ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE

ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE

VOLUME 2

Stuart C. Shapiro, *Editor-in-Chief*
David Eckroth, *Managing editor*
George A. Vallasi, *Chernow Editorial Services, Developmental Editor*

Wiley-Interscience Publication
John Wiley & Sons
New York / Chichester / Brisbane / Toronto / Singapore

Library of Congress Cataloging in Publication Data:

Encyclopedia of artificial intelligence.

"A Wiley-Interscience publication."
1. Artificial intelligence—Dictionaries.
I. Shapiro, Stuart Charles. II. Eckroth, David.
Q335.E53 1987 006.3'03'21 86-26739
ISBN 0-471-80748-6 (set)
ISBN 0-471-62973-1 (Vol. 2)

Printed in the United States of America
10 9 8 7 6 5 4

ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE

VOLUME 2

O

OBJECT-ORIENTED PROGRAMMING. See Languages, object-oriented.

OFFICE AUTOMATION

Office work is of two distinct sorts: subject and overhead. The subject work addresses the business of the organization running the office: for a doctor's office, this includes medicine (e.g., diagnosis, treatment) and scheduling (maintaining an appointment calendar); for a sales office, order maintenance, delivery scheduling, compensation, and order forecasting; for an insurance company, financial planning; and for an oil company, geology. In contrast, the overhead is that work done in the process of "running" the office: this concerns people (hiring, firing, paying, directing), physical arrangements (space, desks, chairs), communication (letters, phones, meetings), bureaucracy (forms, policies, procedures), and organization (managing).

Subject Domains

The domains of subject work cover all of the world's business, a detailed discussion of which is beyond the scope of this article. However, some of these domains are associated more directly with offices, because they appear primarily in office contexts. For example, expert systems (qv) for financial planning, insurance underwriting, legal document preparation, inventory control and sales territory management are all domains traditionally carried out in offices in which AI systems are being used.

Such "expert systems" are most attractive in areas where experts are few, but many are needed (financial advising, insurance underwriting, machinery diagnosis, exploratory geology). Here the expense of an expert system can be justified as enabling less expert people to do the job. Some of the current excitement over AI comes from its notable successes in a number of these domains, particularly ones with high payoff for better answers.

Recently, systems have become available for both mainframes and, more interesting, for personal computers (1), which deliver AI-program construction techniques to nontechnical "users." These provide environments for creating rule sets which potentially capture a domain of interest, and an interpreter for running that rule set on descriptions of particular cases. Sometimes, these construction environments also provide tools for creating attractive user interfaces to the expert systems thus constructed. As a result, AI is playing an increasing role in office systems by acting as the technological base for their construction, making them faster to build and modify and accessible by those lacking the command of more traditional programming practice.

AI programs are not unique here. The very characteristics which make AI-based programs attractive in offices are those which have made word-processing, spreadsheets, and databases attractive: they can be quickly changed and the functionality of the system can be left in the hands of the interested user. In fact, this responsiveness often makes the resulting systems very responsive to the needs of their (often very specific) domains. The abstractions of those domains appear (unusually uninterpreted) in the systems. Therefore, to an observer, they often appear to understand their domains very well. These similarities add fuel to the debates concerning the definition of AI.

Routine and Nonroutine Activity

The fundamental fact about offices is that they are inherently composed of people. People make up an office because there is a changing environment within which the parent organization must function. To do this, the people of the organization must constantly be "making sense" of that world, interpreting themselves and the organization in that context.

Since machines cannot sense that world, they cannot interpret it. To the extent that an interpretation can be settled upon, and conveyed to the machine, and that interpretation made to apply for many cases, the machine can be used to handle the growing volume of transactions which make up modern business. To support this, it is necessary that people preprocess the varied world into input expressed in the terms of the chosen interpretation and reinsert through interpretation the machine's output back into that world. This leads to segregating office work into the routine (for machines) and the nonroutine (for people). Thus "office automation" only automates the office in the sense that it transforms the view of office work to enable automation (computers) to play a role.

From this viewpoint, the role of AI-based programs in offices is not significantly different from that of more traditional office computer systems. The AI-based programs being used in offices, while often improvements over less flexible traditional programs, are still incapable of being really responsive to the world in which they are embedded. On the whole technologies are unaware even that there is a situation to be made sense of. Beyond that, they are not capable of conceiving that the characteristics which they address need to be augmented or modified, much less of determining, from confrontation with the actualities of a presented situation, either the nature or the details of those changes. Thus the systems do not work well when used beyond the limits of their expertise. Indeed, they do not as yet have much sense of their own limits (the judgment of which even people find difficult).

Thus in the end, people must interpret these systems to the world. Therefore, the impact of AI is, at most, to change the boundary between the routine and the nonroutine and make that boundary more maleable, but it does not remove it: the essential office, in which people address an ever-changing world, remains unchanged.

