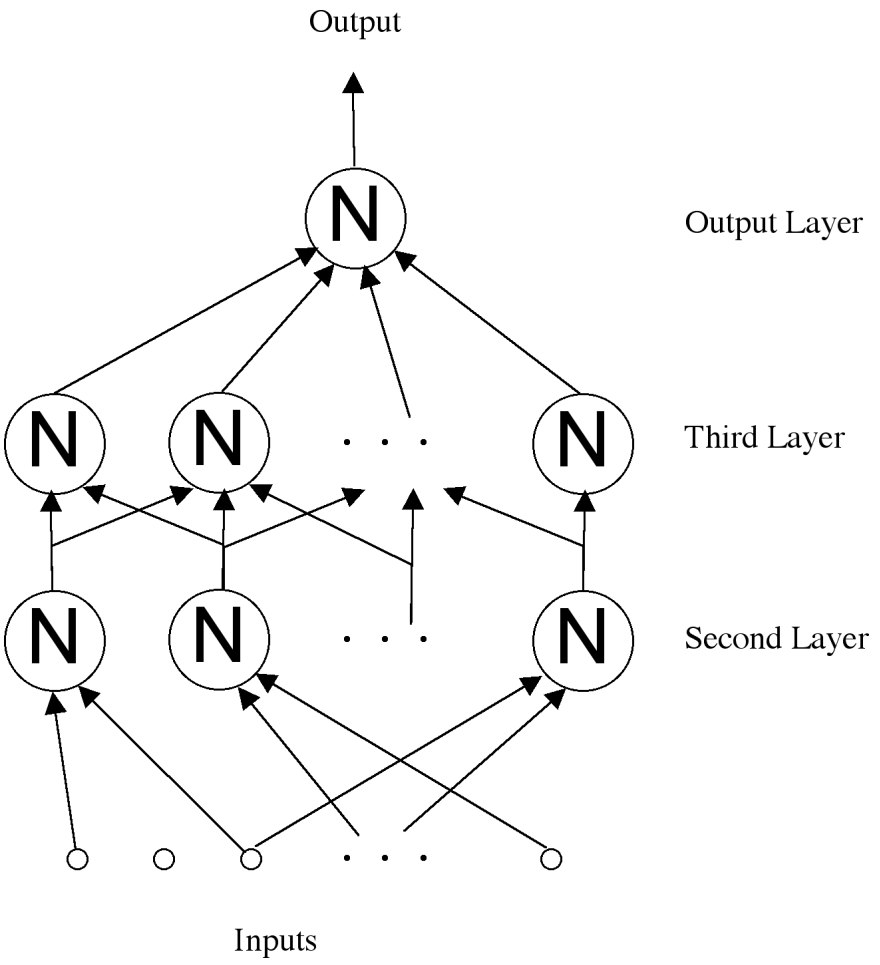


Figure 4.5: A perceptron.

The input layer, shown at the bottom of Fig. 4.5, was typically a rectangular array of 1's and 0's corresponding to cells called "pixels" of a black-and-white image. One of the applications Rosenblatt was interested in was, like Selfridge, character recognition.

I'll use some simple algebra and geometry to show how the neural elements in perceptron networks can be "trained" to produce desired outputs.

Let's consider, for example, a single neural element whose inputs are the values x_1, x_2, and x_3 and whose associated weight values are w_1, w_2, and w_3. When the sum computed by this element is exactly equal to its threshold value, say t, we have the equation

$$w_1x_1 + w_2x_2 + w_3x_3 = t.$$

In algebra, such an equation is called a "linear equation." It defines a linear boundary, that is, a plane, in a three-dimensional space. The plane separates those input values that would cause the neural element to have an output of 1 from those that would cause it to have an output of 0. I show a typical planar boundary in Fig. 4.6.

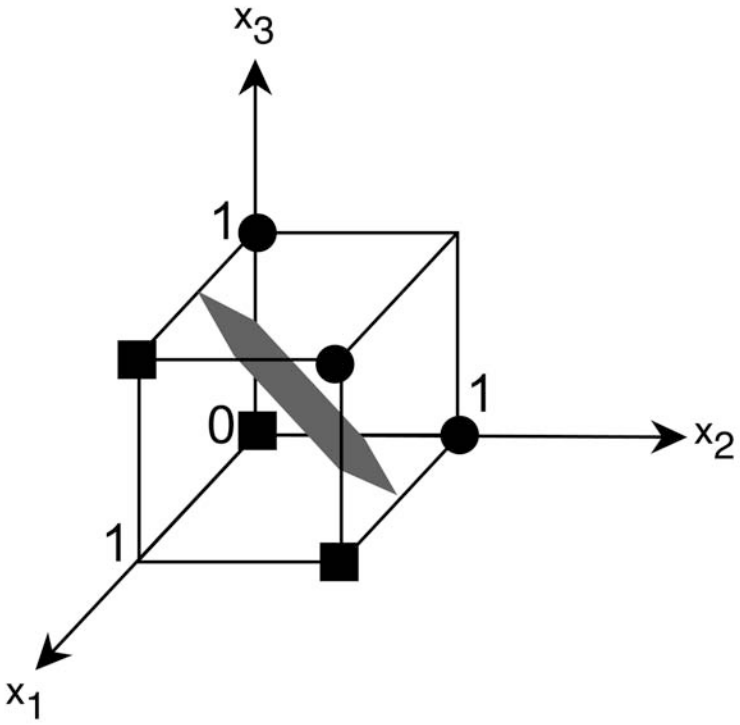


Figure 4.6: A separating plane in a three-dimensional space.

An input to the neural element can be depicted as a point (that is, a vector) in this three-dimensional space. Its coordinates are the values of x_1, x_2, and x_3, each of which can be either 1 or 0. The figure shows six such points, three of them (the small circles, say) causing the element to have an output of 1 and three (the small squares, say) causing it to have an output of 0. Changing the value of the threshold causes the plane to move sideways in a

direction parallel to itself. Changing the values of the weights causes the plane to rotate. Thus, by changing the weight values, points that used to be on one side of the plane might end up on the other side. "Training" takes place by performing such changes. I'll have more to say about training procedures presently.

In dimensions higher than three (which is usually the case), a linear boundary is called a "hyperplane." Although it is not possible to visualize what is going on in spaces of high dimensions, mathematicians still speak of input points in these spaces and rotations and movements of hyperplanes in response to changes in the values of weights and thresholds.

Rosenblatt defined several types of perceptrons. He called the one shown in the diagram a "series-coupled, four-layer perceptron." (Rosenblatt counted the inputs as the first layer.) It was termed "series-coupled" because the output of each neural element fed forward to neural elements in a subsequent layer. In more recent terminology, the phrase "feed-forward" is used instead of "series-coupled." In contrast, a "cross-coupled" perceptron could have the outputs of neural elements in one layer be inputs to neural elements in the same layer. A "back-coupled" perceptron could have the outputs of neural elements in one layer be inputs to neural elements in lower numbered layers.

Rosenblatt thought of his perceptrons as being models of the wiring of parts of the brain. For this reason, he called the neural elements in all layers but the output layer "association units" ("A-units") because he intended them to model associations performed by networks of neurons in the brain.

Of particular interest in Rosenblatt's research was what he called an "alpha-perceptron." It consisted of a three-layer, feed-forward network with an input layer, an association layer, and one or more output units. In most of his experiments, the inputs had values of 0 or 1, corresponding to black or white pixels in a visual image presented on what he called a "retina." Each A-unit received inputs (which were not multiplied by weight values) from some randomly selected subset of the pixels and sent its output, through sets of adjustable weights, to the final output units, whose binary values could be interpreted as a code for the category of the input image.

Various "training procedures" were tried for adjusting the weights of the output units of an alpha-perceptron. In the most successful of these (for pattern-recognition purposes), the weights leading in to the output units were adjusted only when those units made an error in classifying an input. The adjustments were such as to force the output to make the correct classification for that particular input. This technique, which soon became a standard, was called the "error-correction procedure." Rosenblatt used it successfully in a number of experiments for training perceptrons to classify visual inputs, such as alphanumeric characters, or acoustic inputs, such as speech sounds. Professor H. David Block, a Cornell mathematician working with Rosenblatt, was able to prove that the error-correction procedure was guaranteed to find a

hyperplane that perfectly separated a set of training inputs when such a hyperplane existed.[8] (Other mathematicians, such as Albert B. Novikoff at SRI, later developed more elegant proofs.[9] I give a version of this proof in my book *Learning Machines.*[10])

Although some feasibility and design work was done using computer simulations, Rosenblatt preferred building hardware versions of his perceptrons. (Simulations were slow on early computers, thus explaining the interest in building special-purpose perceptron hardware.) The MARK I was an alpha-perceptron built at the Cornell Aeronautical Laboratory under the sponsorship of the Information Systems Branch of the Office of Naval Research and the Rome Air Development Center. It was first publicly demonstrated on 23 June 1960. The MARK I used volume controls (called "potentiometers" by electrical engineers) for weights. These had small motors attached to them for making adjustments to increase or decrease the weight values.

In 1959, Frank Rosenblatt moved his perceptron work from the Cornell Aeronautical Laboratory in Buffalo, New York, to Cornell University, where he became a professor of psychology. Together with Block and several students, Rosenblatt continued experimental and theoretical work on perceptrons. His book *Principles of Neurodynamics* provides a detailed treatment of his theoretical ideas and experimental results.[11] Rosenblatt's last system, called Tobermory, was built as a speech-recognition device.[12] [Tobermory was the name of a cat that learned to speak in *The Chronicles of Clovis*, a group of short stories by Saki (H. H. Munro).] Several Ph.D. students, including George Nagy, Carl Kessler, R. D. Joseph, and others, completed perceptron projects under Rosenblatt at Cornell.

In his last years at Cornell, Rosenblatt moved on to study chemical memory transfer in flatworms and other animals – a topic quite removed from his perceptron work. Tragically, Rosenblatt perished in a sailing accident in Chesapeake Bay in 1969.

Around the same time as Rosenblatt's alpha-perceptron, Woodrow W. (Woody) Bledsoe (1921–1995) and Iben Browning (1918–1991), two mathematicians at Sandia Laboratories in Albuquerque, New Mexico, were also pursuing research on character recognition that used random samplings of input images. They experimented with a system that projected images of alphanumeric characters on a 10×15 mosaic of photocells and sampled the states of 75 randomly chosen pairs of photocells. Pointing out that the idea could be extended to sampling larger groups of pixels, say N of them, they called their method the "N-tuple" method. They used the results of this sampling to make a decision about the category of an input letter.[13]

4.2.2 ADALINES and MADALINES

Independently of Rosenblatt, a group headed by Stanford Electrical Engineering Professor Bernard Widrow (1929–) was also working on neural-network systems during the late 1950s and early 1960s. Widrow had recently joined Stanford after completing a Ph.D. in control theory at MIT. He wanted to use neural-net systems for what he called "adaptive control." One of the devices Widrow built was called an "ADALINE" (for adaptive linear network). It was a single neural element whose adjustable weights were implemented by switchable (thus adjustable) circuits of resistors. Widrow and one of his students, Marcian E. "Ted" Hoff Jr. (who later invented the first microprocessor at Intel), developed an adjustable weight they called a "memistor." It consisted of a graphite rod on which a layer of copper could be plated and unplated – thus varying its electrical resistance. Widrow and Hoff developed a training procedure for their ADALINE neural element that came to be called the Widrow–Hoff least-mean-squares adaptive algorithm. Most of Widrow's experimental work was done using simulations on an IBM1620 computer. Their most complex network design was called a "MADALINE" (for many ADALINEs). A training procedure was developed for it by Stanford Ph.D. student William Ridgway.[14]

4.2.3 The MINOS Systems at SRI

Rosenblatt's success with perceptrons on pattern-recognition problems led to a flurry of research efforts by others to duplicate and extend his results. During the 1960s, perhaps the most significant pattern-recognition work using neural networks was done at the Stanford Research Institute in Menlo Park, California. There, Charles A. Rosen (1917–2002) headed a laboratory that was attempting to etch microscopic vacuum tubes onto a solid-state substrate. Rosen speculated that circuits containing these tubes might ultimately be "wired-up" to perform useful tasks using some of the training procedures being explored by Frank Rosenblatt. SRI employed Rosenblatt as a consultant to help in the design of an exploratory neural network.

When I interviewed for a position at SRI in 1960, a team in Rosen's lab, under the leadership of Alfred E. (Ted) Brain (1923–2004), had just about completed the construction of a small neural network called MINOS (Fig. 4.7). (In Greek mythology, Minos was a king of Crete and the son of Zeus and Europa. After his death, Minos was one of the three judges in the underworld.) Brain felt that computer simulations of neural networks were too slow for practical applications, thus leading to his decision to build rather than to program. (The IBM 1620 computer being used at the same time by Widrow's group at Stanford for simulating neural networks had a basic machine cycle of 21 microseconds and a maximum of 60,000 "digits" of random-access memory.) For adjustable weights, MINOS used magnetic devices designed by Brain.

Rosenblatt stayed in close contact with SRI because he was interested in using these magnetic devices as replacements for his motor-driven potentiometers.

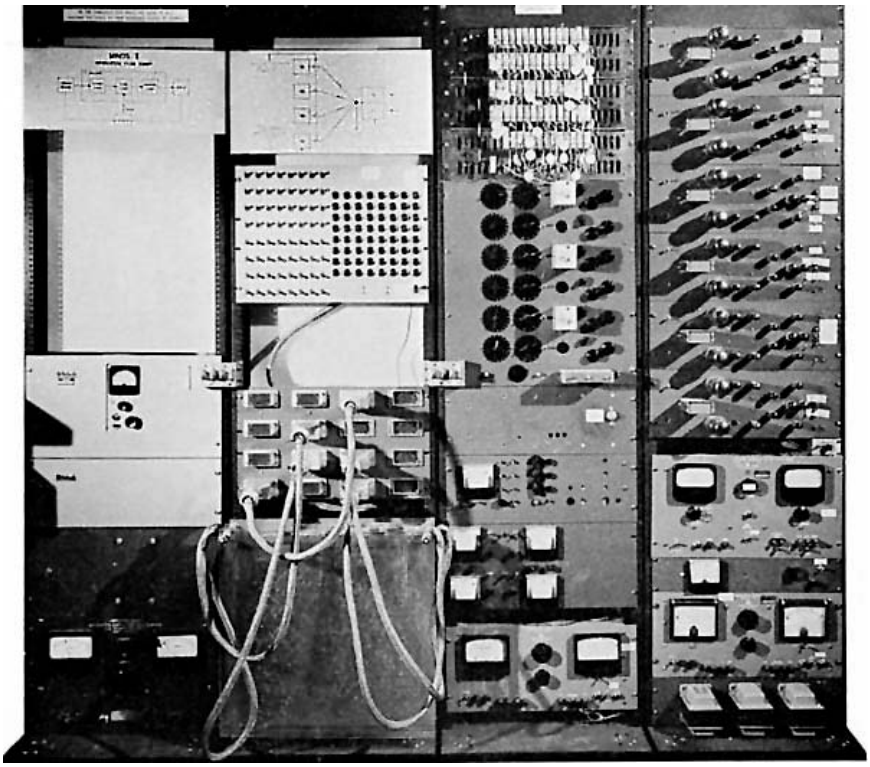


Figure 4.7: MINOS. Note the input switches and corresponding indicator lights in the second-from-the-left rack of equipment. The magnetic weights are at the top of the third rack. (Photograph used with permission of SRI International.)

Rosen's enthusiasm and optimism about the potential for neural networks helped convince me to join SRI. Upon my arrival in July 1961, I was given a draft of Rosenblatt's book to read. Brain's team was just beginning work on the construction of a large neural network, called MINOS II, a follow-on system to the smaller MINOS. (See Fig. 4.8.)

Work on the MINOS systems was supported primarily by the U.S. Army Signal Corps during the period 1958 to 1967. The objective of the MINOS work was "to conduct a research study and experimental investigation of techniques and equipment characteristics suitable for practical application to graphical data processing for military requirements." The main focus of the project was the automatic recognition of symbols on military maps. Other applications – such as the recognition of military vehicles, such as tanks, on aerial photographs and the recognition of hand-printed characters – were also

attempted.[15]

In the first stage of processing by MINOS II, the input image was replicated 100 times by a 10×10 array of plastic lenses. Each of these identical images was then sent through its own optical feature-detecting mask, and the light through the mask was detected by a photocell and compared with a threshold. The result was a set of 100 binary (off–on) values. These values were the inputs to a set of 63 neural elements ("A-units" in Rosenblatt's terminology), each with 100 variable magnetic weights. The 63 binary outputs from these neural elements were then translated into one of 64 decisions about the category of the original input image. (We constructed 64 equally distant "points" in the sixty-three-dimensional space and trained the neural network so that each input image produced a point closer to its own prototype point than to any other. Each of these prototype points was one of the 64 "maximal-length shift-register sequences" of 63 dimensions.)[16]

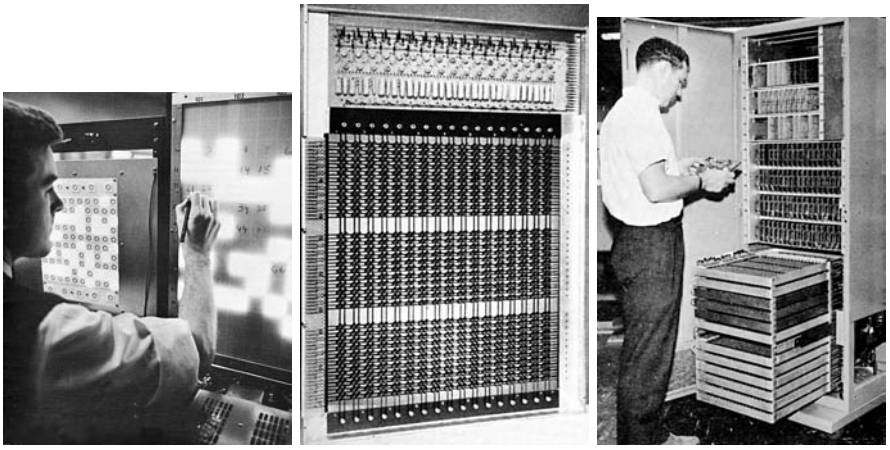


Figure 4.8: MINOS II: operator's display board (left), an individual weight frame (middle), and weight frames with logic circuitry (right). (Photographs used with permission of SRI International.)

During the 1960s, the SRI neural network group, by then called the Learning Machines Group, explored many different network organizations and training procedures. As computers became both more available and more powerful, we increasingly used simulations (at various computer centers) on the Burroughs 220 and 5000 and on the IBM 709 and 7090. In the mid-1960s, we obtained our own dedicated computer, an SDS 910. (The SDS 910, developed at Scientific Data Systems, was the first computer to use silicon transistors.) We used that computer in conjunction with the latest version of our neural network hardware (now using an array of 1,024 preprocessing lenses), a combination we called MINOS III.

One of the most successful results with the MINOS III system was the automatic recognition of hand-printed characters on FORTRAN coding sheets. (In the 1960s, computer programs were typically written by hand and then converted to punched cards by key-punch operators.) This work was led by John Munson (1939–1972; Fig. 4.9), Peter Hart (1941– ; Fig. 4.9), and Richard Duda (1936– ; Fig. 4.9). The neural net part of MINOS III was used to produce a ranking of the possible classifications for each character with a confidence measure for each. For example, the first character encountered in a string of characters might be recognized by the neural net as a "D" with a confidence of 90 and as an "O" with a confidence of 10. But accepting the most confident decision for each character might not result in a string that is a legal statement in the FORTRAN language – indicating that one or more of the decisions was erroneous (where it is assumed that whoever wrote statements on the coding sheet wrote legal statements). Accepting the second or third most confident choices for some of the characters might be required to produce a legal string.

Figure 4.9: John Munson (left), Peter Hart (middle), and Richard Duda (right). (Photographs courtesy of Faith Munson, of Peter Hart, and of Richard Duda.)

The overall confidence of a complete string of characters was calculated by adding the confidences of the individual characters in the string. Then, what was needed was a way of ranking these overall confidence numbers for each of the possible strings resulting from all of the different choices for each character. Among this ranking of all possible strings, the system then selected the most confident *legal* string.

As Richard Duda wrote, however, "The problem of finding the 1st, 2nd, 3rd,... most confident string of characters is by no means a trivial problem." The key to computing the ranking in an efficient manner was to use a method called *dynamic programming*.[17] (In a later chapter, we'll see dynamic programming used again in speech recognition systems.)

An illustration of a sample of the original source and the final output is

shown in Fig. 4.10.

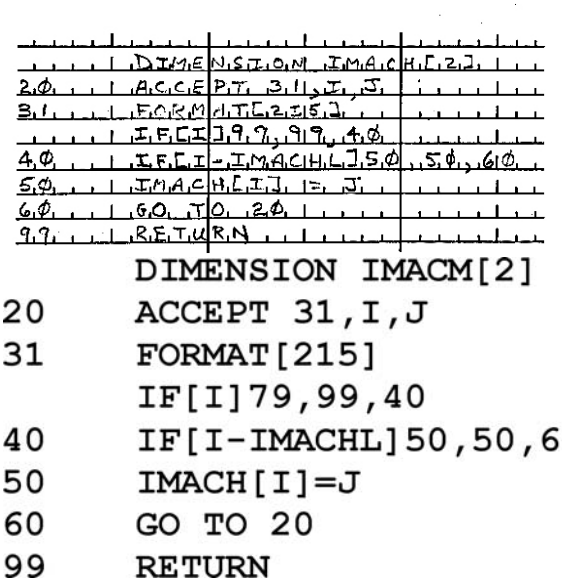


Figure 4.10: Recognition of FORTRAN characters. Input is above and output (with only two errors) is below. (Illustration used with permission of SRI International.)

After the neural net part of the system was trained, the overall system (which decided on the most confident legal string) was able to achieve a recognition accuracy of just over 98% on a large sample of material that was not part of what the system was trained on. Recognizing handwritten characters with this level of accuracy was a significant achievement in the 1960s.[18]

Expanding its interests beyond neural networks, the Learning Machines Group ultimately became the SRI Artificial Intelligence Center, which continues today as a leading AI research enterprise.

4.3 Statistical Methods

During the 1950s and 1960s there were several applications of statistical methods to pattern-recognition problems. Many of these methods bore a close resemblance to some of the neural network techniques. Recall that earlier I explained how to decide which of two tones was present in a noisy radio signal. A similar technique could be used for pattern recognition. For classifying images (or other perceptual inputs), it was usual to represent the input by a

list of distinguishing "features" such as those used by Selfridge and his colleagues. In alphanumeric character recognition for example, one first extracted features from the image of the character to be classified. Usually the features had numerical values, such as the number of times lines of different angles intersected the character or the length of the perimeter of the smallest circle that completely enclosed the character. Selecting appropriate features was often more art than science, but it was critical to good performance.

We'll need a bit of elementary mathematical notation to help describe these statistically oriented pattern-recognition methods. Suppose the list of features extracted from a character is $\{f_1, f_2, \ldots, f_i, \ldots, f_N\}$. I'll abbreviate this list by the bold-face symbol $\mathbf{X}$. Suppose there are k categories, $C_1, C_2, \ldots, C_i, \ldots, C_k$ to which the character described by $\mathbf{X}$ might belong. Using Bayes's rule in a manner similar to that described earlier, the decision rule is the following:

> Decide in favor of that category for which $p(\mathbf{X} \mid C_i)p(C_i)$ is largest, where $p(C_i)$ is the *a priori* probability of category C_i and $p(\mathbf{X} \mid C_i)$ is the likelihood of $\mathbf{X}$ given C_i. These likelihoods can be inferred by collecting statistical data from a large sample of characters.

As I mentioned earlier, researchers in pattern recognition often describe the decision process in terms of geometry. They imagine that the values of the features obtained from an image sample can be represented as a point in a multidimensional space. If we have several samples for each of, say, two known categories of data, we can represent these samples as scatterings of points in the space. In character recognition, scattering can occur not only because the image of the character might be noisy but also because characters in the same category might be drawn slightly differently. I show a two-dimensional example, with features f_1 and f_2, in Fig. 4.11. From the scattering of points in each category we can compute an estimate of the probabilities needed for computing likelihoods. Then, we can use the likelihoods and the prior probabilities to make decisions.

I show in this figure the boundary, computed from the likelihoods and the prior probabilities, that divides the space into two regions. In one region, we decide in favor of category 1; in the other, we decide in favor of category 2. I also show a new feature point, $\mathbf{X}$, to be classified. In this case, the position of $\mathbf{X}$ relative to the boundary dictates that we classify $\mathbf{X}$ as a member of category 1.

There are other methods also for classifying feature points. An interesting example is the "nearest-neighbor" method. In that scheme, invented by E. Fix and J. L. Hodges in 1951,[19] a new feature point is assigned to the same category as that sample feature point to which it is closest. In Fig. 4.11, the new point $\mathbf{X}$ would be classified as belonging to category 2 using the nearest-neighbor method.

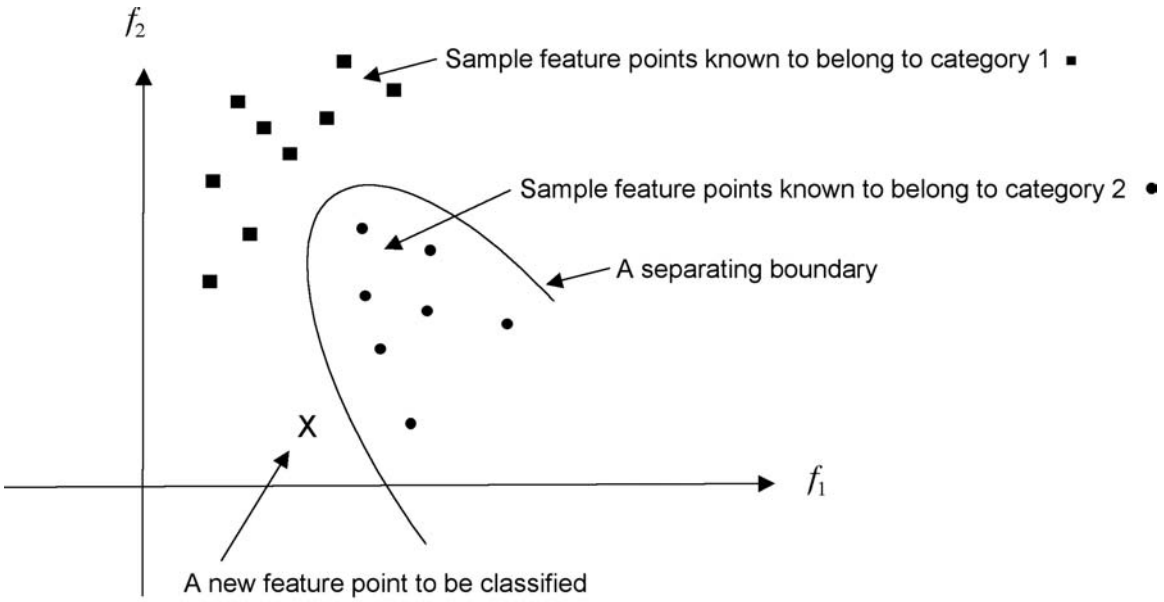


Figure 4.11: A two-dimensional space of feature points and a separating boundary.

An important elaboration of the nearest-neighbor method assigns a new point to the same category as the majority of the k closest points. Such a decision rule seems plausible (in the case in which there are many, many sample points of each category) because there being more sample points of category C_i closer to an unknown point, $\mathbf{X}$, than sample points of category C_j is evidence that $p(\mathbf{X} \mid C_i)p(C_i)$ is greater than $p(\mathbf{X} \mid C_j)p(C_j)$. Expanding on that general observation, Thomas Cover and Peter Hart rigorously analyzed the performance of nearest-neighbor methods.[20]

Any technique for pattern recognition, even those using neural networks or nearest neighbors, can be thought of as constructing separating boundaries in a multidimensional space of features. Another method for constructing boundaries using "potential functions" was suggested by the Russian scientists M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer in the 1960s.[21]

Some important early books on the use of statistical methods in pattern recognition are ones by George Sebestyen,[22] myself,[23] and Richard Duda and Peter Hart.[24] My book also describes some of the relationships between statistical methods and those based on neural networks. The technology of pattern recognition as of the late 1960s is nicely reviewed by George Nagy (who had earlier been one of Frank Rosenblatt's graduate students).[25]

4.4 Applications of Pattern Recognition to Aerial Reconnaissance

The neural network and statistical methods for pattern recognition attracted much attention in many aerospace and avionics companies during the late 1950s and early 1960s. These companies had ample research and development budgets stemming from their contracts with the U.S. Department of Defense. Many of them were particularly interested in the problem of aerial reconnaissance, that is, locating and identifying "targets" in aerial photographs. Among the companies having substantial research programs devoted to this and related problems were the Aeronutronic Division of the Ford Motor Co.,[26] Douglas Aircraft Company (as it was known at that time), General Dynamics, Lockheed Missiles and Space Division, and the Philco Corporation. (Philco was later acquired by Ford in late 1961.)

I'll mention some of the work at Philco as representative. There, Laveen N. Kanal (1931–), Neil C. Randall (1930–), and Thomas Harley (1929–) worked on both the theory and applications of statistical pattern-recognition methods. The systems they developed were for screening aerial photographs for interesting military targets such as tanks. A schematic illustration of one of their systems is shown in Fig. 4.12.[27]

Philco's apparatus scanned material from 9-inch film negatives gathered by a U2 reconnaissance airplane during U.S. Army tank maneuvers at Fort Drum, New York. A small section of the scanned photograph, possibly containing an M-48 tank (in standard position and size), was first processed to enhance edges, and the result was presented to the target detection system as an array of 1's and 0's. The first of their systems used a 22×12 array; later ones used a 32×32 array as shown in Fig. 4.12. The array was then segmented into 24 overlapping 8×8 "feature blocks." The data in each feature block were then subjected to a statistical test to decide whether or not the small area of the picture represented by this block contained part of a tank.

The statistical tests were based on a "training sample" of 50 images containing tanks and 50 samples of terrain not containing tanks. For each 8×8 feature block, statistical parameters were compiled from these samples to determine a (linear) boundary in the sixty-four-dimensional space that best discriminated the tank samples from the nontank samples.

Using these boundaries, the system was then tested on a different set of 50 images containing tanks and 50 images not containing tanks. For each test image, the number of feature blocks deciding "tank present" was tallied to produce a final numerical "score" (such as 21 out of the 24 blocks decided a tank was present). This score could then be used to decide whether or not the image contained a tank.

The authors stated that "the experimental performance of the statistical classification procedure exceeded all expectations." Almost half of the test

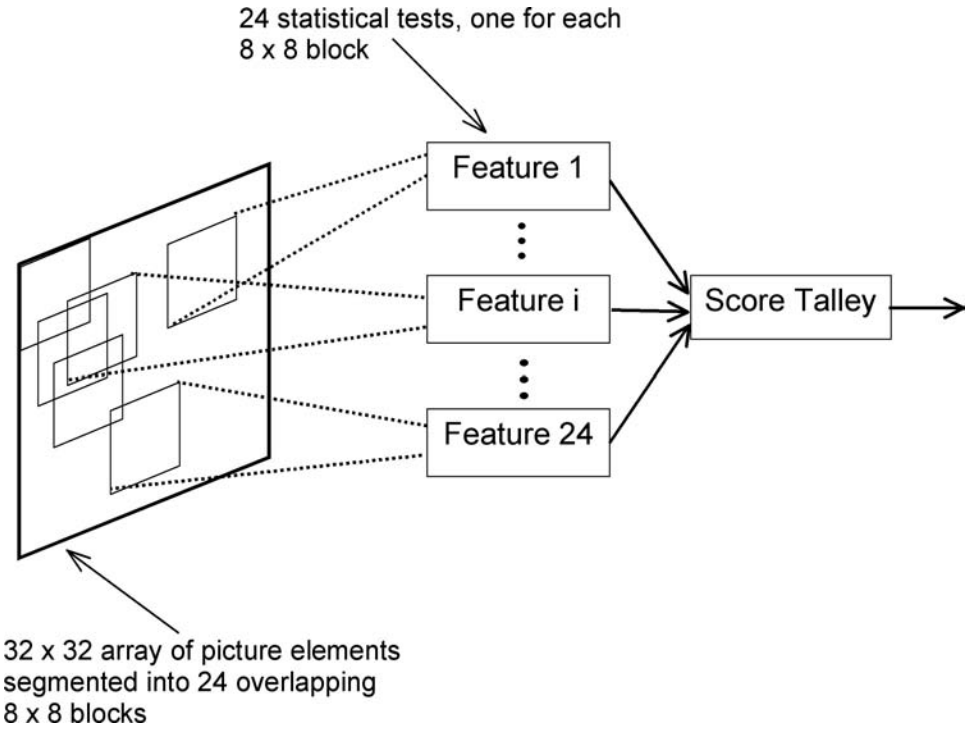


Figure 4.12: A Philco tank-recognition system. (Adapted from Laveen N. Kanal and Neal C. Randall, "Target Detection in Aerial Photography," paper 8.3, *Proceedings of the 1964 Western Electronics Show and Convention (WESCON)*, Los Angeles, CA, Institute of Radio Engineers (now IEEE), August 25–28, 1964.)

samples had perfect scores (that is, all 24 feature blocks correctly discriminated between tank and nontank). Furthermore, all of the test samples containing tanks had a score greater than or equal to 11, and all of the test samples not containing tanks had a score less than or equal to 7.

An early tank-detecting system at Philco was built with analog circuitry – not programmed on a computer. As Thomas Harley, the project leader for this system, later elaborated,[28]

> It is important to remember the technological context of the era in which this work was done. The system we implemented had no built-in computational capabilities. The weights in the linear discriminant function were resistors that controlled the current coming from the (binary) voltage source in the shift register elements. Those currents were added together, and each feature was recognized or not depending whether on the sum of those

> currents exceeded a threshold value. Those binary feature decisions were then summed, again in an analog electrical circuit, not in a computer, and again a decision [tank or no tank] was made depending on whether the sum exceeded a threshold value.

In another system, the statistical classification was implemented by a program, called MULTINORM, running on the Philco 2000 computer.[29] In other experiments Philco used additional statistical tests to weight some of the feature blocks more heavily than others in computing the final score. Kanal told me that these experiments with weighting the outputs of the feature blocks "anticipated the support vector machine (SVM) classification idea. . . [by] using the first layer to identify the training samples close to the boundary between tanks and non-tanks."[30] (I'll describe the important SVM method in a later chapter.)

Of course, these systems had a rather easy task. All of the tanks were in standard position and were already isolated in the photograph. (The authors mention, however, how the system could be adapted to deal with tanks occurring in any position or orientation in the image.) The photograph in Fig. 4.13 shows a typical tank image. (The nontank images are similar, except without the tank.)

I find the system interesting not only because of its performance but also because it is a layered system (similar to Pandemonium and to the alpha-perceptron) and because it is an example in which the original image is divided into overlapping subimages, each of which is independently processed. As I'll mention later, overlapping subimages play a prominent role in some computational models of the neocortex.

Unfortunately, the Philco reports giving details of this work aren't readily available.[31] Furthermore, Philco and some of the other groups engaged in this work have disappeared. Here is what Tom Harley wrote me about the Philco reports and about Philco itself:[32]

> Most of the pattern recognition work done at Philco in the 1960s was sponsored by the DoD [Department of Defense], and the reports were not available for public distribution. Since then, the company itself has really vanished into thin air. Philco was bought by Ford Motor Company in 1961, and by 1966, they had eliminated the Philco research labs where Laveen [Kanal] and I were working. Ford tried to move our small pattern recognition group to Newport Beach, CA [the location of Ford's Aeronutronic Division, whose pattern recognition group folded later also], and when we all decided not to go, they transferred us to their Communications Division, and told us to close out our pattern recognition projects. Laveen eventually went off to the University of Maryland, and in 1975, I transferred to the Ford Aerospace

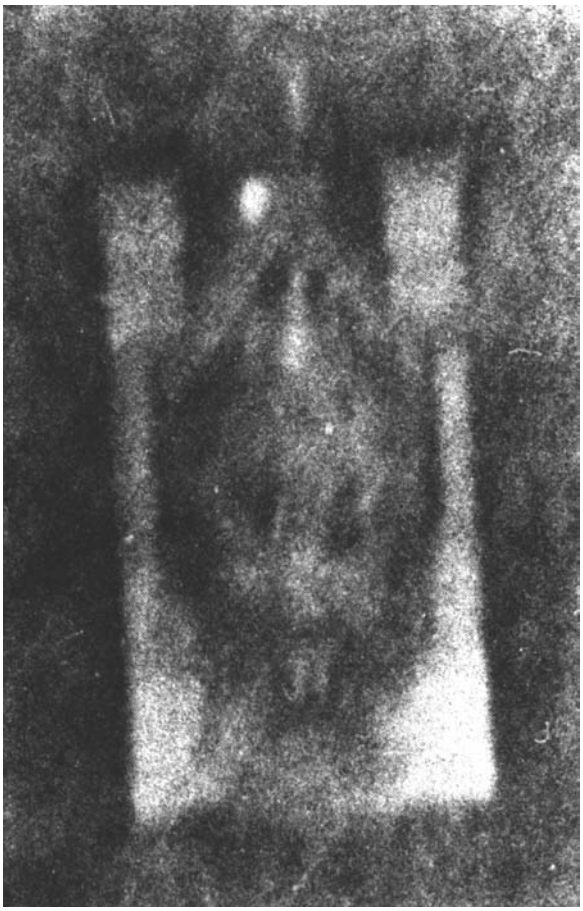


Figure 4.13: A typical tank image. (Photograph courtesy of Thomas Harley.)

> Western Development Labs (WDL) in Palo Alto, where I worked on large systems for the intelligence community. In later years, what had been Philco was sold to Loral, and most of that was later sold to Lockheed Martin. I retired from Lockheed in 2001.

Approaches to AI problems involving neural networks and statistical techniques came to be called "nonsymbolic" to contrast them with the "symbol-processing" work being pursued by those interested in proving theorems, playing games, and problem solving. These nonsymbolic approaches found application mainly in pattern recognition, speech processing, and computer vision. Workshops and conferences devoted especially to those topics began to be held in the 1960s. A subgroup of the IEEE Computer Society (the Pattern Recognition Subcommittee of the Data Acquisition and

Transformation Committee) organized the first "Pattern Recognition Workshop," which was held in Puerto Rico in October 1966.[33] A second one (which I attended) was held in Delft, The Netherlands, in August 1968. In 1966, this subgroup became the IEEE Computer Society Pattern Analysis and Machine Intelligence (PAMI) Technical Committee, which continued to organize conferences and workshops.[34]

Meanwhile, during the late 1950s and early 1960s, the symbol-processing people did their work mainly at MIT, at Carnegie Mellon University, at IBM, and at Stanford University. I'll turn next to describing some of what they did.

Notes

1. See http://www.nist.gov/public_affairs/techbeat/tb2007_0524.htm. [**89**]

2. Russell A. Kirsch *et al.*, "Experiments in Processing Pictorial Information with a Digital Computer," *Proceedings of the Eastern Joint Computer Conference*, pp. 221–229, Institute of Radio Engineering and Association for Computing Machinery, December 1957. [**89**]

3. The proceedings of the conference were published in George L. Fischer Jr. *et al.*, *Optical Character Recognition*, Washington, DC: Spartan Books, 1962. [**90**]

4. From J. Rabinow, "Developments in Character Recognition Machines at Rabinow Engineering Company," in George L. Fischer Jr. *et al.*, *op. cit.*, p. 27. [**90**]

5. From http://www.sri.com/about/timeline/erma-micr.html. [**90**]

6. Oliver G. Selfridge and Ulrich Neisser, "Pattern Recognition by Machine," *Scientific American*, Vol. 203, pp. 60–68, 1960. (Reprinted in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, pp. 237ff, New York: McGraw Hill, 1963.) [**91**]

7. An early reference is Frank Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, Vol. 65, pp. 386ff, 1958. [**92**]

8. H. David Block, "The Perceptron: A Model for Brain Functioning," *Reviews of Modern Physics*, Vol. 34, No. 1, pp. 123–135, January 1962. [**97**]

9. Albert B. J. Novikoff, "On Convergence Proofs for Perceptrons," in *Proceedings of the Symposium on Mathematical Theory of Automata*, pp. 615–622, Brooklyn, NY: Polytechnic Press of Polytechnic Inst. of Brooklyn, 1963. [**97**]

10. Nils J. Nilsson, *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*, New York: McGraw-Hill Book Co., 1965; republished as *The Mathematical Foundations of Learning Machines*, San Francisco: Morgan Kaufmann Publishers, 1990. [**97**]

11. Frank Rosenblatt, *Principles of Neurodynamics*, Washington, DC: Spartan Books, 1962. [**97**]

12. Frank Rosenblatt, "A Description of the Tobermory Perceptron," *Collected Technical Papers*, Vol. 2, Cognitive Systems Research Program, Cornell University, 1963. [**97**]

13. Woodrow W. Bledsoe and Iben Browning, "Pattern Recognition and Reading by Machine," *Proceedings of the Eastern Joint Computer Conference*, pp. 225–232, New York: Association for Computing Machinery, 1959. [**97**]

14. William C. Ridgway, "An Adaptive Logic System with Generalizing Properties," *Stanford Electronics Laboratories Technical Report 1556-1*, Stanford University, Stanford, CA, 1962. [**98**]

15. For a description of MINOS II, see Alfred E. Brain, George Forsen, David Hall, and Charles Rosen, "A Large, Self-Contained Learning Machine," *Proceedings of the Western Electronic Show and Convention*, 1963. The paper was reprinted as Appendix C of an SRI proposal and is available online at http://www.ai.sri.com/pubs/files/rosen65-esu65-1tech.pdf. [**100**]

16. For a discussion of shift-register codes and other codes, see W. Peterson, *Error-Correcting Codes*, New York: John Wiley & Sons, 1961. Our technique was reported in A. E. Brain and N. J. Nilsson, "Graphical Data Processing Research Study and Experimental Investigation," Quarterly Progress Report No. 8, p. 11, SRI Report, June 1962; available online at http://www.ai.sri.com/pubs/files/1329.pdf. [**100**]

17. Robert E. Larsen of SRI suggested using this method. The online encyclopedia Wikipedia has a clear description of dynamic programming. See http://en.wikipedia.org/wiki/Dynamic_programming.