Running the Office: Overhead Domains.

Consider now the overhead of the office, that work which is quintessentially "office work." The office is a sociotechnical environment made up of many systems interacting: physical, technical, intellectual, social, bureaucratic, organizational. The task of an office worker is to do whatever is necessary to achieve certain ends within the organization, with those ends involving other people, taking place in certain physical locations, with certain technical assists. The organization's goals concern business, but also include legality (2), and responsibility (3,4). Many of these goals are unarticulated. Office work is always changing. The mechanisms and agreements set up by people to get the job done are highly individual and specific. They must be easy to set up and dismantle, often being active for just minutes ("Answer the phone while I make a copy, will you?"). Office work often has a large social component. As an important example, consider the role of a secretary: to catalyze and support communication for the people in the office with each other, the organization and the world.

The account just given of the separation into routine and nonroutine is particularly applicable to office overhead work. Further, application of computers to this domain has been made more difficult by unrealistic views of this work, views which see it as much more routine than it is. The perception of regularity in these views is achieved by looking too narrowly at what is happening. Two good examples are in the domains of flow of paper work and calendars.

Bureaucracy presents the picture of an office as a place where forms flow from worker to worker in a pattern specified by "office procedures." A number of research efforts accepted this apparent regularity and attempted to build systems which would aid in the process, either by modelling it for purpose of analysis (5,6), or by supporting it for purpose of reducing the load on workers in remembering and tracking the flow of "paper" (6,7,8). These efforts were not highly successful because they are based on invalid assumptions about the flow of paper in an office: The set of procedures specifying flow is only one of the factors determining the movement of paper in the office, and the others were not addressed by the systems. Moreover the procedures themselves are only a surface manifestation of a set of underlying goals which the paper flow, indeed the paper itself, is a device for achieving (9).

People in offices use calendars to plan their time. Calendar systems have been built to help people with this task. They have recognized that an important part of this is to help with the coordination required for people to arrange meeting times. These systems have not been generally accepted because they look at the use of calendars too narrowly. First, people move about, and an immobile calendar is simply not acceptable (10). More interesting, because a calendar represents a person's commitments, it also represents priorities, and this is highly personal and contextually sensitive information. Consequently the coordination of meeting times is not a matter of simply finding corresponding available slots in various calendars, but rather is a matter of negotiating, which often turns on complex social and organizational power relationships among those concerned.

All this is not to say that computers and AI will not address these and other domains comprising office overhead work. Indeed, forms and calendar systems have already played an important part in supporting the work done in offices. However, to achieve this, they will have to be based on realistic views of office work. The niceties of the various interactions, particularly the social interactions, are well out of reach of today's AI technology. And although more easily changed than traditional programs, they still cannot change as fast as the office does. Therefore even AI computer systems are best viewed as being in that part of the office which handles the routine part of the work.

Office Educator: A New Role for AI Systems

One advantage of AI technology over traditional computer systems is that the knowledge about its application domains is made explicit. Thus AI enables a major new role for computer technology within offices: that of educator and communicator. Through AI applied as an adjunct to other systems, both technological and human, people in offices can better come to understand these complex multistructured environments. Through understanding they can gain better command of them. In addition, through this improved command, AI can even enable new environments which are simply not possible without it. More generally, AI can enable changed relationships between office workers and their environments.

It should be noted that this new role for AI is not confined to the office. Explicit knowledge will support the educator's role wherever AI is used. However, because offices are a very common workplace in today's world, AI's role as educator may be more visible to a large nontechnical audience in the office than elsewhere.

The rest of this article discusses four aspects of the office environment concerning which AI may educate: technology, communications, information domains, and the organization itself. This framework provides a perspective on the AI work that has already been done in office systems. Each section indicates how AI systems can provide people with better understanding and command of their environments.

Understanding Machines

Office machines are becoming more complex. Text editors are difficult to learn (11); manuals for office systems are imposing, even copiers are the subject of study (12). The utility of these machines is limited less by what their developers can provide than by what their users can understand.

A primary role for AI in the offices, then, is to enable better user interfaces. Training tutorials (13,14) can introduce people formally to the machine. Situated online help (15) and intelligent interfaces can assist in the actual use. The studies of what it is to interact with a machine (12) are showing the need for the machine to be able to consider itself as a player in the interaction, and to use its understanding of the interaction itself as a basis for reasoning about that interaction. The work already done on reflection and introspection, reasoning, and repair (16) will be important in achieving this.

One of the most difficult things about the new machines is that they are software-based. The culturally provided way of understanding machinery (as the interaction of comprising subparts) provides no access to this technology. Simulations of the machine while it is running can provide subtle (even enjoyable) support for mental models of the machine which will support reasoning about it (17).