[**101**]

18. The technical details of the complete system are described in two papers: John Munson, "Experiments in the Recognition of Hand-Printed Text: Part I – Character Recognition," and Richard O. Duda and Peter E. Hart, "Experiments in the Recognition of Hand-Printed Text: Part II – Context Analysis," *AFIPS Conference Proceedings*, (of the 1968 Fall Joint Computer Conference), Vol. 33, pp. 1125–1149, Washington, DC: Thompson Book Co., 1968. Additional information can be found in SRI AI Center Technical reports, available online at http://www.ai.sri.com/pubs/files/1343.pdf and http://www.ai.sri.com/pubs/files/1344.pdf. [**102**]

19. E. Fix and J. L. Hodges Jr., "Discriminatory analysis, nonparametric discrimination," USAF School of Aviation Medicine, Randolph Field, Texas, Project 21-49-004, Report 4, Contract AF41(128)-31, February 1951. See also B. V. Dasarathy (ed.), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press, which is a reprint of 1951 unpublished work of Fix and Hodges. [**103**]

20. Thomas M. Cover and Peter E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, pp. 21–27, January 1967. Available online at http://ieeexplore.ieee.org/iel5/18/22633/01053964.pdf. [**104**]

21. See M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer, "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning," *Automation and Remote Control*, Vol. 25, pp. 917–936, 1964, and A. G. Arkadev and E. M. Braverman, *Computers and Pattern Recognition*, (translated from the Russian by W. Turski and J. D. Cowan), Washington, DC: Thompson Book Co., Inc., 1967. [**104**]

22. George S. Sebestyen, *Decision-Making Processes in Pattern Recognition*, Indianapolis, IN: Macmillan Publishing Co., Inc., 1962. [**104**]

23. Nils J. Nilsson, *op. cit.* [**104**]

24. Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, New York: John Wiley & Sons, 1973; updated version: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*, 2nd Edition, New York: John Wiley & Sons, 2000. [**104**]

25. George Nagy, "State of the Art in Pattern Recognition," *Proceedings of the IEEE*, Vol. 56, No. 5, pp. 836–857, May 1968. [**104**]

26. See, for example, Joseph K. Hawkins and C. J. Munsey, "An Adaptive System with Direct Optical Input," *Proceedings of the IEEE*, Vol. 55, No. 6, pp. 1084–1085, June 1967. Available online for IEEE members at http://ieeexplore.ieee.org/iel5/5/31078/01446273.pdf?tp=&arnumber=1446273&isnumber=31078. [**105**]

27. Laveen N. Kanal and Neal C. Randall, "Target Detection in Aerial Photography," paper 8.3, *Proceedings of the 1964 Western Electronics Show and Convention (WESCON)*, Los Angeles, CA, Institute of Radio Engineers (now IEEE), August 25–28, 1964. (Several other

papers on pattern recognition were presented at this conference and are contained in the proceedings.) [**105**]

28. Thomas Harley, personal e-mail communication, July 15, 2007. [**106**]

29. Laveen N. Kanal and Neal C. Randall, *op. cit.*. [**107**]

30. Laveen Kanal, personal e-mail communication, July 13, 2007. [**107**]

31. Laveen N. Kanal, "Statistical Methods for Pattern Classification," Philco Report, 1963; originally appeared in T. Harley *et al.*, "Semi-Automatic Imagery Screening Research Study and Experimental Investigation," Philco Reports VO43-2 and VO43-3, Vol. I, Sec. 6, and Appendix H, prepared for U.S. Army Electronics Research and Development Laboratory under Contract DA-36-039-SC- 90742, March 29, 1963. [**107**]

32. Thomas Harley, personal e-mail communication, July 11, 2007. [**107**]

33. Laveen N. Kanal (ed.), *Pattern Recognition, Proceedings of the IEEE Workshop on Pattern Recognition*, held at Dorado, Puerto Rico, Washington, DC: Thompson Book Co., 1968. [**109**]

34. See the Web page at http://tab.computer.org/pamitc/. [**109**]

Chapter 5

Early Heuristic Programs

5.1 The Logic Theorist and Heuristic Search

Just prior to the Dartmouth workshop, Newell, Shaw, and Simon had programmed a version of LT on a computer at the RAND Corporation called the JOHNNIAC (named in honor of John von Neumann). Later papers[1] described how it proved some of the theorems in symbolic logic that were proved by Russell and Whitehead in Volume I of their classic work, *Principia Mathematica*.[2] LT worked by performing transformations on Russell and Whitehead's five axioms of propositional logic, represented for the computer by "symbol structures," until a structure was produced that corresponded to the theorem to be proved. Because there are so many different transformations that could be performed, finding the appropriate ones for proving the given theorem involves what computer science people call a "search process."

To describe how LT and other symbolic AI programs work, I need to explain first what is meant by a "symbol structure" and what is meant by "transforming" them. In a computer, symbols can be combined in lists, such as $(\mathrm{A}, 7, \mathrm{Q})$. Symbols and lists of symbols are the simplest kinds of symbol structures. More complex structures are composed of lists of lists of symbols, such as $((\mathrm{B}, 3), (\mathrm{A}, 7, \mathrm{Q}))$, and lists of lists of lists of symbols, and so on. Because such lists of lists, etc. can be quite complex, they are called "structures." Computer programs can be written that transform symbol structures into other symbol structures. For example, with a suitable program the structure "(the sum of seven and five)" could be transformed into the structure "(7 + 5)," which could further be transformed into the symbol "12."

Transforming structures of symbols and searching for an appropriate problem-solving sequence of transformations lies at the heart of Newell and Simon's ideas about mechanizing intelligence. In a later paper (the one they gave on the occasion of their receiving the prestigious Turing Award), they

summarized the process as follows:[3]

> The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search – that is, by generating and progressively modifying symbol structures until it produces a solution structure.
>
> . . .
>
> To state a problem is to designate (1) a test for a class of symbol structures (solutions of the problem), and (2) a generator of symbol structures (potential solutions). To solve a problem is to generate a structure, using (2), that satisfies the test of (1).

Understanding in detail how LT itself used symbol structures and their transformations to prove theorems would require some mathematical and logical background. The process is easier to explain by using one of AI's favorite "toy problems" – the "fifteen-puzzle." (See Fig. 5.1.) The fifteen-puzzle is one of several types of sliding-block puzzles. The problem is to transform an array of tiles from an initial configuration into a "goal" configuration by a succession of moves of a tile into an adjacent empty cell.

Figure 5.1: Start (left) and goal (right) configurations of a fifteen-puzzle problem.

I'll use a simpler version of the puzzle – one that uses a 3×3 array of eight sliding tiles instead of the 4×4 array. (AI researchers have experimented with programs for solving larger versions of the puzzle also, such as 5×5 and 6×6.)

Suppose we wanted to move the tiles from their configuration on the left to the one on the right as illustrated in Fig. 5.2.

Following the Newell and Simon approach, we must first represent tile positions for the computer by symbol structures that the computer can deal

Figure 5.2: The eight-puzzle.

with. I will represent the starting position by the following structure, which is a list of three sublists:

$$((2,8,3),(1,6,4),(7,\mathtt{B},5)).$$

The first sublist, namely, $(2,8,3)$, names the occupants of the first row of the puzzle array, and so on. `B` stands for the empty cell in the middle of the third row.

In the same fashion, the goal configuration is represented by the following structure:

$$((1,2,3),(8,\mathtt{B},4),(7,6,5)).$$

Next, we have to show how a computer can transform structures of the kind we have set up in a way that corresponds to the allowed moves of the puzzle. Note that when a tile is moved, it swaps places with the blank cell; that is, the blank cell moves too. The blank cell can either move within its row or can change rows.

Corresponding to these moves of the blank cell, when a tile moves within its row, `B` swaps places with the number either to its left in its list (if there is one) or to its right (if there is one). A computer can easily make either of these transformations. When the blank cell moves up or down, `B` swaps places with the number in the corresponding position in the list to the left (if there is one) or in the list to the right (if there is one). These transformations can also be made quite easily by a computer program.

Using the Newell and Simon approach, we start with the symbol structure representing the starting configuration of the eight-puzzle and apply allowed transformations until a goal is reached. There are three transformations of the starting symbol structure. These produce the following structures:

$$((2,8,3),(1,6,4),(\mathtt{B},7,5)),$$

$$((2,8,3),(1,6,4),(7,5,\mathtt{B})),$$

and

$$((2,8,3),(1,\mathtt{B},4),(7,6,5)).$$

None of these represents the goal configuration, so we continue to apply transformations to each of these and so on until a structure representing the goal is reached. We (and the computer) can keep track of the transformations made by arranging them in a treelike structure such as shown in Fig. 5.3. (The arrowheads on both ends of the lines representing the transformations indicate that each transformation is reversible.)

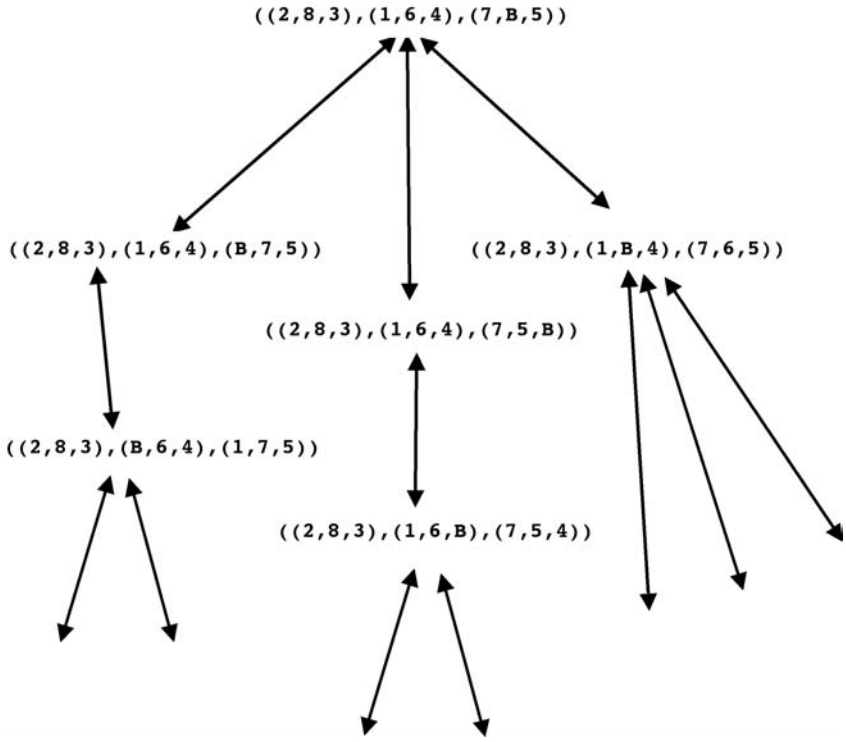


Figure 5.3: A search tree.

This version of the eight-puzzle is relatively simple, so not many transformations have to be tried before the goal is reached. Typically though (especially in larger versions of the puzzle), the computer would be swamped by all of the possible transformations – so much so that it would never generate a goal expression. To constrain what was later called "the combinatorial explosion" of transformations, Newell and Simon suggested using "heuristics" to generate only those transformations guessed as likely to be on the path to a solution.

In one of their papers about LT, they wrote "A process that *may* solve a problem, but offers no guarantees of doing so, is called a *heuristic* for that problem." Rather than blindly striking out in all directions in a search for a

proof, LT used search guided by heuristics, or "heuristic search." Usually, as was the case with LT, there is no guarantee that heuristic search will be successful, but when it is successful (and that is quite often) it eliminates much otherwise fruitless search effort.

The search for a solution to an eight-puzzle problem involves growing the tree of symbol structures by applying transformations to the "leaves" of the tree and thus extending it. To limit the growth of the tree, we should use heuristics to apply transformations only to those leaves thought to be on the way to a solution. One such heuristic might be to apply a transformation to that leaf with the smallest number of tiles out of position compared to the goal configuration. Because sliding tile problems have been thoroughly studied, there are a number of heuristics that have proved useful – ones much better than the simple number-of-tiles-out-of-position one I have just suggested.

Using heuristics keyed to the problem being solved became a major theme in artificial intelligence, giving rise to what is called "heuristic programming." Perhaps the idea of heuristic search was already "in the air" around the time of the Dartmouth workshop. It was implicit in earlier work by Claude Shannon. In March 1950, Shannon, an avid chess player, published a paper proposing ideas for programming a computer to play chess.[4] In his paper, Shannon distinguished between what he called "type A" and "type B" strategies. Type A strategies examine every possible combination of moves, whereas type B strategies use specialized knowledge of chess to focus on lines of play thought to be the most productive. The type B strategies depended on what Newell and Simon later called heuristics. And Minsky is quoted as saying "...I had already considered the idea of heuristic search obvious and natural, so that the Logic Theorist was not impressive to me."[5]

It was recognized quite early in AI that the way a problem is set up, its "representation," is critical to its solution. One example of how a representation affects problem solving is due to John McCarthy and is called the "mutilated checkerboard" problem.[6] Here's the problem: "Two diagonally opposite corner squares are removed from a checkerboard. Is it possible to cover the remaining squares with dominoes?" (A domino is a rectangular tile that covers two adjacent squares.) A naive way of searching for a solution would be to try to place dominoes in all possible ways over the checkerboard. But, if one uses the information that a checkerboard consists of 32 squares of one color and 32 of another color, and that the opposite corner squares are of the same color, then one realizes that the mutilated board consists of 30 squares of one color and 32 of another. Because a domino covers two squares of opposite colors, there is no way that a set of them can cover the remaining colors. McCarthy was interested in whether or not people could come up with "creative" ways to formulate the puzzle so that it could be solved by computers using methods based on logical deduction.

Another classic puzzle that has been used to study the effects of different representations is the "missionary and cannibals" problem: Three cannibals

and three missionaries must cross a river. Their boat can only hold two people. If the cannibals outnumber the missionaries, on either side of the river, the missionaries on that side perish. Each missionary and each cannibal can row the boat. How can all six get across the river safely? Most people have no trouble formulating this puzzle as a search problem, and the solution is relatively easy. But it does require making one rather nonintuitive step. The computer scientist and AI researcher Saul Amarel (1928–2002) wrote a much-referenced paper analyzing this puzzle and various extended versions of it in which there can be various numbers of missionaries and cannibals.[7] (The extended versions don't appear to be so easy.) After moving from one representation to another, Amarel finally developed a representation for a generalized version of the problem whose solution required virtually no search. AI researchers are still studying how best to represent problems and, most importantly, how to get AI systems to come up with their own representations.

5.2 Proving Theorems in Geometry

Nathan Rochester returned to IBM after the Dartmouth workshop excited about discussions he had had with Marvin Minsky about Minsky's ideas for a possible computer program for proving theorems in geometry. He described these ideas to a new IBM employee, Herb Gelernter (1929–). Gelernter soon began a research project to develop a geometry-theorem-proving machine. He presented a paper on the first version of his program at a conference in Paris in June 1959,[8] acknowledging that

> [t]he research project itself is a consequence of the Dartmouth Summer Research Project on Artificial Intelligence held in 1956, during which M. L. Minsky pointed out the potential utility of the diagram to a geometry theorem-proving machine.

Gelernter's program exploited two important ideas. One was the explicit use of subgoals (sometimes called "reasoning backward" or "divide and conquer"), and the other was the use of a diagram to close off futile search paths.

The strategy taught in high school for proving a theorem in geometry involves finding some subsidiary geometric facts from which, if true, the theorem would follow immediately. For example, to prove that two angles are equal, it suffices to show that they are corresponding angles of two "congruent" triangles. (A triangle is congruent to another if it can be translated and rotated, possibly even flipped over, in such a way that it matches the other exactly.) So now, the original problem is transformed into the problem of showing that two triangles are congruent. One way (among others) to show that two triangles are congruent is to show that two corresponding sides and the enclosed angle of the two triangles all have the

same sizes. This backward reasoning process ends when what remains to be shown is among the premises of the theorem.

Readers familiar with geometry will be able to follow the illustrative example shown in Fig. 5.4. There, on the left-hand side, we are given triangle ABC with side AB equal to side AC and must prove that angle ABC is equal to angle ACB. The triangle on the right side is a flipped-over version of triangle ABC.

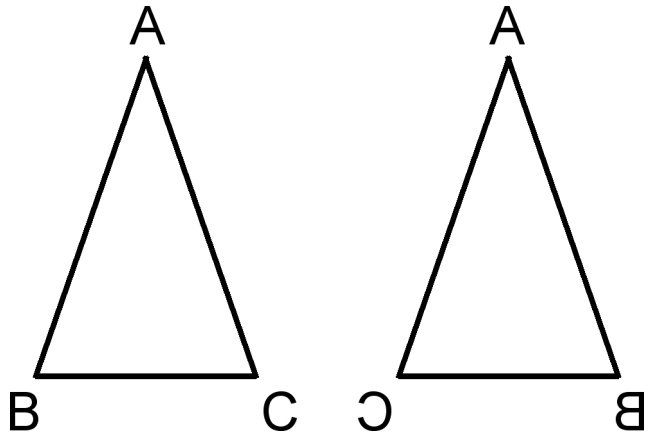


Figure 5.4: A triangle with two equal sides (left) and its flipped-over version (right).

Here is how the proof goes: If we could prove that triangle ABC is congruent to triangle AƆƎ, then the theorem would follow because the two angles are corresponding angles of the two triangles. These two triangles can be proved congruent if we could establish that side AB (of triangle ABC) is equal to side AƆ (of triangle AƆƎ) and that side AC (of triangle ABC) is equal to side AƎ (of triangle AƆƎ) and that angle A (of triangle ABC) is equal to angle A (of triangle AƆƎ). But the premises state that side AB is equal to side AC, and these lengths don't change in the flipped-over triangle. Similarly, angle A is equal to its flipped-over version – so we have our proof.

Before continuing my description of Gelernter's program, a short historical digression is in order. The geometry theorem just proved is famous – being the fifth proposition in Book I of Euclid's *Elements*. Because Euclid's proof of the proposition was a difficult problem for beginners it became known as the *pons asinorum* or "fools bridge." The proof given here is simpler than Euclid's – a version of it was given by Pappus of Alexandria (circa 290–350 CE).

Minsky's "hand simulation" of a program for proving theorems in geometry, discussed at Dartmouth, came up with this very proof (omitting what I think is the helpful step of flipping the triangle over). Minsky wrote[9]

> In 1956 I wrote two memos about a hand-simulated program for proving theorems in geometry. In the first memo, the procedure found the simple proof that if a triangle has two equal sides then the corresponding angles are equal. It did this by noticing that triangle ABC was congruent to triangle CBA because of "side-angle-side." What was interesting is that this was found after a very short search – because, after all, there weren't many things to do. You might say the program was too stupid to do what a person might do, that is, think, "Oh, those are both the same triangle. Surely no good could come from giving it two different names." (The program has a collection of heuristic methods for proving Euclid-Like theorems, and one was that "if you want to prove two angles are equal, show that they're corresponding parts of congruent triangles." Then it also had several ways to demonstrate congruence. There wasn't much more in that first simulation.) But I can't find that memo anywhere.

As Minsky said, this is a very easy problem for a computer. Gelernter's program proved much more difficult theorems, and for these his use of a diagram was essential. The program did not literally draw and look at a diagram. Instead, as Gelernter wrote,

> [The program is] supplied with the diagram in the form of a list of possible coordinates for the points named in the theorem. This point list is accompanied by another list specifying the points joined by segments. Coordinates are chosen to reflect the greatest possible generality in the figures.

So, for example, the points named in the problem about proving two angles equal are the vertices of the triangle ABC, namely, points A and B and C. Coordinates for each of these points are chosen, and care is taken to make sure that these coordinates do not happen to satisfy any special unnamed properties.

Gelernter's program worked by setting up subgoals and subsubgoals such as those I used in the example just given. It then searched for a chain of these ending in subgoals that could be established directly from the premises. Before any subgoal was selected by the program to be worked on however, it was first tested to see whether it held in the diagram. If it did hold, it might possibly be provable and could therefore be considered as a possible route to a proof. But, if it did not hold in the diagram, it could not possibly be true. Thus, it could be eliminated from further consideration, thereby "pruning" the search tree and saving what would certainly be fruitless effort. Later work in AI would also exploit "semantic" information of this sort.

We can see similarities between the strategies used in the geometry program and those used by humans when we solve problems. It is common for

us to work backward – transforming a hard problem into subproblems and those into subsubproblems and so on until finally the problems are trivial. When a subproblem has many parts, we know that we must solve all of them. We also recognize when a proposed subproblem is patently impossible and thus can reject it. The next program I describe was based explicitly on what its authors thought were human problem-solving strategies.

5.3 The General Problem Solver

At the same 1959 Paris conference where Gelernter presented his program, Allen Newell, J. C. Shaw, and Herb Simon gave a paper describing their recent work on mechanizing problem solving.[10] Their program, which they called the "General Problem Solver (GPS)," was an embodiment of their ideas about how humans solve problems. Indeed, they claimed that the program itself was a theory of human problem-solving behavior. Newell and Simon were among those who were just as interested (perhaps even more interested) in explaining the intelligent behavior of humans as they were in building intelligent machines. They wrote "It is often argued that a careful line must be drawn between the attempt to *accomplish* with machines the same tasks that humans perform, and the attempt to *simulate* the processes humans actually use to accomplish these tasks. ... GPS maximally confuses the two approaches – with mutual benefit."[11]

GPS was an outgrowth of their earlier work on the Logic Theorist in that it was based on manipulating symbol structures (which they believed humans did also). But GPS had an important additional mechanism among its symbol-manipulating strategies. Like Gelernter's geometry program, GPS transformed problems into subproblems, and so on. GPS's innovation was to compute a "difference" between a problem to be solved (represented as a symbol structure) and what was already known or given (also represented as a symbol structure). The program then attempted to reduce this difference by applying some symbol-manipulating "operator" (known to be relevant to this difference) to the initial symbol structure. Newell and Simon called this strategy "means–ends analysis." (Note the similarity to feedback control systems, which continuously attempt to reduce the difference between a current setting and a desired setting.) To do so, it would have to show that the operator's applicability condition was satisfied – a subproblem. The program then started up another version of itself to work on this subproblem, looking for a difference and so on.

For example, suppose the goal is to have Sammy at school when Sammy is known to be at home.[12] GPS computes a "difference," namely, Sammy is in the wrong place, and it finds an operator relevant to reducing this difference, namely, driving Sammy to school. To drive Sammy to school requires that the car be in working order. To make the problem interesting, we'll suppose that

the car's battery is dead, so GPS can't apply the drive-car operator because that operator requires a working battery. Getting a working battery is a subproblem to which GPS can apply a version of itself. This "lower" version of GPS computes a difference, namely, the need for a working battery, and it finds an operator, namely, calling a mechanic to come and install a new battery. To call a mechanic requires having a phone number (and let us suppose we have it), so GPS applies the call-mechanic operator, resulting in the mechanic coming to install a new battery. The lower version of GPS has successfully solved its problem, so the superordinate GPS can now resume – noting that the condition for drive-car, namely, having a working battery, is satisfied. So GPS applies this operator, Sammy gets to school, and the original problem is solved. (This example illustrates the general workings of GPS. A real one using actual symbol structures, differences, and operators with their conditions and so on would be cumbersome but not more revealing.)

When GPS works on subproblems by starting up a new version of itself, it uses a very important idea in computer science (and in mathematics) called "recursion." You might be familiar with the idea that computer programmers organize complex programs hierarchically. That is, main programs fire up subprograms, which might fire-up subsubprograms, and so on. When a main program "calls" a subprogram, the main program suspends itself until the subprogram completes what it is supposed to do (possibly handing back data to the main program), and then the main program resumes work. In AI (and in other applications also), it is common to have a main program call a version of itself – taking care that the new version works on a simpler problem so as to avoid endless repetition and "looping." Having a program call itself is called "recursion."

Do people use subprograms and recursion in their own thinking? Quite possibly, but their ability to recall how to resume what some higher level thought process was doing when that process starts up a chain of lower level processes is certainly limited. I don't believe that GPS attempted to mimic this limitation of human thinking.

Newell and Simon believed that the methods used by GPS could be used to solve a wide variety of different problems, thus giving rise to the term "general." To apply it to a specific problem, a "table of differences" for that problem would have to be supplied. The table would list all the possible differences that might arise and match them to operators, which, for that problem, would reduce the corresponding differences. GPS was, in fact, applied to a number of different logical problems and puzzles[13] and inspired later work in both artificial intelligence and in cognitive science. Its longevity as a problem-solving program itself and as a theory of human problem solving was short, however, and lives on only through its various descendants (about which more will be discussed later).

Heuristic search procedures were used in a number of AI programs developed in the early 1960s. For example, another one of Minsky's Ph.D.

students, James Slagle, programmed a system called SAINT that could solve calculus problems, suitably represented as symbol structures. It solved 52 of 54 problems taken from MIT freshman calculus final examinations.[14] Much use of heuristics was used in programs that could play board games, a subject to which I now turn.

5.4 Game-Playing Programs

I have already mentioned some of the early work of Shannon and of Newell, Shaw, and Simon on programs for playing chess. Playing excellent chess requires intelligence. In fact, Newell, Shaw, and Simon wrote that if "one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavor."[15]

Thinking about programs to play chess goes back at least to Babbage. According to Murray Campbell, an IBM researcher who helped design a world-champion chess-playing program (which I'll mention later), Babbage's 1845 book, *The Life of a Philosopher*, contains the first documented discussion of programming a computer to play chess.[16] Konrad Zuse, the German designer and builder of the Z1 and Z3 computers, used his programming language called Plankalkül to design a chess-playing program in the early 1940s.

In 1946 Turing mentioned the idea of a computer showing "intelligence," with chess-playing as a paradigm.[17] In 1948, Turing and his former undergraduate colleague, D. G. Champernowne, began writing a chess program. In 1952, lacking a computer powerful enough to execute the program, Turing played a game in which he simulated the computer, taking about half an hour per move. (The game was recorded. You can see it at http://www.chessgames.com/perl/chessgame?gid=1356927.) The program lost to a colleague of Turing, Alick Glennie; however, it is said that the program won a game against Champernowne's wife.[18]

After these early programs, work on computer chess programs continued, with off-again–on-again effort, throughout the next several decades. According to John McCarthy, Alexander Kronrod, a Russian AI researcher, said "Chess is the Drosophila of AI" – meaning that it serves, better than more open-ended intellectual tasks do, as a useful laboratory specimen for research. As Minsky said, "It is not that the games and mathematical problems are chosen because they are clear and simple; rather it is that they give us, for the smallest initial structures, the greatest complexity, so that one can engage some really formidable situations after a relatively minimal diversion into programming."[19] Chess presents very difficult problems for AI, and it was not until the mid-1960s that the first competent chess programs appeared. I'll return to discuss these in a subsequent chapter.

More dramatic early success, however, was achieved on the simpler game of checkers (or draughts as the game is known in British English). Arthur Samuel (Fig. 5.5) began thinking about programming a computer to play checkers in the late 1940s at the University of Illinois where he was a Professor of Electrical Engineering. In 1949, he joined IBM's Poughkeepsie Laboratory and completed his first operating checkers program in 1952 on IBM's 701 computer. The program was recoded for the IBM 704 in 1954. According to John McCarthy,[20] "Thomas J. Watson Sr., the founder and President of IBM, remarked that the demonstration [of Samuel's program] would raise the price of IBM stock 15 points. It did."

[Apparently, Samuel was not the first to write a checkers-playing program. According to the *Encyclopedia Brittanica, Online*, "The earliest successful AI program was written in 1951 by Christopher Strachey, later director of the Programming Research Group at the University of Oxford. Strachey's checkers (draughts) program ran on the Ferranti Mark I computer at the University of Manchester, England. By the summer of 1952 this program could play a complete game of checkers at a reasonable speed."][21]

Samuel's main interest in programming a computer to play checkers was to explore how to get a computer to learn. Recognizing the "time consuming and costly procedure[s]" involved in programming, Samuel wrote "Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort."[22] Samuel's efforts were among the first in what was to become a very important part of artificial intelligence, namely, "machine learning." His first program that incorporated learning was completed in 1955 and demonstrated on television on February 24, 1956.

Before describing his learning methods, I'll describe in general how Samuel's program chose moves. The technique is quite similar to how moves were chosen in the eight-puzzle I described earlier. Except now, provision must be made for the fact that the opponent chooses moves also. Again, a tree of symbolic expressions, representing board positions, is constructed. Starting with the initial configuration, all possible moves by the program (under the assumption that the program moves first) are considered. The result is all the possible resulting board configurations branching out from the starting configuration. Then, from each of these, all possible moves of the opponent are considered – resulting in more branches, and so on.

If such a tree could be constructed for an entire game, a winning move could be computed by examination of the tree. Unfortunately, it has been estimated that there are about 5×10^{20} possible checkers positions. A leading expert in programming computers to play games, Jonathan Schaeffer, was able to "solve" checkers (showing that optimal play by both players results in a draw) by time-consuming analysis of around 10^{14} positions. He wrote me that "This was the result of numerous enhancements aimed at focussing the search at the parts of the search space where we were most likely to find what we

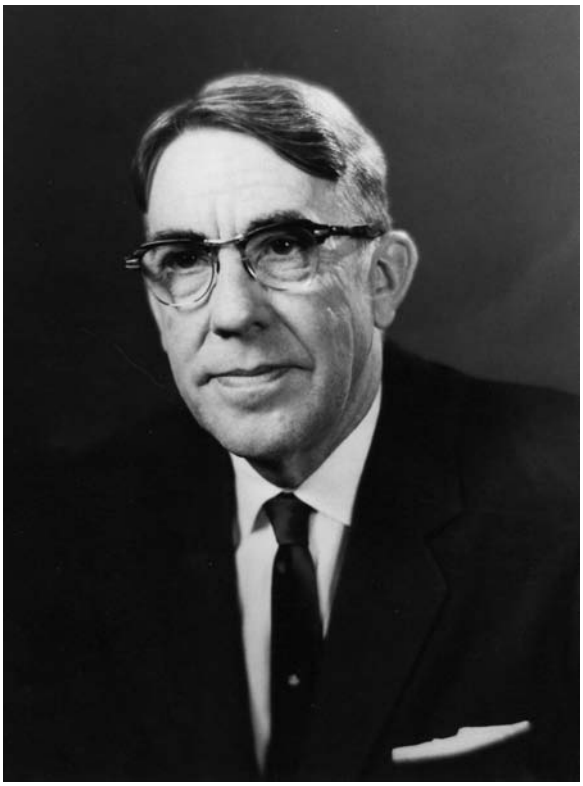


Figure 5.5: Arthur Samuel. (Photograph courtesy of Donna Hussain, Samuel's daughter.)

needed."[23] I'll describe his work in more detail later.

Samuel's program then could necessarily construct only a part of the tree – that is, it could look only a few moves ahead. How far ahead it looked, along various of its branches, depended on a number of factors that need not concern us here. (They involved such matters as whether or not an immediate capture was possible.) Looking ahead about three moves was typical, although some branches might be explored (sparsely) to a depth of as many as ten moves. A diagram from Samuel's paper, shown in Fig. 5.6, gives the general idea. Samuel said that the "actual branchings are much more numerous."

So, how is the program to choose a move from such an incomplete tree? This problem is faced by all game-playing programs, and they all use methods that involve computing a score for the positions at the tips, or "leaves," of the tree (that is, the leaves of the incomplete tree generated by the program) and then "migrating" this score back up to the positions resulting from moves from the current position. First, I will describe how to compute the score, then how

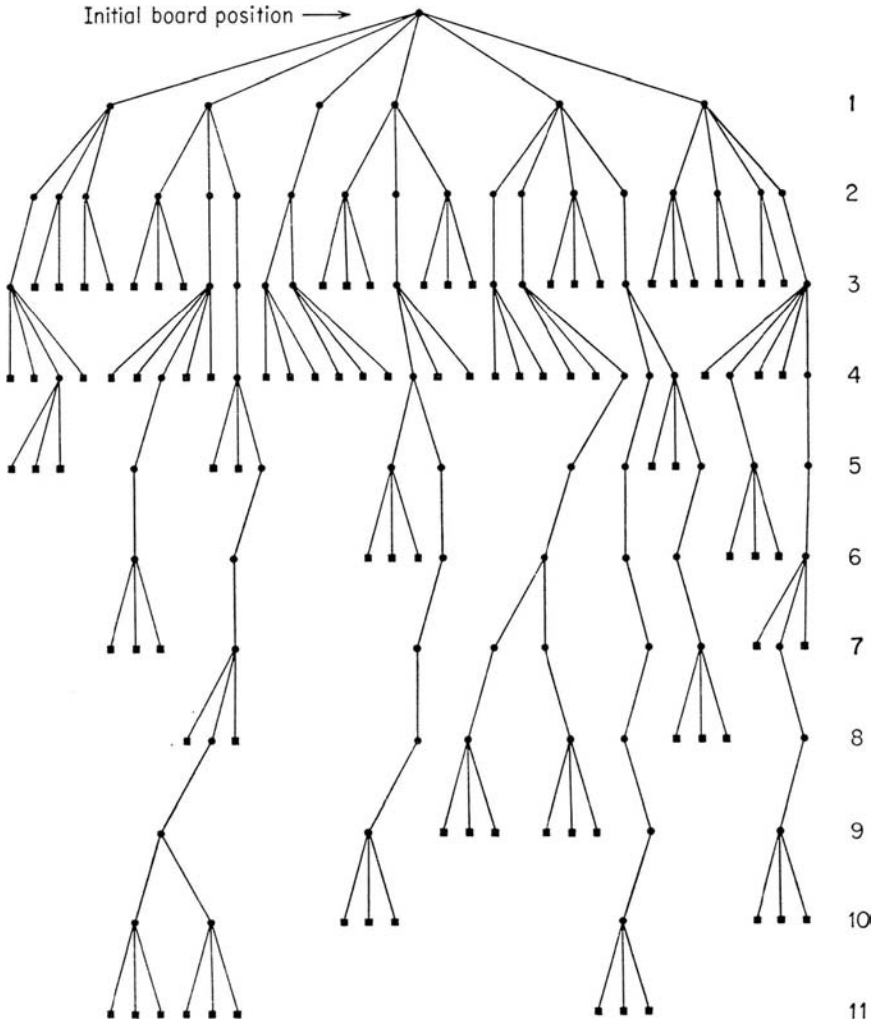


Figure 5.6: An illustrative checkers game tree. (From p. 74 of Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, New York: McGraw Hill, 1963.)

to migrate it back, and then how Samuel used learning methods to improve performance.

Samuel's program first computed the points to be awarded to positions at the leaves of the tree based on their overall "goodness" from the point of view of the program. Among the features contributing points were the relative piece advantage (with kings being worth more than ordinary pieces), the overall "mobility" (freedom to move) of the program's pieces, and center control.

(The program had access to 38 such features but only used the 16 best of these at any one time.) The points contributed by each feature were then multiplied by a "weight" (reflecting the relative importance of its corresponding feature), and the result was summed to give an overall score for a position.

Starting with a position immediately above those at the tip of the tree, if it is a position for which it is the program's turn to move, we can assume that the program would want to move to that position with the highest score, so that highest score is migrated back to this "immediately above" position. If, however, it is a position from which it is the opponent's turn to move, we assume that the opponent would want to move to that position with the lowest score. In that case, the lowest score is migrated back to this immediately above position. This alternately "highest–lowest" migration strategy is continued back all the way up the tree and is called the "minimax" strategy.

[A simple modification of this strategy, called the "alpha–beta" procedure, is used to infer (correctly) from already-migrated scores that certain branches need not be examined at all – thus allowing other branches to be explored more deeply. Opinions differ about who first thought of this important modification. McCarthy and Newell and Simon all claim credit. Samuel told me he used it but that it was too obvious to write about.]

If one assumes that it is the program's turn to move from the current position, and that scores have already been migrated back to the positions just below it, the program would make its move to that position with the highest score. And then the game would continue with the opponent making a move, another stage of tree growth, score computation and migration, and so on until one side wins or loses.

One of the learning methods in Samuel's program adjusted the values of the weights used by the scoring system. (Recall that weight adjustments in Pandemonium and in neural networks were ways in which those systems learned.) The weights were adjusted so that the score of a board position (as computed by the sum of the weighted feature scores) moved closer to the value of its migrated score after finishing a search. For example, if the score of an initial position was computed (using the weights before adjustment) to be 22, and the migrated score of that position after search was 30, then the weights used to compute the score of the initial position were adjusted in a manner so that the new score (using the adjusted value of the weights) was made closer to 30, say 27. (This technique foreshadowed a very important learning method later articulated by Richard Sutton called "temporal-difference learning.") The idea here was that the migrated score, depending as it did on looking ahead in the game, was presumed to be a better estimate than the original score. The estimating procedure was thereby improved so that it produced values more consistent with the "look-ahead" score.

Samuel also used another method called "rote learning" in which the program saved various board positions and their migrated scores encountered

during actual play. Then, at the end of a search, if a leaf position encountered was the same as one of these stored positions, its score was already known (and would not have to be computed using the weights and features.) The known score, based as it was on a previous search, would presumably be a better indicator of position value than would be the computed score.

Samuel's program also benefitted from the use of "book games," which are records of the games of master checkers players. In commenting about Samuel's work, John McCarthy wrote that "checker players have many volumes of annotated games with the good moves distinguished from the bad ones. Samuel's learning program used *Lee's Guide to Checkers*[24] to adjust its criteria for choosing moves so that the program would choose those thought good by checker experts as often as possible."

Samuel's program played very good checkers and, in the summer of 1962, beat Robert Nealey, a blind checkers master from Connecticut. (You can see a game played between Mr. Nealey and Samuel's program at http://www.fierz.ch/samuel.htm.) But, according to Jonathan Schaeffer and Robert Lake, "In 1965, the program played four games each against Walter Hellman and Derek Oldbury (then playing a match for the World Championship), and lost all eight games."[25]

Notes

1. A. Newell and H. A. Simon, "The Logic Theory Machine: A Complex Information Processing System," *Proceedings IRE Transactions on Information Theory*, Vol. IT-2, pp. 61–79, September 1956, and A. Newell, J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," *Proceedings of the 1957 Western Joint Computer Conference*, Institute of Radio Engineers, pp. 218–230, 1957. [**113**]

2. Alfred North Whitehead and Bertrand Russell, *Principia Mathematica*, Vol. 1, Cambridge: Cambridge University Press, 1910. [**113**]

3. Allen Newell and Herbert A. Simon, "Computer Science as Empirical Inquiry: Symbols and Search," *Communications of the ACM*, Vol. 19, No. 3, pp. 113–126, March 1976. [**114**]

4. Claude E. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*, Ser. 7, Vol. 41, No. 314, March 1950. Text available online at http://www.pi.infn.it/~carosi/chess/shannon.txt. (The paper was first presented in March 1950 at the National Institute for Radio Engineers Convention in New York.) [**117**]

5. Pamela McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, p. 106, San Francisco: W. H. Freeman and Co., 1979. [**117**]

6. John McCarthy, "A Tough Nut for Theorem Provers," Stanford Artificial Intelligence Project Memo No. 16, July 17, 1964; available online at http://www-formal.stanford.edu/jmc/toughnut.pdf. [**117**]

7. Saul Amarel, "On Representations of Problems of Reasoning About Actions," in Donald Michie (ed.), *Machine Intelligence 3*, pp. 131–171, Edinburgh: Edinburgh University Press, 1968. [**118**]

8. Herbert Gelernter, "Realization of a Geometry-Theorem Proving Machine," *Proceedings of the International Conference on Information Processing"*, pp. 273–282, Paris: UNESCO House, Munich: R. Oldenbourg, and London: Butterworths, 1960. Also in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, pp. 134–152, New York: McGraw Hill, 1963. [**118**]

9. From http://www.math.niu.edu/~rusin/known-math/99/minsky. [**119**]

10. Allen Newell, J. C. Shaw, and Herbert A. Simon, "Report on a General Problem-Solving Program," *Proceedings of the International Conference on Information Processing"*, pp. 256–264, Paris: UNESCO House, Munich: R. Oldenbourg, and London: Butterworths, 1960. [**121**]

11. For more about GPS as a theory and explanation for human problem solving, see Allen Newell and Herbert Simon, "GPS, a Program That Simulates Human Thought," in H. Billings (ed.), *Lernende Automaten*, pp. 109–124, Munich: R. Oldenbourg KG, 1961. Reprinted in *Computers and Thought*, pp. 279–293. [**121**]

12. I adapt an example from
http://www.math.grinnell.edu/~stone/events/scheme-workshop/gps.html. [**121**]

13. See George Ernst and Allen Newell, *GPS: A Case Study in Generality and Problem Solving*, New York: Academic Press, 1969. [**122**]

14. James R. Slagle, "A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus," Ph.D. dissertation, MIT, May 1961. For an article about SAINT, see James R. Slagle, "A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus," *Journal of the ACM*, Vol. 10, No. 4, pp. 507–520, October 1963. [**123**]

15. Allen Newell, J. Shaw, and Herbert Simon, "Chess-Playing Programs and the Problem of Complexity," *IBM Journal of Research and Development*, Vol. 2, pp. 320–335, October 1958. [**123**]

16. Chapter 5 of *Hal's Legacy: 2001's Computer as Dream and Reality*, David G. Stork (ed.), Cambridge, MA: MIT Press, 1996. See the Web site at
http://mitpress.mit.edu/e-books/Hal/chap5/five3.html. [**123**]

17. Andrew Hodges, "Alan Turing and the Turing Test," in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, Robert Epstein, Gary Roberts, and Grace Beber (ed.), Dordrecht, The Netherlands: Kluwer, 2009. See A. M. Turing, "Proposed Electronic Calculator," report for National Physical Laboratory, 1946, in *A. M. Turing's ACE Report of 1946 and Other Papers*, B. E. Carpenter and R. W. Doran (eds.), Cambridge, MA: MIT Press, 1986. [**123**]

18. http://en.wikipedia.org/wiki/Alan_Turing. [**123**]

19. Marvin Minsky (ed.), "Introduction," *Semantic Information Processing*, p. 12, Cambridge, MA: MIT Press, 1968. [**123**]

20. From a Web retrospective at
http://www-db.stanford.edu/pub/voy/museum/samuel.html. [**124**]

21. See Christopher Strachey, "Logical or Non-mathematical Programmes," *Proceedings of the 1952 ACM National Meeting (Toronto)*, pp. 46–49, 1952. [**124**]

22. Arthur L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 210–229, 1959. Reprinted in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, p. 71, New York: McGraw Hill, 1963. [**124**]

23. E-mail of February 14, 2009. [**125**]

24. John W. Dawson, *Lee's Guide to the Game of Draughts*, Revised Edition, London: E. Marlborough, 1947. [**128**]

25. Jonathan Schaeffer and Robert Lake, "Solving the Game of Checkers," *Games of No Chance*, pp. 119–133, MSRI Publications, Vol. 29, 1996. (Available online at http://www.msri.org/communications/books/Book29/files/schaeffer.pdf.) [**128**]

Chapter 6

Semantic Representations

The computer programs I have described so far performed transformations on relatively simple symbol structures, which were all that were required for the mathematical problems, puzzles, and games that these programs dealt with. The main effort was in coming up with and using problem-specific heuristics (such as features to be used in computing the value of a checkers position, for example) to limit the number of transformations of these structures in searches for solutions. As Minsky put it, "The most central idea of the pre-1962 period was that of finding heuristic devices to control the breadth of a *trial-and-error search*."[1] In the early 1960s, several Ph.D. research projects, some performed under Minsky's direction at MIT, began to employ more complex symbol structures in programs for performing various intellectual tasks. Because of their rich, articulated content of information about their problem topic, these structures were called *semantic representations*.[2] As Minsky wrote, "Within the small domain in which each program operates, the performance [of these programs] is not too bad compared with some human activities. . . . But much more important than what these particular experiments achieve are the methods they use to achieve what they do, *for each is a first trial of previously untested ideas*."[3] I'll describe some examples of these sorts of projects and the new methods that they employed.