New paradigms for the control of the interaction between user and computer system are also enabled by AI. For example, the work on ONCOCIN (see Medical advice systems) includes exploration of partitioning the responsibility: the user (in this case a doctor treating patients with drugs as part of a medical experiment) is responsible for the dosages given to the patient; the machine is responsible for the acceptibility of the dosages given for use in research on treatment methods. To achieve these new styles of interaction between user and machine, the machine must educate the user in the role that they are to play with respect to the machine.

Learning is often best done in a hands-on, "learn through doing" fashion. To support this, the environment must encourage exploration, most particularly by rendering errors hard to make inadvertently or painless to recover from. AI technology has led the way in showing how to do this (18). This attitude will be required by all users, since "closed" systems which can be completely learned will become less and less the rule. A complementary need is to develop in users an attitude that admits partial, incomplete, possible even wrong, understanding of the system as an acceptable state of knowledge: the "purposes at hand" determine what is needed. Online help systems carry this message; dynamically modifiable ones would further it.

Finally, that machines must suit many different styles of usage will require that they be "tailorable" to individual needs. This tailoring can be done explicitly (through specification) or implicitly (e.g., by example). AI can help with each. Explicit specification requires models of the machine and how specification can change it. Tailoring by example requires pattern generalization and recognition (19).

Understanding Communications

Communications are central to running an office. Indeed communications are the primary subject domain for people like secretaries. AI is beginning to play a role in helping with electronic mail, through aid with addressing (20), with selecting recipients (21), and with sorting mail when received. Voice mail and integrated telephone systems are being explored.

Although these roles for AI will be important, the more profound effects of AI will be in helping people understand the office communication patterns in which they are involved. To large measure, people's knowledge of communications systems is tacit. AI can educate directly by making this information explicit. This should include not only details of the technology (e.g., the need to use protection mechanisms, and the ability to find the person who can supervene them) but also the social mores associated with using it (22) (e.g., who may use a distribution list for what purpose; and expectations, such as how soon messages must be answered, and the social implications of not meeting these expectations). AI can provide descriptions of the communications media, both inside and outside the machine, which will permit users not only to answer the questions concerning the usage of the media, but also to know what questions to ask. Further, with the availability of this information to the communications systems themselves, AI can support the creating of agents for dealing with situated problems (e.g., knowledge-based descriptions of standing-orders) (23).

Educating indirectly, AI can change the very nature of the communications media, providing new opportunities in the office. For example, electronic mail is a new medium with a new set of social values. In particular, it is more formal than the telephone and less formal than office memos. In creating messages, the impact of these felicity conditions on formality is subtle. Having spelling and grammar checkers available may reduce fears associated with sending a message in haste. On the other hand, it may be important to also provide mechanisms which guarantee that a message looks hurried, thereby communicating to the receiver the intent of the sender. Although powerful and potentially very useful, the full impact of such AI systems on communications in the office must be studied with care.

Understanding Information Domains

One of the cornerstones of AI technology is the explicit representation of the ontology of its application domains. (This is the "knowledge base" of the domain, in terms of which the "data" of the domain is understood and encoded.) In addition to supporting the application, this explicit encoding of domain ontology can be used to teach the user about the domain. This is important in training new personnel and in reducing memory load for experienced users. The presentation can be direct (tools which present the information explicitly to the user) or indirect (e.g., during data acquisition or information retrieval (24)). Explicit ontology can also be used to support its own change. This is particularly useful in the office environment where the ontology of information is often changing rapidly, not only by extension, but also by reinterpretation. With the support of change and the capacity for querying the ontology of the data comes a potential for information change not currently available: the information system can itself act as a communication system for discussing and controlling the terms in which the domains addressed are described (25). This is only one form of "idea generation" in which the systems themselves support the exploration of not only ideas but the frameworks in which those ideas are expressed.

Another dimension of difference between AI ontologies and traditional ones (e.g., data dictionaries, relational databases) is the sophistication of AI knowledge representation (qv) techniques in representing partial, hypothetical, alternative, dependent, and contingent information. This sophistication offers fundamentally new capabilities in the office, where such incomplete and complex information is the rule rather than the exception. AI ontologies also handle heterogeneity in a way not found in traditional ontologies, permitting the various relationships between terms in the ontology to be expressed in detail. Such heterogeneity presents a new solution to an old problem, that of determining, when retrieving information, the terms in which to state that query. Retrieval by reformulation (24) offers a mode of relationship with the information system which contrasts sharply with existing technology: the information system can help with the formulation of the query itself. The user's attitude toward this task can therefore change radically.

Finally, a wholly new continuity to information can be achieved through the explicit representation of the ontology of that information when the changes in that ontology are recorded as well. All the information, both past and present, can be viewed as part of a single structure and the changes in its meaning explored. Tracking the history of information (26) can provide a view of the office which carries with it not only the potential for reviewing that history, but also the possibility for reasoning about it.

Understanding Organizations

The people in an office are located in a matrix of different structures: physical, technical, intellectual, social, bureaucratic, organizational. When AI capabilities are coupled with communications, the people using the systems can be supplied with information concerning these structures.

As discussed earlier, support of office work has focussed on a bureaucratic view which sees office work as policies, procedures, forms and forms-flow (7,27,28,29). Although this limited view does not have the power to even adequately describe the work involved in the paperwork of the office, this information is an important driver of that work (9). Making this information explicit within the system offers for the organization all the advantages described above for information spaces: learning, increased sophistication, and history.

More radically, explicit encoding of the structures composing the office matrix offers the possibility of new structures themselves, and hence significantly altered views of the office and office work. At the low end, the explicit encoding of the content of a group discussion can provide new forms of interaction within meetings. "Working at home" can be enhanced by explicit encoding of project information. Distributed work groups may not have either an "office" or a corresponding organizational structure, but rather only agreements to collaborate and a sense of presence of participants maintained through explicitly encoded group information. An even less formal, yet still explicit, form of organization is that provided by distribution lists and teleconferencing within electronic mail. Coupling this with explicit encoding of the structural matrix provides for yet other new interactions within the "office"; the organizationally constructed structure can itself be an important driver of the interactions (21). Finally, small organizations have a fluid structural matrix which makes them particularly productive: there is no need to create organizational or bureaucratic entities to correspond to projects. It may be possible to extend such fluid and complex structures to larger organizations through explicit encoding and manipulation of these organizational relationships.

So by explicitly encoding the structural matrix of an organization and by making that structure a domain for attention, AI technology can be used proactively to create awareness of, and changes in, the processes by which offices and organizations run.

Summary

AI technology is being used to create office systems for addressing both subject and overhead domains of office work. This technology provides better system construction and maintenance techniques and extended system functionality. However, the essential core of office work is matching the organization to its diverse and changing environment. Since even AI systems are, as yet, not directly responsive to their environments, all office systems must still be regarded as doing the routine work for their human partners who handle diversity and change. But, unlike traditional systems, AI systems explicitly encode information about their domains. Therefore they can extend their limited roles to include that of office educator, providing people in offices with new ways to gain and manipulate the information necessary to command their machines, their communications media, their subject domains and even the organization itself.