6.1 Solving Geometric Analogy Problems

Thomas G. Evans (1934–) programmed a system that was able to perform well on some standard geometric analogy tests. It was apparently the largest program written up to that time in John McCarthy's new programming language, LISP (which I'll describe later). I quote from an article based on Evans's 1963 dissertation, which presented this work:[4]

> We shall be considering the solution by machine of so-called "geometric-analogy" intelligence-test questions. Each member of this class of problems consists of a set of labeled line drawings. The task to be performed can be described by the question: "Figure A is to Figure B as Figure C is to which of the following figures?" For example [in Fig. 6.1] it seems safe to say that most people would agree with the program we are about to describe, in choosing [number 4] as the desired answer.

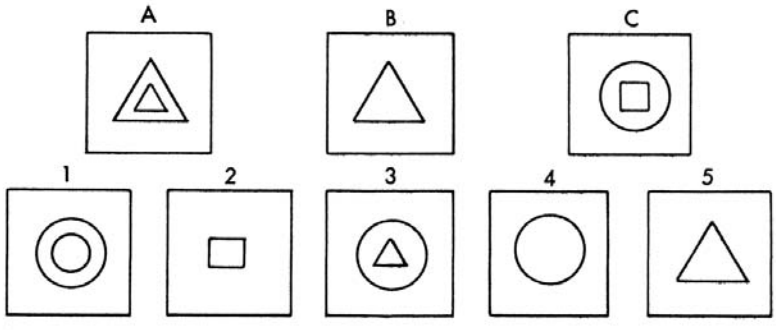
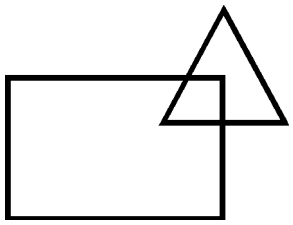


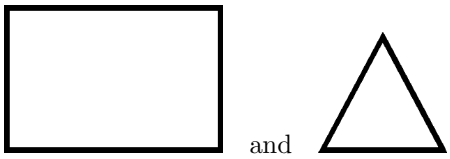
Figure 6.1: An analogy problem.

He further noted that "problems of this type are widely regarded as requiring a high degree of intelligence for their solution and in fact are used as a touchstone of intelligence in some general intelligence tests used for college admission and other purposes." So, again, AI research concentrated on mechanizing tasks requiring human intelligence.

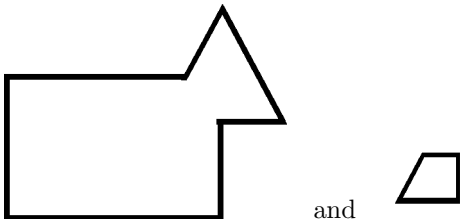
Evans's program first transformed the diagrams presented to it so that they revealed how they were composed out of parts. He called these "articular" representations. Of the possibly several decompositions possible, the one chosen by the program depended on its "context." (This choice is one example of a heuristic used by the program.) For example, the diagram



could either be decomposed into

 and

or into

 and

But if the analogy problem contained another diagram (part of the context):

 then the first decomposition would be chosen.

Evans represented diagrams and their parts as complex symbol structures consisting of rather elaborate combinations of lists and lists of lists whose elements indicated which parts were inside or outside (or above or below) which other parts, and so on. Those details need not concern us here, but they did allow Evans to specify "rules" for his program that could be used to show how one diagram could be transformed into another. The program was able to infer which combinations of these rules transformed Figure A of a given problem into Figure B. Then it could apply this transformation to Figure C. If one of the multiple-choice answers resulted, it would give that one as its answer. Otherwise, the program "weakened" the transformation just enough so that one of the answers was produced, and that would be the program's answer.

Evans summarized his results as follows:

> Allowing ourselves only [the parts of the program actually implemented], our estimate would be that of the 30 geometric-analogy problems on a typical edition of the ACE tests, [the program] can successfully solve at least 15 and possibly as

many as 20 problems.

He notes that this level of performance compares favorably with the average high school student.

6.2 Storing Information and Answering Questions

Another of Minsky's Ph.D. students during the early 1960s, Bertram Raphael (1936–), focused on the problem of "machine understanding." In his dissertation,[5] Raphael explained that

> a computer should be considered able to "understand" if it can converse intelligently, i.e., if it can remember what it is told, answer questions, and make responses which a human observer considers reasonable.

Raphael wanted to be able to tell things to a computer and then ask it questions whose answers could be deduced from the things it had been told. (The telling and asking were to be accomplished by typing sentences and queries.) Here are some examples of the kinds of things he wanted to tell it:

> Every boy is a person.
>
> A finger is part of a hand.
>
> There are two hands on each person.
>
> John is a boy.
>
> Every hand has five fingers.

Given this information, Raphael would want his system to be able to deduce the answer to the question "How many fingers does John have?"

Because Raphael wanted his system to communicate with people, he wanted its input and output languages to be "reasonably close to natural English." He recognized that "the linguistic problem of transforming natural language input into a usable form will have to be solved before we obtain a general semantic information retrieval system." This "linguistic problem" is quite difficult and still not "solved" even though much progress has been made since the 1960s. Raphael used various "devices" (as he called them and which are not germane to our present discussion) to "bypass [the general problem of dealing with natural language] while still utilizing understandable English-like input and output."

The main problem that Raphael attacked was how to organize facts in the computer's memory so that the relevant deductions could be made. As Raphael put it, "The most important prerequisite for the ability to 'understand' is a suitable internal representation, or model, for stored information. The model should be structured so that information relevant for question-answering is easily accessible."[6]

Raphael called his system SIR, for Semantic Information Retrieval, (which he programmed in LISP). He used the word "semantic" because SIR modeled sentences in a way dependent on their meanings. The sentences that SIR could deal with involved "entities" (such as John, boy, hand, finger, and so on) and relations among these entities (such as "set-membership," "part–whole," "ownership," "above," "beside," and other spatial relationships.) The model, then, had to have ways for representing entities and the relationships among them.

Entities such as John and boy were represented by the LISP computer words `JOHN` and `BOY`, respectively. (Of course, the computer had no way of knowing that the computer word `JOHN` had anything to do with the person John. Raphael could have just as well represented John in the computer by `X13F27` so long as he used that representation consistently for John. Using the computer word `JOHN` was a mnemonic convenience for the programmer – not for the computer!) When representing the fact that John is a boy, SIR would "link" a computer expression `(SUPER-SET JOHN BOY)` to the expression `JOHN` and link a computer expression `(SUB-SET BOY JOHN)` to the expression `BOY`. Thus, if SIR were asked to name a boy, it could reply "`JOHN`" by referring to `BOY` in its model, looking at its `SUB-SET` link and retrieving `JOHN`. (I have simplified the representations somewhat to get the main ideas across; SIR's actual representations were a bit more complicated.)

SIR could deal with dozens of different entities and relations among them. Every time it was told new information, it would add new entities and links as needed. It also had several mechanisms for making logical deductions and for doing simple arithmetic. The very structure of the model facilitated many of its deductions because, as Minsky pointed out in his discussion of Raphael's thesis, "the direct predicate-links... almost physically chain together the immediate logical consequences of the given information."[7]

SIR was also the first AI system to use the "exception principle" in reasoning. This principle is best explained by quoting directly from Raphael's thesis:

> General information about "all the elements" of a set is considered to apply to particular elements only in the absence of more specific information about those elements. Thus it is not necessarily contradictory to learn that "mammals are land animals" and yet "a whale is a mammal which always lives in water." In the program, this idea is implemented by always referring for desired information

> to the property-list [that is, links] of the individual concerned *before* looking at the descriptions of sets to which the individual belongs.
>
> The justification for this departure from the no-exception principles of Aristotelian logic is that this precedence of specific facts over background knowledge seems to be the way people operate, and I wish the computer to communicate with people as naturally as possible.
>
> The present program does not experience the uncomfortable feeling people frequently get when they must face facts like "a whale is a mammal which lives in water although mammals as a rule live on land."

The exception principle was studied by AI researchers in much more detail later and led to what is called default reasoning and nonmonotonic logics, as we shall see.

6.3 Semantic Networks

It is instructive to think of SIR's representational scheme in terms of a network. The entities (such as JOHN and BOY) are the "nodes" of the network, and the relational links (such as SUB-SET) are the connections between nodes. SIR was an early version of what would become an important representational idea in artificial intelligence, namely, *semantic networks.* It was not the first, however. John Sowa, who has written extensively about semantic networks, claims that the "oldest known semantic network was drawn in the 3rd century AD by the Greek philosopher Porphyry in his commentary on Aristotle's categories."[8] In 1961 Margaret Masterman (1910–1986), Director of the Cambridge Language Research Unit, used a semantic network in a translation system in which concepts were ordered in a hierarchy.[9]

M. Ross Quillian, a student of Herb Simon's at the Carnegie Institute of Technology, was interested, along with Newell and Simon, in computational models of human mental processes, specifically memory organization. He developed a memory model consisting of a semantic network of nodes representing English words. The nodes were interconnected by what he called "associative links." In Quillian's words, "In the memory model, ingredients used to build up a concept are represented by the token nodes naming other concepts, while the configurational meaning of the concept is represented by the particular structure of interlinkages connecting those token nodes to each other."

Quillian goes on to write that "[t]he central question asked in this research has been: What constitutes a reasonable view of how semantic information is organized within a person's memory? In other words: What

sort of representational format can permit the 'meanings' of words to be stored, so that humanlike use of these meanings is possible?"[10]

I can illustrate how Quillian's network format represents meaning by using one of his examples. Consider the different meanings of the word "plant." One such meaning is given by linking the node `PLANT` to other nodes, such as `LIVE`, `LEAF`, `FOOD`, `AIR`, `WATER`, and `EARTH`, through connections that represent that a plant (according to this meaning of the word) is alive, has leaves, and gets its food from air, water, and earth. Another meaning of "plant" links `PLANT` to other nodes, such as `PEOPLE`, `PROCESS`, and `INDUSTRY`, through connections that represent that a plant (according to this other meaning of the word) is an apparatus that uses people for engaging in a process used in industry.

According to Quillian, the meaning of a term is represented by its place in the network and how it is connected to other terms. This same idea is used in dictionaries where the meaning of a word is given by mentioning the relationship of this word to other words. The meanings of those other words are, in turn, given by their relationships to yet other words. So we can think of a dictionary as being like a large semantic network of words linked to other words.

By using this view, the *full* meaning of a concept can be quite extensive. As Quillian puts it,

> Suppose that a person were asked to state everything he knows about the concept "machine." ... This information will start off with the more "compelling" facts about machines, such as that they are usually man-made, involve moving parts, and so on, and will proceed "down" to less and less inclusive facts, such as the fact that typewriters are machines, and then eventually will get to much more remote information about machines, such as the fact that a typewriter has a stop which prevents its carriage from flying off each time it is returned. We are suggesting that this information can all usefully be viewed as part of the subject's concept of "machine."

In what way is Quillian's network a model of human memory organization? Quillian explored two capabilities of human memory modeled by his network. One was comparing and contrasting two different words. Quillian proposed that this be done by a process that came to be called "spreading activation." Conceptually, one starts at the nodes representing the two words and gradually traverses the links emanating from them, "activating" the nodes along the way. This process continues until the two "waves" of activation intersect, thus producing a "path" between the two original nodes. Quillian proposed that the total "distance" along this path between the two words could be used as a measure of their similarity. The path can be used to produce an account comparing the two words. (Quillian's program had mechanisms for expressing this account in simple English.)

To use one of Quillian's examples, suppose we wanted to compare the words "cry" and "comfort." The spreading activations would intersect at the word "sad," and the English account would express something like "to cry is to make a sad sound, and to comfort is to make something less sad."

Quillian was also interested in how the network could be used to "disambiguate" two possible uses of the same word. Consider, for example, the sentence "After the strike, the president sent him away." The network can encode different meanings of the word "strike." One such might involve a labor dispute, another might involve baseball, and yet another involve a raid by military aircraft. Which of these meanings is intended by the sentence? Presumably, activation proceeding outward from the word "president" would eventually reach concepts having to do with labor disputes before reaching concepts having to do with baseball or the military. Thus, the "labor dispute" meaning would be preferred because it is "closer," given that the word "president" is in the sentence. In contrast, a different conclusion would be reached for the sentence "After the strike, the umpire sent him away."

Quillian's model differs from some later semantic networks in that it does not have a predetermined hierarchy of superclasses and subclasses. As Quillian puts it, "every word is the patriarch of its own separate hierarchy *when some search process starts with it.* Similarly, every word lies at various places down within the hierarchies of (i.e., is an ingredient in) a great many other word concepts, when processing starts with them."

Notes

1. Marvin Minsky (ed.), "Introduction," *Semantic Information Processing*, p. 9, Cambridge, MA: MIT Press, 1968. [**131**]

2. It might be argued that the diagram used by Gelernter's geometry program was an earlier use of a semantic representation. [**131**]

3. Marvin Minsky, *op. cit.*, p. 1. [**131**]

4. Thomas G. Evans, "A Program for the Solution of a Class of Geometric-Analogy Intelligence-Test Questions," in Marvin L. Minsky, *op. cit.*, p. 271. [**131**]

5. Bertram Raphael, "SIR: Semantic Information Retrieval," in Marvin Minsky, *op. cit.*, pp. 33–145. (This is a partial reprint of his 1964 Ph.D. dissertation.) [**134**]

6. Marvin Minsky, *op. cit.*, p. 35. [**135**]

7. Marvin Minsky, *op. cit.*, p. 17. [**135**]

8. From an article by John F. Sowa at http://www.jfsowa.com/pubs/semnet.htm. (This is a revised and extended version of one that was originally written for the *Encyclopedia of Artificial Intelligence*, edited by Stuart C. Shapiro, Wiley, 1987, second edition, 1992.) [**136**]

9. Margaret Masterman, "Semantic Message Detection for Machine Translation, Using an Interlingua," in *Proceedings of the 1961 International Conference on Machine Translation of Languages and Applied Language Analysis*, pp. 438–475, London: Her Majesty's Stationery Office, 1962. [**136**]

10. M. Ross Quillian, "Semantic Memory," Ph.D. dissertation, Carnegie Institute of Technology (now Carnegie Mellon University), October 1966. (This work also appears as Report AFCRL-66-189 and is partially reprinted in M. Minsky (ed.), *Semantic Information Processing*, pp. 216–270, Cambridge, MA: MIT Press, 1968.) [**137**]

Chapter 7

Natural Language Processing

Beyond pattern recognition of individual alphanumeric characters, whether they be of fixed font or handwritten, lies the problem of understanding strings of characters that form words, sentences, or larger assemblages of text in a "natural" language, such as English. To distinguish languages such as English from the languages used by computers, the former are usually called "natural languages." In artificial intelligence, "understanding" natural language input usually means either converting it to some kind of memory model (such as the one used by Raphael in his SIR system or the semantic network used by Quillian) or the evocation of some action appropriate to the input.

Natural languages are spoken as well as written. And, because speech sounds are not as well segmented as are the characters printed on a page, speech understanding presents additional difficulties, which I'll describe in a later chapter.

The inverse of understanding natural language input is generating natural language output – both written and spoken. Translating from one language to another involves both understanding and generation. So does carrying on a conversation. All of these problems – understanding, generation, translation, and conversing – fall under the general heading of "natural language processing" (sometimes abbreviated as NLP).

7.1 Linguistic Levels

Linguists and others who study language recognize several levels at which language can be analyzed. These levels can be arranged in a sort of hierarchy, starting with those dealing with the most basic components of language

(sounds and word parts) and proceeding upward to levels dealing with sequences of sentences. If speech is being dealt with, there are the levels of *phonetics* (language sounds) and *phonology* (organization of sounds into words). For both speech and text, *morphology* deals with how whole words are put together from smaller parts. For example, "walking" consists of "walk" plus "-ing."

Next, *syntax* is concerned with sentence structure and grammar. It attempts to describe rules by which a string of words in a certain language can be labeled either grammatical or not. For example, the string "John hit the ball" is grammatical but the string "ball the hit John" is not. Together with the dictionary definitions of words, syntax comes next in importance for understanding the meaning of a sentence. For example, the sentence "John saw the man with a telescope" has two different meanings depending on its syntactic structure (that is, depending on whether "with a telescope" refers to "the man" who had a telescope or to "saw.")

But grammaticality alone is insufficient for determining meaning. For example, the sentence "Colorless green ideas sleep furiously" might be considered grammatical, but it is nonsensical. The *semantics* level helps to determine the meaning (or the meaninglessness) of a sentence by employing logical analyses. For example, through semantic analysis, an "idea" can't be both "colorless" and "green."

Next comes the *pragmatics* level, which considers the context of a sentence to pin down meaning. For example, "John went to the bank" would have a different meaning in a sentence about stream fishing than it would in a sentence about commerce. Pragmatics deals with meanings in the context of specific situations.

One of these levels in particular, namely, syntax, was the subject of much early study and continues to be an important aspect of NLP. In 1957, the American linguist Noam Chomsky published a ground-breaking book titled *Syntactic Structures* in which he proposed sets of grammatical rules that could be used for generating the "legal" sentences of a language.[1] The same rules could also be used to analyze a string of words to determine whether or not they formed a legal sentence of the language. I'll illustrate how this analysis is done using what Chomsky called a *phrase-structure grammar* (PSG).[2] The process is very similar to how we all "diagrammed" sentences back in grade school.

Grammars are defined by stating rules for replacing words in the string by symbols corresponding to syntactic categories, such as noun or verb or adjective. Grammars also have rules for replacing strings of these syntactic symbols by additional symbols. To illustrate these ideas, I'll use a very simple grammar adapted from one of Chomsky's examples. This grammar has only three syntactic categories: determiner, noun, and verb. Those three are sufficient for analysing strings such as "the man hit the ball."

One of the rules in this illustrative grammar states that we can replace either of the words "the" or "a" by the symbol "DET" (for determiner). Linguists write this rule as follows:

> the | a → DET
> (The symbol | is used to indicate that *either* of the words that surround it can be replaced by the syntactic symbol to the right of the arrow.)

Here are some other rules, written in the same format:

> man | ball | john → N
> (The words "man," "ball," and "john" can be replaced by the symbol "N" for noun.)
>
> hit | took | threw → V
> (The words "hit," "took," and "threw" can be replaced by the symbol "V" for verb.)
>
> DET N → NP
> (The string of symbols "DET" and "N" can be replaced by the symbol "NP" for noun phrase.)
>
> V NP → VP
> (The string of symbols "V" and "NP" can be replaced by the symbol "VP" for verb phrase.)
>
> NP VP → S
> (The string of symbols "NP" and "VP" can be replaced by the symbol "S" for sentence.)

Symbols such as "S," "DET," "NP," and so on are called the "nonterminal" symbols of the language defined by the grammar, whereas vocabulary words such as "ball," "john," and "threw" are the "terminal" symbols of the language.

We can apply these rules to the string "the man hit the ball" to transform it into "S." Any string that can be changed into "S" in this way is said to be grammatical – a legal sentence in the language defined by this very simple grammar. One way to illustrate the rule applications, called a *parse tree*, is shown in Fig. 7.1.[3]

This example was based on a small set of syntactic categories and replacement rules just to illustrate the main ideas about syntactic analysis. To make the grammar slightly more realistic, we would need to include symbols and replacement rules for adjectives, adverbs, prepositions, and so on. And, of course, we would have to include many more vocabulary words.

Grammars are called *context-free grammars* (CFGs) if all of their rules have just a single nonterminal symbol on the right side of the arrow. They are

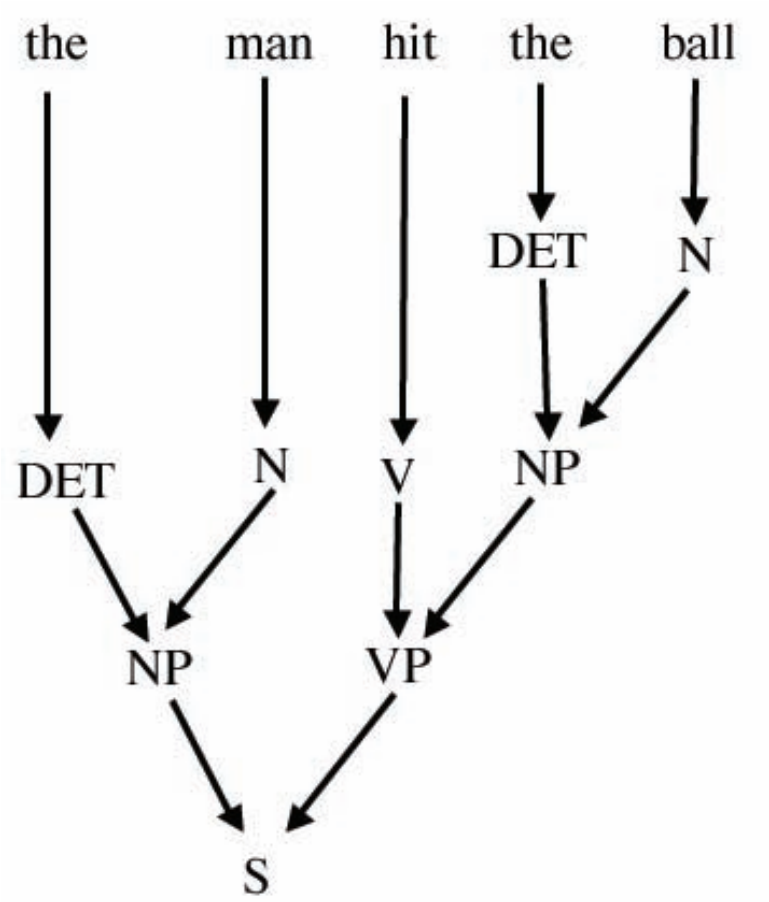


Figure 7.1: A parse tree for analyzing a sentence.

called that because when the rules are used in reverse (to generate rather than to analyze grammatical sentences), the way in which a nonterminal symbol is replaced does not depend on the presence of any other symbols. PSGs are context free.

The diagram in Fig. 7.2 shows how the rules of our simple grammar can be used to generate sentences. In this case, it starts with the symbol for sentence, namely, "S," and generates the sentence "John threw the ball."

This simple grammar certainly can't generate all of the sentences we would claim to be legal or acceptable. It also generates sentences that we would not ordinarily want to accept, such as "the john threw the ball." Chomsky's book presents much more complex grammars, and later work has

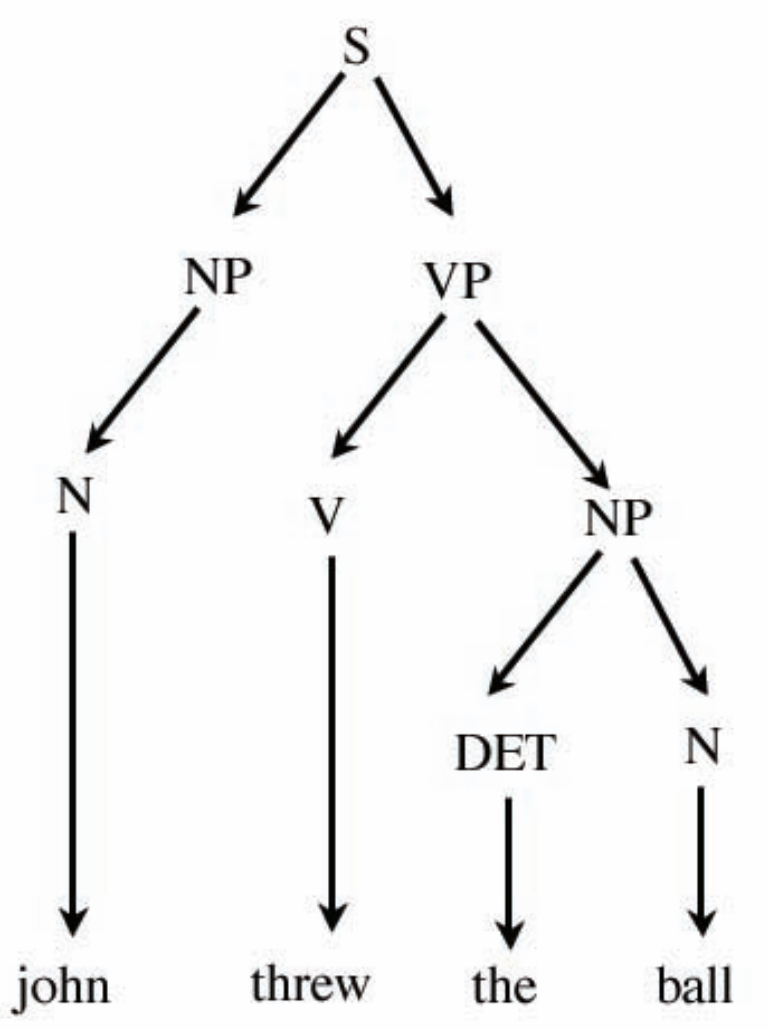


Figure 7.2: A parse tree for generating a sentence.

produced quite elaborate ones. By the early 1960s, several grammars had been encoded in computer programs that could parse samples of English text.[4] I'll be mentioning several different grammars, some more complex than CFGs in succeeding chapters. Nevertheless, even the most complex grammars can't cleanly distinguish between sentences we would accept as grammatically correct and those we would not. I will return to this difficulty and one way to deal with it in a later chapter.

The way a sentence is parsed by a grammar can determine its meaning, so

an important part of natural language processing involves using the grammar rules to find acceptable parse trees for sentences. Finding a parse tree involves search – either for the several different ways that the nonterminal symbols, beginning with "S," can be replaced using grammar rules in an attempt to match a target sentence or for the several different ways the words in a target sentence can be replaced by nonterminal symbols in an attempt to produce the symbol "S." The first of these kinds of searches is called "top-down" (from "S" to a sentence); the second is called "bottom-up" (from a sentence to "S").

It is often (if not usually) the case that, given a grammar, sentences can have more than one parse tree, each with a different meaning. For example, "the man hit the ball in the park" could have a parse tree in which "in the park" is part of a verb phrase along with "hit" or a parse tree in which "in the park" is part of a noun phrase along with "ball." Moreover, as I have already mentioned, some parsings of sentences might be meaningless. For example, according to my simple grammar, "the ball threw the man" is a legal but probably meaningless sentence. Deciding which parse tree is appropriate is part of the process of deciding on meaning and is a job for the semantics (and possibly even the pragmatics) level. During the late 1950s and throughout most of the 1960s and beyond, syntactic analysis was more highly developed than was semantics.

Semantic analysis usually involves using the parse tree to guide the transformation of the input sentence into an expression in some well-defined "meaning representation language" or into a program that responds in the appropriate way to the input sentence. For example, "the man threw the ball" might be transformed into a logical expression such as

$$(\exists \mathtt{x}, \mathtt{y}, \mathtt{z})[\mathtt{Past}(\mathtt{z}) \wedge \mathtt{Man}(\mathtt{x}, \mathtt{z}) \wedge \mathtt{Ball}(\mathtt{y}, \mathtt{z}) \wedge \mathtt{Event}(\mathtt{z}) \wedge \mathtt{Throws}(\mathtt{x}, \mathtt{y}, \mathtt{z})],$$

which can be interpreted as "there are x, y, and z, such that z is an event that occurred in the past, x is a man in that event, y is a ball in that event, and x throws y in that event."

Semantic analysis might also transform the sentence "the man threw the ball" into a program that, in some way, simulates a man throwing the ball in the past.

7.2 Machine Translation

Some of the first attempts to use computers for more than the usual numerical calculations were in automatic translation of sentences in one language into sentences of another. Word dictionaries could be stored in computer memory (either on tapes or on punched cards), and these could be used to find English equivalents for foreign words. It was thought that selecting an appropriate equivalent for each foreign word in a sentence, together with a modest amount

of syntactic analysis, could be used to translate a sentence in a foreign language (Russian, for example) into English.

Reporting about a new computer[5] being developed by a team led by Harry D. Huskey at the National Bureau of Standards (now called the National Institute of Standards and Technology), the *New York Times* reported the following on May 31, 1949:[6]

> A new type of "electric brain" calculating machine capable not only of performing complex mathematical problems but even of translating foreign languages, is under construction here at the United States Bureau of Standards Laboratory at the University of California's Institute of Numerical Analysis. While the exact scope the machine will have in the translating field has not been decided, the scientists working on it say it would be quite possible to make it encompass the 60,000 words of the Webster Collegiate Dictionary with equivalents for each word in as many as three foreign languages.

Explaining how the machine might do translation, the *Times* reporter wrote

> When a foreign word for translation is fed into the machine, in the form of an electro-mathematical symbol on a tape or card, the machine will run through its "memory" and if it finds that symbol as record, will automatically emit a predetermined equivalent – the English word.
>
> . . .
>
> This admittedly will amount to a crude word-for-word translation, lacking syntax, but will nevertheless be extremely valuable, the designers say, for such purposes as scientists' translations of foreign technical papers in which vocabulary is far more of a problem than syntax.

The machine had not actually performed any translations – the idea of doing so was still just a possibility envisioned by Huskey. But even nonscientists could imagine the difficulties. An editorial in the *New York Times* the next day put the problem well:

> We have our misgivings about the accuracy of every translation. How is the machine to decide if the French word "pont" is to be translated as "bridge" or "deck" or to know that "operation" in German means a surgical operation? All the machine can do is to simplify the task of looking up words in a dictionary and setting down their English equivalents on a tape, so that the translator

> still has to frame the proper sentences and give the words their contextual meaning.

In a 1947 letter to Norbert Wiener, Warren Weaver, a mathematician and science administrator, mentioned the possibility of using digital computers to translate documents between natural human languages. Wiener was doubtful about this possibility. In his reply to Weaver, Wiener wrote "I frankly am afraid the boundaries of words in different languages are too vague and the emotional and international connotations are too extensive to make any quasi-mechanical translation scheme very hopeful." Nevertheless, by July 1949, Weaver had elaborated his ideas into a memorandum, titled "Translation" that he sent to several colleagues.

Weaver began his memorandum by stating the following:

> There is no need to do more than mention the obvious fact that a multiplicity of languages impedes cultural interchange between the peoples of the earth, and is a serious deterrent to international understanding. The present memorandum, assuming the validity and importance of this fact, contains some comments and suggestions bearing on the possibility of contributing at least something to the solution of the world-wide translation problem through the use of electronic computers of great capacity, flexibility, and speed.

According to the editors of the published volume[7] in which the memorandum was reprinted, "When he sent it to some 200 of his acquaintances in various fields, it was literally the first suggestion that most had ever seen that language translation by computer techniques might be possible." Weaver's document is often credited with initiating the field of machine translation (often abbreviated as MT).[8]

In June 1952 at MIT, Yehoshua Bar-Hillel (1915–1975), an Israeli logician who was then at MIT's Research Laboratory for Electronics, organized the first conference devoted to machine translation.[9] Originally optimistic about the possibilities, Bar-Hillel was later to conclude that full automatic translation was impossible.

In January 1954, automatic translation of samples of Russian text to English was demonstrated at IBM World Headquarters, 57th Street and Madison Avenue, New York City. The demonstration, using a small vocabulary and limited grammar, was the result of a collaboration between IBM and Georgetown University. The project was headed by Cuthbert Hurd, director of the Applied Sciences Division at IBM, and Léon Dostert of Georgetown. According to an IBM press release[10] on January 8, 1954,

> Russian was translated into English by an electronic "brain" today for the first time.

> Brief statements about politics, law, mathematics, chemistry, metallurgy, communications and military affairs were submitted in Russian by linguists of the Georgetown University Institute of Languages and Linguistics to the famous 701 computer of the International Business Machines Corporation. And the giant computer, within a few seconds, turned the sentences into easily readable English.
>
> A girl who didn't understand a word of the language of the Soviets punched out the Russian messages on IBM cards. The "brain" dashed off its English translations on an automatic printer at the breakneck speed of two and a half lines per second.
>
> "Mi pyeryedayem mislyi posryedstvom ryechyi," the girl punched. And the 701 responded: "We transmit thoughts by means of speech."
>
> "Vyelyichyina ugla opryedyelyayetsya otnoshyenyiyem dlyini dugi k radyiusu," the punch rattled. The "brain" came back: "Magnitude of angle is determined by the relation of length of arc to radius."

Although the demonstration caused a great deal of excitement and led to increased funding for translation research, subsequent work in the field was disappointing.[11] Evaluating MT work in a 1959 report circulated among researchers, Bar-Hillel had become convinced that fully automatic, high-quality translation (which he dubbed FAHQT) was not feasible "not only in the near future but altogether." His expanded report appeared in a 1960 paper that enjoyed wide distribution.[12]

One of the factors leading Bar-Hillel to his negative conclusions was the apparent difficulty of giving computers the "world knowledge" they would need for high-quality translation. He illustrated the problem with the following story:

> Little John was looking for his toy box. Finally he found it. The box was in the pen. John was very happy.

How should one translate "The box was in the pen"? Bar-Hillel argued that even if there were only two definitions of "pen" (a writing utensil and an enclosure where small children play), a computer knowing only those definitions would have no way of deciding which meaning was intended. In addition to its knowledge of vocabulary and syntax, a translating computer would need to know "the relative sizes of pens, in the sense of writing implements, toy boxes, and pens, in the sense of playpens." Such knowledge, Bar-Hillel claimed, was not at the disposal of the electronic computer. He said that giving a computer such encyclopedic knowledge was "utterly chimerical and hardly deserves any further discussion."

As later researchers would finally concede, Bar-Hillel was right about his claim that highly competent natural language processing systems (indeed, broadly competent AI systems in general) would need to have encyclopedic knowledge. However, most AI researchers would disagree with him about the futility of attempting to give computers the required encyclopedic knowledge. Bar-Hillel was well known for being a bit of a nay-sayer regarding artificial intelligence. (Commenting on John McCarthy's "Programs with Common Sense" paper at the 1958 Teddington Conference, Bar-Hillel said "Dr. McCarthy's paper belongs in the Journal of Half-Baked Ideas, the creation of which was recently proposed by Dr. I. J. Good.")[13]

In April 1964, the National Academy of Sciences formed the Automatic Language Processing Advisory Committee (ALPAC), with John R. Pierce (1910–2002) of Bell Laboratories as chair, to "advise the Department of Defense, the Central Intelligence Agency, and the National Science Foundation on research and development in the general field of mechanical translation of foreign languages." The committee issued its report in August 1965 and concluded, among other things, that "...there is no immediate or predictable prospect of useful machine translation."[14] They recommended support for basic linguistics science and for "aids" to translation, but not for further support of fully automatic translation. This report caused a dramatic reduction of large-scale funding of research on machine translation. Nonetheless, machine translation survived and eventually thrived, as we shall see in later chapters.

The Association for Machine Translation and Computational Linguistics (AMTCL) held its first meeting in 1962. In 1968, it changed its name to the Association for Computational Linguistics (ACL) and has become an international scientific and professional society for people working on problems involving natural language and computation. It publishes the quarterly journal *Computational Linguistics* and sponsors conferences and workshops.[15]

7.3 Question Answering

In addition to work on machine translation, researchers began exploring how sentences in a natural language, such as English, could be used to communicate with computers. You will recall Weizenbaum's ELIZA program that was able to engage a person in a conversation even though the program "understood" nothing about what was being said. And, I have already mentioned Raphael's SIR system that could represent information given to it and then answer questions.

I'll mention a few other projects to give a flavor of natural language processing work during this period. A program called BASEBALL (written in IPL-V, a special list-processing programming language developed by Newell, Shaw, and Simon to be described later) was developed at the Lincoln

Laboratory under the direction of Bert Green, a professor of Psychology at the Carnegie Institute of Technology.[16] It could answer simple English questions about baseball using a database about baseball games played in the American League during a single year. For example, it could answer a question such as "Where did the Red Sox play on July 7?" The questions had to be of a particularly simple form and restricted to words in the program's vocabulary. In the authors' words,[17]

> Questions are limited to a single clause; by prohibiting structures with dependent clauses the syntactic analysis is considerably simplified. Logical connectives, such as *and*, *or*, and *not*, are prohibited, as are constructions implying relations like *most* and *highest*. Finally, questions involving sequential facts, such as "Did the Red Sox ever win six games *in a row*?" are prohibited.

The program worked by converting a question into a special form called a "specification list" using both special-purpose syntactic and semantic analyses. This list would then be used to access the program's database to find an answer to the question. For example, the question "Where did the Red Sox play on July 7?" would first be converted to the list:

Place = ?
Team = Red Sox
Month = July
Day = 7

The authors claimed that their "restrictions were temporary expedients that will be removed in later versions of the program." As far as I know, there were no later versions of the program. (As we will see as my history of AI unfolds, there are several instances in which it proved very difficult to remove "temporary" restrictions.)

Another natural language program, SAD SAM, was written in IPL-V in 1962–1963 by Robert Lindsay at the Carnegie Institute of Technology.[18] It could analyze English sentences about family relationships and encode these relationships in a family tree. Using the tree, it could then answer English questions about relationships.

For example, if SAD SAM received the sentence "Joe and Jane are Tom's offspring," it would construct a treelike list structure for a certain "family unit" in which Tom is the father and Joe and Jane are the children. Then, if it received the sentence "Mary is Jane's mother," it would add Mary to this structure as Tom's wife. It would then be able to answer the question "Who is Joe's mother?"

SAD SAM is an acronym for Sentence Appraiser and Diagrammer and Semantic Analyzing Machine. The SAD part parsed the input sentences and

passed them to SAM, which extracted the semantic information needed for building family trees and for finding answers to questions. The program could accept a wide variety of sentences in Basic English – a system of grammar and a vocabulary of about 850 words defined by Charles K. Ogden.[19]

Robert F. Simmons (1925–1994), a psychologist and linguist at the Systems Development Corporation (SDC) in Santa Monica, California, had grander goals for his own work in natural language processing. According to an "In Memoriam" page written by Gordon Novak, one of his Ph.D. students at the University of Texas in Austin where Simmons took up a position as Professor of Computer Sciences and Psychology,[20]

> Simmons' dream was that one could have "a conversation with a book;" the computer would read the book, and then the user could have a conversation with the computer, asking questions to be answered from the computer's understanding of the book.

Accomplishing this "dream" would turn out to be as hard as AI itself. In a 1961 note about his proposed "Synthex" project, Simmons described how he would begin:[21]

> The objective of this project is to develop a research methodology and a vehicle for the design and construction of a general purpose computerized system for synthesizing complex human cognitive functions. The initial vehicle, proto-synthex, will be an elementary language-processing device which reads simple printed material and answers simple questions phrased in elementary English.

By 1965, Simmons and Lauren Doyle had conducted some experiments with their Protosynthex system. According to a report by Trudi Bellardo Hahn,[22] "A small prototype full-text database of chapters from a child's encyclopedia (*Golden Book*) was loaded on the system. Protosynthex could respond to simple questions in English with an 'answer.' . . . it was a pioneering effort in the use of natural language for text retrieval."