BIBLIOGRAPHY

1. T. J. Schwartz, "Artificial Intelligence in the Personal Computer Environment, Today and Tomorrow," *Proc. of the Ninth Int. Jt. Conf. on Artif. Intell.*, Los Angeles, CA, 1985, 1261–1262.
2. M. A. Boden, "Panel: Artificial Intelligence and Legal Responsibility," *Proc. of the Ninth Int. Jt. Conf. on Artif. Intell.*, Los Angeles, CA, 1985, 1267–1268.
3. H. Thompson, "Empowering Automatic Decision-Making Systems: General Intelligence, Responsibility and Moral Sensibility," *Proc. of the Ninth Int. Jt. Conf. on Artif. Intell.*, Los Angeles, CA, 1985, 1281–1283.
4. Y. Wilks, "Responsible Computers?", *Proc. of the Ninth Int. Jt. Conf. on Artif. Intell.*, Los Angeles, CA, 1985, 1279–1280.
5. G. Bracchi and B. Pernici, "The Design Requirements of Office Systems," *ACM Transaction on Office Information Systems* **2**(2), 151–170 (April 1984).
6. G. J. Nutt and C. A. Ellis, "Computer Science and Office Information Systems," SSL-79-6, Xerox Corporation, Palo Alto, CA, 1980.
7. G. Barber, "Supporting Organizational Problem Solving with a Work Station," *ACM Transaction on Office Information Systems* **1**(1), 45–67 (January 1983).
8. S. B. Yao, A. R. Hevner, Z. Hi, and D. Luo, "FORMANAGER: An Office Management System," *ACM Transaction on Office Information Systems* **2**(3), 235–262 (July 1984).
9. L. A. Suchman, "Office Procedure as Practical Action: Models of Work and System Design," *ACM Transaction on Office Information Systems* **1**(4), 320–328 (October 1983).
10. C. M. Kincaid, P. B. DuPont, and A. R. Kaye, "Electronic Calendars in the Office: An Assessment of User Needs and Current Technology," *ACM Transaction on Office Information Systems* **3**(1), 89–102 (January 1985).
11. R. L. Mack, C. H. Lewis, and J. M. Carroll, "Learning to Use Word Processors: Problems and Prospects," *ACM Transaction on Office Information Systems* **1**(3), 254–271 (July 1983).
12. L. A. Suchman, "Plans and Situated Actions: The problem of human–machine communication," Dissertation. University of California, Berkeley, 1984.
13. J. S. Brown, R. R. Burton, and J. deKleer, "Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III," in D. Sleeman, J. S. Brown, *Intelligent Tutoring Systems* Academic Press, London, 1982.
14. R. R. Burton and J. S. Brown, "An investigation of computer coaching for informal learning activities," in D. Sleeman, J. S. Brown, *Intelligent Tutoring Systems,* Academic Press, London, 1982.
15. G. Fischer, A. Lemke, and T. Schwab, "Knowledge-based Help Systems," *Proc. CHI'85 Human Factors in Computer Systems,* ACM, New York, 161–167, 1985.
16. J. S. Brown and K. VanLehn, "Repair Theory: A Generative Theory of Bugs in Procedural Skills," *Cognitive Science* **4**(4), 379–476 1980.
17. T. Moran, "Getting Into a System: External-Internal Task Mapping Analysis," *Proc. CHI'83 Human Factors in Computer Systems,* ACM, New York, 45–49, 1983.
18. W. Teitelman, "Toward a Programming Laboratory," *Proc. of the First Int. Jt. Conf. on Artif. Intell.,* Washington, DC, 1969, 1–8a.
19. D. C. Halbert, "Programming by example," Dissertation. University of California, Berkeley, 1984.
20. D. Tsichritzis, "Message Addressing Schemes," *ACM Transaction on Office Information Systems* **2**(1), 58–77 (January 1984).
21. T. W. Malone, K. R. Grant, and F. A. Turbak, "The Information Lens: An Intelligent Sharing in Organizations," *Proc. CHI'86: Human Factors in Computing Systems,* April 1986, 1–8.
22. D. K. Brotz, "Message System Mores: Etiquette in Laurel," *ACM Transaction on Office Information Systems* **3**(2), 179–192 (April 1983).
23. T. Kaczmarek, W. Mark, and N. Sondheimer, "The Consul/CUE Interface: An Integrated Interactive Environment," *Proc. CHI'83 Human Factors in Computer Systems,* ACM, New York, 98–102, 1983.
24. F. N. Tou, M. D. Williams, R. E. Fikes, D. A. Henderson, T. W. Malone, "RABBIT: An intelligent database assistant," *Proc. of the Second National Conference on Artificial Intelligence, AAAI,* Pittsburgh, PA, 1982, 314–318.

25. D. A. Henderson, Jr., "The Trillium User Interface Design Environment," *Proc. CHI'86: Human Factors in Computing Systems,* April 1986, 221–227.
26. D. Bobrow and I. Goldstein, "An Experimental Description-based Programming Environment: Four Reports," CSL-81-3, Xerox Corporation, Palo Alto, CA, 1981.
27. R. E. Fikes, "Odyssey: a knowledge-based assistant," *Artificial Intelligence Journal* **3**(16), 331–362.
28. R. E. Fikes, "On supporting the use of procedures in Office Work", *Proc. of the First National Conference on Artificial Intelligence,* Stanford, CA, 1980, 202–207.
29. R. E. Fikes, "A committment-based framework for describing informal cooperative work," *Cognitive Science* **6**(4), 331–347, 1982.

D. Austin Henderson, Jr.
Xerox PARC

OPS-5

OPS-5 is a domain-independent production system. It is the latest in a series that started with "OPS" (see C. L. Forgy and J. McDermott, OPS, a Domain-Independent Production System, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* Cambridge, MA, pp. 933–939, 1977). The main source is the OPS-5 User's Manual (see C. L. Forgy, OPS-5 User's Manual, Technical Report, CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh, PA, (July, 1981)). The OPS-5 environment is built on top of LISP (qv) (however, there exists one implementation in "BLISS") and contains a working memory and a production memory. One production consists of a left-hand side (LHS) and a right-hand side (RHS) that correspond approximately to the IF-part and THEN-part of a normal programming language. The RHS of a production is executed if its LHS matches the content of the working memory. If several LHSs match, a conflict-resolution strategy is used. If no LHS matches, the "program" terminates. The VAX configuration program R1 is a commercially used product based on OPS-4 [see J. McDermott, "R1: A rule-based configurer of computer systems," *Artif. Intell.* **19**(1), 39–88 (September 1982)]. R1 has been followed up by a new version called XCON (qv).