In the meantime, Daniel G. Bobrow (1935–), a Ph.D. student of Marvin Minsky's at MIT, wrote a set of programs, called the STUDENT system, that could solve algebra "story problems" given to it in a restricted subset of English. Here is an example of a problem STUDENT could solve:

> The distance from New York to Los Angeles is 3000 miles. If the average speed of a jet plane is 600 miles per hour, find the time it takes to travel from New York to Los Angeles by jet.

STUDENT solved the problem by using some known relationships about speed and distance to set up and solve the appropriate equations. Bobrow's dissertation gave several other examples of problems STUDENT could solve and the methods used.[23]

Notes

1. Noam Chomsky, *Syntactic Structures,* 's-Gravenhage: Mouton & Co., 1957. [**142**]

2. The basic structure of PSGs was independently invented by computer scientist John Backus to describe the syntax of the ALGOL programming language. See John Backus, "The Syntax and Semantics of the Proposed International Algebraic Language of the Zürich ACM-GAMM Conference," *Proceedings on the International Conference on Information Processing*, pp. 125–132, UNESCO, 1959. [**142**]

3. According to C. George Boeree (see http://www.ship.edu/~cgboeree/wundtjames.html), Wilhelm Wundt "invented the tree diagram of syntax we are all familiar with in linguistics texts." [**143**]

4. For a survey of work during this period, see Daniel Bobrow, "Syntactic Analysis of English by Computer: A Survey," *Proceedings of the 1963 Fall Joint Computer Conference*, Vol. 24, pp. 365–387, Baltimore: Spartan Books, 1963. [**145**]

5. The Standards Western Automatic Computer (later abbreviated to SWAC) [**147**]

6. The quotation appears in John Hutchins, "From First Conception to First Demonstration: The Nascent Years of Machine Translation, 1947–1954. A chronology," *Machine Translation*, Vol. 12 No. 3, pp. 195–252, 1997. (A corrected 2005 version, with minor additions, appears at http://www.hutchinsweb.me.uk/MTJ-1997-corr.pdf.) [**147**]

7. W. N. Locke and A. D. Booth (eds.), *Machine Translation of Languages: Fourteen Essays*, pp. 15–23, Cambridge, MA: MIT Press, 1955. [**148**]

8. For a history of MT, see W. John Hutchins, "Machine Translation: A Brief History," in E. F. K. Koerner and R. E. Asher (eds.), *Concise History of the Language Sciences: From the Sumerians to the Cognitivists*, pp. 431–445, Oxford: Pergamon Press, 1995. (Also available online at http://www.hutchinsweb.me.uk/ConcHistoryLangSci-1995.pdf.) Hutchins also has a Web page devoted to his publications at http://ourworld.compuserve.com/homepages/WJHutchins/#History1. [**148**]

9. For reports about this conference see E. Reifler, "The First Conference on Mechanical Translation," *Mechanical Translation*, Vol. 1 No. 2, pp. 23–32, 1954, and A. C. Reynolds, "The Conference on Mechanical Translation Held at MIT, June 17–20, 1952," *Mechanical Translation*, Vol. 1, No. 3, pp. 47–55, 1954. [**148**]

10. http://www-03.ibm.com/ibm/history/exhibits/701/701_translator.html. [**148**]

11. For a summary of the IBM–Georgetown work, see W. John Hutchins, "The Georgetown–IBM Experiment Demonstrated in January 1954," in Robert E. Frederking and Kathryn B.Taylor (eds.), *Proceedings of Machine Translation: From Real Users to Research*, 6th Conference of the Association for Machine Translation in the Americas, AMTA-2004, pp. 102–114, Washington DC, USA, September 28–October 2, 2004, Berlin: Springer, 2004. An online version is available at http://www.hutchinsweb.me.uk/ATMA-2004.pdf. [**149**]

12. Yehoshua Bar-Hillel, "The Present Status of Automatic Translation of Languages," *Advances in Computers*, Vol. 1, No. 1, pp. 91–163, 1960. [**149**]

13. In D. V. Blake and A. M. Uttley (eds.), *Proceedings of the Symposium on Mechanisation of Thought Processes*, p. 85, London: Her Majesty's Stationary Office, 1959. [**150**]

14. John R. Pierce *et al.*, *Language and Machines: Computers in Translation and Linguistics*, ALPAC Report, National Academy of Sciences Publication 416, National Research Council, Washington, DC, 1966. [**150**]

15. See http://www.aclweb.org/. [**150**]

16. Bert F. Green Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery, "BASEBALL: An Automatic Question Answerer," pp. 219–224, *Proceedings of the Western Joint*

Computer Conference, May 1961. Reprinted in Edward A. Feigenbaum and Julian Feldman (eds.), *Computers and Thought*, pp. 207–216, New York: McGraw Hill, 1963, and in B. Grosz, K. Spark Jones, and B. Lynn Webber (eds.), *Readings in Natural Language Processing*, Morgan Kaufman, Los Altos, CA, 1986. [**151**]

17. *Ibid.* [**151**]

18. See Robert K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Language," in Edward A. Feigenbaum, and Julian Feldman, *op. cit.*, pp. 217–233. [**151**]

19. Charles K. Ogden, *Basic English: A General Introduction with Rules and Grammar*, 4th edition, London: Kegan, Paul, Trench, Trubner & Co., Ltd., 1933. (Lindsay says 1,700 words; other sources say 850.) [**152**]

20. From http://www.cs.utexas.edu/users/ai-lab/simmons.html. [**152**]

21. Robert F. Simmons, "Synthex," *Communications of the ACM*, Vol. 4 , No. 3, p. 140, March 1961. [**152**]

22. From "Text Retrieval Online: Historical Perspective on Web Search Engines," by Trudi Bellardo Hahn, *ASIS Bulletin*, April/May 1998. Available online at http://www.asis.org/Bulletin/Apr-98/hahn.html. [**152**]

23. Daniel G. Bobrow, "Natural Language Input for a Computer Problem Solving System," MIT Artificial Intelligence Project Memo 66, Memorandum MAC-M-148, March 30, 1964. Available online at http://dspace.mit.edu/bitstream/handle/1721.1/5922/AIM-066.pdf?sequence=2. An article based on the dissertation is Chapter 3 of Marvin Minsky (ed.), *Semantic Information Processing*, Cambridge, MA: MIT Press, 1968. [**152**]

Chapter 8

1960s' Infrastructure

The technical developments during the 1960s were aided (indeed, one might say made possible) by several systems support and societal factors. New computer languages made it much easier to build AI systems. Researchers from mathematics, from cognitive science, from linguistics, and from what soon would be called "computer science" came together in meetings and in newly formed laboratories to attack the problem of mechanizing intelligent behavior. In addition, government agencies and companies, concluding that they had an important stake in this new enterprise, provided needed research support.

8.1 Programming Languages

Newell and Simon were among the first to realize that a specialized computer language would be useful for manipulating the symbolic expressions that were at the heart of their approach to mechanizing intelligence. The most elementary kind of symbolic expression is a list of symbols, such as $(7, \mathtt{B}, 5)$. More complex structures can be composed by creating lists of lists of symbols and lists of lists of lists, and so on.

In my description of symbol structures for the eight-puzzle, I mentioned the kinds of manipulations that are needed. Recall that the starting position of the eight-puzzle was represented by the expression

$$((2, 8, 3), (1, 6, 4), (7, \mathtt{B}, 5)).$$

What was needed was a language for writing programs that could produce expressions representing the positions corresponding to moves of the puzzle. For example, one of the moves that can be made from the starting position is represented by the expression

$$((2, 8, 3), (1, 6, 4), (\mathtt{B}, 7, 5)).$$

To produce this expression, the program must copy the starting position expression and then interchange the first and second elements of the third list in that expression.

Newell, Shaw, and Simon set about to develop a language in which these kinds of manipulations could be programmed. Starting around 1954 at the RAND Corporation, they created a series of languages all called IPL (for information-processing language). Several versions of the language were developed. IPL-I was not actually implemented but served as a design specification. IPL-II was implemented in 1955 for the RAND Corporation's JOHNNIAC computer. Later versions (through IPL-VI) were implemented at Carnegie Tech.

The IPL languages were used to program several early AI programs, including LT, GPS, NSS (the Newell, Shaw, Simon chess-playing program), and the programs written by Newell's and Simon's students, such as Quillian and George Ernst. After the Dartmouth summer project, John McCarthy also began thinking about using list-processing languages. He was aware of the use of FLPL (FORTRAN fortified by some list-processing operations) in Gelernter's geometry theorem-proving machine. Ultimately, however, McCarthy concluded a new language was needed that was easier to use than IPL and more powerful than FLPL.

Starting in the fall of 1958 at MIT, McCarthy began the implementation of a programming language he called LISP (for list processing). He based it (loosely) on a branch of mathematics of special interest in computation called recursive function theory. LISP had several elementary operations for copying a list, stripping off elements of a list, adding an element to a list, and checking to see whether something were an element of a list. From these, arbitrarily complex manipulations of lists could be composed. An important feature of LISP was that programs for manipulating lists were themselves represented as lists. Such programs could thus be elements of other lists and could have subprograms embedded in them. A program could even have a version of itself embedded in it. As I have already mentioned, programs that can activate versions of themselves as part of their operation are called "recursive" and are very useful (if used with the care needed to avoid endless circularity).[1]

Because it was easier to use, LISP soon replaced IPL as the primary language of artificial intelligence research and applications. The programs produced by Minsky's students, Evans, Raphael, Bobrow, Slagle, and others, were all written in LISP. (Interestingly, Arthur Samuel did not use a list-processing language for writing his checkers-playing programs. Rather heroically, he programmed them in the base language of elementary machine operations to make them run efficiently and use memory sparingly.)

Besides developing LISP, McCarthy proposed a method, called "time-sharing," by which a single computer could be made to serve several users simultaneously – acting as if each user had his or her own private

machine.[2] Working initially with Ed Fredkin at Bolt, Beranek, and Newman (BBN) and later with others, McCarthy developed an early time-sharing system at MIT using a DEC PDP-1 computer.[3]

8.2 Early AI Laboratories

In 1955, Newell moved from the RAND Corporation to Carnegie Tech (which became Carnegie Mellon University, CMU, in 1967) to work on a Ph.D. degree in industrial management under Herb Simon. After completing his degree, Newell stayed on as a professor at Carnegie, and he and Simon began advising a number of Ph.D. students – using the phrase "complex information processing (CIP)" to describe their work. (For several years they avoided the AI sobriquet.) In the fall of 1956, Herb Simon took delivery of an IBM 650, which was the first computer used for CIP work. Later, they used an IBM 704, followed by a series of DEC machines.

John McCarthy moved from Dartmouth to MIT in the fall of 1958. Minsky joined MIT a year later. As Minsky puts it,[4]

> [McCarthy and I] were walking down the hall and we met Jerry Wiesner or Zimmerman or someone and he said how's it going and we said well, we're working on these artificial intelligence ideas but we need a little more room and support for some graduate students. So then a room appeared a few days later...

The "room" soon developed into the MIT Artificial Intelligence Project. Initially, the group used MIT's IBM 704 computer, which proved not to have sufficient memory for the programs being written. So it began to use a DEC PDP-1 belonging to BBN. With funding from another project at MIT, it bought its own PDP-1, which was followed by the PDP-6 and PDP-10. Several of the group's Ph.D. students did their work at BBN and at the nearby Lincoln Laboratory where Oliver Selfridge continued his AI research – mainly on pattern recognition and machine learning. In 1962, McCarthy moved to Stanford where he began an AI project. Seymour Papert (1928–), a mathematician who had worked with Jean Piaget, joined Minsky as co-director of the AI Lab in 1963.

By 1965 at Stanford, McCarthy and colleagues had created a time-sharing system, called Thor, on a PDP-1 computer. It included twelve Philco display terminals, which made it the first display-oriented time-sharing system anywhere in the world.

With the help of Lester Earnest (1930–), who had moved to Stanford from Lincoln Laboratory, McCarthy set up the Stanford AI Laboratory (SAIL) in 1965. Outgrowing its on-campus facilities, SAIL moved to a building in the Stanford foothills during the summer of 1966. (See Fig. 8.1.) With additional

support from ARPA, the Lab took delivery of a DEC PDP-6 computer and, later, a PDP-10 computer. In addition to its work in AI (which I'll describe in subsequent chapters), SAIL was involved in many other computer-related projects including the development of a precursor to computer "windows" and the early installation of terminals in everyone's offices.[5]

Figure 8.1: Site of the Stanford AI Lab from 1966 until 1980. (Photograph courtesy of Lester Earnest.)

Since their early days, the groups at CMU, MIT, and Stanford have been among the leaders of research in AI. Often graduates of one of these institutions became faculty members of one of the other ones.

Around 1965 another world-class AI center emerged at the University of Edinburgh in Scotland. Its founder was Donald Michie (1923–2007; Fig. 8.2), who had worked with Alan Turing and I. J. (Jack) Good at Bletchley Park during the Second World War. Discussions there with Turing and Good about intelligent machines captivated Michie. As he reported in an October 2002 interview, "I resolved to make machine intelligence my life as soon as such an enterprise became feasible."[6] Because computer facilities in the mid- to late 1940s were primitive and scarce, Michie became a geneticist and molecular biologist.

Pursuing his interest in machine intelligence, from the sidelines as it were, in 1960 he put together a "contraption of matchboxes and glass beads" that could learn to play tic-tac-toe (noughts and crosses). He named his "machine" MENACE, an acronym for Matchbox Educable Noughts and Crosses Engine.[7]

Figure 8.2: Donald Michie. (Photograph courtesy of the Michie Family.)

(See Fig. 8.3.) (As I'll explain later, MENACE foreshadowed work in what is now called "reinforcement learning.") During a year-long visit to Stanford (sponsored by the Office of Naval Research) in the early 1960s, Michie met John McCarthy, Bernard Widrow, and others working in AI (including me). While there, he worked on a learning program for balancing a pole on a motor-driven cart.

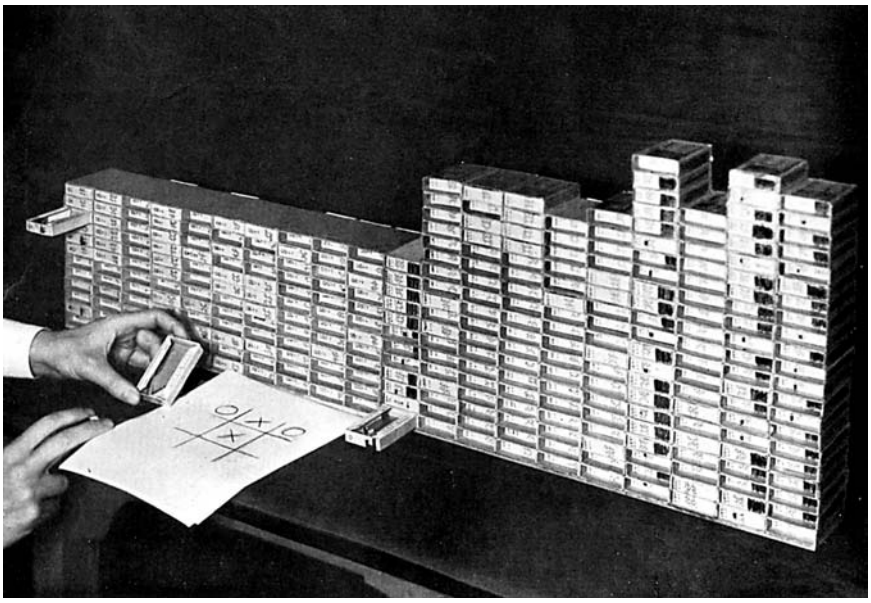


Figure 8.3: Michie's MENACE for learning how to play tic-tac-toe.

In January 1965, Michie became the Director of the UK's first AI laboratory, the Experimental Programming Unit, at the University of Edinburgh. This group was to become the Department of Machine Intelligence and Perception in October 1966. Michie recruited some top-flight computer talent, including Rod Burstall, Robin Popplestone, and John Collins. Those three developed a list-processing language called POP-2, which was the language used for AI program-writing by members of the Unit. (I'll describe some of these programs later.) For many years, Michie's group worked collaboratively with a nearby University of Edinburgh group, the Metamathematics Unit under Bernard Meltzer (circa 1916–2008). The Metamathematics Unit is famous for the work of Robert Boyer and J Strother Moore in mechanized theorem proving and of Robert Kowalski in developing some of the principles of logic programming.[8]

At IBM in Poughkeepsie, Nathan Rochester and Herb Gelernter continued AI research for a short time after the Dartmouth workshop. This research resulted in the geometry-theorem-proving machine. However, soon after, according to a book about government support for computing research, "in spite of the early activity of Rochester and other IBM researchers, the corporation's interest in AI cooled. Although work continued on computer-based checkers and chess, an internal report prepared about 1960 took a strong position against broad support for AI."[9] Perhaps IBM wanted to emphasize how computers *helped* people perform tasks rather than how they might *replace* people. McCarthy's view about all of this is that "IBM thought that artificial intelligence [that machines were as smart as people] was bad for IBM's image... This may have been associated with one of their other image slogans, which was 'data processing, not computing.'"[10]

8.3 Research Support

As the computing systems needed for AI research became larger and more expensive, and as AI laboratories formed, it became necessary to secure more financial support than was needed in the days when individual investigators began work in the field. Two of the major sources of funding during the late 1950s and early 1960s were the Office of Naval Research (ONR) and the Advanced Research Projects Agency (ARPA), each a part of the U.S. defense establishment.

ONR was formed shortly after the end of the Second World War. Its mission was "to plan, foster, and encourage scientific research in recognition of its paramount importance as related to the maintenance of future naval power and the preservation of national security." Its Information Systems Branch was set up in the mid-1950s under the direction of Marshall Yovits. The branch supported AI work at several institutions and also sponsored conferences and workshops on self-organizing systems, cybernetics, optical character

recognition, and artificial intelligence. All of this was done in anticipation that these technologies would be generally useful to the U.S. Navy. (A later director, Marvin Denicoff, supported some of my research and my AI textbook writing.)

The formation of ARPA was, in part, a response to the successful launch of the Soviet satellite *Sputnik* in 1957. ARPA's mission was to provide significant amounts of research funds to attack problem areas important to U.S. defense. One of its most important projects in the late 1950s was the development of ablative nose cones to absorb and dissipate heat during ballistic missile reentry. Its Information Processing Techniques Office (IPTO) was set up in 1962 under the direction of J. C. R. (Lick) Licklider (1915–1990; Fig. 8.4).

"Lick" (as he was called by all who knew him) was a psychoacoustician who worked first at Lincoln Laboratory and MIT and later at BBN. Lick's 1960 paper, "Man-Computer Symbiosis," proposed that men and computers should "cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs."[11]

Lick was persuaded that computers would play a very important role in defense – especially in those applications in which people and computers worked together. At ARPA, he provided funds to MIT for the formation of Project MAC (an acronym for Machine-Aided Cognition and perhaps for Multi-Access Computing or Man And Computers). [Project MAC, initially founded in July 1963, was later to become the Laboratory for Computer Science (LCS), and still later to evolve into the Computer Science and Artificial Intelligence Laboratory (CSAIL).] Project MAC took Minsky and McCarthy's Artificial Intelligence Project under its wing and also supported the development of MIT's Compatible Time-Sharing System (CTSS) under Fernando Corbató. (CTSS work was separate from McCarthy's time-sharing project.)

ARPA funds helped to establish "centers of excellence" in computer science. Besides MIT, these centers included Stanford, Carnegie Mellon, and SRI. ARPA also supported computer science work at the RAND Corporation, the Systems Development Corporation, and BBN, among others. AI was just one of ARPA's interests. IPTO also supported research that led to graphical user interfaces (and the mouse), supercomputing, computer hardware and very-large-scale integrated circuits (VLSI), and, perhaps most famously, research that led to the Internet. According to Licklider, "ARPA budgets did not even include AI as a separate line item until 1968."[12]

But as far as AI was concerned, Lick believed that Newell and Simon, Minsky, and McCarthy ought to be provided with research funds adequate to support big AI projects. With regard to the situation at Stanford (and probably to that at MIT and CMU also), Paul Edwards explained that[13]

> [F]unding from ARPA was virtually automatic; Licklider simply

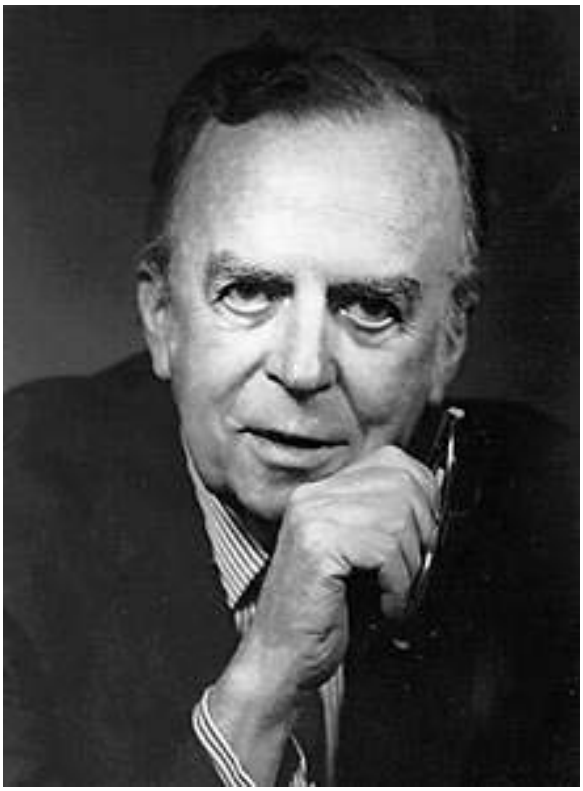


Figure 8.4: J. C. R. Licklider. (Photograph by Koby-Antupit from MIT Collection (JCL8).)

> asked McCarthy what he wanted and then gave it to him, a procedure unthinkable for most other government agencies. Licklider remembered that 'it seemed obvious to me that he should have a laboratory supported by ARPA. . . . So I wrote him a contract at that time."

McCarthy remembers all of this somewhat differently. Soon after arriving at Stanford in 1962, he sent a proposal to Licklider "to do AI." McCarthy claims that Licklider demurred at first – citing their close relationship when McCarthy was at MIT and Licklider at BBN – but then gave him "a small contract."[14] But perhaps it was not so "small" compared with how research was usually supported (say by the National Science Foundation) at the time. Les Earnest claims that McCarthy "obtained financial support for a small activity (6 persons) from the Advanced Research Projects Agency (ARPA) beginning June 15, 1963."[15]

Later, ARPA was renamed DARPA (for Defense Advanced Research Projects Agency) to emphasize its role in defense-related research. DARPA projects and grants were typically much larger than those of ONR and allowed the purchase of computers and other equipment as well as support for personnel. It's hardly an exaggeration to say that a good part of today's computer-based infrastructure is the result of DARPA research support.

8.4 All Dressed Up and Places to Go

By the mid-1960s AI was well prepared for further advances. Flushed with early successes it was poised to make rapid progress during the rest of the 1960s and 1970s. Indeed, many people made enthusiastic predictions. For example, in a 1957 talk[16] Herb Simon predicted that within ten years "a digital computer will be the world's chess champion unless the rules bar it from competition." He made three other predictions too. Within ten years computers would compose music, prove a mathematical theorem, and embody a psychological theory as a program. He said "it is not my aim to surprise or shock you... but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until – in a visible future – the range of problems they can handle will be coextensive with the range to which the human mind has been applied."[17] Later Simon said that his predictions were part of an attempt "to give some feeling for what computers would mean" to society.

One could argue that Simon's predictions about computers composing music and proving a mathematical theorem were realized soon after he made them, but a computer chess champion was not to emerge until forty years later. And, we are still far, I think, from achieving things "coextensive" with what the human mind can achieve.

Simon was not alone in being optimistic. According to Hubert Dreyfus, "Marvin Minsky, head of MITs Artificial Intelligence Laboratory, declared in a 1968 press release for Stanley Kubrick's movie, *2001: A Space Odyssey*, that 'in 30 years we should have machines whose intelligence is comparable to man's.' "[18] The difficulty in assessing these sorts of predictions is that "human-level intelligence" is multifaceted. By the year 2000, AI programs *did* outperform humans in many intellectual feats while still having a long way to go in most others.

Even so, what had already been accomplished was an impressive start. More important perhaps than the specific demonstrations of intelligent behavior by machines was the technical base developed during the 1950s and early 1960s. AI researchers now had the means to represent knowledge by encoding it in networks, as logical formulas, or in other symbol structures tailored to specific problem areas. Furthermore, they had accumulated

experience with heuristic search and other techniques for manipulating and using that knowledge. Also, researchers now had new programming languages, IPL, LISP, and POP-2, that made it easier to write symbol-processing programs. Complementing all of this symbol-processing technology were neural networks and related statistical approaches to pattern recognition. These technical assets, along with the organizational and financial ones, provided a solid base for the next stage of AI's development.

Notes

1. For McCarthy's own history of the development of LISP, see http://www-formal.stanford.edu/jmc/history/lisp.html. Also see Herbert Stoyan's history of LISP at http://www8.informatik.uni-erlangen.de/html/lisp-enter.html. [**156**]

2. See McCarthy's memo proposing how to build a time-sharing system at http://www-formal.stanford.edu/jmc/history/timesharing-memo.html. [**157**]

3. For more about these early days of computing at MIT and of time-sharing work there (among other things), see the interview with John McCarthy conducted by William Aspray of the Charles Babbage Institute on March 2, 1989. It is available online at http://www.cbi.umn.edu/oh/display.phtml?id=92. [**157**]

4. From an interview conducted by Arthur L. Norberg on November 1, 1989, for the Charles Babbage Institute. Available online at http://www.cbi.umn.edu/oh/display.phtml?id=107. [**157**]

5. For a history of AI work in the lab up to 1973, see Lester Earnest (ed.), "Final Report: The First Ten Years of Artificial Intelligence Research at Stanford," Stanford Artificial Intelligence Laboratory Memo AIM-228 and Stanford Computer Science Department Report No. STAN-CS-74-409, July 1973. (Available online at http://www-db.stanford.edu/pub/cstr/reports/cs/tr/74/409/CS-TR-74-409.pdf.) For other SAIL history, see "SAIL Away" by Les Earnest at http://www.stanford.edu/~learnest/sailaway.htm. [**158**]

6. A textscript of the interview can be found online at http://www.aiai.ed.ac.uk/events/ccs2002/CCS-early-british-ai-dmichie.pdf. [**158**]

7. Donald Michie, "Experiments on the Mechanisation of Game Learning: 1. Characterization of the Model and its Parameters," *Computer Journal*, Vol. 1, pp. 232–263, 1963. [**158**]

8. For a history of these Edinburgh groups, see Jim Howe's online 1994 article "Artificial Intelligence at Edinburgh University: A Perspective" at http://www.dai.ed.ac.uk/AI_at_Edinburgh_perspective.html. [**160**]

9. National Research Council, *Funding a Revolution: Government Support for Computing Research*, Washington, DC: National Academy Press, 1999. (An html version of this book, which contains a rather conservative account of AI history, is available from http://www.nap.edu/catalog.php?record_id=6323#toc.) [**160**]

10. From "An Interview with John McCarthy," conducted by William Aspray on 2 March 1989, Palo Alto, CA, Charles Babbage Institute, The Center for the History of Information Processing, University of Minnesota, Minneapolis. [**160**]

11. J. C. R. Licklider, "Man–Computer Symbiosis," *IRE Transactions on Human Factors in Electronics*, HFE-1, pp. 4–11, 1960. Available online at http://memex.org/licklider.html. [**161**]

12. J. C. R. Licklider, "The Early Years: Founding IPTO," p. 220 in Thomas C. Bartee (ed.), *Expert Systems and Artificial Intelligence: Applications And Management*, Indianapolis: Howard W. Sams, 1988. [**161**]

13. Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America*, p. 270, Cambridge, MA: MIT Press, 1996. [**161**]

14. From "An Interview with John McCarthy," *op. cit.* [**162**]

15. Lester Earnest (ed.), "Final Report: The First Ten Years of Artificial Intelligence Research at Stanford," Stanford Artificial Intelligence Laboratory Memo AIM-228 and Stanford Computer Science Department Report No. STAN-CS-74-409, July 1973. (Available online at http://www-db.stanford.edu/pub/cstr/reports/cs/tr/74/409/CS-TR-74-409.pdf.) [**162**]

16. 12th National Meeting of the Operations Research Society (ORSA) in Pittsburgh. [**163**]

17. The published version of this talk is in Herbert Simon and Allen Newell, "Heuristic Problem Solving: The Next Advance in Operations Research," *Operations Research*, Vol. 6, January–February 1958. [**163**]

18. Hubert L. Dreyfus, "Overcoming the Myth of the Mental," *Topoi*, Vol. 25, pp. 43–49, 2006. [**163**]

8.4

Part III

Efflorescence: Mid-1960s to Mid-1970s

During the 1960s and well into the 1970s, AI research blossomed and progress seemed rapid. The laboratories established at MIT, Carnegie Mellon, Stanford, SRI, and Edinburgh expanded, and several new groups got started at other universities and companies. Achievements during the preceding years, even though modest in retrospect, were exciting and full of promise, which enticed several new people into the field, myself included. Many of us were just as optimistic about success as Herb Simon and Marvin Minsky were when they made their predictions about rapid progress.

AI entered a period of flowering that led to many new and important inventions. Several ideas originated in the context of Ph.D. dissertation research projects. Others emerged from research laboratories and from individual investigators wrestling with theoretical problems. In this part, I'll highlight some of the important projects and research results. Although not a complete account, they typify much of what was going on in AI during the period. I'll begin with work in computer vision.

9.0

Chapter 9

Computer Vision

Sighted humans get much of their information through vision. That part of AI called "computer vision" (or, sometimes, "machine vision") deals with giving computers this ability. Most computer vision work is based on processing two-dimensional images gathered from a three-dimensional world – images gathered by one or more television cameras, for example. Because the images are two-dimensional projections of a three-dimensional scene, the imaging process loses information. That is, different three-dimensional scenes might produce the same two-dimensional image. Thus, the problem of reconstructing the scene faithfully from an image is impossible in principle.

Yet, people and other animals manage very well in a three-dimensional world. They seem to be able to interpret the two-dimensional images formed on their retinas in a way that gives them reasonably accurate and useful information about their environments.

Stereo vision, using two eyes, helps provide depth information. Computer vision too can use "stereopsis" by employing two or more differently located cameras looking at the same scene. (The same effect can be achieved by having one camera move to different positions.) When two cameras are used, for example, the images formed by them are slightly displaced with respect to each other, and this displacement can be used to calculate distances to various parts of the scene. The computation involves comparing the relative locations in the images that correspond to the objects in the scene for which depth measurements are desired. This "correspondence problem" has been solved in various ways, one of which is to seek high correlations between small areas in one image with small areas in the other. Once the "disparity" of the location of an image feature in the two images is known, the distance to that part of the scene giving rise to this image feature can be calculated by using trigonometric calculations (which I won't go into here.)[1]

Perhaps surprisingly, a lot of depth information can be obtained from

other cues besides stereo vision. Some of these cues are inherent in a single image, and I'll be describing these in later chapters. Even more importantly, background knowledge about the kinds of objects one is likely to see accounts for much of our ability to interpret images. Consider the image shown in Fig. 9.1 for example.

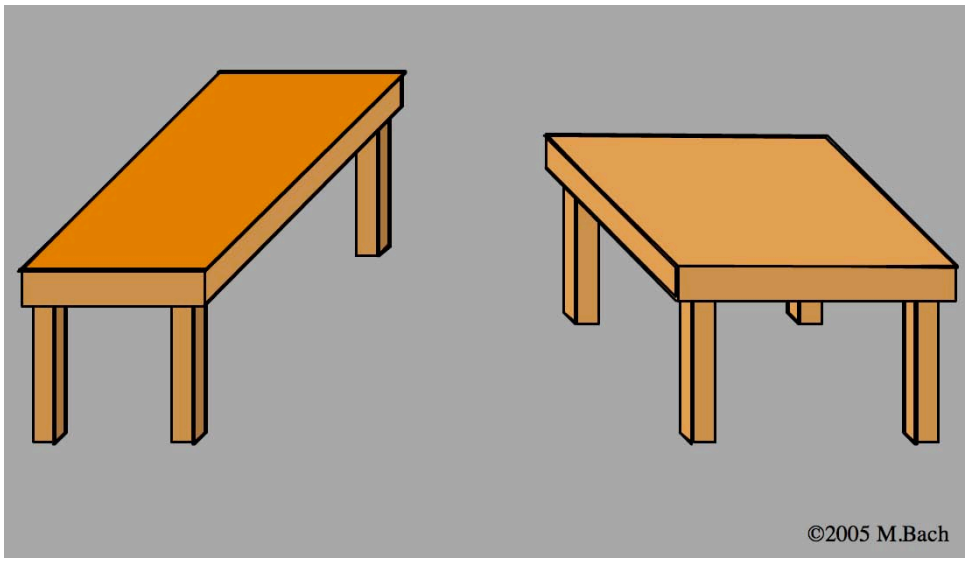


Figure 9.1: Two tables. (Illustration courtesy of Michael Bach.)

Most people would describe this image as being of two tables, one long and narrow and the other more-or-less square. Yet, if you measure the actual table tops in the image itself, you might be surprised to find that they are exactly the same size and shape! (The illustration is based on an illusion called "turning the tables" by the psychologist Roger Shepherd and is adapted from Michael Bach's version of Shepherd's diagram. If you visit Bach's Web site, http://www.michaelbach.de/ot/sze_shepardTables/, you can watch while one table top moves over to the other without changing shape.)

Something apart from the image provides us with information that induces us to make inferences about the shapes of the three-dimensional tables captured in the two-dimensional image shown in Fig. 9.1. As we shall see, that extra information consists of two things: knowledge about the image-forming process under various lighting conditions and knowledge about the kinds of things and their surfaces that occur in our three-dimensional world. If we could endow computers with this sort of knowledge, perhaps they too would be able to see.

9.1 Hints from Biology

There has been a steady flow of information back and forth between scientists attempting to understand how vision works in animals and engineers working on computer vision. An early example of work at the intersection of these two interests was described in an article titled "What the Frog's Eye Tells the Frog's Brain"[2] by four scientists at MIT. Guided by previous biological work, the four, Jerome Lettvin, H. R. Maturana, Warren McCulloch, and Walter Pitts, probed the parts of the frog's brain that processed images. They found that the frog's visual system consisted of "detectors" that responded only to certain kinds of things in its visual field. It had detectors for small, moving convex objects (such as flies) and for a sudden darkening of illumination (such as might be caused by a looming predator). These, together with a couple of other simple detectors, gave the frog information about food and danger. In particular, the frog's visual system did not, apparently, construct a complete three-dimensional model of its visual scene. As the authors wrote,

> The frog does not seem to see or, at any rate, is not concerned with the detail of stationary parts of the world around him. He will starve to death surrounded by food if it is not moving. His choice of food is determined only by size and movement. He will leap to capture any object the size of an insect or worm, providing it moves like one. He can be fooled easily not only by a bit of dangled meat but by any moving small object. His sex life is conducted by sound and touch. His choice of paths in escaping enemies does not seem to be governed by anything more devious than leaping to where it is darker. Since he is equally at home in water and on land, why should it matter where he lights after jumping or what particular direction he takes?

Other experiments produced further information about how the brain processes visual images. Neurophysiologists David Hubel (1926–) and Torsten Wiesel (1924–) performed a series of experiments, beginning around 1958, which showed that certain neurons in the mammalian visual cortex responded selectively to images and parts of images of specific shapes. In 1959 they implanted microelectrodes in the primary visual cortex of an anesthetized cat. They found that certain neurons fired rapidly when the cat was shown images of small lines at one angle and that other neurons fired rapidly in response to small lines at another angle. In fact, they could make a "map" of this area of the cat's brain, relating neuron location to line angle. They called these neurons "simple cells" – to be distinguished from other cells, called "complex cells," that responded selectively to lines moving in a certain direction. Later work revealed that other neurons were specialized to respond to images containing more complex shapes such as corners, longer lines, and large edges.[3] They found that similar specialized neurons also existed in the brains

of monkeys.[4] Hubel and Wiesel were awarded the Nobel Prize in Physiology or Medicine in 1981 (jointly with Roger Sperry for other work).[5]

As I'll describe in later sections, computer vision researchers were developing methods for extracting lines (both large and small) from images. Hubel and Wiesel's work helped to confirm their view that finding lines in images was an important part of the visual process. Yet, straight lines seldom occur in the natural environments in which cats (and humans) evolved, so why do they (and we) have neurons specialized for detecting them? In fact, in 1992 the neuroscientists Horace B. Barlow and David J. Tolhurst wrote a paper titled "Why Do You Have Edge Detectors?"[6] As a possible answer to this question, Anthony J. Bell and Terrence J. Sejnowski later showed mathematically that natural scenes can be analyzed as a weighted summation of small edges even though the scenes themselves do not have obvious edges.[7]

9.2 Recognizing Faces

In the early 1960s at his Palo Alto company, Panoramic Research, Woodrow (Woody) W. Bledsoe (who later did work on automatic theorem proving at the University of Texas), along with Charles Bisson and Helen Chan (later Helen Chan Wolf), developed techniques for face recognition supported by projects from the CIA.[8] Here is a description of their approach taken from a memorial article:[9]

> This [face-recognition] project was labeled man-machine because the human extracted the coordinates of a set of features from the photographs, which were then used by the computer for recognition. Using a GRAFACON, or RAND TABLET, the operator would extract the coordinates of features such as the center of pupils, the inside corner of eyes, the outside corner of eyes, point of widows peak, and so on. From these coordinates, a list of 20 distances, such as width of mouth and width of eyes, pupil to pupil, were computed. These operators could process about 40 pictures an hour. When building the database, the name of the person in the photograph was associated with the list of computed distances and stored in the computer. In the recognition phase, the set of distances was compared with the corresponding distance for each photograph, yielding a distance between the photograph and the database record. The closest records are returned.

Bledsoe continued this work with Peter Hart at SRI after leaving Panoramic in 1966.[10]

Then, in 1970, a Stanford Ph.D. student, Michael D. Kelly, wrote a computer program that was able automatically to detect facial features in

pictures and use them to identify people.[11] The task for his program was, as he put it,

> to choose, from a collection of pictures of people taken by a TV camera, those pictures that depict the same person. ...
>
> In brief, the program works by finding the location of features such as eyes, nose, or shoulders in the pictures. ... The interesting and difficult part of the work reported in this thesis is the detection of these features in digital pictures. The nearest-neighbor method is used for identification of individuals once a set of measurements has been obtained.

Another person who did pioneering work in face recognition was vision researcher Takeo Kanade, now a professor at Carnegie Mellon University. In a 2007 speech at the Eleventh IEEE International Conference on Computer Vision, he reflected on his early work in this field:[12] "I wrote my face recognition program in an assembler language, and ran it on a machine with 10 microsecond cycle time and 20 kB of main memory. It was with pride that I tested the program with 1000 face images, a rare case at the time when testing with 10 images was called a 'large-scale' experiment." (By the way, Kanade has continued his face recognition work up to the present time. His face-recognition Web page is at http://www.ri.cmu.edu/labs/lab_51.html.)

Face recognition programs of the 1960s and 1970s had several limitations. They usually required that images be of faces of standard scale, pose, expression, and illumination. Toward the end of the book, I'll describe research leading to much more robust automatic face recognition.

9.3 Computer Vision of Three-Dimensional Solid Objects

9.3.1 An Early Vision System

Lawrence G. Roberts (1937–), an MIT Ph.D. student working at Lincoln Laboratory, was perhaps the first person to write a program that could identify objects in black-and-white (gray-scale) photographs and determine their orientation and position in space. (His program was also the first to use a "hidden-line" algorithm, so important in subsequent work in computer graphics. As chief scientist and later director of ARPA's Information Processing Techniques Office, Roberts later played an important role in the creation of the Arpanet, the forerunner of the Internet.)

In the introduction to his 1963 MIT Ph.D. dissertation,[13] Roberts wrote

> The problem of machine recognition of pictorial data has long been a challenging goal, but has seldom been attempted with anything more complex than alphabetic characters. Many people have felt that research on character recognition would be a first step, leading the way to a more general pattern recognition system. However, the multitudinous attempts at character recognition, including my own, have not led very far. The reason, I feel, is that the study of abstract, two-dimensional forms leads us away from, not toward, the techniques necessary for the recognition of three-dimensional objects. The perception of solid objects is a process which can be based on the properties of three-dimensional transformations and the laws of nature. By carefully utilizing these properties, a procedure has been developed which not only identifies objects, but also determines their orientation and position in space.

Roberts's system first processed a photograph of a scene to produce a representation of a line drawing. It then transformed the line drawing into a three-dimensional representation. Matching this representation against a stored list of representations of solid objects allowed it to classify the object it was viewing. It could also produce a computer-graphics image of the object as it might be seen from any point of view.

Our main interest here is in how Roberts processed the photographic image. After scanning the photograph and representing it as an array of numbers (pixels) representing intensity values, Roberts used a special calculation, later called the "Roberts Cross," to determine whether or not each small 2×2 square in the array corresponded to a part of the image having an abrupt change in image intensity. (The Roberts Cross was the first example of what were later called "gradient operators.") He then rerepresented the image "lighting up" only those parts of the image where the intensity changed abruptly and leaving "dark" those parts of the image with more-or-less uniform intensity. The result of this process is illustrated in Fig. 9.2 for a typical image used in Roberts's dissertation. As can be seen in that figure, large changes in image intensity are usually associated with the edges of objects. Thus, gradient operators, such as the Roberts Cross, are often called "edge detectors."

Further processing of the image on the right attempted to connect the dots representing abrupt intensity changes by small straight-line segments, then by longer line segments. Finally, a line drawing of the image was produced. This final step is shown in Fig. 9.3.