J. Geller
SUNY at Buffalo

OPTICAL FLOW

When an observer moves relative to the environment, the two-dimensional (2-D) image that is projected onto the eye undergoes complex changes. The pattern of movement of features in the image is referred to as the optical flow field (1). Optical flow can be represented by a 2-D field of velocity vectors as shown in Figure 1. In Figure 1*a* the optical flow is generated by the movement of an observer relative to a stationary environment. The "observer" is a camera mounted on an airplane that is flying over terrain. A single snapshot from a sequence of images is shown with reduced contrast. The black vectors superimposed on the image represent the optical flow, or velocity field. The direction and length of these vectors indicate the direction and speed of movement of features across the image as the airplane flies along. Optical flow is also generated by the motion of objects in the environment. Figure 1*b* shows three views of a three-dimensional (3-D) wire-frame object that is rotating about a central vertical axis. Figure 1*c* shows a snapshot of the object at a particular moment in time, with vectors superimposed that indicate the velocities of individual points on the object.

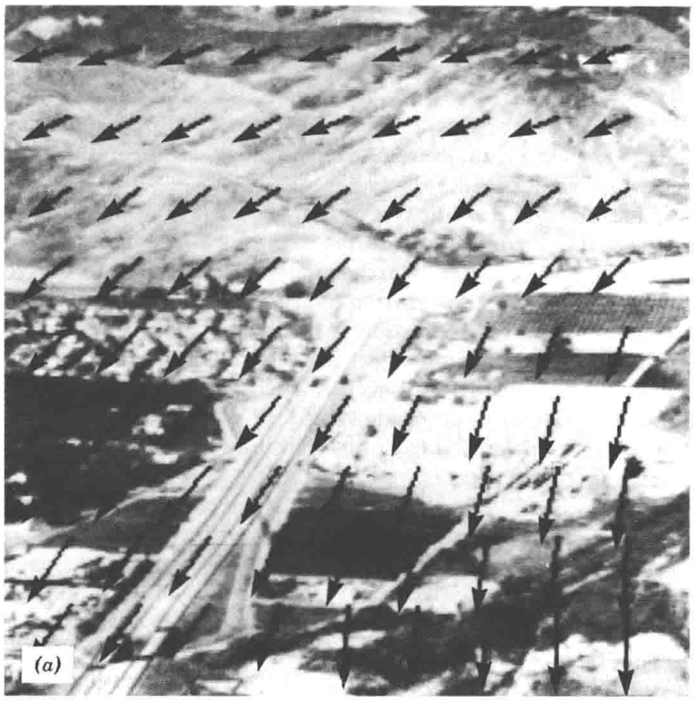
The analysis of the optical flow can be divided into two parts: The first is the measurement of optical flow from the changing image, and the second is the use of optical flow to recover important properties of the environment. The motion of features in the image is not provided to the visual system directly but must be inferred from the changing pattern of intensity that reaches the eye. Variations in the measured optical flow across the image (also known as motion parallax) can then be used to recover the movement of the observer, the 3-D shape of visible surfaces, and the locations of object boundaries. For example, from a sequence of optical flows such as that shown in Figure 1*a*, it is possible to recover the motion of the airplane relative to the ground. The variation in speed of movement of points on the wire-frame object of Figure 1*c* allows the recovery of its 3-D structure from the changing 2-D projection. Sharp changes in the optical flow field indicate the presence of object boundaries in the scene. The measurement and use of optical flow are discussed below.

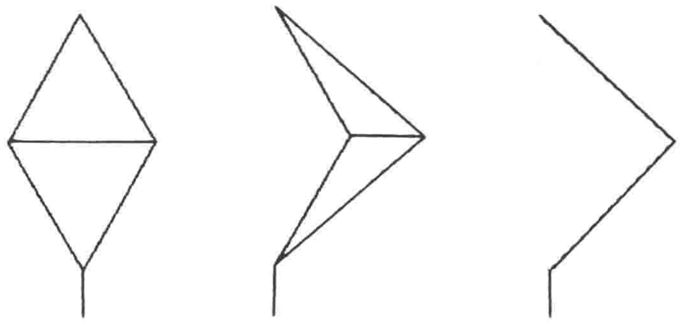
Measurement of Optical Flow

Computational studies offer a broad range of methods for measuring optical flow (for reviews, see Refs. 2–5). Some methods compute the instantaneous optical-flow field directly. Others track features across the image and thus compute a correspondence between features from one moment to the next. Methods for measuring motion also differ in the stage of image processing at which movement is first analyzed. For example, some infer movement directly from changes in the image intensities, and others first filter the image, or extract features such as edges. The range of techniques for motion measurement (see Motion analysis) are reflected in a broad range of application domains, from the simple tracking of objects along a conveyor belt in an industrial setting to the analysis of more complex motions such as that of clouds in satellite weather data, heart walls in X-Ray images, or cells in cell cultures. The analysis of optical flow is also becoming essential in autonomous navigation (see Autonomous vehicles; Robots, mobile) and robotic assembly (see Computer-integrated manufacturing; Robotics).