Roberts's program was able to analyze many different photographs of solid objects. He commented that "The entire picture-to-line-drawing process is not optimal but works for simple pictures." Roberts's success stimulated further work on programs for finding lines in images and for assembling these lines into representations of objects. Perhaps primed by Roberts's choice of

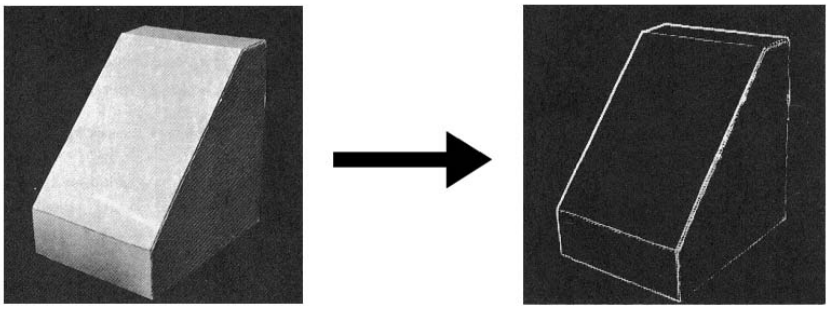


Figure 9.2: Detecting changes in intensity. (Photographs used with permission of Lawrence Roberts.)

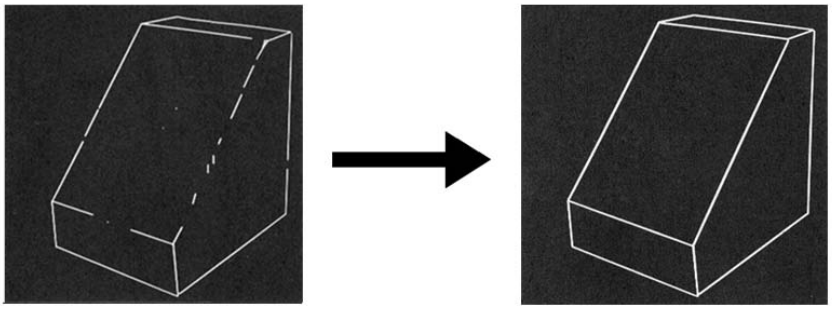


Figure 9.3: Producing the final line drawing. (Photographs used with permission of Lawrence Roberts.)

solid objects, much of the subsequent work dealt with toy blocks (or "bricks" as they are called in Britain).

9.3.2 The "Summer Vision Project"

Interestingly, Larry Roberts was a student of MIT information theory professor Peter Elias, not of Marvin Minsky. But Minsky's group soon began to work on computer vision also. In the summer of 1966, the mathematician and psychologist Seymour Papert, a recent arrival at MIT's Artificial Intelligence Group, launched a "summer vision project." Its goal was to develop a suite of programs that would analyze a picture from a "videsector" (a kind of scanner) to "actually name objects [such as balls, cylinders, and blocks] by matching them with a vocabulary of known objects." One motivation for the project was "to use our summer workers effectively in the

construction of a significant part of a visual system."[14]

Of course, the problem of constructing "a significant part of a visual system" was much more difficult than Papert expected. Nevertheless, the project was successful in that it began a sustained effort in computer vision research at MIT, which continues to this day.

After these early forays at MIT (and similar ones at Stanford and SRI to be described shortly), computer vision research focused on two areas. The first was what might be called "low-level" vision – those first stages of image processing that were aimed at constructing a representation of the image as a line drawing, given an image that was of a scene containing rather simple objects. The second area was concerned with how to analyze the line drawing as an assemblage of separate objects that could be located and identified. An important part of low-level vision was "image filtering," to be described next.

9.3.3 Image Filtering

The idea of filtering an image to simplify it, to correct for noise, and to enhance certain image features had been around for a decade or more. I have already mentioned, for example, that in 1955 Gerald P. Dinneen processed images to remove noise and enhance edges. Russell Kirsch and colleagues had also experimented with image processing.[15] (Readers who have manipulated their digital photography pictures on a computer have used some of these image filters.) Filtering two-dimensional images is not so very different from filtering one-dimensional electronic signals – a commonplace operation. Perhaps the simplest operation to describe is "averaging," which blurs fine detail and removes random noise specks. As in all averaging operations, image averaging takes into account adjacent values and combines them. Consider, for example, the image array of intensity values shown in Fig. 9.4 containing a 3×3 "averaging window" outlined in bold. These intensity values correspond to an image whose right side is bright and whose left side is dark with a sharp edge between. (I adopt the convention that large numbers, such as 10 correspond to brightly illuminated parts of the image, and the number 0 corresponds to black.)

The averaging operation moves the averaging window over the entire image so that its center lies over each pixel in turn. For each placement of the window, the value of the intensity at its center is replaced (in the filtered version) by the average intensity of the values within the window. (The process of moving a window around the image and doing calculations based on the numbers in the window is called *convolution*.) In this example, the 0 at the center of the window would be replaced by 3.33 (perhaps rounded down to 3). One can see that averaging blurs the sharp edge – with the 10 fading to (a rounded) 7 fading to 3 fading to 0 as one moves from right to left. However, intensities well within evenly illuminated regions are not changed.

0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10
0	0	0	0	0	10	10	10	10	10

Figure 9.4: An array of image intensity values and an averaging window.

I have already mentioned another important filtering operation, the Roberts Cross, for detecting abrupt brightness changes in an image. Another one was developed in 1968 by a Ph.D. student at Stanford, Irwin Sobel. It was dubbed the “Sobel Operator” by Raj Reddy who described it in a Computer Vision course at Stanford.[16] The operator uses two filtering windows – one sensitive to large gradients (intensity changes) in the vertical direction and one to large gradients in the horizontal direction. These are shown in Fig. 9.5.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Figure 9.5: Sobel’s vertical (left) and horizontal (right) filters.

Each of the Sobel filters works the same way as the averaging filter, except that the image intensity at each point is multiplied by the number in the corresponding cell of the filtering window before adding all of the numbers. The sum would be 0 inside regions of uniform illumination. If the vertical filter is centered over a vertical edge (with the right side brighter than the left), the sum would be positive. (I’ll let you think about the other possibilities.) Results from the two filtering windows are combined mathematically to detect abrupt changes in any direction.

The images in Fig. 9.6 illustrate the Sobel Operator. The image on the right is the result of applying the Sobel Operator to the image on the left.

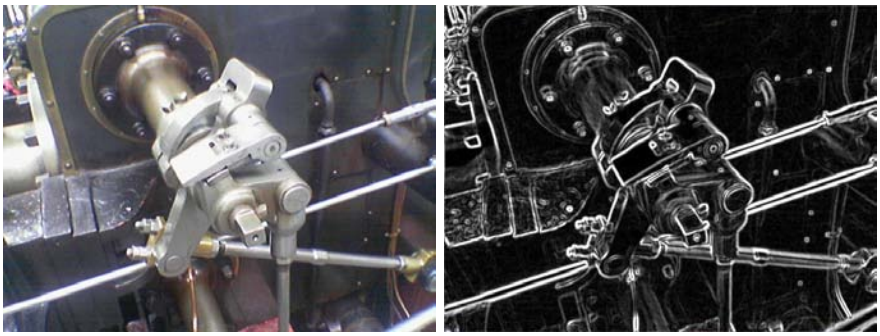


Figure 9.6: Finding abrupt changes in image brightness with the Sobel Operator. (Photographs taken by George Miller and available at http://en.wikipedia.org/wiki/Sobel_operator. Used under the terms of the GNU Free Documentation License.)

A number of other more complex and robust image processing operations have been proposed and used for finding edges, lines, and vertices of objects in images.[17] A particularly interesting one for finding edges was proposed by the British neuroscientist and psychologist David Marr (1945–1980) and Ellen Hildreth.[18] The Marr–Hildreth edge detector uses a filtering window called a "Laplacian of Gaussian (LoG)." (The name arises because a mathematical operator called a "Laplacian" is used on a bell-shaped curve called a "Gaussian," commemorating two famous mathematicians, namely, Pierre-Simon Laplace and Carl Friedrich Gauss.) In Fig. 9.7, I show an example of LoG numbers in a 9×9 filtering window. This window is moved around an image, multiplying image numbers and adding them up, in the same way as the other filtering windows I have already mentioned.

0	0	3	2	2	2	3	0	0
0	2	3	5	5	5	3	2	0
3	3	5	3	0	3	5	3	3
2	5	3	−12	−23	−12	3	5	2
2	5	0	−23	−40	−23	0	5	2
2	5	3	−12	−23	−12	3	5	2
3	3	5	3	0	3	5	3	3
0	2	3	5	5	5	3	2	0
0	0	3	2	2	2	3	0	0

Figure 9.7: A Laplacian of Gaussian filtering window.

If LoG numbers are plotted as "heights" above (and below) a plane, an interesting-looking surface results. An example is shown in Fig. 9.8. This LoG function is often called, not surprisingly, a Mexican hat or sombrero function.

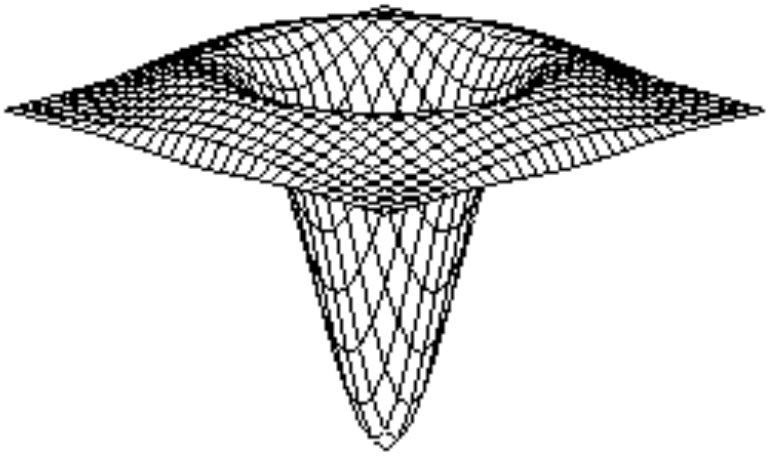


Figure 9.8: A Laplacian of Gaussian surface.

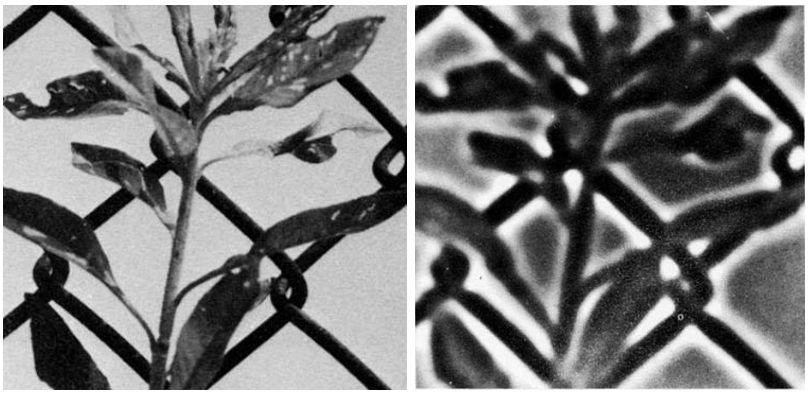


Figure 9.9: An image (left) and its LoG-processed version (right). (Images taken from David Marr and E. Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London*, Series B, Biological Sciences, Vol. 207, No. 1167, p. 198, February 1980.)

Marr and Hildreth used the LoG filtering window on several example images. One example, taken from their paper, is shown in Fig 9.9. Notice that the image on the right has whitish bands surrounding darker parts of the image. The Marr–Hildreth edge detector employs a second image-processing operation that looks for the transitions from light to dark (and vice versa) in the LoG-processed image to produce a final "line drawing," as shown in Fig. 9.10.

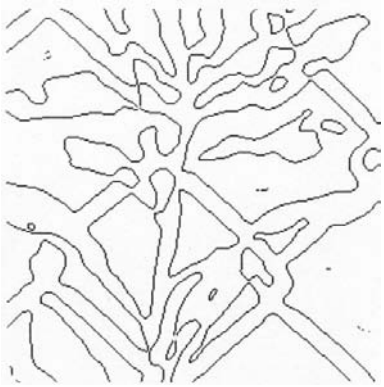


Figure 9.10: The final result of a Marr–Hildreth edge-detecting operation. (From David Marr and E. Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London*, Series B, Biological Sciences, Vol. 207, No. 1167, p. 198, February 1980.)

Further advances have been made in edge detection since Marr and Hildreth's work. Among the currently best detectors are those related to one proposed by John Canny called the Canny edge detector.[19]

As a neurophysiologist, Marr was particularly interested in how the human brain processes images. In a 1976 paper,[20] he proposed that the first stage of processing produces what he called a "primal sketch." As he puts it in his summary of that paper,

> It is argued that the first step of consequence is to compute a primitive but rich description of the grey-level changes present in an image. The description is expressed in a vocabulary of kinds of intensity change (EDGE, SHADING-EDGE, EXTENDED-EDGE, LINE, BLOB etc.). ... This description is obtained from the intensity array by fixed techniques, and it is called the primal sketch.

Marr and Hildreth put forward their edge detector as one of the operations the brain uses in producing a primal sketch. They stated that their

theory "explains several basic psychophysical findings, and... forms the basis for a physiological model of simple [nerve] cells."

Marr's promising career in vision research ended when he succumbed to cancer in 1980. During the last years of his life he completed an important book detailing his theories of human vision.[21] I'll describe some of Marr's ideas about other visual processing steps in a subsequent chapter.

9.3.4 Processing Line Drawings

Assuming, maybe somewhat prematurely, that low-level vision routines could produce a line-drawing version of an image, many investigators moved on to develop methods for analyzing line drawings to find objects in images.

Adolfo Guzman-Arenas (1943–), a student in Minsky's AI Group, focused on how to segment a line drawing of a scene containing blocks into its constituents, which Guzman called "bodies." His LISP program for accomplishing this separation was called SEE and ran on the MIT AI Group's PDP-6 computer.[22] The input to SEE was a line-drawing representation of a scene in terms of its surfaces, lines (where two surfaces came together), and vertices (where lines came together).

SEE's analysis of a scene began by sorting its vertices into a number of different types. For each vertex, depending on its type, SEE connected adjacent planar surfaces with "links." The links between surfaces provide evidence that those surfaces belong to the same body. For example, some links for a scene analyzed by SEE are shown in Fig. 9.11.

SEE performed rather well on a wide variety of line drawings. For example, it correctly found all of the bodies in the scene shown in Fig. 9.12.

For most of his work, Guzman assumed that somehow other programs would produce his needed line drawings from actual images. As he wrote in a paper describing his research,[23]

> The scene itself is not obtained from a visual input device, or from an array of intensities of brightness. Rather, it is assumed that a preprocessing of some sort has taken place, and the scene to be analyzed is available in a symbolic format... in terms of points (vertices), lines (edges), and surfaces (regions)."

Additionally, Guzman did not concern himself with what might be done after the scene had been separated into bodies:

> ...it cannot find "cubes" or "houses" in a scene, since it does not know what a "house" is. Once SEE has partitioned a scene into bodies, some other program will work on them and decide which of those bodies are "houses."

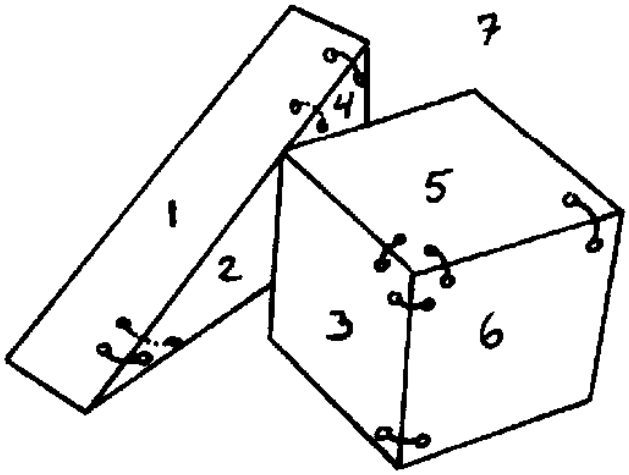


Figure 9.11: Links established by SEE for a sample scene. (Illustration used with permission of Adolpho Guzman.)

Later extensions to SEE, reported in the final version of his thesis, involved some procedures for image capture. But the images were of specially prepared scenes, as he recently elaborated:[24]

> Originally SEE worked on hand-drawn scenes, "perfect scenes" (drawings of lines)...
>
> Later, I constructed a bunch of wooden polyhedra (mostly irregular), painted them black, carefully painted their edges white, piled several of them together, and took pictures of the scenes. The pictures were scanned, edges found, and given to SEE. It worked quite well on them.

Although SEE was capable of finding bodies in rather complex scenes, it also could make mistakes, and it could not identify blocks that had holes in them.

The next person to work on the problem of scene articulation was David Huffman (1925–1999), a professor of Electrical Engineering at MIT. (Huffman was famous for his invention, while a graduate student at MIT, of what came to be called "Huffman coding," an efficient scheme that is used today in many applications involving the compression and transmission of digital data.) Huffman was bothered by what he considered Guzman's incomplete analysis of what kinds of objects could correspond to what kinds of line drawings. After

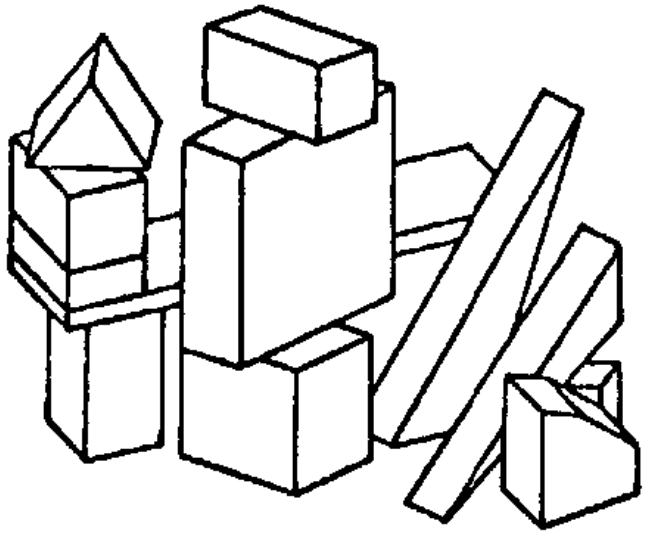


Figure 9.12: A scene analyzed by SEE. (Illustration used with permission of Adolpho Guzman.)

leaving MIT in 1967 to become a professor of Information and Computer Science at the University of California at Santa Cruz, he completed a theory for assigning labels to the lines in drawings of trihedral solids – objects in which exactly three planar surfaces join at each vertex of the object. The labels depended on the ways in which planes could come together at a vertex. (I got to know Huffman well at that time because he consulted frequently at the Stanford Research Institute.)

Huffman pointed out that there are only four ways in which three plane surfaces can come together at a vertex.[25] These are shown in Fig. 9.13. In addition to these four kinds of vertices, a scene might contain what Huffman called "T-nodes" – line intersection types caused by one object in a scene occluding another. These all give rise to a number of different kinds of labels for the lines in the scene; these labels specify whether the lines correspond to convex, concave, or occluding edges.

Huffman noted that the labels of the lines in a drawing might be locally consistent (around some vertices) but still be globally inconsistent (around all of the vertices). Consider, for example, Roger Penrose's famous line drawing of an "impossible object" shown in Fig. 9.14.[26] (It is impossible because no three-dimensional object, viewed in "general position," could produce this image.) No "real scene" can have a line with two different labels.

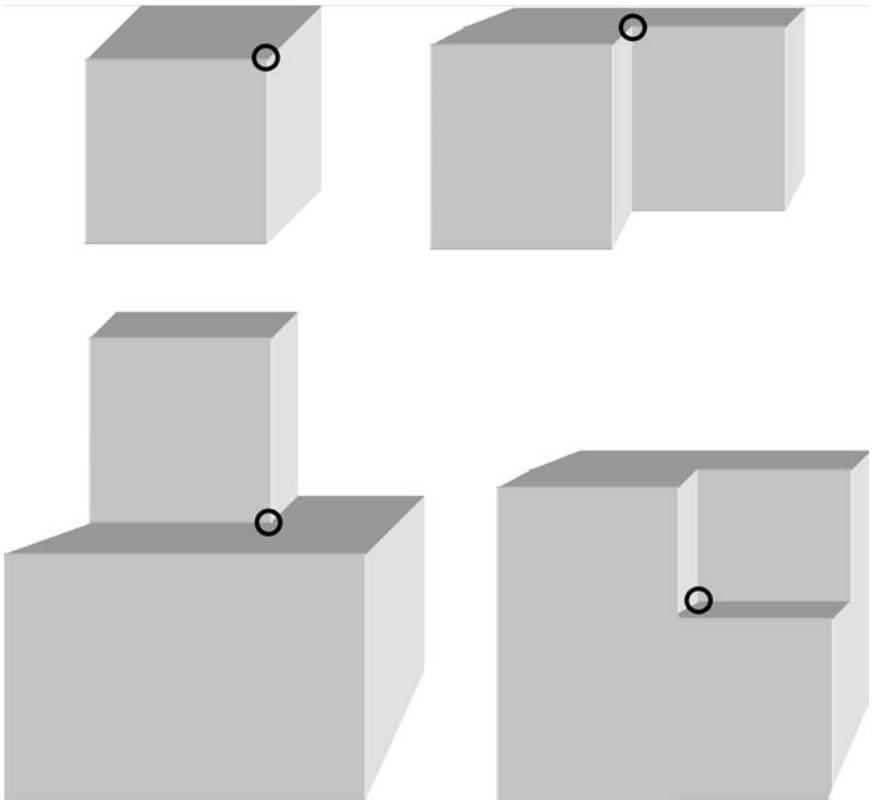


Figure 9.13: The four different kinds of vertices that can occur in trihedral solids.

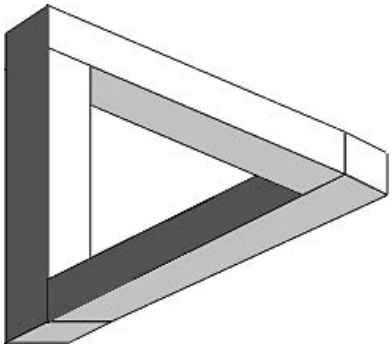


Figure 9.14: An impossible object.

Max Clowes (circa 1944–1981) of Sussex University in Britain developed similar ideas independently,[27] and the labeling scheme is now generally known as Huffman–Clowes labeling.

Next comes David Waltz (1943–). In his 1972 MIT Ph.D. thesis, he extended the Huffman–Clowes line-labeling scheme to allow for line drawings of scenes with shadows and possible "cracks" between two adjoining objects.[28] Waltz's important contribution was to propose and implement an efficient computational method for satisfying the constraint that all of the lines must be assigned one and only one label. (For example, an edge can't be concave at one end and convex at the other.) In Fig. 9.15, I show an example of a line drawing that Waltz's program could correctly segment into its constituents.

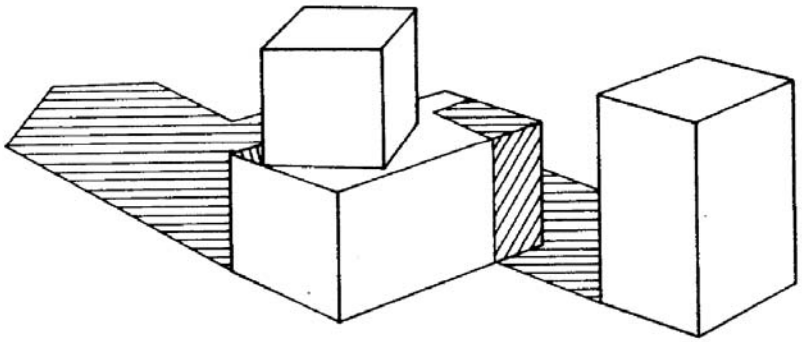


Figure 9.15: A scene with shadows analyzed by Waltz's program. (Illustration used with permission of David Waltz.)

Summarizing some of the work on processing line drawings at MIT, Patrick Winston says that "Guzman was the experimentalist, Huffman the theoretician, and Waltz the encyclopedist (because Waltz had to catalog thousands of junctions, in order to deal with cracks and shadows)."[29]

Meanwhile, similar work for finding, identifying, and describing objects in three-dimensional scenes was being done at Stanford. By 1972 Electrical Engineering Ph.D. student Gilbert Falk could segment scenes of line drawings into separate objects using techniques that were extensions of those of Guzman.[30] And by 1973, Computer Science Ph.D. student Gunnar Grape performed segmentation of scenes containing parallelepipeds and wedges using models of those objects.[31]

Other work on analysis of scenes containing polyhedra was done by Yoshiaki Shirai while he was visiting MIT's AI Lab[32] and by Alan Mackworth at the Laboratory of Experimental Psychology of the University of Sussex.[33]

Notes

1. For a thorough treatment, see David Forsyth and Jean Ponce, *Computer Vision: A Modern Approach*, Chapter 13, Upper Saddle River, NJ: Prentice Hall, 2003. [**169**]

2. Lettvin *et al.*, "What the Frog's Eye Tells the Frog's Brain," *Proceedings of the IRE*, Vol. 47, No. 11, pp. 1940–1951, 1959. [Reprinted as Chapter 7 in William C. Corning and Martin Balaban (eds.), *The Mind: Biological Approaches to Its Functions*, pp. 233–258, 1968.] [**171**]

3. David H. Hubel and Torsten N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *Journal of Physiology*, Vol. 160, pp. 106–154, 1962. [**171**]

4. David H. Hubel and Torsten N. Wiesel, "Receptive Fields and Functional Architecture of Monkey Striate Cortex, *Journal of Physiology*, Vol. 195, pp. 215–243, 1968. [**172**]

5. An interesting account of Hubel's and Wiesel's work and descriptions about how the brain processes visual images can be found in Hubel's online book *Eye, Brain, and Vision* at http://neuro.med.harvard.edu/site/dh/index.html. [**172**]

6. Horace B. Barlow and D. J. Tolhurst, "Why Do You Have Edge Detectors?," in *Proceedings of the 1992 Optical Society of America Annual Meeting*, Technical Digest Series, Vol. 23, pp. 172, Albuquerque, NM, Washington: Optical Society of America, 1992. [**172**]

7. Anthony J. Bell and Terrence J. Sejnowski, "Edges Are the 'Independent Components' of Natural Scenes," *Advances in Neural Information Processing Systems*, Vol. 9, Cambridge, MA: MIT Press, 1996. Available online at ftp://ftp.cnl.salk.edu/pub/tony/edge.ps.Z. [**172**]

8. Woodrow W. Bledsoe and Helen Chan, "A Man–Machine Facial Recognition System: Some Preliminary Results," Technical Report PRI 19A, Panoramic Research, Inc., Palo Alto, CA, 1965. [**172**]

9. Michael Ballantyne, Robert S. Boyer, and Larry Hines, "Woody Bledsoe: His Life and Legacy," *AI Magazine*, Vol. 17, No. 1, pp. 7–20, 1996. Also available online at http://www.utexas.edu/faculty/council/1998-1999/memorials/Bledsoe/bledsoe.html. [**172**]

10. Woodrow W. Bledsoe, "Semiautomatic Facial Recognition," Technical Report SRI Project 6693, Stanford Research Institute, Menlo Park, CA, 1968. [**172**]

11. Michael D. Kelly, Visual Identification of People by Computer, Stanford AI Project, Stanford, CA, Technical Report AI-130, 1970. [**173**]

12. http://iccv2007.rutgers.edu/TakeoKanadeResponse.htm. [**173**]

13. Lawrence G. Roberts, "Machine Perception of Three-Dimensional Solids," MIT Ph.D. thesis, 1963; published as Lincoln Laboratory Technical Report #315, May 22, 1963; appears in J. T. Tippett *et al.* (eds.), *Optical and Electro-Optical Information Processing*, pp. 159–197, Cambridge, MA: MIT Press, 1965. Available online at http://www.packet.cc/files/mach-per-3D-solids.html. [**173**]

14. The project is described in MIT's Artificial Intelligence Group Vision Memo No. 100 available at ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-100.pdf. [**176**]

15. Russell A. Kirsch *et al.*, "Experiments in Processing Pictorial Information with a Digital Computer," *Proceedings of the Eastern Joint Computer Conference*, pp. 221–229, Institute of Radio Engineers and Association Association for Computing Machinery, December 1957. [**176**]

16. According to Sobel, he and a fellow student, Gary Feldman, first presented the operator in a Stanford AI seminar in 1968. It was later described in Karl K. Pingle, "Visual Perception by a Computer," in A. Grasselli (ed.), *Automatic Interpretation and Classification of Images*, pp. 277–284, New York: Academic Press, 1969. It was also

mentioned in Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, pp. 271–272, New York: John Wiley & Sons, 1973. [**177**]

17. See, for example, M. H. Hueckel, "An Operator Which Locates Edges in Digitized Pictures," *Journal of the ACM*, Vol. 18, No. 1, pp. 113–125, January 1971, and Berthold K. P. Horn, "The Binford–Horn Line Finder," MIT AI Memo 285, MIT, July 1971 (revised December 1973 and available online at http://people.csail.mit.edu/bkph/AIM/AIM-285-OPT.pdf). [**178**]

18. David Marr and Ellen Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London*, Series B, Biological Sciences, Vol. 207, No. 1167, pp. 187–217, February 1980. [**178**]

19. John E. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions Pattern Analysis and Machine Intelligence*, Vol. 8, pp. 679–714, 1986. [**180**]

20. David Marr, "Early Processing of Visual Information," *Philosophical Transactions of the Royal Society of London*, Series B, Biological Sciences, Vol. 275, No. 942, pp. 483–519, October 1976. [**180**]

21. David Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, San Francisco: W.H. Freeman and Co., 1982. [**181**]

22. Guzman's 1968 Ph.D. thesis is titled "Computer Recognition of Three Dimensional Objects in a Visual Scene" and is available online at http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-059.pdf. [**181**]

23. Adolfo Guzman, "Decomposition of a Visual Scene into Three-Dimensional Bodies," *AFIPS*, Vol. 33, pp. 291–304, Washington, DC: Thompson Book Co., 1968. Available online as an MIT AI Group memo at ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-171.pdf. [**181**]

24. Personal communication, September 14, 2006. [**182**]

25. David A. Huffman, "Impossible Objects as Nonsense Sentences," in B. Meltzer and D. Michie (eds.), *Machine Intelligence 6*, pp. 195–234, Edinburgh: Edinburgh University Press, 1971, and David A. Huffman, "Realizable Configurations of Lines in Pictures of Polyhedra," in E. W. Elcock and D. Michie (eds.), *Machine Intelligence 8*, pp. 493–509, Chicester: Ellis Horwood, 1977. [**183**]

26. According to Wikipedia, this impossible object was first drawn by the Swedish artist Oscar Reutersvärd in 1934. [**183**]

27. Max B. Clowes, "On Seeing Things," *Artificial Intelligence*, Vol. 2, pp. 79–116, 1971. [**185**]

28. David L. Waltz, "Generating Semantic Descriptions from Drawings of Scenes with Shadows," MIT AI Lab Technical Report No. AITR-271, November 1, 1972. Available online at https://dspace.mit.edu/handle/1721.1/6911. A condensed version appears in Patrick Winston (ed.), *The Psychology of Computer Vision*, pp. 19–91, New York: McGraw-Hill, 1975. [**185**]

29. Personal communication, September 20, 2006. [**185**]

30. Gilbert Falk, "Computer Interpretation of Imperfect Line Data as a Three-Dimensional Scene," Ph.D. thesis in Electrical Engineering, Stanford University, Artificial Intelligence Memo AIM-132, and Computer Science Report No. CS180, August 1970. Also see Gilbert Falk, "Interpretation of Imperfect Line Data as a Three-Dimensional Scene," *Artificial Intelligence*, Vol. 3, pp. 101–144, 1972. [**185**]

31. Gunnar Rutger Grape, "Model Based (Intermediate Level) Computer Vision," Stanford Computer Science Ph.D. thesis, Artificial Intelligence Memo AIM-204, and Computer Science Report No. 266, May 1973. [**185**]

32. Yoshiaki Shirai, "A Heterarchical Program for Recognition of Polyhedra," MIT AI

Memo No. 263, June 1972. Available online at
ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-263.pdf. [**185**]

33. Alan K. Mackworth, "Interpreting Pictures of Polyhedral Scenes," *Artificial Intelligence*, Vol. 4, No. 2, pp. 121–137, June 1973. [**185**]

Chapter 10

"Hand–Eye" Research

The motivation for much of the computer vision research that I have described during this period was to provide information to guide a robot arm. Because the images that could be analyzed best were of simple objects such as toy blocks, work was concentrated on getting a robot arm to stack and unstack blocks. I'll describe some typical examples of this "hand–eye" research, beginning with a project that did not actually involve an "eye."

10.1 At MIT

A computer-guided mechanical "hand" was developed by Heinrich A. Ernst in 1961 as part of his Electrical Engineering Sc.D. work at MIT.[1] (His advisor was Claude Shannon.) The hand, named MH-1, was a "mechanical servomanipulator [an American Machine and Foundry model 8] adapted for operation by the TX-0 computer." It used tactile sensors mounted on the hand to guide it because, as Ernst wrote, "organs for vision are too difficult to build at the present time." The abstract of Ernst's thesis describes some of what the system could do:

> [O]ne program consisting of nine statements will make the hand do the following: Search the table for a box, remember its position, search the table for blocks, take them and put them into the box. The position of the objects is irrelevant as long as they are on the table. If as a test for the built-in mechanical intelligence, the box should be taken away and placed somewhere else while the hand searches for blocks, MH-1 will remember the new position of the box and continue to work with it as soon as it has realized the change in the situation, that is, has bumped into the box while looking for blocks. This will be done automatically, without any

need to mention it in the specific program for this block-and-box performance.

Actually, MH-1 was not the first computer-guided hand, although it was the first to employ touch sensors to guide its motion. One was developed and patented in 1954 by George Devol, an American engineer. Based on this invention, he and another engineer, Joseph F. Engelberger, founded Unimation, Inc. Soon after, they installed a prototype of their first industrial robot, called a "Unimate," in the General Motors Corporation Ternstedt Division plant near Trenton, New Jersey.

Back at MIT, Ph.D. students Patrick Winston (later an MIT professor and director of its AI Laboratory), Thomas O. Binford (later a Stanford professor), Berthold K. P. Horn (later an MIT professor), and Eugene Freuder (later a University of New Hampshire professor) developed a system that used an AMF Versatran robot arm to "copy" a configuration of blocks. The scene consisting of the blocks was first scanned, and lines were extracted from the image using a "line-finder," which was under development by Binford and Horn.[2] Using these lines, objects in the image were identified, and a plan was made for the robot arm to disassemble the blocks in the scene. The robot arm then carried out this plan and reassembled the blocks in their original configuration. The system was demonstrated in December 1970 for various configurations of blocks. An example block configuration successfully handled by their "copy demo" is shown in Fig. 10.1. (A film, called *Eye of the Robot*, showing the copy demo in action is available at http://projects.csail.mit.edu/films/aifilms/digitalFilms/9mpeg/88-eye.mpg.)

The system depended on precise illumination and carefully constructed blocks. Attempts to extend the range of computer vision to less constrained scenes led to further concentration at MIT and elsewhere on the early stages of vision processing. I'll describe some of the ensuing work on these problems in more detail later.[3]

10.2 At Stanford

Meanwhile at John McCarthy's SAIL, a team led by Professor Jerome Feldman (1938–) pursued work on hand–eye projects using the PDP-1 and later the PDP-6 and PDP-10 computers.[4] McCarthy later told me that he got interested in robots because of his interest in computer vision. He was not very excited about the work in pattern recognition – it was "discrimination" rather than "description." "If you want to pick something up, you have to describe it not merely recognize it."[5]

In 1966 SAIL had acquired a Rancho Los Amigos Hospital electromechanical prosthetic arm. By the spring of 1967, a hand–eye system was developed by Karl Pingle, Jonathan Singer, and Bill Wichman that could

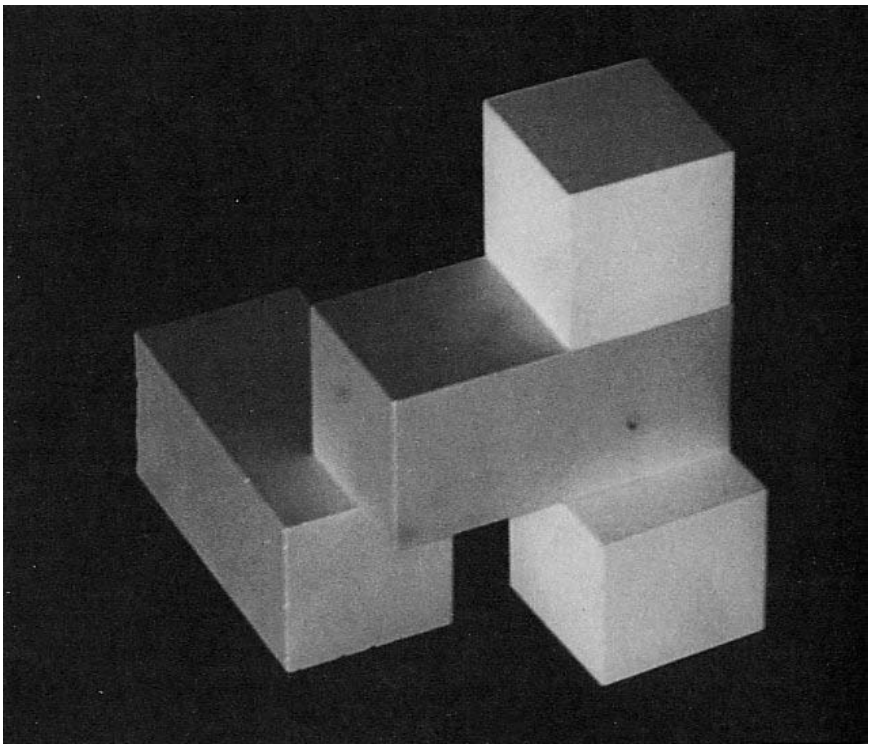


Figure 10.1: A block arrangement for the MIT copy demo. (Used with permission of Berthold Horn.)

use a TV camera and primitive vision routines to locate blocks scattered on a table. Using the information thus obtained, it could control the arm to sort the blocks.[6] According to the authors,

> One section of the system scans the TV image to find the outer edge of an object, then traces around the outside edges of the object using a gradient operator [an edge detector] to find the location and direction of the edge. Curve fitting routines fit straight lines to a list of points found on the edges and calculate the position of the corners. . . . A second section of the system is devoted to control of the arm. . . . The sections which control it consist of a solution program which calculates the angular position required at each actuator and a servo program which drives the arm to the desired positions. . .

Les Earnest wrote that this was "the first robotics visual feedback system," although "only the outer edges of the blocks were observed, [and] the

hand had to be removed from view when visual checking was done. . . ."[7]

Several versions of this block-sorting and stacking system were demonstrated. In one, the system located colored blocks on a table and placed them in separate stacks of red and blue blocks.[8]

By 1971, a vision-guided block stacking system solved the "instant insanity" puzzle.[9] In that puzzle, four cubes, each face of which has one of four different colors, must be arranged in a tower such that each side of the tower shows four different colors. The system, running on SAIL's PDP-10 computer, used a TV camera equipped with a turret of four lenses and a color wheel to locate four cubes on a table top. The arm picked up and turned each cube to expose all faces to the camera. Then, knowing the color of each face and having found a solution to the puzzle, the arm stacked the cubes in a tower exhibiting the solution. [A silent, 16-mm color six-minute film, titled *Instant Insanity*, was made by Richard Paul and Karl Pingle in August 1971 and shown at the second International Joint Conference on Artificial Intelligence (IJCAI) in London. The film can be seen at http://www.youtube.com/watch?v=O1oJzUSITeY.]

Dabblal Rajagopal "Raj" Reddy (1937– ; Fig. 10.2) was the first Ph.D. student of Stanford's new Department of Computer Science. His thesis research was on speech recognition. After obtaining his Ph.D. in 1966, Reddy joined Stanford's faculty and continued research on speech recognition at SAIL. While there he participated in a project to control a hand–eye system by voice commands.[10] As stated in a project review, "Commands as complicated as 'Pick up the small block in the lower lefthand corner,' are recognized and the tasks are carried out by the computer controlled arm." (The system was demonstrated in a 1969 fifteen-minute, 16-mm color, sound film showing some of Reddy's results on speech recognition. It is titled *Hear Here* and was produced by Raj Reddy, Dave Espar, and Art Eisensen. The film is available at http://www.archive.org/details/sailfilm_hear.) In 1969 Reddy moved to CMU where he pursued research in speech recognition and later became Dean of CMU's School of Computer Science.

In the early 1970s, the Stanford team used a vision system and a new electromechanical hand designed by mechanical engineering student Vic Scheinman[11] to assemble a model "T" Ford racing water pump.[12] An industrial-style setup was used with tools in fixed places, screws in a feeder, and the pump body and cover on a pallet. A diagram of the workspace is shown in Fig. 10.3.

The hand–eye system executed the following complex set of steps that was computed previously:

> locate the pump base, move it into standard position, determine the final grasping position by touch, place the pump base in its standard position, insert two pins to guide the alignment of the gasket and cover, put on the gasket, visually check the position of

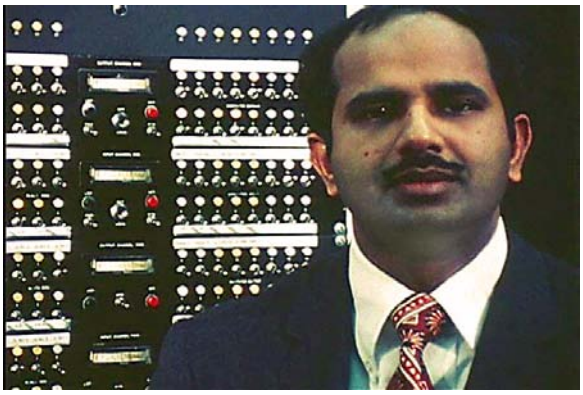


Figure 10.2: Raj Reddy. (Photograph courtesy of Raj Reddy.)

> the gasket, locate the cover by touch, put on the cover over the guide pins, pick up a hex head power screw driver, pick up a screw from the feeder, screw in the first two screws, remove the aligning pins, screw in the last four screws, and finally check the force required to turn the rotor.

A film of the water pump assembly can be seen at http://www.archive.org/details/sailfilm_pump. It is also available at http://www.saildart.org/films/, along with several other Stanford AI Lab films.

10.3 In Japan

Hand–eye work was also being pursued at Hitachi's Central Research Laboratory in Tokyo. There, Masakazu Ejiri and colleagues developed a robot system called HIVIP consisting of three subsystems called EYE, BRAIN, and HAND. (See Fig. 10.4.) One of EYE's two television cameras looked at a plan drawing depicting an assembly of blocks. The other camera looked at some blocks on a table. Then, BRAIN figured out how to pick up and assemble the blocks as specified in the drawing, and HAND did the assembly.[13]

10.4 Edinburgh's "FREDDY"

During the late 1960s and into the 1970s, researchers under the direction of Professor Donald Michie in the Department of Machine Intelligence and Perception at the University of Edinburgh developed robot systems generally

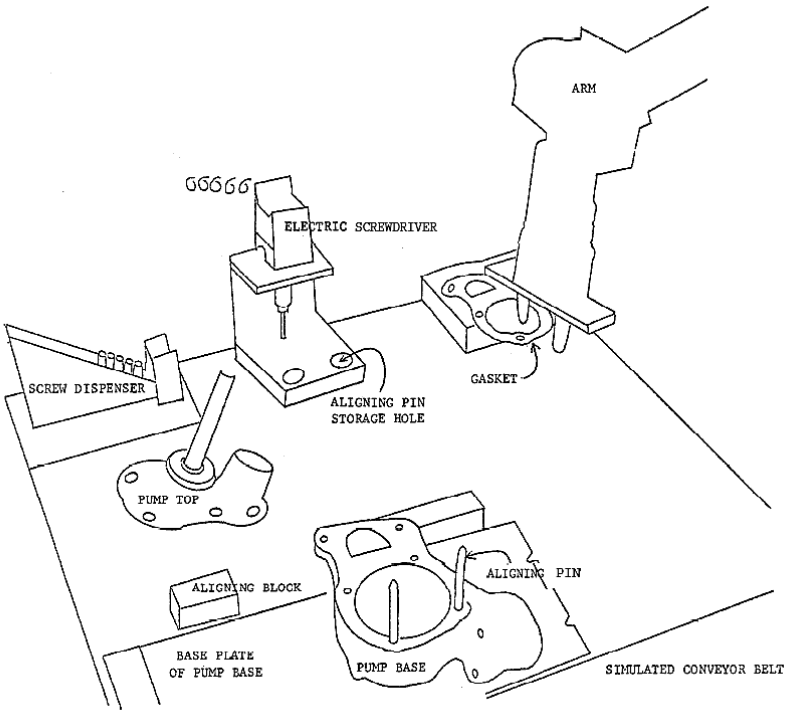


Figure 10.3: Diagram of a water pump assembly workspace. (Illustration used with permission of Robert Bolles.)

called "FREDDY."[14] The best known of these was the hand–eye system FREDDY II, which had a large robot arm and two TV cameras suspended over a moving table. Even though the arm did not move relative to the room, it did relative to its "world," the table. The setup is shown in Fig. 10.5.

A demonstration task for FREDDY II was to construct complete assemblies, such as a toy car or boat, from a kit of parts dumped onto the table. The aim was to develop AI techniques that could provide the basis for better industrial assembly robots, that is, robots that were more versatile, more reliable, and more easily programmed than those in operation at that time.

At the beginning, the component parts were in an unorganized jumble. FREDDY had to find and identify them and then lay them out neatly. Once it had found all the parts needed, FREDDY could then perform the assembly sequence, using a small workstation with a vice.

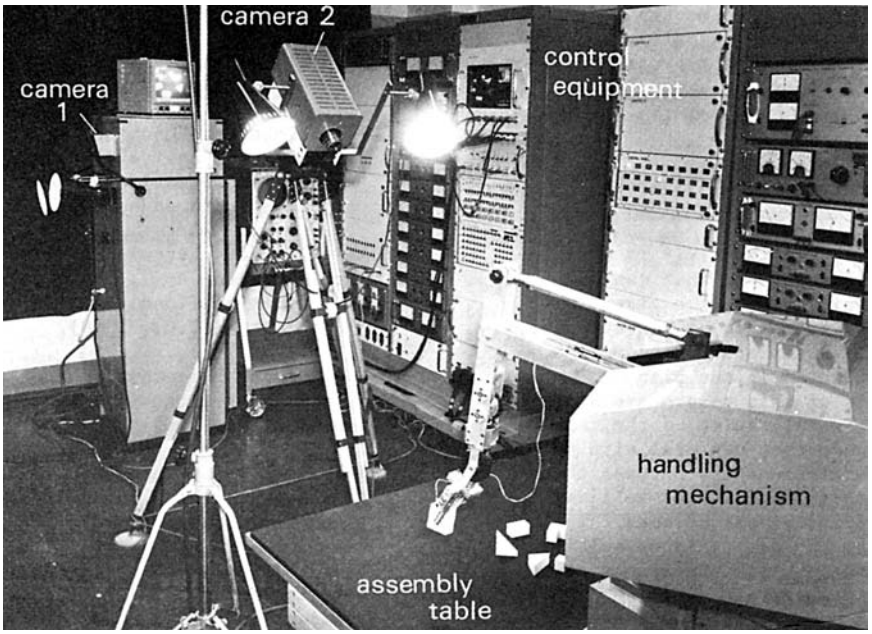


Figure 10.4: Hitachi's HIVIP robotic assembly system.

Isolated parts were recognized from features of their outline (corners, curves, etc.), their holes, and their general properties. These were taught to FREDDY by showing it different views of each of the parts in a prior training phase.

To deal with heaps of parts, FREDDY applied several tactics: It could try to find something protruding from the heap, which it could grasp and pull out; it could attempt to lift something (unknown) off the top; or it could simply plough the hand through the heap to try to split it into two smaller ones.

Constructing the assembly was performed by following a sequence of instructions that had been programmed interactively during the training phase. Some instructions were simple movements, but others were much more sophisticated and used the force sensors in the hand. For example, in a "constrained move," the hand would slide the part it held along a surface until it hit resistance; in "hole fitting," the hand would fit one part (such as an axle) into a hole in another (such as a wheel) by feel, as humans do.

FREDDY could assemble both the car and the boat when the two kits were mixed together and dumped on the table. It took about four hours to do so, primarily because of the limited power of FREDDY's two computers.[15] The main computer was an Elliot 4130 with 64k 24-bit words (later upgraded to 128k) and with a clock speed of 0.5 MHz. It was programmed in the POP-2

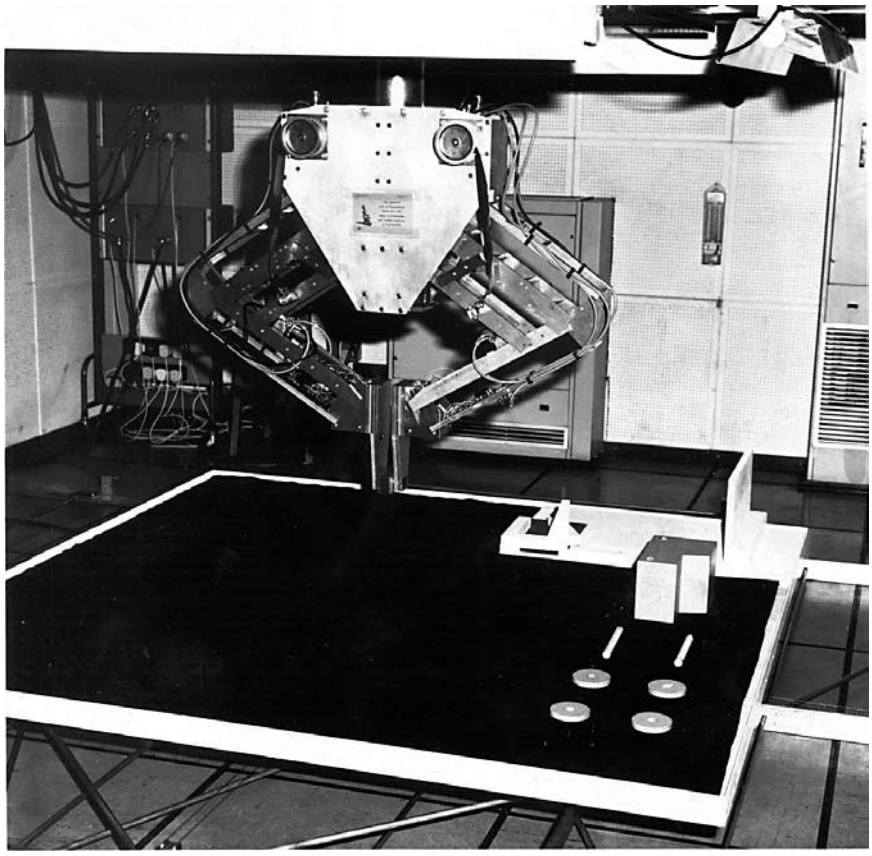


Figure 10.5: FREDDY II, the University of Edinburgh robot. (Photograph courtesy of University of Edinburgh.)

language. The robot motors and cameras were controlled by a Honeywell H316, with 4k 16-bit words (later upgraded to 8k words, at a cost of about $8,000 for the additional 8k bytes!).

Harry Barrow (1943–), a key person involved in the work, later gave this account of FREDDY's operation:[16]

> [A]cquiring an image from the TV camera took quite a few seconds and processing took even longer, and in a single run FREDDY took between 100 and 150 pictures! It took a picture every time it picked up an object to check it has successfully lifted it and not dropped it, and it took a picture every time it put an object down to verify the space was empty. It also scanned the entire world (which required multiple pictures) several times to make a map,

> and it looked at each object from two different cameras to do some stereo-style estimation of position and size. In fact, the system made the most intensive use of image data of any robot system in the world. The Stanford and MIT systems only took a very small number of pictures to perform their tasks, and relied heavily on dead reckoning and things not going wrong. We, on the other hand, assumed that things were likely to go wrong (objects dropped, rolling, etc.) and made our system highly robust. I really believe that in many ways it was probably the most advanced hand–eye system in existence at the time.

FREDDY is now on permanent exhibition in the Royal Scottish Museum in Edinburgh, with a continuous-loop movie of FREDDY assembling the mixed model car and boat kits.

Hand–eye research at Edinburgh was suspended during the mid-1970s in part owing to an unfavorable assessment of its prospects in a study commissioned by the British Science Research Council. (I'll have more to say about that assessment later.)

Notes

1. Heinrich A. Ernst, "MH-1, A Computer-Operated Mechanical Hand," Sc.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, 1962. Available online at http://dspace.mit.edu/bitstream/handle/1721.1/15735/09275630.pdf?sequence=1. [**189**]

2. Reported in Berthold K. P. Horn, "The Binford–Horn Line Finder," MIT AI Memo 285, MIT, July 1971 (revised December 1973). Available online at http://people.csail.mit.edu/bkph/AIM/AIM-285-OPT.pdf. [**190**]

3. Patrick Winston gives a nice description of the MIT programs just mentioned (including the copy demo and Winston's own thesis work on learning structural descriptions of object configurations) in Patrick H. Winston, "The MIT Robot," *Machine Intelligence 7*, Bernard Meltzer and Donald Michie (eds.), pp. 431–463, New York: John Wiley and Sons, 1972. [**190**]

4. For a brief review, see Lester Earnest (ed.), "Final Report: The First Ten Years of Artificial Intelligence Research at Stanford," Stanford Artificial Intelligence Laboratory Memo AIM-228 and Stanford Computer Science Department Report No. STAN-CS-74-409, July 1973. (Available online at http://www-db.stanford.edu/pub/cstr/reports/cs/tr/74/409/CS-TR-74-409.pdf.) For more details see Jerome A. Feldman *et al.*, "The Stanford Hand–Eye Project," *Proceedings of the IJCAI*, pp. 521–526, Washington, DC, 1969, and Jerome A. Feldman *et al.*, "The Stanford Hand-Eye Project – Recent Results," presented at IFIP Congress, Stockholm, 1974. [**190**]

5. John McCarthy, private communication, August 11, 2007. [**190**]

6. The system is described in Karl K. Pingle, Jonathan A. Singer, and William M. Wichman, "Computer Control of a Mechanical Arm through Visual Input," *Proceedings of the IFIP Congress (2)*, pp. 1563–1569, 1968. The vision part of the system is described in William Wichman, "Use of Optical Feedback in the Computer Control of an Arm," Stanford

Electrical Engineering Department Engineers thesis, August 1967 (and also appears as Stanford Artificial Intelligence Memo AIM-56, 1967.) [**191**]

7. Lester Earnest, *op. cit.*. [**192**]

8. *Butterfinger*, an 8-minute, 16-mm color film showing a version of this sorting system in operation was produced and directed by Gary Feldman in 1968. The film is available at http://projects.csail.mit.edu/films/aifilms/digitalFilms/1mp4/09-robot.mp4. [**192**]

9. The system is described in Jerome Feldman *et al.*, "The Use of Vision and Manipulation to Solve the 'Instant Insanity' Puzzle," *Proceedings of the IJCAI*, pp. 359–364, London: British Computer Society, September 1971. [**192**]

10. The system is described in Les Earnest *et al.*, "A Computer with Hands, Eyes, and Ears," *Proceedings of the 1968 Fall Joint Computer Conference*, Washington, DC: Thompson, 1968. [**192**]

11. Victor D. Scheinman, "Design of a Computer Manipulator," Stanford AI Memo AIM-92, June 1, 1969. [**192**]

12. This task is described in Robert Bolles and Richard Paul, "The Use of Sensory Feedback in a Programmable Assembly System," Stanford AI Laboratory Memo AIM-220, Stanford Computer Science Department Report STAN-CS-396, October 1973, which is available online at ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/73/396/CS-TR-73-396.pdf. [**192**]

13. See Masakazu Ejiri *et al.*, "An Intelligent Robot with Cognition and Decision-Making Ability," *Proceedings of the IJCAI*, pp. 350–358, London: British Computer Society, September 1971, and Masakazu Ejiri *et al.*, "A Prototype Intelligent Robot That Assembles Objects from Plan Drawings," *IEEE Transactions on Computers*, Vol. 21, No. 2, pp. 161–170, February 1972. [**193**]

14. This is an abbreviation, according to Donald Michie, of Frederick, an acronym of Friendly Robot for Education, Discussion and Entertainment, the Retrieval of Information, and the Collation of Knowledge. [**194**]

15. http://www.aiai.ed.ac.uk/project/freddy/. The key reference is A. P. Ambler, H. G. Barrow, C. M. Brown, R. M. Burstall, and R. J. Popplestone, "A Versatile Computer-Controlled Assembly System," *Artificial Intelligence*, Vol. 6, pp. 129–156, 1975. [**195**]

16. E-mail note from Harry Barrow of January 3, 2009. [**196**]

11.1

Chapter 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.

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

1. $(\forall \mathtt{x})[\mathtt{Man}(\mathtt{x}) \supset \mathtt{Mortal}(\mathtt{x})]$

   (The expression "$(\forall \mathtt{x})$" is a way of writing "for all x"; and the expression "$\supset$" is a way of writing "implies that." "$\mathtt{Man}(\mathtt{x})$" is a way of writing "x is a man"; and "$\mathtt{Mortal}(\mathtt{x})$" 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 $\mathtt{P}$ and $\mathtt{P} \supset \mathtt{Q}$, then we are justified in deducing the statement $\mathtt{Q}$.

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 $\neg \mathtt{P} \vee \mathtt{Q}$ and $\mathtt{P}$ produces $\mathtt{Q}$. (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 $\mathtt{P}$ and the $\neg \mathtt{P}$ in the two statements. (Resolution is a kind of generalized *modus ponens* as can be seen from the fact that $\neg \mathtt{P} \vee \mathtt{Q}$ is logically equivalent to $\mathtt{P} \supset \mathtt{Q}$.) 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 $\neg \mathtt{P}(\mathtt{x}) \vee \mathtt{Q}(\mathtt{x})$ 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 ∨'s and ¬'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. $(\forall x)[\texttt{MACHINE}(x) \supset \neg\texttt{ANIMAL}(x)]$

   (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. $(\forall x)\texttt{ANIMAL}(x)$

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

4. $x = \texttt{Rob}$

   (This indicates that $\neg\texttt{ANIMAL}(\texttt{Rob})$ 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

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

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

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

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

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

Here "$(\exists \mathtt{s})$" 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 $(\exists \mathtt{s})\mathtt{On}(\mathtt{A}, \mathtt{B}, \mathtt{s})$, 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 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.

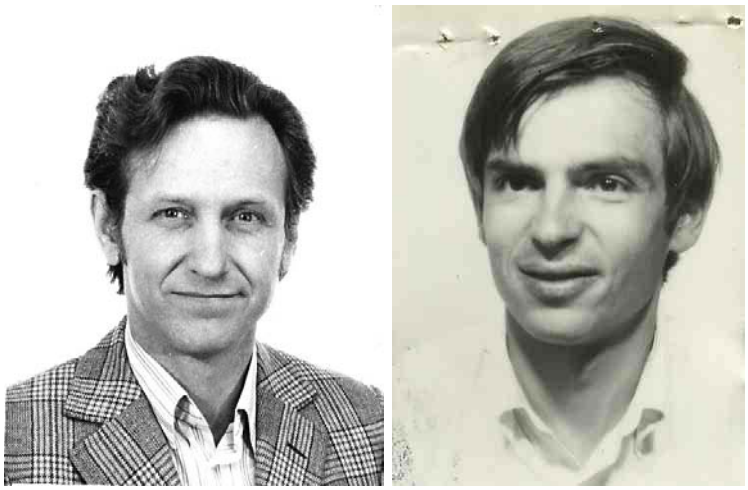


Figure 11.1: Robert Kowalski (left) and Alain Colmerauer (right). (Photographs courtesy of Robert Kowalski and of Alain Colmerauer.)

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 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 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]

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

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.

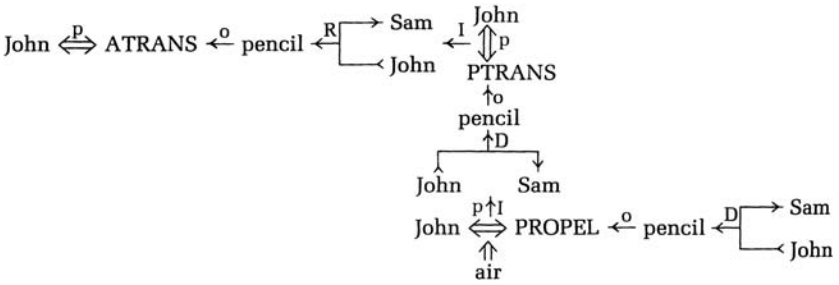


Figure 11.3: Conceptual structure for "John threw the pencil to Sam." (From Roger C. Schank, "Identification of Conceptualizations Underlying Natural Langauge," in Roger Schank and Kenneth Colby (eds.), *Computer Models of Thought and Language*, p. 226, San Francisco: W. H. Freeman and Co., 1973.)

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.

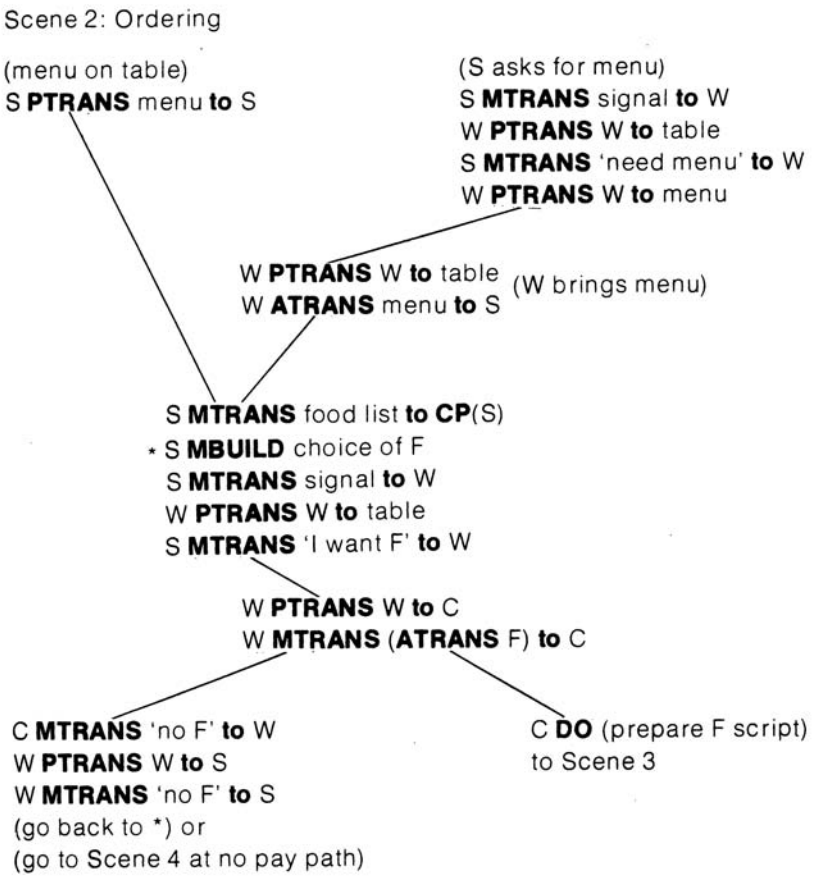


Figure 11.4: A scene in the restaurant script. (From Roger C. Schank and Robert P. Abelson, *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*, p. 43, Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.)

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 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.

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. 134), and 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. [**200**]

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. [**200**]

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. [**200**]

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. [**200**]

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. [**200**]

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. [**201**]

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. [**201**]

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/. [**201**]

9. William McCune, "Solution of the Robbins Problem," *Journal of Automated Reasoning*, Vol. 19, No. 3, pp. 263–276, 1997. [**201**]

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. [**202**]

11. Robert A. Kowalski and Donald Kuehner, "Linear Resolution with Selection Function," *Artificial Intelligence*, Vol. 2, Nos. 3–4, pp. 227–260, 1971. [**204**]

12. From one of Kowalski's Web pages: http://www.doc.ic.ac.uk/~rak/history.html. [**204**]

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/HistoireProlog/19november92.pdf. [**204**]

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. [**205**]

15. John R. Anderson and Gordon H. Bower, *Human Associative Memory*, Washington, DC: Winston and Sons, 1973. [**205**]

16. From the Web site http://rumelhartprize.org/john.htm. [**205**]

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. [**205**]

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. [**205**]

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. [**205**]

20. The SNePS Web page is at http://www.cse.buffalo.edu/sneps/. [**206**]

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. [**206**]

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. [**206**]

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. [**206**]

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. [**207**]

25. *Ibid.* [**209**]

26. Roger C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge: Cambridge University Press, 1982. [**209**]

27. Roger C. Schank, *Dynamic Memory Revisited*, Cambridge: Cambridge University Press, 1999. [**209**]

28. See http://www.edge.org/3rd_culture/bios/schank.html. [**209**]

29. See http://aigp.csres.utexas.edu/~aigp/researcher/show/192. [**209**]

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. [**209**]

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. [**210**]

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. [**210**]

Chapter 12

Mobile Robots

The hand–eye systems described earlier might be thought of as "robots," but they could not move about from their fixed base. Up to this time, very little work had been done on mobile robots even though they figured prominently in science fiction. I have already mentioned Grey Walter's "tortoises," which were early versions of autonomous mobile robots. In the early 1960s researchers at the Johns Hopkins University Applied Physics Laboratory built a mobile robot they called "The Beast." (See Fig. 12.1.) Controlled by on-board electronics and guided by sonar sensors, photocells, and a "wallplate-feeling" arm, it could wander the white-walled corridors looking for dark-colored power plugs. Upon finding one, and if its batteries were low, it would plug itself in and recharge its batteries. The system is described in a book by Hans Moravec.[1]

Beginning in the mid-1960s, several groups began working on mobile robots. These included the AI Labs at SRI and at Stanford. I'll begin with an extended description of the SRI robot project for it provided the stimulus for the invention and integration of several important AI technologies.

12.1 Shakey, the SRI Robot

In November 1963, Charles Rosen, the leader of neural-network research at SRI, wrote a memo in which he proposed development of a mobile "automaton" that would combine the pattern-recognition and memory capabilities of neural networks with higher level AI programs – such as were being developed at MIT, Stanford, CMU, and elsewhere. Rosen had previously attended a summer course at UCLA on LISP given by Bertram Raphael, who was finishing his Ph.D. (on SIR) at MIT.

Rosen and I and others in his group immediately began thinking about mobile robots. We also enlisted Marvin Minsky as a consultant to help us.

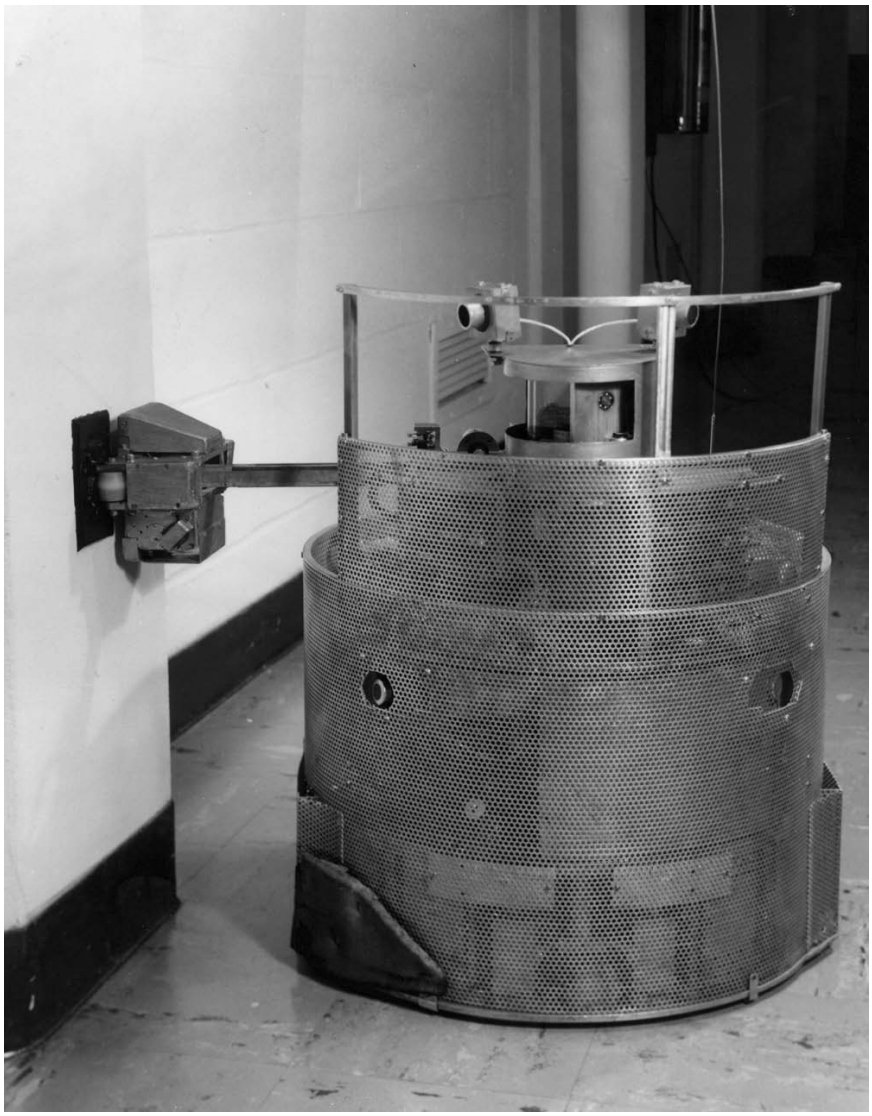


Figure 12.1: The Johns Hopkins "Beast." (Photograph courtesy of the Johns Hopkins University Applied Physics Laboratory.)

Minsky spent two weeks at SRI during August 1964. We made the first of many trips to the ARPA office (in the Pentagon at that time) to generate interest in supporting mobile robot research at SRI. We also talked with Ruth Davis, the director of the Department of Defense Research and Engineering (DDR&E) – the office in charge of all Defense Department research. We wrote

a proposal in April 1964 to DDR&E for "Research in Intelligent Automata (Phase I)" that would, we claimed, "ultimately lead to the development of machines that will perform tasks that are presently considered to require human intelligence."[2] The proposal, along with several trips and discussions culminated, in November 1964, in a "work statement" issued by the then-director of ARPA's Information Processing Techniques Office, Ivan Sutherland. The excerpt in Fig. 12.2 describes the goals of the program.[3]

A RESEARCH AND DEVELOPMENT PROGRAM IN APPLICATIONS OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

Goals

The long-range goal of this program will be to develop automatons capable of gathering processing and transmitting information in a hostile environment. The time period involved is 1970-1980.

The first short-range goal of the program will be to design and develop a mobile automaton to accomplish non-trivial missions in a real environment. External control will be exercised over the automaton from a computer. The automaton will have at least visual and tactile sensor capability.

The second short-range goal will be to design and develop a mobile automaton to accomplish non-trivial missions in a real environment in a self-contained mode, e.g., with little or no external control.

.....

Such a long-range goal attained by stepping through a number of intermediary ones is believed essential to knit together as many of the constituent subject areas of "artificial intelligence" as possible. It has been so stated as to require the successful application of many techniques with all the attendant problems of interaction and feed-back. It is difficult of realization but by the same token it will provide solutions to existing pressing military problems.

Figure 12.2: Excerpt from the typescript of the automaton work statement.

In the meantime, Bertram Raphael completed his MIT Ph.D. degree in 1964 and took up a position at UC Berkeley for an academic year. In April 1965, he accepted our offer to join SRI to provide our group with needed AI expertise. After several research proposal drafts and discussions with people in the relevant offices in the Defense Department (complicated by the fact that Ivan Sutherland left ARPA during this time), SRI was finally awarded a rather large (for the time) contract based essentially on Sutherland's work statement. The "start-work" date on the project, which was administered for ARPA by the Rome Air Development Center (RADC) in Rome, New York, was March

17, 1966. (Coincidentally, just before joining SRI in 1961, I had just finished a three-year stint of duty as an Air Force Lieutenant at RADC working on statistical signal-processing techniques for radar systems.) Ruth Davis played a prominent role in getting ARPA and RADC to move forward on getting the project started. The "knitting together" of several disparate AI technologies was one of the primary challenges and one of the major contributions of SRI's automaton project.[4]

One of the tasks was the actual construction of a robot vehicle whose activities would be controlled by a suite of programs. Because of various engineering idiosyncrasies, the vehicle shook when it came to an abrupt stop. We soon called it "Shakey," even though one of the researchers thought that sobriquet too disrespectful. [Shakey was inducted into the "Robot Hall of Fame" (along with C-3PO among others) in 2004.[5] It was also named as the fifth-best robot ever (out of 50) by *Wired Magazine* in January 2006. *Wired*'s numbers 2 and 4 were fictional, "Spirit" and "Opportunity" (the Mars robots) were number 3, and "Stanley" (winner of the 2005 DARPA "Grand Challenge") was named "the #1 Robot of All Time." Shakey is now exhibited at the Computer History Museum in Mountain View, California.][6]

Shakey had an on-board television camera for capturing images of its environment, a laser range finder (triangulating, not time-of-flight) for sensing its distance from walls and other objects, and cat-whisker-like bump detectors. Shakey's environment was a collection of "rooms" connected by doorways but otherwise separated by low walls that we could conveniently see over but Shakey could not. Some of the rooms contained large objects, as shown in Fig. 12.3. The size of Shakey can be discerned from inspection of Fig. 12.4.

Most of the programs that we developed to control Shakey were run on a DEC PDP-10 computer. Between the PDP-10 and the mobile vehicle itself were a PDP-15 peripheral computer (for handling the lower level communications and commands to on-board hardware) and a two-way radio and video link. The PDP-10 programs were organized in what we called a "three-layer" hierarchy. Programs in the lowest level drove all of the motors and captured sensory information. Programs in the intermediate level supervised primitive actions, such as moving to a designated position, and also processed visual images from Shakey's TV camera. Planning more complex actions, requiring the execution of a sequence of intermediate-level actions, was done by programs in the highest level of the hierarchy. The Shakey project involved the integration of several new inventions in search techniques, in robust control of actions, in planning and learning, and in vision. Many of these ideas are widely used today. The next few subsections describe them.

12.1.1 A*: A New Heuristic Search Method

One of the first problems we considered was how to plan a sequence of "way points" that Shakey could use in navigating from place to place. In getting

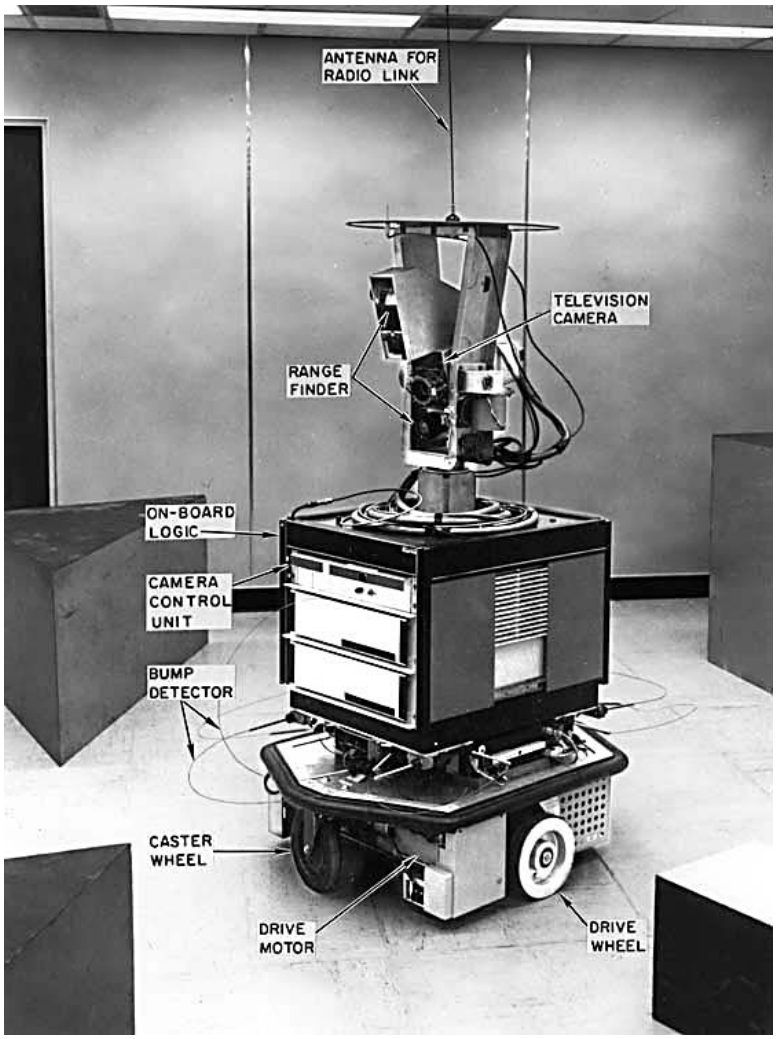


Figure 12.3: Shakey as it existed in November 1968 (with some of its components labeled). (Photograph courtesy of SRI International.)

around a single obstacle lying between its initial position and a goal position, Shakey should first head toward a point near an occluding boundary of the obstacle and then head straight for the unobstructed final goal point. However, the situation becomes more complicated if the environment is littered with several obstacles, and we sought a general solution to this more difficult problem.

Shakey kept information about the location of obstacles and about its

Figure 12.4: Charles A. Rosen with Shakey. (Photograph courtesy of SRI International.)

own position in a "grid model," such as the one shown in Fig. 12.5. (To obtain the required accuracy, grid cells were decomposed into smaller cells near the objects. I think this was one of the first applications of adaptive cell

decomposition in robot motion planning and is now a commonly used technique.) Consider, for example, the navigation problem in which Shakey is at position R and needs to travel to G (where R and G are indicated by the shaded squares). It can use a computer representation of the grid model to plan a route before beginning its journey – but how? The map shows the positions of three objects that must be avoided. It is not too difficult to compute the locations of some candidate way points near the corners of the objects. (These way points must be sufficiently far from the corners so that Shakey wouldn't bump into the objects.) The way points are indicated by shaded stars and labeled "A," "B," and so on through "K." Using techniques now familiar in computer graphics, it also is not difficult to compute which way points are reachable using an obstacle-free, straight-line path from any other way point and from R and G.

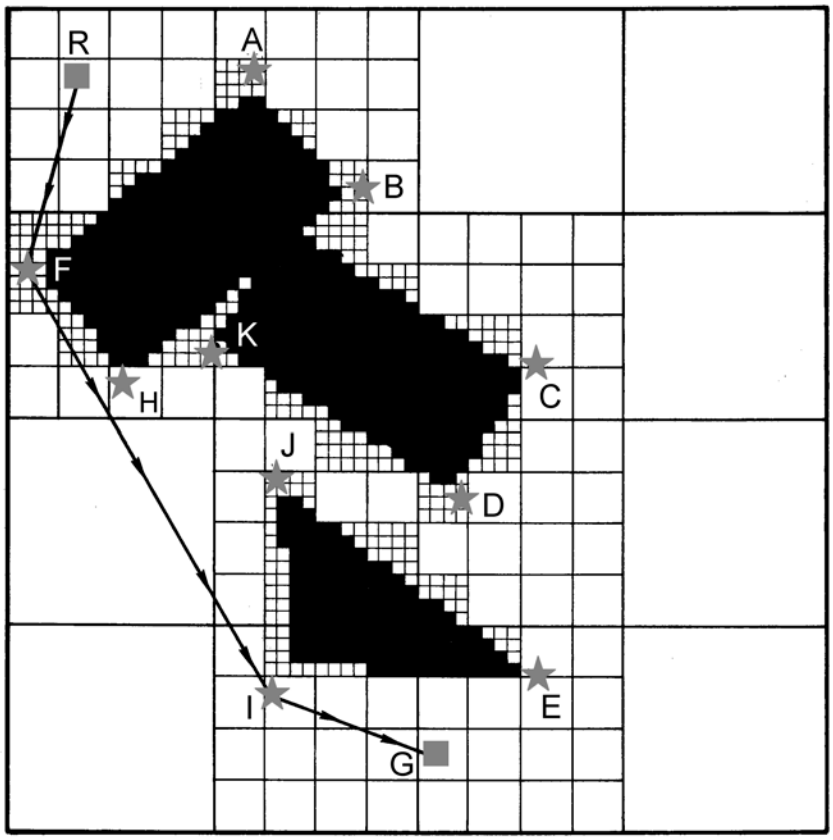


Figure 12.5: A navigation problem for Shakey. (Illustration used with permission of SRI International.)

Looked at in this way, Shakey's navigation problem is a search problem, similar to ones I have mentioned earlier. Here is how a search tree can be constructed and then searched for a shortest path from R to G. First, because A and F are directly reachable by obstacle-free, straight-line paths from R, these are set up as direct-descendant "nodes" of R in the search tree. We continue the process of computing descendant nodes (along obstacle-free, straight-line paths) from each of A and F and so on until G is added to the tree. Then, it is a simple matter to identify the shortest path from R to G.

Several methods for searching trees (and their more general cousins, graphs) were already in use by the mid-1960s. One point in favor of these known methods was that they were guaranteed to find shortest paths when used to solve Shakey's navigation problems. However, they could be computationally inefficient for difficult problems. Of course, solving simple navigation problems (such as the one in the diagram) does not involve much search, so any search method would solve such problems quickly. But we were interested in general methods that would work efficiently on larger, more difficult problems. I was familiar with the heuristic search method proposed by J. Doran and Donald Michie for solving the eight-piece, sliding-tile puzzle. They assigned a numerical value to each node in the search tree, based on the estimated difficulty of reaching the goal from that node. The node with the lowest score was the one that was selected next to have its descendants generated.[7]

I reasoned that a good "heuristic" estimate for the difficulty of getting from a way point position to the goal (before actually searching further) would be the "airline distance," ignoring any intervening obstacles, from that position to the goal. I suggested that we use that estimate as the score of the corresponding node in the search tree. Bertram Raphael, who was directing work on Shakey at that time, observed that a better value for the score would be the sum of the distance traveled so far from the initial position plus my heuristic estimate of how far the robot had to go.[8]

Raphael and I described this idea to Peter Hart, who had recently obtained a Ph.D. from Stanford and joined our group at SRI. Hart recalls[9] "going home that day, sitting in a particular chair and staring at the wall for more than an hour, and concluding" that if the estimate of remaining distance (whatever it might be) was never larger than the actual remaining distance, then the use of such an estimate in our new scoring scheme would *always* find a path having the shortest distance to the goal. (Of course, my heuristic airline distance satisfied Hart's more-general condition.) Furthermore, he thought such a procedure would generate search trees no larger than any other procedures that were also guaranteed to find shortest paths and that used heuristic estimates no better than ours.

Together, Hart, Raphael, and I were able to construct proofs for these claims, and we named the resulting search process "A*." (The "A" was for algorithm and the "$*$" denoted its special property of finding shortest paths.

I think Hart and Raphael did most of the heavy lifting in devising the proofs.) When paths have costs associated with them that depend on more than just distance, and when such costs (rather than distances) are taken into account in computing scores, A* is guaranteed to find lowest cost paths.[10]

The inclusion of the estimate of remaining distance (or cost) to the goal contributes to searching in the general direction of the goal. The inclusion of the actual distance (or cost) incurred so far ensures that the search process will not forever be led down promising but perhaps futile paths and will be able to "leak around" obstacles.