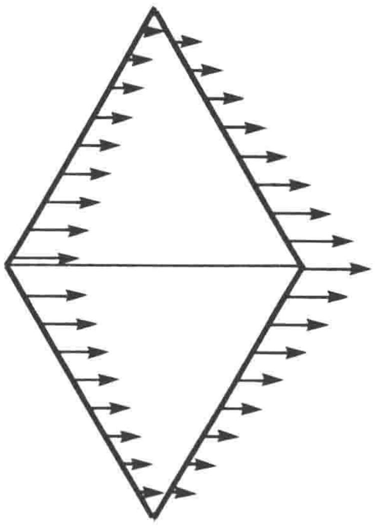
The measurement of optical flow poses two fundamental problems for computer-vision systems. First, the changing pattern of image intensity provides only partial information about the true motion of features in the image due to a problem often referred to as the aperture problem. Second, when the general motion of objects is allowed, there does not exist a unique optical-flow field that is consistent with the changing image. In theory, there exist infinite possible interpretations of the motion of features in the image. Additional constraint is required to identify the most plausible interpretation from a physical viewpoint.

The aperture problem is illustrated in Figure 2. Suppose that the movement of features in the image were first detected using operations that examine only a limited area of the image. Such operations can provide only partial information about the true motion of features in the image (2–7). In Figure



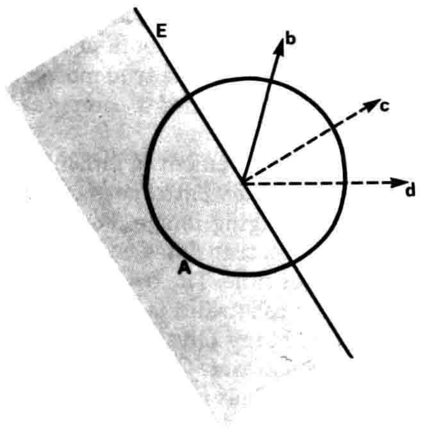


(b)

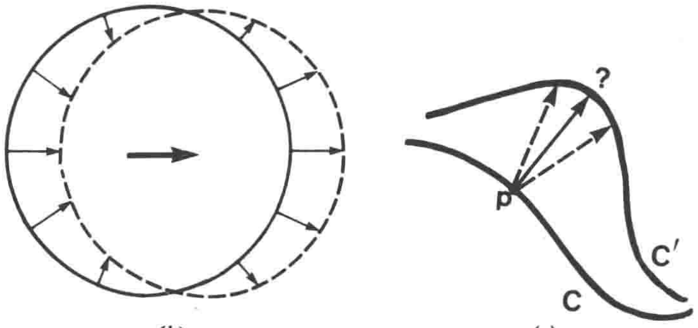


(c)

Figure 1. (a) Optical-flow field, represented by black arrows, is superimposed on a natural image that was taken from an airplane flying over terrain. (b) Three views of a wire-frame object rotating about a central vertical axis. (c) Projected pattern of velocities of individual points on the object are shown superimposed on a snapshot of the object in motion (an orthographic, or parallel projection is used).



(a)



(b) (c)

Figure 2. (a) Operation that examines the moving edge **E** through the limited aperture **A** can compute only the component of motion c in the direction perpendicular to the orientation of the edge. The true motion of the edge is ambiguous. (b) A circle undergoes pure translation to the right. The arrows along the circle represent the perpendicular components of motion that can be measured directly from the changing image. (c) A contour **C** rotates, translates, and deforms to yield the contour **C**′. The motion of the point p is ambiguous.

2a the extended edge **E** moves across the image, and its movement is observed through a window defined by the circular aperture **A**. Through this window, it is only possible to observe the movement of the edge in the direction perpendicular to its orientation. The component of motion along the orientation of the edge is invisible through this limited aperture. Thus, it is not possible to distinguish between motions in the directions b, c, and d. This property is true of any motion detection operation that examines only a limited area of the image. As a consequence of the aperture problem, the measurement of optical flow requires two stages of analysis: The first measures components of motion in the direction perpendicular to the orientation of image features; the second combines these components of motion to compute the full 2-D pattern of movement in the image. In Figure 2b a circle undergoes pure translation to the right. The arrows along the contour represent the perpendicular components of velocity that can be measured directly from the changing image. These component measurements each provide some constraint on the possible motion of the circle. Its true motion, however, can be determined only by combining the constraints imposed by these component measurements. The movement of some features such as corners or small patches and spots can be measured unambiguously in the changing image. Several methods for measuring motion rely on the tracking of such isolated features (2–4,6). In general, however, the first measurements of movement provide only partial information about the true movement of features in the image and must be combined to compute the full optical-flow field.