A* has been extended in many ways – especially by Richard Korf to make it more practical when computer memory is limited.[11] Today, A* is used in many applications including natural language parsing,[12] the computation of driving directions,[13] and interactive computer games.[14]

12.1.2 Robust Action Execution

The A* algorithm was embedded in Shakey's programs for navigating from one place to another within a room containing obstacles and for pushing an object from one place to another. Navigation programs, along with others, occupied the middle level of the hierarchy of Shakey's programs. These intermediate-level programs were all designed to achieve certain goals, such as getting an object in front of a doorway for example. They were also quite robust in that they "kept trying" even in the face of unforeseen difficulties. For example, if an object being pushed happened accidentally to slip off the front "pushing bar," the push program noticed this problem (through built-in contact sensors in the pushing bar) and repositioned Shakey so that it could reengage the object and continue pushing.

In thinking about how to achieve this robustness, I was inspired both by Miller, Galanter, and Pribram's TOTE units and by the idea of homeostasis. (Recall that a TOTE unit for driving in a nail keeps pounding until the nail is completely driven in and that homeostatic systems take actions to return them to stability in the face of perceived environmental disturbances.) I wanted the mid-level programs to seek and execute that action that was both "closest" to achieving their goals and that could actually be executed in the current situation. If execution of that action produced a situation in which, as anticipated, an action even closer to achieving the goal could be executed, fine; the mid-level program was at least making progress. If not, or something unexpected caused a setback, some other action would be executed next to get back on track. Richard Duda and I developed a format, called "Markov tables," for writing these intermediate-level programs having this "keep-trying" property.[15]

12.1.3 STRIPS: A New Planning Method

The mid-level programs could accomplish a number of simple tasks, such as getting Shakey from one place to another in the same room, pushing objects, and getting Shakey through a doorway into an adjoining room. However, to go to some distant room and push an object there into some designated position would require joining together a sequence of perhaps several of these mid-level programs. Just as humans sometimes make and then execute plans for accomplishing their tasks, we wanted Shakey to be able to assemble a plan of actions and then to execute the plan. The plan would consist of a list of the programs to be executed.

Information needed for planning was stored in what was called an "axiom model." This model contained logical statements in the language of the predicate calculus (which I talked about earlier.) For example, Shakey's location was represented by a statement such as `AT(ROBOT, 7,5)`, the fact that Box1 was pushable was represented by the statement `PUSHABLE(BOX1)`, and the fact that there was a doorway named D1 between rooms R1 and R2 was represented by the statement `JOINSROOMS(D1, R1,R2)`. The axiom model had close to two-hundred statements such as these and was the basis of Shakey's reasoning and planning abilities.

Our first attempt at constructing plans for Shakey used the QA3 deduction system and the situation calculus. We would ask QA3 to prove (using a version of the axiom model) that there existed a situation in which Shakey's goal (for example, being in some distant room) was true. The result of the deduction (if successful) would name that situation in terms of a list of mid-level actions to be executed.[16]

The use of the situation calculus for planning how to assemble mid-level actions involved using logical statements to describe the effects of these actions on situations. Not only did we have to describe how a mid-level action changed certain things about the world, but we also had to state that it left many things unaffected. For example, when Shakey pushed an object, the position of that object in the resulting situation was changed, but the positions of all other objects were not. That most things in Shakey's world did not change had to be explicitly represented as logical statements and, worse, reasoned about by QA3. This difficulty, called the "frame problem," has been the subject of a great deal of research in AI, and there have been many attempts to mitigate it, if not solve it.[17] Because of the frame problem, QA3 could be used only for putting together the simplest two- or three-step plans. Any attempt to generate plans very much longer would exhaust the computer's memory.

The problem with the situation calculus (as it was used then) was that it assumed that all things might change unless it was explicitly stated that they did not change. I reasoned that a better convention would be to assume that all things remained unchanged unless it was explicitly stated that they did change. To employ a convention like that, I proposed a different way of

updating the collection of logical statements describing a situation. The idea was that certain facts, specifically those that held before executing the action but might not hold after, should be deleted and certain new facts, namely, those caused by executing the action, should be added. All other facts (those not slated for deletion) should simply be copied over into the collection describing the new situation. Besides describing the *effects* of an action in this way, each action description would have a *precondition*, that is, a statement of what had to be true of a situation to be able to execute the action in that situation. (A year or so earlier, Carl Hewitt, a Ph.D. student at MIT, was developing a robot programming language called PLANNER that had mechanisms for similar kinds of updates.)[18]

For example, to describe the effects of the program `goto((X1,Y1), (X2,Y2))` for moving Shakey from some position (X1,Y1) to some position (X2,Y2), one should delete the logical statement `AT(ROBOT, X1,Y1)`, add the statement `AT(ROBOT, X2,Y2)`, and keep all of the other statements. Of course, to execute `goto((X1,Y1), (X2,Y2))`, Shakey would already have to be at position (X1,Y1); that is, the axiom model had to contain the precondition statement `AT(ROBOT,X1,Y1)`, or at least contain statements from which `AT(ROBOT,X1,Y1)` could be proved.

Around this time (1969), Richard Fikes (1942–) had just completed his Ph.D. work under Allen Newell at Carnegie and joined our group at SRI. Fikes's dissertation explored some new ways to solve problems using procedures rather than using logic as in QA3. Fikes and I worked together on designing a planning system that used preconditions, delete lists, and add lists (all expressed as logical statements) to describe actions. Fikes suggested that in performing a search for a goal-satisfying sequence of actions, the system should use the "means–ends" analysis heuristic central to Newell, Shaw, and Simon's General Problem Solver (GPS). Using means–ends analysis, search would begin by identifying those actions whose add lists contained statements that helped to establish the goal condition. The preconditions of those actions would be set up as subgoals, and this backward reasoning process would continue until a sequence of actions was finally found that transformed the initial situation into one satisfying the goal.

By 1970 or so, Fikes had finished programming (in LISP) our new planning system. We called it STRIPS, an acronym for Stanford Research Institute Problem Solver.[19] After its completion, STRIPS replaced QA3 as Shakey's system for generating plans of action. Typical plans consisting of six or so mid-level actions could be generated on the PDP-10 in around two minutes.

The STRIPS planning system itself has given way to more efficient AI planners, but many of them still describe actions in terms of what are called "STRIPS operators" (sometimes "STRIPS rules") consisting of preconditions, delete lists, and add lists.

12.1.4 Learning and Executing Plans

It's one thing to make a plan and quite another to execute it properly. Also, we wanted to be able to save the plans already made by STRIPS for possible future use. We were able to come up with a structure, called a "triangle table," for representing plans that was useful not only for executing plans but also for saving them. (John Munson originally suggested grouping the conditions and effects of robot actions in a triangular table. Around 1970, Munson, Richard Fikes, Peter Hart, and I developed the triangle table formalism to represent plans consisting of STRIPS operators.) The triangle table tabulated the preconditions and effects of each action in the plan so that it could keep track of whether or not the plan was being executed properly.

Actions in the plans generated by STRIPS had specific values for their parameters. For example, if some `goto` action was part of a plan, actual place coordinates were used to name the place that Shakey was to go from and the place it was to go to, perhaps `goto((3,7),(8,14))`. Although we might want to save a plan that had that specific `goto` as a component, a more generally applicable plan would have a `goto` component with nonspecific parameters that could be replaced by specific ones depending on the specific goal. That is, we would want to generalize something like `goto((3,7),(8,14))`, for example, to `goto((x1,y1),(x2,y2))`. One can't willy-nilly replace constants by variables, but one must make sure that any such generalizations result in viable and executable plans for all values of the variables. We were able to come up with a procedure that produced correct generalizations, and it was these generalized plans that were represented in the triangle table.

After a plan was generated, generalized, and represented in the triangle table, Shakey's overall executive program, called "PLANEX," supervised its execution.[20] In the environment in which Shakey operated, plan execution would sometimes falter, but PLANEX, using the triangle table, could decide how to get Shakey back on the track toward the original goal. PLANEX gave the same sort of "keep-trying" robustness to plan execution that the Markov tables gave to executing mid-level actions.

12.1.5 Shakey's Vision Routines

Shakey's environment consisted of the floor it moved about on, the walls bounding its rooms, doorways between the rooms, and large rectilinear objects on the floor in some of the rooms. We made every effort to make "seeing" easy for Shakey. A dark baseboard separated the light-colored floor from the light-colored walls. The objects were painted various shades of red, which appeared dark to the vidicon camera and light to the infrared laser range finder. Even so, visual processing still presented challenging problems.

Rather than attempt complete analyses of visual scenes, our work concentrated on using vision to acquire specific information that Shakey

needed to perform its tasks. This information included Shakey's location and the presence and locations of objects – the sort of information that was required by the mid-level actions. The visual routines designed to gather that information were embedded in the programs for performing those actions. Known properties of Shakey's environment were exploited in these routines.

Exploiting the fact that the objects, the floor, and the wall contained planes of rather constant illumination, Claude Brice and Claude Fennema in our group developed image-processing routines that identified regions of uniform intensity in an image.[21] Because the illumination on a single plane, say the face of an object, might change gradually over the region, the region-finding routine first identified rather small regions. These were then merged across region boundaries in the image if the intensity change across the boundary was not too great. Eventually, the image would be partitioned into a number of large regions that did a reasonable job of representing the planes in the scene. The boundaries of these regions could then be fitted with straight-line segments.

Another vision routine was able to identify straight-line segments in the image directly. Richard Duda and Peter Hart developed a method for doing this based on a modern form of the "Hough transform."[22] After edge-detection processing had identified the locations and directions of small line segments, the Hough transform was used to construct those longer lines that were statistically the most likely, given the small line segments as evidence.

Both region finding and line detection were used in various of the vision routines for the mid-level actions. One of these routines, called `obloc`, was used to refine the location of an object whose location was known only roughly. The pictures in Fig. 12.6 show a box, how it appears as a TV image from Shakey's camera, and two of the stages of `obloc`'s processing. From the regions corresponding to the box and the floor (and using the fact that Shakey is on the same floor as the box), straightforward geometric computations could add the box and its location to Shakey's models.

Shakey ordinarily kept track of its location by dead-reckoning (counting wheel revolutions), but this estimate gradually accumulated errors. When Shakey determined that it should update its location, it used another vision routine, called `picloc`. A nearby "landmark," such as the corner of a room, was used to update Shakey's position with respect to the landmark. The pictures in Fig. 12.7 show how `obloc` traces out the baseboard and finds the regions corresponding to the walls and the floor. The final picture shows the discrepancy between Shakey's predicted location of the corner (based on Shakey's estimate of its own location) and the actual location based on `picloc`. This discrepancy was used to correct Shakey's estimate of its position.

Before Shakey began a straight-line motion in a room where the presence of obstacles might not be known, it used a routine called `clearpath` to determine whether its path was clear. This routine checked the image of its

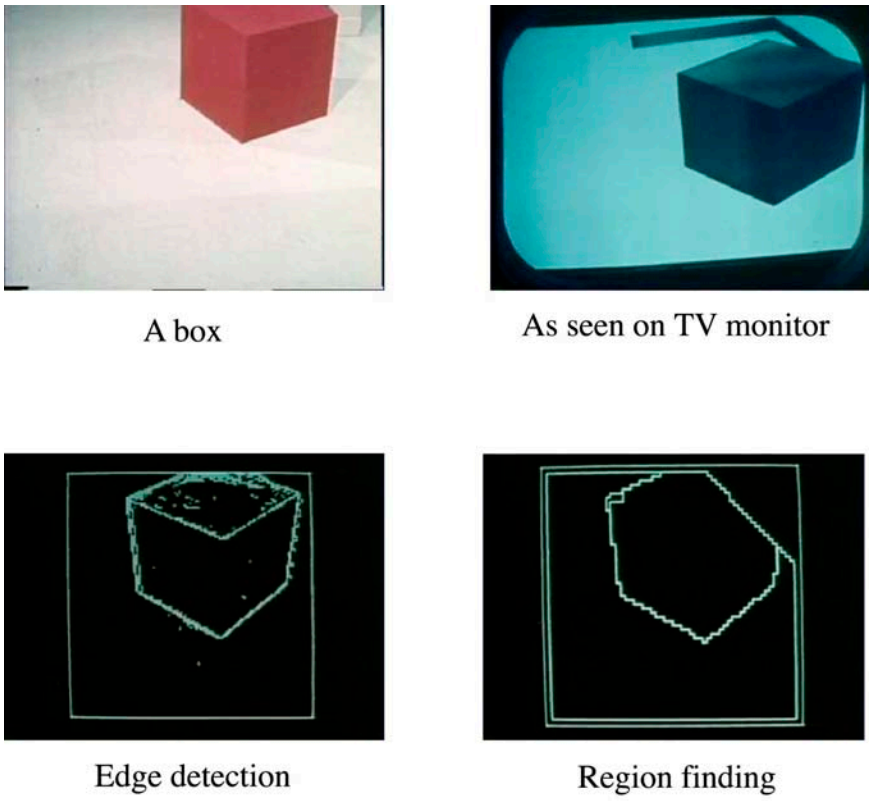


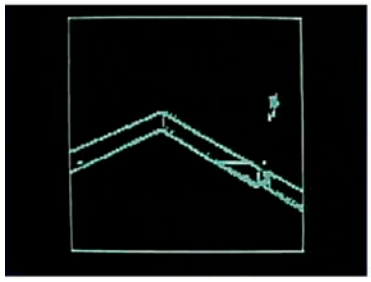
Figure 12.6: Using vision to locate an object. (From the film *Shakey: An Experiment in Robot Planning and Learning.* Used with permission of SRI International.)

path on the floor (a trapezoidal-shaped region) for changes in brightness that might indicate the presence of an obstacle.

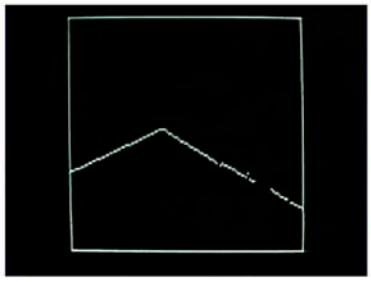
In appraising Shakey's visual performance, it is important to point out that it was really quite primitive and subject to many errors – even in Shakey's specially designed environment. As one report acknowledges, "Regions that we wish to keep distinct – such as two walls meeting at a corner – are frequently merged, and fragments of meaningful regions that should be merged are too often kept distinct." Regarding `clearpath`, for example, this same report notes that "... shadows and reflections can still cause false alarms, and the only solution to some of these problems is to do more thorough scene analysis."[23] Nevertheless, vision played an important part in Shakey's overall performance, and many of the visual processing techniques developed during the Shakey project are still used (with subsequent improvements) today.[24]

Corner of a room

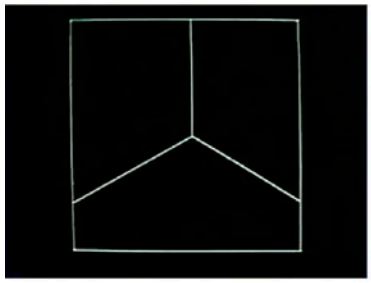
As seen on TV monitor



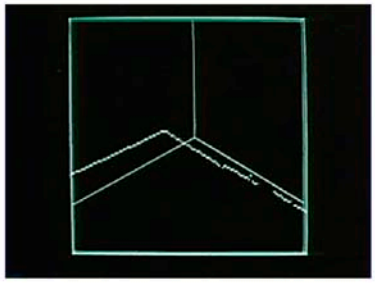
Baseboard detected



Boundary between floor and wall



Predicted corner



Discrepancy used to correct error

Figure 12.7: Using vision to update position. (From the film *Shakey: An Experiment in Robot Planning and Learning.* Used with permission of SRI International.)

12.1.6 Some Experiments with Shakey

To illustrate Shakey's planning and plan-execution and learning methods in action, we set up a task in which Shakey was to push a specified box in front of a specified doorway in a nonadjacent room. To do so, Shakey had to use STRIPS to make a plan to travel to that room and then to push the box. Before beginning its execution of the plan, Shakey saved it in the generalized form described earlier. In the process of executing the plan, we arranged for Shakey to encounter an unexpected obstacle. Illustrating its robust plan execution procedure, Shakey was able to find a different version of the generalized plan that would take it on a somewhat different route to the target room where it could carry on.[25]

One of the researchers working on the Shakey project was L. Stephen Coles (1941–), who had recently obtained a Ph.D. degree under Herb Simon at Carnegie Mellon University working on natural language processing. Coles wanted to give Shakey tasks stated in English. He developed a parser and semantic analysis system that translated simple English commands into logical statements for STRIPS. For example, the task of box pushing just mentioned was posed for Shakey in English as follows:

> Use BOX2 to block door DPDPCLK from room RCLK.

(BOX2, DPDPCLK, and RCLK were the names Shakey used to identify the box, door, and room in question. We were obliging enough to use Shakey's names for things when giving it tasks to perform.)

Coles's program, called ENGROB,[26] translated this English command into the following condition to be made true (expressed in the language of the predicate calculus):

> BLOCKED(DPDPCLK, RCLK, BOX2)

This condition was then given to STRIPS to make a plan for achieving it.

Coles was also interested in getting Shakey to solve problems requiring indirect reasoning. He set up an experiment in which Shakey was to push a box off an elevated platform. To do so, it would have to figure out that it would need to push a ramp to the platform, roll up the ramp, and then push the box. This task was given to Shakey in English as "Push the box that is on the platform onto the floor." The task was successfully executed and described in one of the Shakey technical reports.[27]

The "push-the-box-off-the-platform" task was Coles's way of showing that Shakey could solve problems like the "monkey-and-bananas" problem. That problem, made famous by John McCarthy as an example for deductive reasoning, involved a monkey, a box, and some bananas hanging out of reach. The monkey was supposed to be able to reason that to get the bananas, it

would have to push the box under the bananas, climb up on the box, and then grab the bananas.[28] McCarthy is said to have heard Karl Lashley at the 1948 Caltech Hixon symposium describe a similar problem for demonstrating intelligent problem solving by chimpanzees.

One of the persons who was impressed with Shakey was Bill Gates, who later co-founded Microsoft. He saw the 1972 Shakey film as a junior in high school and drove down from Seattle to SRI (with Paul Allen, who would be the other co-founder of Microsoft) to have a look. According to one source, he was "particularly excited about Shakey moving things around so it could go up a ramp."[29]

12.1.7 Shakey Runs into Funding Troubles

Shakey was the first robot system having the abilities to plan, reason, and learn; to perceive its environment using vision, range-finding, and touch sensors; and to monitor the execution of its plans. It was, perhaps, a bit ahead of its time. Much more research (and progress in computer technology generally) would be needed before practical applications of robots with abilities such as these would be feasible. We mentioned some of the limiting assumptions that were being made by robot research projects at that time in one of our reports about Shakey:

> Typically, the problem environment [for the robot] is a dull sort of place in which a single robot is the only agent of change – even time stands still until the robot moves. The robot itself is easily confused; it cannot be given a second problem until it finishes the first, even though the two problems may be related in some intimate way. Finally, most robot systems cannot yet generate plans containing explicit conditional statements or loops.

Even though the SRI researchers had grand plans for continuing work on Shakey, DARPA demurred, and the project ended in 1972. This termination was unfortunate, because work on planning, vision, learning, and their integration in robot systems had achieved a great deal of momentum and enthusiasm among SRI researchers. Furthermore, several new ideas for planning and visual perception were being investigated. Many of these were described in detail in a final report for the Shakey project.[30]

Among these ideas, a particularly important one involved techniques for constructing plans in a hierarchical fashion. To do so, an overall plan consisting of just "high-level" actions must be composed first. Such a plan can be found with much less searching than one consisting of all of the lowest level actions needed. For example, one's plan for getting to work might involve only the decision either to take the subway or to drive one's car. Then, gradually, the high-level plan must be refined in more and more detail until actions at

the lowest level (such as which set of car keys should be used) would eventually be filled in.

A Stanford computer science graduate student working at SRI, Earl Sacerdoti (1948–), proposed two novel methods for hierarchical planning. First (as part of his master's degree work), he programmed a system he called ABSTRIPS.[31] It consisted of a series of applications of STRIPS – beginning with an easy-to-compose plan that ignored all but the most important operator preconditions. Subsequent applications of STRIPS, guided by the higher level plans already produced, would then gradually take the more detailed preconditions into account. The result was a series of ever-more-detailed plans, culminating in one that could actually be executed.

For his Ph.D. work, Sacerdoti went on to develop a more powerful hierarchical planning system he called NOAH (for Nets of Action Hierarchies).[32] Unlike ABSTRIPS, whose action operators were all at the same level of detail (albeit with preconditions that could be selectively ignored), NOAH employed action operators at several levels of detail. Each operator came equipped with specifications for how it could be elaborated by operators at a lower level of detail. Furthermore, NOAH's representation of a plan, in a form Sacerdoti called a "procedural network," allowed indeterminacy about the order in which plan steps at one level might be carried out. This "delayed commitment" about ordering permitted the more detailed steps of the elaborations of nonordered plans at one level to be interleaved at the level below, often with a consequent improvement in overall efficiency.

Sacerdoti was hoping to use his hierarchical planning ideas in the Shakey project, so he and the rest of us at SRI were quite disappointed that DARPA was not going to support a follow-on project. (Basic research on robots was one of the casualties of the DARPA emphasis on applications work that began in the early 1970s.) However, we were able to talk DARPA into a project that had obvious military relevance but still allowed us to continue work on automatic planning, vision, and plan execution. Interestingly, the project was pretty much a continuation of our research work on Shakey but with a human carrying out the planned tasks instead of a robot. We called it the "computer-based consultant (CBC) project." I'll describe it in a subsequent chapter.

Sacerdoti and the SRI researchers were not alone in recognizing the importance of hierarchical planning. As part of his Ph.D. work at the University of Edinburgh, Austin Tate (1951–) was developing a network-based planning system called INTERPLAN.[33] In 1975 and 1976, supported by the British Science Research Council, Tate and colleagues from operations research produced a hierarchical planner called NONLIN.[34] The planner took its name from the fact that, like NOAH, some of the plan steps were left unordered until they were elaborated at lower levels of the hierarchy.

Other planning systems grew out of the NOAH and NONLIN tradition. One was the interactive plan-generation and plan-execution system SIPE-2 developed by David E. Wilkins at SRI International.[35] Another was O-PLAN developed by Tate and colleagues at the Artificial Intelligence Applications Institute (AIAI) at the University of Edinburgh.[36] These systems have been widely used, extended, and applied.[37]

12.2 The Stanford Cart

In the early 1960s, James Adams, a Mechanical Engineering graduate student at Stanford (and later a Stanford professor), began experimenting with a four-wheeled, mobile cart with a TV camera and a radio control link. Lester Earnest wrote (in his history of the several projects using this cart) "Among other things, Adams showed in his dissertation that with a communication delay corresponding to the round trip to the Moon (about $2\frac{1}{2}$ seconds) the vehicle could not be reliably controlled if traveling faster than about 0.2 mph (0.3 kph)."[38]

After Earnest joined the Stanford AI Laboratory, he and Rodney Schmidt, an Electrical Engineering Ph.D. student, got an upgraded version of the cart to "follow a high contrast white line [on the road around the Lab] under controlled lighting conditions at a speed of about 0.8 mph (1.3 kph)." Other AI graduate students also experimented with the cart from time to time during the early 1970s. A picture of the cart (as it appeared around this time) is shown in Fig. 12.8.

When Hans Moravec came to Stanford to pursue Ph.D. studies on visual navigation, he began work with the cart, "but suffered a setback in October 1973 when the cart toppled off an exit ramp while under manual control and ended up with battery acid throughout its electronics." By 1979 Moravec got the refurbished cart, now equipped with stereo vision, to cross a cluttered room without human intervention. But it did this very slowly. According to Moravec,[39]

> The system was reliable for short runs, but slow. The Cart moved 1 m every 10 to 15 min, in lurches. After rolling a meter it stopped, took some pictures, and thought about them for a long time. Then it planned a new path, executed a little of it, and paused again. It successfully drove the Cart through several 20-m courses (each taking about 5 h) complex enough to necessitate three or four avoiding swerves; it failed in other trials in revealing ways.

A short video of the cart in action can be seen at http://www.frc.ri.cmu.edu/users/hpm/talks/Cart.1979/Cart.final.mov. Along with Shakey, the Stanford Cart resides in the Computer History Museum in

Figure 12.8: The Stanford cart. (Photograph courtesy of Lester Earnest.)

Mountain View, California. They were the progenitors of a long line of robot vehicles, which will be described in subsequent chapters.

Notes

1. Hans P. Moravec, *Robot: Mere Machine to Transcendent Mind*, pp. 18–19, Oxford: Oxford University Press, 1999. [**213**]

2. A copy of the proposal is available online at http://www.ai.sri.com/pubs/files/1320.pdf. Its cover page says "Prepared by Nils J. Nilsson," but it was really a team effort, and many of the ideas were elaborations of those put forward in Rosen's 1963 memo. [**215**]

3. A copy of the complete statement can be found at http://ai.stanford.edu/~nilsson/automaton-work-statement.pdf. [**215**]

4. Online copies of SRI's proposals for the automaton project, subsequent progress reports, and related papers can be found at http://www.ai.sri.com/shakey/. [**216**]

5. See http://www.robothalloffame.org/. [**216**]

6. See http://www.computerhistory.org/timeline/?category=rai. [**216**]

7. J. Doran and Donald Michie, "Experiments with the Graph Traverser Program," *Proceedings of the Royal Society of London*, Series A, Vol. 294, pp. 235–259, 1966. [**220**]

8. The first written account of this idea was in Charles A. Rosen and Nils Nilsson, "Application of Intelligent Automata to Reconnaissance," pp. 21–22, SRI Report, December 1967. Available online at http://www.ai.sri.com/pubs/files/rosen67-p5953-interim3.pdf. [**220**]

9. Personal communication, October 24, 2006. [**220**]

10. See Peter Hart, Nils Nilsson, and Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions System Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107, 1968, and Peter Hart, Nils Nilsson, and Bertram Raphael, "Correction to 'A Formal Basis for the Heuristic Determination of Minimum-Cost Paths,"' *SIGART Newsletter*, No. 37, pp. 28–29, December 1972. [**221**]

11. See, for example, Korf's publications at http://www.cs.ucla.edu/~korf/publications.html. [**221**]

12. In a 2003 paper titled "A* Parsing: Fast Exact Viterbi Parse Selection," Dan Klein and Christopher Manning wrote "The use of A* search can dramatically reduce the time required to find a best parse by conservatively estimating the probabilities of parse completions." [**221**]

13. In an e-mail to Peter Hart dated March 27, 2002, Brian Smart, the chief technical officer of a vehicle-navigation company, wrote "Like most of the 'location based services' and 'vehicle navigation' industry, we use a variant of A* for computing routes for vehicle and pedestrian navigation applications." [**221**]

14. In an e-mail to me dated June 14, 2003, Steven Woodcock, a consultant on the use of AI in computer games, wrote "A* is far and away the most used... and most useful... algorithm for pathfinding in games today. At GDC roundtables since 1999, developers have noted that they make more use of A* than any other tool for pathfinding." [**221**]

15. See Bertram Raphael *et al.*, "Research and Applications – Artificial Intelligence," pp. 27–32, SRI Report, April 1971. Available online at http://www.ai.sri.com/pubs/files/raphael71-p8973-semi.pdf. [**221**]

16. For a description of how QA3 developed a plan for Shakey to push three objects to the same place, for example, see Nils J. Nilsson, "Research on Intelligent Automata," Stanford Research Institute Report 7494, pp. 10ff, February 1969; available online at http://www.ai.sri.com/pubs/files/nilsson69-p7494-interim1.pdf. [**222**]

17. McCarthy and Hayes first described this problem in John McCarthy and Patrick Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Donald Michie and Bernard Meltzer (eds.), *Machine Intelligence*, Vol. 4, pp. 463–502, 1969. Reprinted in Matthew Ginsberg (ed.), *Readings in Nonmonotonic Reasoning*, pp. 26–45, San Francisco: Morgan Kaufmann Publishers, Inc., 1987. Preprint available online at http://www-formal.stanford.edu/jmc/mcchay69/mcchay69.html. [**222**]

18. See Carl Hewitt, " PLANNER: A Language for Proving Theorems in Robots," *Proceedings of the First International Joint Conference on Artificial Intelligence*, pp. 295–301, 1969. [**223**]

19. See Richard Fikes and Nils Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, Nos. 3–4, pp. 189–208, 1971. Available online at http://ai.stanford.edu/users/nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf. [**223**]

20. The generalization and execution mechanisms are described in Richard Fikes, Peter Hart, and Nils Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, No. 4, pp. 251–288, 1972. Available online (as an SRI report) at http://www.ai.sri.com/pubs/files/tn070-fikes72.pdf. [**224**]

21. See Claude Brice and Claude Fennema, "Scene Analysis Using Regions," *Artificial Intelligence*, Vol. 1, No. 3, pp. 205–226, 1970. [**225**]

22. Richard O. Duda and Peter E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, Vol. 15, pp. 11–15, January 1972. See also, Peter E. Hart, "How the Hough Transform Was Invented," *IEEE Signal Processing Magazine*, November, 2009. [**225**]

23. Bertram Raphael *et al.*, "Research and Applications – Artificial Intelligence," Part V, SRI Final Report, December 1971; available online at http://www.ai.sri.com/pubs/files/raphael71-p8973-final.pdf. [**226**]

24. For more information about Shakey's visual routines, in addition to the final report just cited, see Richard O. Duda, "Some Current Techniques for Scene Analysis," SRI Artificial Intelligence Group Technical Note 46, October 1970, available online at http://www.ai.sri.com/pubs/files/tn046-duda70.pdf. [**226**]

25. This experiment, as well as other information about Shakey, is described in Bertram Raphael *et al.*, "Research and Applications – Artificial Intelligence," SRI Final Report, December 1971, available online at http://www.ai.sri.com/pubs/files/raphael71-p8973-final.pdf; in Nils Nilsson (ed.), "Shakey The Robot," SRI Technical Note 323, April 1984, available online at http://www.ai.sri.com/pubs/files/629.pdf; and in a 1972 film, *Shakey: An Experiment in Robot Planning and Learning*, available online at http://www.ai.sri.com/movies/Shakey.ram. [**228**]

26. L. Stephen Coles, "Talking with a Robot in English," *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, DC, May 7–9, Bedford, MA: The MITRE Corporation, 1969. [**228**]

27. See L. Stephen Coles *et al.*, "Application of Intelligent Automata to Reconnaissance," SRI Final Report, November 1969, available online at http://www.ai.sri.com/pubs/files/coles69-p7494-final.pdf. [**228**]

28. The problem was introduced by McCarthy in his July 1963 memo "Situations, Actions, and Causal Laws," reprinted as Section 7.2 of his paper "Program with Commonsense," which appeared in Marvin Minsky (ed.), *Semantic Information Processing*, pp. 403–418, Cambridge, MA: MIT Press, 1968. [**229**]

29. E-mail from Eric Horvitz of May 12, 2003. [**229**]

30. Peter E. Hart *et al.*, "Artificial Intelligence – Research and Applications," Technical Report, Stanford Research Institute, December 1972. (Available online at http://www.ai.sri.com/pubs/files/hart72-p1530-annual.pdf.) See also Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, "Some New Directions in Robot Problem Solving," in *Machine Intelligence 7*, Bernard Meltzer and Donald Michie (eds.), pp. 405–430, Edinburgh: Edinburgh University Press, 1972. (The SRI Technical Note 68 version is available online at http://www.ai.sri.com/pubs/files/1484.pdf.) [**229**]

31. Earl D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," pp. 412–422, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, 1973. (The SRI AI Center Technical Note 78 version is available online at http://www.ai.sri.com/pubs/files/1501.pdf.) [**230**]

32. Earl D. Sacerdoti, "The Non-Linear Nature of Plans," *Proceedings of the International Joint Conference on Artificial Intelligence*, 1975. (The SRI AI Center Technical Note 101 version is available online at http://www.ai.sri.com/pubs/files/1385.pdf.) Also see Earl D. Sacerdoti, *A Structure for Plans and Behavior*, New York: Elsevier North-Holland, 1977. (The SRI AI Center Technical Note No. 109 version is available online at http://www.ai.sri.com/pubs/files/762.pdf.) [**230**]

33. Austin Tate, "Interacting Goals and Their Use," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 215–218, Tbilisi, USSR,

September 1975; available online at http://www.aiai.ed.ac.uk/project/oplan/documents/1990-PRE/1975-ijcai-tate-interacting-goals.pdf. Austin Tate, "Using Goal Structure to Direct Search in a Problem Solver," Ph.D. thesis, University of Edinburgh, September 1975; available online at http://www.aiai.ed.ac.uk/project/oplan/documents/1990-PRE/. [**230**]

34. Austin Tate, "Generating Project Networks," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)* pp. 888–893, Boston, MA, August 1977; available online at http://www.aiai.ed.ac.uk/project/oplan/documents/1990-PRE/1977-ijcai-tate-generating-project-networks.pdf. [**230**]

35. See the SIPE-2 Web page at http://www.ai.sri.com/~sipe/. [**231**]

36. Ken Currie and Austin Tate, "O-PLAN: The Open Planning Architecture," *Artificial Intelligence*, Vol. 52, pp. 49–86, 1991. Available online at http://www.aiai.ed.ac.uk/project/oplan/documents/1991/91-aij-oplan-as-published.pdf. [**231**]

37. See, for example, AIAI's "Planning and Activity Management" Web page at http://www.aiai.ed.ac.uk/project/plan/. [**231**]

38. Lester Earnest, "Stanford Cart," August 2005; available online at http://www.stanford.edu/~learnest/cart.htm. [**231**]

39. Hans P. Moravec, "The Stanford Cart and the CMU Rover," *Proceedings of the IEEE*, Vol. 71, No. 7, pp. 872–884, July 1983. [**231**]

Chapter 13

Progress in Natural Language Processing

As mentioned previously, the problems of understanding, generating, and translating material in ordinary human (rather than computer) languages fall under the heading of natural language processing. During the "early explorations" phase of AI research, some good beginnings were made on NLP problems. In the subsequent phase, the late 1960s to early 1970s, new work built on these foundations, as I'll describe in this part of the book.

13.1 Machine Translation

W. John Hutchins, who has written extensively about the history of machine translation (MT), has called the period 1967 to 1976, "the quiet decade."[1] Inactivity in the field during this period is due in part to the ALPAC report, which, as I have already said, was pessimistic about the prospects for machine translation. Hutchins claimed "The influence of the ALPAC report was profound. It brought a virtual end to MT research in the USA for over a decade and MT was for many years perceived as a complete failure. . . . The focus of MT activity switched from the United States to Canada and to Europe."[2]

One exception to this decade-long lull in the United States was the development of the Systran (System Translator) translating program by Petr Toma, a Hungarian-born computer scientist and linguistics researcher who had worked on the Georgetown Russian-to-English translation system. In 1968, Toma set up a company called Latsec, Inc., in La Jolla, California, to continue the Systran development work he had begun earlier in Germany. The U.S. Air Force gave the company a contract to develop a Russian-to-English translation

system. It was tested in early 1969 at the Wright-Patterson Air Force Base in Dayton, Ohio, "where it continues to provide Russian–English translations for the USAF's Foreign Technology Division to this day."[3] Systran has evolved to be one of the main automatic translation systems. It is marketed by the Imageforce Corporation in Tampa, Florida.[4]

How well does Systran translate? It all depends on how one wants to measure performance. Margaret Boden mentions two measures, namely, "intelligibility" and "correctness." Both of these measures depend on human judgement. For the first, one asks "Can the translation be generally understood?" For the second, one asks "Do human 'post-editors' need to modify the translation?" Boden states that "in the two-year period from 1976 to 1978, the intelligibility of translations generated by Systran rose from 45 to 78 percent for [raw text input]..." She also notes that human translations score only 98 to 99 percent, not 100 percent. Regarding correctness, Boden states that in 1978 "only 64 percent of the words were left untouched by human post-editors. Even so, human post-editing of a page of Systran output took only twenty minutes in the mid-1980s, whereas normal (fully human) translation would have taken an hour."[5]

13.2 Understanding

Although the late 1960s and early 1970s might have been a "quiet decade" for machine translation, it was a very active period for other NLP work. Researchers during these years applied much more powerful syntactic, semantic, and inference abilities to the problem of understanding natural language. Typical of the new attitude was the following observation by Terry Winograd, an MIT Ph.D. student during the late 1960s:[6]

> If we really want computers to understand us, we need to give them ability to use more knowledge. In addition to a grammar of the language, they need to have all sorts of knowledge about the subject they are discussing, and they have to use reasoning to combine facts in the right way to understand a sentence and to respond to it. The process of understanding a sentence has to combine grammar, semantics, and reasoning in a very intimate way, calling on each part to help with the others.

13.2.1 SHRDLU

Perhaps the NLP achievement that caused the greatest excitement was the SHRDLU natural language dialog system programmed by Terry Winograd (1946– ; Fig. 13.1) for his Ph.D. dissertation (under Seymour Papert) at MIT.[7]

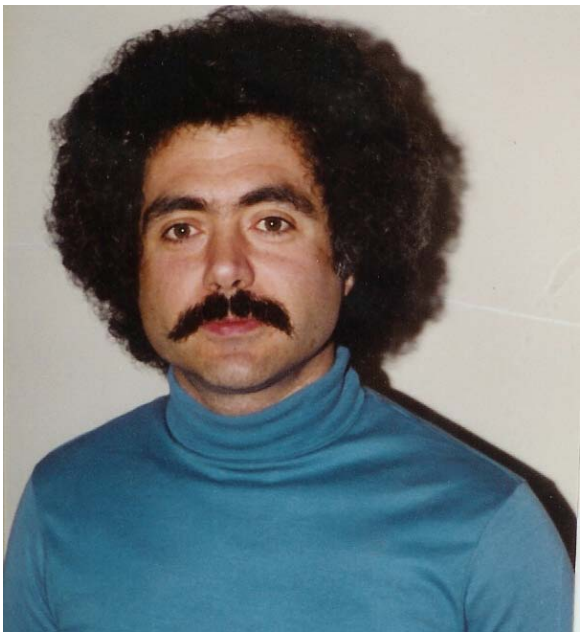


Figure 13.1: Terry Winograd. (Photograph courtesy of Terry Winograd.)

SHRDLU was able to carry on a dialog about what Winograd called a "micro-world," a world consisting of toy blocks and a "gripper" for moving them about. Unlike the the world of real blocks used in previous MIT and Stanford hand–eye research, Winograd's blocks world was simulated on a DEC PDP-10 computer and viewed on a DEC 340 black-and-white graphics display such as that shown in Fig. 13.2. SHRDLU could be given commands, typed in ordinary English, to manipulate the blocks using its gripper. It could also be asked questions about the state of its world. It could "understand" those commands and questions, carry out requested manipulations, and ask the user clarifying questions in English about questions or commands it did not understand.

Before describing something about SHRDLU's design, let us look at an example of part of a dialog.[8] In the following, the user's input is numbered and shown in lowercase letters, and SHRDLU's response is in all caps, Winograd's commentary follows in the same typeface, and mine is in parentheses in normal type:

```
1.  pick up a big red block.

In order to pick up the red block, it had to clear it
off by finding a space for the green one and moving the
green one away.
```

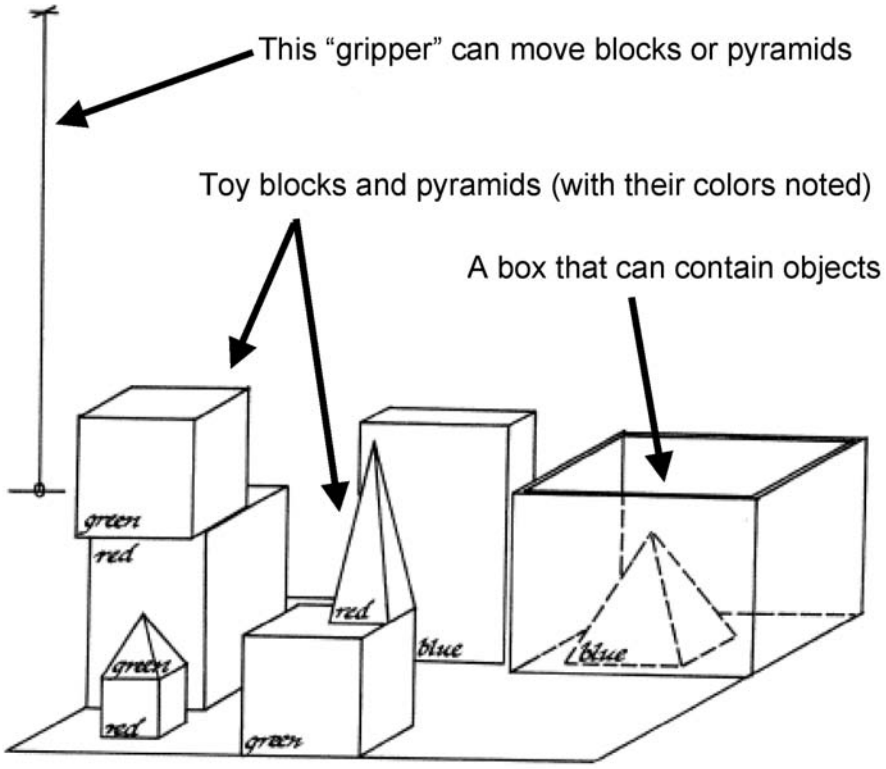


Figure 13.2: SHRDLU's world. (Illustration used with permission of Terry Winograd.)

```
OK.
The system answers "OK" when it carries out a command.
```

(I won't bother to show what the block configuration looks like as the dialog proceeds, assuming you can use your own mental "model" to keep track of things. The gripper is now holding the red block.)

```
2.  grasp the pyramid.
There are three pyramids in the scene, so the program
does nothing.
I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
3.  find a block which is taller than the one you are
holding and put it into the box.
```

(The user has now apparently changed his mind about pyramid grasping.)

```
The system has to figure out what is meant by "one" and
"it," as well as using deduction to decide which block
it is holding.

BY "IT," I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER
THAN THE ONE I AM HOLDING.

Printed out to keep us informed.

The hand has to set down the red block to pick up the
taller blue one.

OK.

4.  what does the box contain?

THE BLUE PYRAMID AND THE BLUE BLOCK.

5.  what is the pyramid supported by?

Earlier the system responded to "the pyramid" by saying
"I don't understand which pyramid you mean." now, it
knows that we must mean the one which was just
mentioned.

THE BOX
```

This dialog illustrating SHRDLU's capabilities goes on for forty-two more user inputs and system responses.[9]

To engage in dialogs like this, SHRDLU needed "knowledge." It needed to know about the syntax of that part of English it used, and it needed to know what various words and word combinations mean in the context of their use. It also needed to know about its blocks world – how blocks can be manipulated and what it means for an object to be inside of the "box." It needed to keep track of the dialog so that it could decide to which object mentioned previously a word such as "it" referred.

All of this needed knowledge was represented in LISP programs, or "procedures," as Winograd called them. Knowledge about syntax was represented as a collection of procedures based on the principles of "systemic grammar."[10] Knowledge about the meanings of words in context was represented in procedures that could refer to a dictionary of word meanings, to other parts of the sentence in which the word was used, and to the discourse. Knowledge about the blocks world was represented in two ways: There was a model that gave the locations of all of the objects and there were procedures that could infer the predicted effects (in the model) of manipulations by the gripper on the various objects. The object-moving procedures had information both about the preconditions and about the effects of these manipulations. These procedures were encoded in a version of Hewitt's PLANNER language, which, as mentioned previously, bore some resemblance to STRIPS operators.

Additional procedures in the PLANNER language were used for other types of inference needed by the system. Logical rules were expressed as programs, which were capable of making both forward and backward deductions.

SHRDLU's processes for language understanding can be divided into three parts, namely, syntax, semantics, and inference, but doing so is somewhat misleading because the interplay among these parts was a key feature of the system. As Winograd stated, "Since each piece of knowledge can be a procedure, it can call on any other piece of knowledge of any type." For example, Winograd wrote, "As it finds each piece of the syntactic structure, it checks its semantic interpretation, first to see if it is plausible, then (if possible) to see if it is in accord with the system's knowledge of the world, both specific and general."

Winograd's procedural representation of knowledge (together with Hewitt's PLANNER language for encoding such representations) can be contrasted with McCarthy's use of logical formulas to represent knowledge declaratively. The success of SHRDLU fueled a debate among AI researchers about the pros and cons of these two knowledge representation strategies – procedural versus declarative. Actually, the use of LISP to represent procedures blurs this distinction to some extent because, as Winograd pointed out, "LISP allows us to treat programs as data and data as programs." So, even though SHRDLU's knowledge was represented procedurally, it was able to incorporate some declarative new knowledge (presented to it as English sentences) into its procedures.

SHRDLU's performance was indeed quite impressive and made some natural language researchers optimistic about future success.[11] However, Winograd soon abandoned this line of research in favor of pursuing work devoted to the interaction of computers and people. Perhaps because he had first-hand experience of how much knowledge was required for successful language understanding in something so simple as the blocks world, he despaired of ever giving computers enough knowledge to duplicate the full range of human verbal competence. In a 2004 e-mail, Winograd put SHRDLU's abilities in context with those of humans:[12]

> There are fundamental gulfs between the way that SHRDLU and its kin operate, and whatever it is that goes on in our brains. I don't think that current research has made much progress in crossing that gulf, and the relevant science may take decades or more to get to the point where the initial ambitions become realistic. In the meantime AI took on much more doable goals of working in less ambitious niches, or accepting less-than-human results (as in translation).

13.2.2 LUNAR

On their return from the first manned moon landing, the Apollo 11 astronauts brought back several pounds of moon rocks for scientific study. Various data about these rocks were stored in databases that could be accessed by geologists and other scientists. To make retrieval of this information easier for lunar geologists, NASA asked William A. Woods, a young computer scientist at BBN, about the possibility of designing some sort of natural-language "front end" so that the databases could be queried in English instead of in arcane computer code. Woods had just completed his Ph.D. research at Harvard on question-answering systems.[13]

Sponsored by NASA's Manned Spacecraft Center, Woods and BBN colleagues Ron Kaplan and Bonnie Webber developed a system they called "LUNAR" for answering questions about the moon rocks.[14] LUNAR used both syntactic and semantic processes to transform English questions into moon rock database queries. Syntactic analysis was performed by using "augmented transition networks" (ATNs), a methodology developed by Woods during his Harvard Ph.D. research. (I'll describe what ATNs are all about shortly.) The semantic component, guided by the ATN-derived parse trees, transformed English sentences into what Woods called a "meaning representation language" (MRL). This language was a logical language (like that of the predicate calculus) but extended with procedures that could be executed. MRL was originally conceived by Woods at Harvard and further developed at BBN.

LUNAR was able to "understand" and answer a wide variety of questions, including, for example,

> "What is the average concentration of aluminum in high alkali rocks?"
>
> "How many breccias contain olivine?"
>
> "What are they?" (LUNAR recognized that "they" referred to the breccias named as answers to the last question.)

LUNAR was the first question-answering system to publish performance data. It was able to answer successfully 78% of the questions put to it by geologists at the Second Annual Lunar Science Conference held in Houston in January 1971. Reportedly, 90% would have been answerable with "minor fixes" to the system.

In a June 2006 talk[15] about LUNAR, Woods mentioned some of its limitations. The following dialog illustrates one shortcoming:

> User: What is a breccia?
>
> LUNAR: S10018.
>
> User: What is S10018?
>
> LUNAR: S10018.

Woods said, "LUNAR simply finds referents of referring expressions and gives their names. There is no model of the purpose behind the user's question or of different kinds of answers for different purposes."

Although LUNAR could recognize several different ways of phrasing essentially the same question, Woods claimed that "there are other requests which (due to limitations in the current grammar) must be stated in a specific way in order for the grammar to parse them and there are others which are only understood by the semantic interpreter when they are stated in certain ways."[16]

13.2.3 Augmented Transition Networks

Many people realized that context-free grammars (like the ones I discussed earlier) were too weak for most practical natural language processing applications. For example, if we were to expand the illustrative grammar I described in Section 7.1 so that it included (in addition to "threw" and "hit" and "man") the present-tense verbs "throw," "throws," and "hits" and the plural noun "men," then the strings "the men hits the ball" and "the man throw the ball" would be inappropriately accepted as grammatical sentences. To expand a context-free grammar to require that nouns and verbs must agree as to number would involve an impractically large collection of rules. Also, allowing for passive sentences, such as "the ball was hit by the men," would require even further elaboration. Clearly, the sorts of sentences that geologists might ask about moon rocks required more powerful grammars – such as the augmented transition networks that Woods and others had been developing.

In Chomsky's 1957 book[17] he had proposed a hierarchy of grammatical systems of which context-free grammars were just one example. His more powerful grammars had a "transformational component" and were able, for example, to parse a sentence such as "the ball was hit by the man" and give it the same "deep structure" as it would give the sentence "the man hit the ball." Augmented transition network grammars could also perform these kinds of transformations but in a more computationally satisfying way.

An augmented transition network is a maplike graphical structure in which the nodes represent points of progress in the parsing process, and the paths connecting two nodes represent syntactic categories. We can think of parsing a sentence as traversing a path through the network from the start node (no progress at all yet) to an end node (where the sentence has been successfully parsed). Traversing the path builds the syntactic structure of the sentence in the form of a parse tree. Analysis of a sentence involves peeling off the words in left-to-right fashion and using them to indicate which path in the network to take.

Syntactic analysis could begin by peeling off a single word and finding out from a lexicon whether it was a noun, a determiner, an auxiliary (such as

"does"), an adjective, or some other "terminal" syntactic category. Or it could begin by peeling off a group of words and checking to see whether this group was a noun phrase, a verb phrase, a prepositional phrase, or what have you. In the first case, depending on the category of the single word, we would take a path corresponding to that category leading out from the start node. To accommodate the second case, there would be possible paths corresponding to a noun phrase and the other possible higher level syntactic categories.

But how would we decide whether or not we could take the noun-phrase path, for example? The answer proposed by Woods and others was that there would be additional transition networks corresponding to these higher level categories. We would be permitted to take the noun-phrase path in the main transition network only if we could successfully traverse the noun-phrase network. And because one path in the noun-phrase network might start with a prepositional phrase, we would have to check to see whether we could take that path (in the noun-phrase network) by successfully traversing a prepositional-phrase network. This process would continue with one network "calling" other networks in a manner similar to the way in which a program can fire up (or "call") other programs, possibly *recursively*. (You will recall my discussion of recursive programs: programs that can call versions of themselves.) For this reason, assemblages of networks like these are called recursive transition networks.

The first networks of this kind were developed at the University of Edinburgh in Scotland by James Thorne, Paul Bratley, and Hamish Dewar.[18] Later, Dan Bobrow and Bruce Fraser proposed a transition network system that elaborated on the Scottish one.[19] Both of these systems also performed auxiliary computations while traversing their networks. These "augmentations" allowed the construction of a "deep structure" representation of the sentence being analyzed. Woods's work on "augmented, recursive transition networks" built on and refined these ideas and introduced an elegant network definition language.[20]

As an example, Woods described how one of his networks analyzed the sentence "John was believed to have been shot."[21] After all of the calls to subsidiary networks and all of the auxiliary computations were performed the parse tree shown in Fig. 13.3 was obtained. We can observe two things about this parse tree. First, note the occurrence of "PAST" and "PAST PERFECT" as tense markers. Second, note that the form of the original passive-voice sentence has been transformed to an active-voice sentence using a presumed pronoun "SOMEONE." As Woods notes, the structure can be paraphrased as "Someone believed that someone had shot John." Network grammars get at the "deep structure" of sentences by transforming then into a standard form.

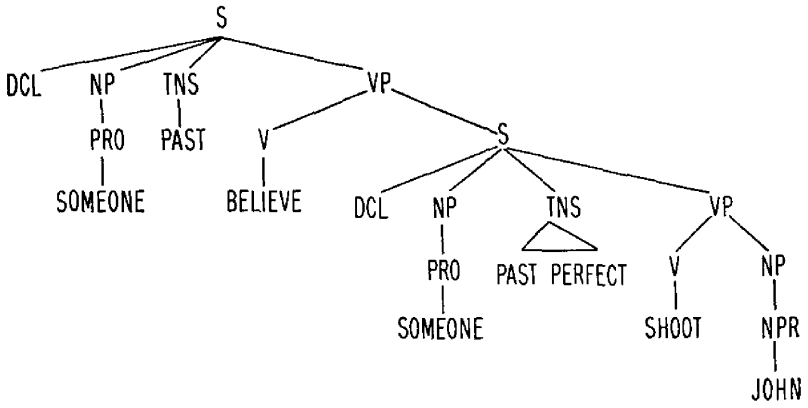


Figure 13.3: A parse tree obtained for the sentence "John was believed to have been shot." (From William A. Woods, "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10, pp. 591–606, 1970.)

13.2.4 GUS

Dan Bobrow and Ronald Kaplan, two of the researchers doing NLP work at BBN, moved to the newly formed Palo Alto Research Center (PARC) of Xerox in the mid 1970s. One of the projects that they and other researchers worked on there was GUS (an acronym for Genial Understander System), which was "intended to engage a sympathetic and highly cooperative human in an English dialog, directed towards a specific goal within a very restricted domain of discourse."[22] The restricted domain was that of a travel agent engaged in "a conversation with a client who wants to make a simple return trip to a single city in California." The following transcript of an actual dialog shows off some of its abilities:

> GUS: Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go?
>
> (1) *Client*: I want to go to San Diego on May 28.
>
> GUS: What time do you want to leave?
>
> (2) *Client*: I must be in San Diego before 10 am.
>
> GUS: Would you like P.S.A. flight 102 that arrives at 9.15 am?
>
> (3) *Client*: What is the next flight?
>
> GUS: Air California flight 310 that leaves at 8.30 am and arrives at 10.00 am.
>
> (4) *Client*: I'll take the first one.

> GUS: In what name should I make a reservation, please?
>
> (5) *Client*: Dan Bobrow.
>
> GUS: I have confirmed the following flight: P.SA. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am. What date do you want to return on?
>
> (6) *Client*: On Friday in the evening.
>
> GUS: Would you like the flight that leaves at 7.45 pm?
>
> (7) *Client*: That's fine.
>
> GUS: I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm. Thank you for calling. Goodbye.

Although the abilities of the system were certainly limited, GUS was able to deal with a number of problems. One of these involves what NLP researchers call "resolving anaphora," by which they mean deciding on the objects or events to which various words or phrases in a dialog refer. Several examples, keyed to the numbered sentences in the dialog above, are mentioned in the paper about GUS:

> At line (3), for example, the client's query refers to the flight mentioned in GUS's immediately preceding utterance. In (4) there is a reference to the flight mentioned earlier in the conversation, [following line (2)]. Note that "next flight" in (3) was to be interpreted relative to the order of flights in the airline guide whereas "first one" in (4) refers to the order in which the flights were mentioned. Another implicit referent underlies the use of "Friday" to specify a date in (6). Resolution of this reference requires some complicated reasoning involving both the content and the context of the conversation. Since May 28 has been given as the departure date, it must presumably be the following Friday that the client has in mind. On the other hand, suppose that the specifications were reversed and Friday had been given as the departure date at line (1). It would then be most readily interpretable as referring to the Friday immediately following the conversation.

GUS was a combination of several communicating subsystems, a morphological analyzer for dealing with word components, a syntactic analyzer for generating parse trees, a "reasoner" for figuring out a user's meanings and intentions, and a language generator for responding. Controlling these components was done by using an "agenda" mechanism. As the authors explain,

> GUS operates in a cycle in which it examines this agenda, chooses the next job to be done, and does it. In general, the execution of the selected task causes entries for new tasks to be created and placed on the agenda. Output text generation can be prompted by reasoning processes at any time, and inputs from the client are handled whenever they come in. There are places at which information from a later stage (such as one involving semantics) are fed back to an earlier stage (such as the parser). A supervisory process can reorder the agenda at any time.

The syntactic component of GUS had "access to a main dictionary of more than 3,000 stems and simple idioms." The syntactic analyzer was based on a system developed earlier by Ronald Kaplan, which used a transition-network grammar and was called a "General Syntactic Processor."[23] Client sentences were encoded in "frames" (which are related to Minsky's frames but closer in form to semantic networks). Some frames described the sequence of a normal dialog, whereas others represented the attributes of a date, a trip plan, or a traveler. GUS's reasoning component used the content and structure of the frames to deduce how best to interpret client sentences.

Besides anaphora, the paper mentioned several other problems that GUS was able to deal with. However, it also cautioned that "it is much too easy to extrapolate from [the sample dialog] a mistaken notion that GUS contained solutions to far more problems than it did." Sample dialogs recorded between human clients and humans playing the role of a GUS revealed numerous instances in which the computer GUS would fail. The authors concluded that if users of systems like GUS departed "from the behavior expected of them in the minutest detail, or if apparently insignificant adjustments are made in their structure," the systems would act as if they had "gross aphasia" or had just simply died. The authors conceded that "GUS itself is not very intelligent, but it does illustrate what we believe to be essential components of [an intelligent language understanding] system. . . . [It] must have a high quality parser, a reasoning component, and a well structured data base of knowledge." Subsequent work on NLP at PARC and many other places sought to improve all of these components.

The systems developed by researchers such as Winograd, Woods, Bobrow, and their colleagues were very impressive steps toward conversing with computers in English. Yet, there was still a long way to go before natural language understanding systems could perform in a way envisioned by Winograd in the preface to his Ph.D. dissertation:

> Let us envision a new way of using computers so they can take instructions in a way suited to their jobs. We will talk to them just as we talk to a research assistant, librarian, or secretary, and they will carry out our commands and provide us with the information we ask for. If our instructions aren't clear enough, they will ask for

more information before they do what we want, and this dialog will all be in English.

Notes

1. W. John Hutchins, "Machine Translation: A Brief History," in E. F. K. Koerner and R. E. Asher (eds.), *Concise History of the Language Sciences: From the Sumerians to the Cognitivists*, pp. 431–445, Oxford: Pergamon Press, 1995. (Also available online at http://ourworld.compuserve.com/homepages/WJHutchins/Conchist.htm.) [**237**]

2. *Ibid.* [**237**]

3. W. John Hutchins, *Machine Translation: Past, Present, Future*, Chichester: Ellis Horwood, 1986. An updated (2003) version is available online at http://www.hutchinsweb.me.uk/PPF-TOC.htm. Some of the technical details of Systran's operation are described in the book. [**238**]

4. http://www.translationsoftware4u.com/. [**238**]

5. Margaret A. Boden, *Mind as Machine: A History of Cognitive Science*, p. 683, Oxford: Oxford University Press, 2006. [**238**]

6. This quote is from the preface of Winograd's Ph.D. dissertation. [**238**]

7. SHRDLU is described in Winograd's dissertation "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language." It was issued as an MIT AI Technical Report No. 235, February 1971, and is available online at https://dspace.mit.edu/bitstream/1721.1/7095/2/AITR-235.pdf. The thesis was also published as a full issue of *Cognitive Psychology*, Vol. 3, No. 1, 1972, and as a book *Understanding Natural Language*, New York: Academic Press, 1972. The letters in SHRDLU comprise the second column of keys in linotype machines, which were used to set type before computers were used for that. This nonsense word was often used in *MAD* magazine, which Winograd read in his youth. Failing to think of an acceptable acronym to use to name his system, Winograd used SHRDLU. For Winograd's account, see http://hci.stanford.edu/~winograd/shrdlu/name.html. [**238**]

8. Taken from Section 1.3 of Winograd's thesis. [**239**]

9. Readers interested in the entire dialog can see it either in Winograd's thesis or on one of his Web sites at http://hci.stanford.edu/~winograd/shrdlu/. [**241**]

10. Winograd cites, among others, M. A. K. Halliday, "Categories of the Theory of Grammar," *Word*, Vol, 17, No. 3, pp. 241-292, 1961. [**241**]

11. For a short film of SHRDLU in action, see http://projects.csail.mit.edu/films/aifilms/digitalFilms/3mpeg/26-robot.mpg. [**242**]

12. From http://www.semaphorecorp.com/misc/shrdlu.html. [**242**]

13. William A. Woods, "Semantics for a Question-Answering System," Ph.D. dissertation, Harvard University, August 1967. Reprinted as a volume in the series *Outstanding Dissertations in the Computer Sciences*, New York: Garland Publishing, 1979. [**243**]

14. William A. Woods, Ron M. Kaplan, and Bonnie Nash-Webber, "The Lunar Sciences Natural Language Information System: Final Report," BBN, Cambridge, MA, June 1, 1972. See also William A. Woods, "Progress in Natural Language Understanding – An Application to Lunar Geology," *AFIPS Conference Proceedings*, Vol. 42, pp. 441–450, Montvale, New Jersey: AFIPS Press, 1973. [**243**]

15. See http://www.ils.albany.edu/IQA06/Files/Bill_Woods_IQA06.pdf. [**243**]

16. William A. Woods, "Progress in Natural Language Understanding – An Application to Lunar Geology," *AFIPS Conference Proceedings*, Vol. 42, pp. 441–450, Montvale, New Jersey: AFIPS Press, 1973. [**244**]

17. Noam Chomsky, *Syntactic Structures,* 's-Gravenhage: Mouton & Co., 1957. [**244**]

18. James Thorne, Paul Bratley, and Hamish Dewar, "The Syntactic Analysis of English by Machine," in D. Michie (ed.), *Machine Intelligence 3*, pp. 281–309, New York: American Elsevier Publishing Co., 1968; Hamish Dewar, Paul Bratley, and James Thorne, "A Program for the Syntactic Analysis of English Sentences," *Communications of the ACM*, Vol. 12, No. 8, pp. 476–479, August 1969. [**245**]

19. Daniel Bobrow and Bruce Fraser, "An Augmented State Transition Network Analysis Procedure," *Proceedings of the International Joint Conferenece on Artificial Intelligence*, pp. 557–567, Washington, DC, 1969. [**245**]

20. William A. Woods, "Augmented Transition Networks for Natural Language Analysis," Report CS-1, Aiken Computation Laboratory, Harvard University, Cambridge, MA, December 1969; William A. Woods, "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10, pp. 591–606, 1970. The ACM article has been reprinted in Yoh-Han Pao and George W. Ernest (eds.), *Tutorial: Context-Directed Pattern Recognition and Machine Intelligence Techniques for Information Processing*, Silver Spring, MD: IEEE Computer Society Press, 1982, and in Barbara Grosz, Karen Sparck Jones, and Bonnie Webber (eds.), *Readings in Natural Language Processing*, San Mateo, CA: Morgan Kaufmann, 1986. [**245**]

21. William A. Woods, *op. cit.*, pp. 602ff. [**245**]

22. Daniel G. Bobrow *et al.*, "GUS, A Frame-Driven Dialog System," *Artificial Intelligence*, Vol. 8, pp. 155–173, 1977. [**246**]

23. Ronald Kaplan, "A General Syntactic Processor," in R. Rustin (ed.), *Natural Language Processing*, New York: Algorithmics Press, 1973. [**248**]

14.0

Chapter 14

Game Playing

I have already mentioned attempts to program computers to play board games, such as chess and checkers. The most successful of these was Arthur Samuel's checker-playing program. In 1967, Samuel published a paper describing an improved version of his program.[1] He had refined the program's search procedure and incorporated better "book-learning" capabilities, and instead of calculating the estimated value of a position by adding up weighted feature values, he used hierarchically organized tables. According to Richard Sutton, "This version learned to play much better than the 1959 program, though still not at a master level."[2]

Between 1959 and 1962, a group of MIT students, advised by John McCarthy, developed a chess-playing program. It was based on earlier programs for the IBM 704 written by McCarthy. One of the group members, Alan Kotok (1941–2006) described the program in his MIT bachelor's thesis.[3] The program was written in a combination of FORTRAN and machine (assembly) code and ran on the IBM 7090 computer at MIT. It used the alpha–beta procedure (as discussed earlier) to avoid generating branches of the search tree that could be eliminated without altering the final result. Kotok claimed that his program did not complete any games but "played four long game fragments in which it played chess comparable to an amateur with about 100 games experience. . . . Most of the machine's moves are neither brilliant nor stupid. It must be admitted that it occasionally blunders."[4] When McCarthy moved to Stanford, he took the program along with him and continued to work on it.

In the meantime, a computer chess program was being developed by Georgi Adelson-Velskiy and colleagues in Alexander Kronrod's laboratory at the Institute for Theoretical and Experimental Physics (ITEP) in Moscow.[5] During a visit to the Soviet Union in 1965, McCarthy accepted a challenge to have the Kotok–McCarthy program play the Soviet program. Beginning on November 22, 1967, and continuing for about nine months, the

Kotok–McCarthy program (running on a DEC PDP-6 at Stanford) played the Russian program (running on the Russian M-20 computer at ITEP) – the first match to be played by a computer against a computer. In each of the first two games, the Stanford program eked out a draw (by surviving until the agreed-upon limit of 40 moves) against a weak version of the Russian program. However, it lost the last two games against a stronger version of the ITEP program. McCarthy later claimed that, although Stanford had the better computer, ITEP had the better programs.[6] The ITEP program was the forerunner of the much improved Kaissa program developed later by Misha Donskoy, Vladimir Arlazarov, and Alexander Ushkov at the Institute of Control Science in Moscow.

Richard Greenblatt, an expert programmer at the AI Lab at MIT, thought he could improve on Kotok's chess program. His work on computer chess eventually led to a program he called MAC HACK VI.[7] Being an expert chess player himself, he was able to incorporate a number of excellent heuristics for choosing moves and for evaluating moves in his program. Running on the AI Lab's DEC PDP-6 and written in efficient machine code, MAC HACK VI was the first program to play in tournaments against human chess players. In an April 1967 tournament, it won two games and drew two, achieving a rating of 1450 on the U.S. Chess Federation rating scale, about the level of an amateur human player. (According to the international rating system for human chess players, the highest level is that of Grand Masters. Then come International Masters, National Masters, Experts, Class A, Class B, and so on. MAC HACK VI played at the high Class C to low Class B level, which is still quite far from master play.) It became an honorary member of the U.S. Chess Federation and of the Massachusetts Chess Association. In a famous match at MIT in 1967,[8] Greenblatt's program beat Hubert Dreyfus (1929–), an AI critic who had earlier observed that "no chess program could play even amateur chess."[9] Although Dreyfus's observation was probably true in 1965, Greenblatt's MAC HACK VI was playing at the amateur level two years later.

Perhaps encouraged by MAC HACK's ability, in 1968 Donald Michie and John McCarthy made a bet of £250 each with David Levy (1945–), a Scottish International Master, that a computer would be able to beat him within ten years. (The following year Seymour Papert joined in, and in 1971 Ed Kozdrowicki of the University of California at Davis did also, bringing the total bet to £1000. In 1974, Donald Michie raised the total to £1250.) In 1978, Levy collected on his bet – as we shall see later.[10]

Around 1970, three students at Northwestern University in Illinois, David Slate, Larry Atkin, and Keith Gorlen, began writing a series of chess programs. The first of these, CHESS 3.0, running on a CDC 6400 computer, won the first Association for Computing Machinery's computer chess tournament (computers against computers) in New York in 1970. There were six entries – MAC HACK VI not among them. According to David Levy, "CHESS 3.0 evaluated approximately 100 positions per second and played at the 1400

level on the U.S. Chess Federation rating scale." Subsequent Northwestern programs, up through CHESS 4.6, achieved strings of wins at this annual event. Meanwhile, however, CHESS 4.2 was beaten in an early round of the first World Computer Chess Championship tournament held at the International Federation of Information Processing Societies (IFIPS) meeting in Stockholm in 1974. The Russian program, Kaissa, won all four games in that tournament, thereby becoming the world computer chess champion.[11]

These years, the late 1960s through the mid-1970s, saw computer chess programs gradually improving from beginner-level play to middle-level play. Work on computer chess during the next two decades would ultimately achieve expert-level play, as we shall see in a subsequent chapter. Despite this rapid progress, it was already becoming apparent that there was a great difference between how computers played chess and how humans played chess. As Hans Berliner, a chess expert and a chess programming expert, put it in an article in *Nature*,[12]

> [A human] uses prodigious amounts of knowledge in the pattern-recognition process [to decide on a good maneuver] and a small amount of calculation to verify the fact that the proposed solution is good in the present instance. ... However, the computer would make the same maneuver because it found at the end of a very large search that it was the most advantageous way to proceed out of the hundreds of thousands of possibilities it looked at. CHESS 4.6 has to date made several well known maneuvers without having the slightest knowledge of the maneuver, the conditions for its applications, and so on; but only knowing that the end result of the maneuver was good.

Berliner summed up the difference by saying that "The basis of human chess strength, by contrast [with computers], is *accumulated knowledge*" (my italics). Specific knowledge about the problem being solved, as opposed to the use of massive search in solving the problem, came to be a major theme of artificial intelligence research during this period. (Later, however, massive search regained some of its importance.) Perhaps the most influential proponents of the use of knowledge in problem solving were Edward Feigenbaum and his colleagues at Stanford. I'll turn next to their seminal work.

Notes

1. Arthur L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers II – Recent Progress," *IBM Journal of Research and Development*, Vol. 11, No. 6, pp. 601–617, 1967. [**251**]

2. http://www.cs.ualberta.ca/~sutton/book/11/node3.html. [**251**]

3. Alan Kotok, "A Chess Playing Program for the IBM 7090 Computer" MIT bachelor's thesis in Electrical Engineering, June 1962. Online versions of the thesis are available at http://www.kotok.org/AK-Thesis-1962.pdf and http://www.kotok.org/AI_Memo_41.html. (The latter is an MIT memo in which Kotok pointed out that "...this report, while written by me, represents joint work of 'the chess group,' which consisted of me, Elwyn R. Berlekamp (for the first year), Michael Lieberman, Charles Niessen, and Robert A. Wagner (for the third year). We are all members of the MIT [undergraduate] Class of 1962.) [**251**]

4. The Computer History Museum has a video "oral history" of Kotok available at http://www.computerhistory.org/chess/alan_kotok.oral_history_highlight.102645440/index.php?iid=orl-433444ecc827d. [**251**]

5. G. M. Adelson-Velskiy, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii and A.V. Uskov, "Programming a Computer to Play Chess," *Russian Mathematical Surveys* 25, March–April 1970, pp. 221–262, London: Cleaver-Hume Press. (Translation of *Proceedings of the 1st Summer School on Mathematical Programming*, Vol. 2, pp. 216–252, 1969.) [**251**]

6. See the oral presentation about the history of computer chess at http://video.google.com/videoplay?docid=-1583888480148765375. [**252**]

7. Richard D. Greenblatt, Donald E. Eastlake III, and Stephen D. Crocker, "The Greenblatt Chess Program," AI Memo 174, April 1969. Available online at https://dspace.mit.edu/bitstream/1721.1/6176/2/AIM-174.pdf. [**252**]

8. See an account in the *SIGART Newsletter*, December 1968. [**252**]

9. Hubert L. Dreyfus, "Alchemy and Artificial Intelligence," RAND paper P-3244, p. 10, The RAND Corporation, Santa Monica, CA, December 1965. Available online at http://www.rand.org/pubs/papers/2006/P3244.pdf. [**252**]

10. For first-hand details about the bet, see David Levy, *Robots Unlimited: Life in a Virtual Age*, p. 83, Wellesley, MA: A K Peters, Ltd., 2006. [**252**]

11. See the Computer History Museum's exhibits on the history of computer chess at http://www.computerhistory.org/chess/index.php. For a concise timeline of computer chess history compiled by Bill Wall, visit http://www.geocities.com/SiliconValley/Lab/7378/comphis.htm. [**253**]

12. Hans J. Berliner, "Computer Chess," *Nature*, Vol. 274, p. 747, August 1978. [**253**]

15.0

Chapter 15

The Dendral Project

After Ed Feigenbaum moved from UC Berkeley to Stanford in 1965, he became interested in "creating models of the thinking processes of scientists, especially the processes of empirical induction by which hypotheses and theories were inferred from data." As he put it, "What I needed was a specific task environment in which to study these issues concretely."[1] Feigenbaum recalls attending a Behavioral Sciences workshop at Stanford and hearing a talk by Joshua Lederberg (1925–2008; Fig. 15.1), a Nobel Prize–winning geneticist and founder of the Stanford Department of Genetics. Lederberg talked about the problem of discerning the structure of a chemical compound from knowledge of its atomic constituents and from its mass spectrogram. This sounded like the kind of problem Feigenbaum was looking for, and he and Lederberg soon agreed to collaborate on it.[2]

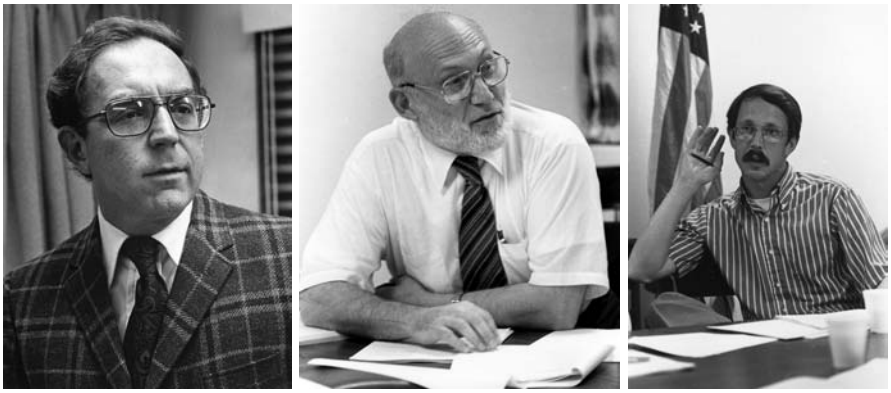


Figure 15.1: Edward Feigenbaum (left), Joshua Lederberg (middle), and Bruce Buchanan (right). (Photographs courtesy of Edward Feigenbaum.)

Chemical molecules are described by formulas that give their atomic constituents. For example, the formula for propane is C_3H_8, indicating that it consists of three carbon atoms and eight hydrogen atoms. But there is more to know about a compound than what atoms it is made of. The atoms composing a molecule are arranged in a geometric structure, and chemists want to know what that structure is. The three carbon atoms in propane, for example, are attached together in a chain. The two carbon atoms at the ends of the chain each have three hydrogen atoms attached to them, and the single carbon atom in the middle of the chain has two hydrogen atoms attached to it. Chemists represent this structure by the diagram shown in Fig. 15.2.

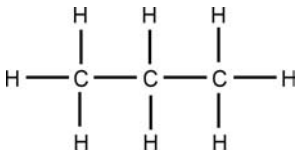


Figure 15.2: The structure of the propane molecule.

Chemists have found that it is not too difficult to discern the structure of simple compounds like propane. However, it is more difficult for more complex compounds, such as 2-methyl-hexan-3-one, a ketone with chemical formula C_7H_{140}. One method that chemists have used to infer the structure of a compound is to bombard it with high-energy electrons in a mass spectrometer. The electron beam of a mass spectrometer breaks the compound into fragments, and the resulting fragments are sorted according to their masses by a magnetic field within the spectrometer. A sample mass spectrogram is shown in Fig. 15.3.

The fragments produced by the mass spectrometer tend to be composed of robust substructures of the compound, and the masses of these reveal hints about the main structure. An experienced chemist uses "accumulated knowledge" (to use Berliner's phrase) about how compounds tend to break up in the mass spectrometer to make good guesses about a compound's structure.

Feigenbaum and Lederberg, together with their colleague Bruce Buchanan (1940–), who had joined Stanford in 1966 after obtaining a Ph.D. in Philosophy at the University of Michigan, set about attempting to construct computer programs that could use mass spectrogram data, together with the chemical formula of a compound, to "elucidate" (as they put it) the structure of the compound.

Lederberg had already developed a computer procedure called Dendral (an acronym for Dendritic Algorithm) that could generate all topologically possible acyclic structures given the chemical formula and other basic chemical information about how atoms attach to other atoms. (An acyclic structure is one that does not contain any rings. You might recall, for example, that

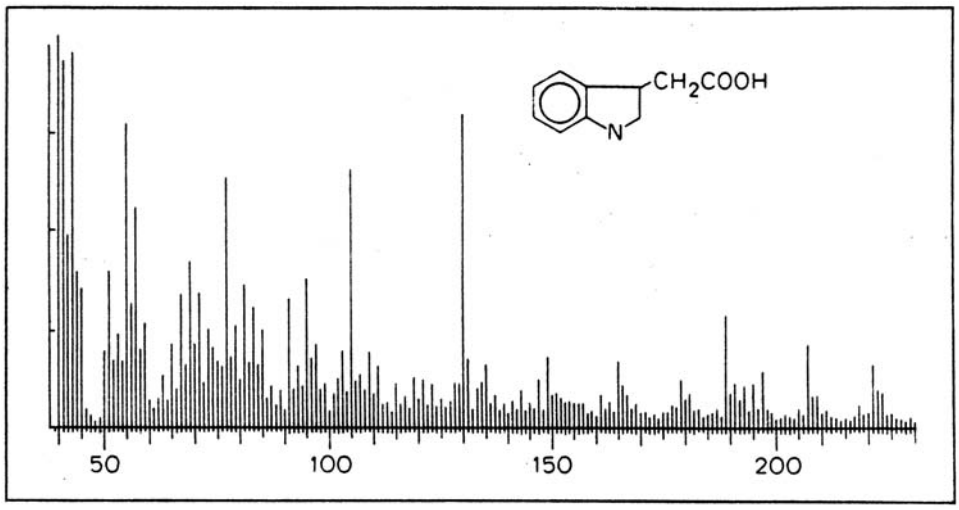


Figure 15.3: A mass spectrogram. (Illustration used with permission of Edward Feigenbaum.)

benzene contains six carbon atoms arranged in a hexagon, which chemists call a ring. Each of the carbon atoms has a hydrogen atom attached to it.) Lederberg's algorithm proceeded incrementally by generating partial structures from the main formula, then generating more articulated partial structures from these and so on in a treelike fashion. The tips or leaves of the tree would contain the final, fully articulated topologically possible structures. Finding the actual structure of a compound (or at least the most plausible actual structures) can be likened to a search down the tree to the appropriate tip or tips.

Feigenbaum and colleagues proposed using the knowledge that skilled chemists used when interpreting mass-spectral data. The chemists knew that certain features of the spectrograms implied that the molecule under study would contain certain substructures and would not contain other ones. This knowledge could be used to limit the possible structures generated by Lederberg's Dendral algorithm. Knowledge of this sort was represented as "rules." Here is one example of a Dendral rule:

```
Rule 74:
    IF   The spectrum for the molecule has two peaks
         at masses X1 and X2 such that:
             X1 + X2 = M + 28
                 and
             X1 - 28 is a high peak
                 and
```

```
          X2 - 28 is a high peak
              and
          at least one of X1 or X2 is high
THEN The molecule contains a ketone group
```

The first program to employ this kind of knowledge was called HEURISTIC DENDRAL. (The adjective "heuristic" was used because knowledge from the chemists was used to control search down the Dendral tree.) It used as input the chemical formula and mass-spectrometer data (and sometimes nuclear-magnetic-resonance data) and produced as output an ordered set of chemical structure descriptions hypothesized to explain the data. Early work with HEURISTIC DENDRAL was limited to elucidating the structure of acyclic compounds because these were the only ones that Lederberg's algorithm could handle. These included saturated acyclic ethers, alcohols, thioethers, thiols, and amines. Here is one example of the power of their early program: There are 14,715,813 possible structures of *N*,*N*-dimethyl-1-octadecyl amine. Using the mass spectrum of that compound, HEURISTIC DENDRAL reduced the number to 1,284,792. Using the mass spectrum and nuclear-magnetic-resonance data, just one structure survived.[3]

The name "DENDRAL" came to describe a whole collection of programs for structure elucidation developed during the Dendral project, which continued to the end of the 1970s. Many of these programs are used by chemists today. Computer scientists and chemists working on the project were able to extend Lederberg's algorithm to handle cyclic compounds. After Lederberg persuaded Stanford chemist Carl Djerassi to join the project, performance was expanded greatly in both breadth and depth.[4]

An important innovation made during the Dendral project was a simulation of how a chemical structure would break up in a mass spectrometer. After HEURISTIC DENDRAL produced some candidate structures for a particular compound, these structures were subjected to analysis in the simulated mass spectrometer. The outputs were then compared with the actual mass spectrometer output. That structure whose simulated spectrogram was closest to the actual spectrogram was likely to be the actual structure of the compound. This process of "analysis by synthesis" came to be widely used in artificial intelligence, especially in computer vision.

From his experience during the DENDRAL years, Feigenbaum went on to champion the importance of specific knowledge about the problem domain in AI applications (as opposed to the use of general inference methods). He proposed what he called the "knowledge-is-power" hypothesis, which he later called the "knowledge principle."[5] Here is how he later described it:[6]

> We must hypothesize from our experience to date that the problem solving power exhibited in an intelligent agent's performance is primarily a consequence of the specialist's knowledge employed by the agent, and only very secondarily related to the generality and

> power of the inference method employed. Our agents must be knowledge-rich, even if they are methods-poor.

Embedding the knowledge of experts in AI programs led to the development of many "expert systems," as we shall see later. It also led to increased concentration on specific and highly constrained problems and away from focusing on the general mechanisms of intelligence, whatever they might be.

Notes

1. The quotation taken from "Comments by Edward A. Feigenbaum" in Edward H. Shortliffe and Thomas C. Rindfleisch, "Presentation of the Morris F. Collen Award to Joshua Lederberg," *Journal of the American Medical Informatics Association*, Vol. 7, No. 3, pp. 326–332, May–June 2000. Available online at http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=61437. [**255**]

2. For an interesting account of the history of their collaboration, see "How DENDRAL Was Conceived and Born," by Joshua Lederberg, a paper presented at the Association for Computing Machinery (ACM) Symposium on the History of Medical Informatics at the National Library of Medicine on November 5, 1987. Later published in Bruce I. Blum and Karen Duncan (eds.), *A History of Medical Informatics*, pp. 14–44, New York: Association for Computing Machinery Press, 1990. Typescript available online at http://profiles.nlm.nih.gov/BB/A/L/Y/P/_/bbalyp.pdf. [**255**]

3. Robert K. Lindsay, Bruce G. Buchanan, Edward A. Feigenbaum, and Joshua Lederberg, *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project*, p. 70, New York: McGraw-Hill Book Co., 1980. [**258**]

4. For a thorough account of achievements of the Dendral project, see *ibid.* [**258**]

5. The hypothesis seems to have been implicit in Edward A. Feigenbaum, "Artificial Intelligence: Themes in the Second Decade," *Supplement to Proceedings of the IFIP 68 International Congress*, Edinburgh, August 1968. Published in A. J. H. Morrell (ed.), *Information Processing 68*, Vol. II, pp. 1008–1022, Amsterdam: North-Holland, 1969. [**258**]

6. Edward A. Feigenbaum, "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 1014–1029, 1977. See also Edward A. Feigenbaum, "The Art of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering," Stanford Heuristic Programming Project Memo HPP-77-25, August 1977, which is available online at http://infolab.stanford.edu/pub/cstr/reports/cs/tr/77/621/CS-TR-77-621.pdf. [**258**]