The measurement of movement is difficult because in theory, there are infinitely many patterns of motion that are consistent with a given changing image. For example, in Figure 2*c*, the contour **C** rotates, translates, and deforms to yield the contour **C**′ at some later time. The true motion of the point *p* is ambiguous. Additional constraint is required to identify a single pattern of motion. Many physical assumptions could provide this additional constraint. One possibility is the assumption of pure translation. That is, it is assumed that velocity is constant over small areas of the image. This assumption has been used both in computer-vision studies and in biological models of motion measurement (2–6,8). Methods that assume pure translation are useful for detecting sudden movements and tracking objects across the visual field. These methods have led to fast algorithms for computing a rough estimate of the motion of objects, which is often sufficient in applications of motion analysis. Tasks such as the recovery of 3-D structure from motion require a more detailed measurement of relative motion in the image. The analysis of variations in motion such as those illustrated in Figure 2*c* requires the use of a more general physical assumption.

Other computational studies have assumed that velocity varies smoothly across the image (5,7). This is motivated by the assumption that physical surfaces are generally smooth. Variations in the structure of a surface are usually small compared with the distance of the surface from the viewer. When surfaces move, nearby points tend to move with similar velocities. There exist discontinuities in movement at object boundaries, but most of the image is the projection of relatively smooth surfaces. Thus, it is assumed that image velocities vary smoothly over most of the visual field. A unique pattern of movement can be obtained by computing a velocity field that is consistent with the changing image and has the least amount of variation possible. The use of the smoothness assumption allows general motion to be analyzed and can be embodied into the optical-flow computation in a way that guarantees a unique solution (5). The optical-flow fields shown in Figure 1 were computed with an algorithm that uses the smoothness assumption (5).

Use of Optical Flow

Most computational studies of the use of optical flow have focused on the recovery of the movement of the observer and 3-D structure and movement of surfaces on the scene (for reviews; see Refs. 9–11). Some have also addressed the interpretation of discontinuities in the optical-flow field (e.g., Ref. 12). Theoretically, the two problems of recovering the 3-D movement of the environment and observer are closely related. A fundamental problem faced by both is that there does not exist a unique interpretation of the 2-D image in terms of the 3-D motion of the observer and visible surfaces. Additional constraint is required to obtain a unique interpretation.

With regard to the recovery of 3-D structure from motion, computational studies have used the assumption of rigidity to derive a unique interpretation. These studies assume that if it is possible to interpret a changing 2-D image as the projection of a rigid 3-D object in motion, such an interpretation should be chosen (for reviews, see Refs. 4,9–11). When the rigidity assumption is used in this way, the recovery of structure from motion requires the computation of the rigid 3-D objects that would project onto a given 2-D image. The rigidity assumption was suggested by perceptual studies that described a tendency for the human visual system to choose a rigid interpretation of moving elements (13,14).

Computational studies have shown that it is possible to use the rigidity assumption to derive a unique 3-D structure from a changing 2-D image. Furthermore, it is possible to derive this unique 3-D interpretation by integrating image information only over a limited extent in space and in time. For example, suppose that a rigid object in motion is projected onto the image using a parallel projection such as that illustrated in Figure 1*c*. Three distinct views of four points on the moving object are sufficient to compute a unique rigid 3-D structure for the points (9). In general, if only two views of the moving points are considered or fewer points are observed, there are multiple rigid 3-D structures consistent with the changing 2-D projection. If a perspective projection of objects onto the image is used instead, two distinct views of seven points in motion are usually sufficient to compute a unique 3-D structure for the points (11). Other theoretical results address the information that can be obtained directly from the optical-flow field (8,9). These and other theoretical results are summarized in Ref. 9. These results are important for two reasons. First, they show that by using the rigidity assumption, it is possible to recover a unique structure from motion information alone. Second, they show that it is possible to recover this structure by integrating image information over a small extent in space and time. Computational studies of the recovery of structure from motion provide algorithms for deriving the structure of moving objects (9–11). These algorithms can also be used to recover the motion of an observer relative to a stationary scene.

BIBLIOGRAPHY

1. J. J. Gibson, *The Perception of the Visual World*, Houghton Mifflin, Boston, MA, 1950.
2. W. B. Thompson and S. T. Barnard, "Low-level estimation and interpretation of visual motion," *IEEE Comput.* **14,** 47–56 (1980).
3. S. Ullman, "Analysis of visual motion by biological and computer vision systems," *IEEE Comput.* **14,** 57–69 (1981).
4. D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
5. E. C. Hildreth, *The Measurement of Visual Motion*, MIT Press, Cambridge, MA, 1984.
6. D. T. Lawton, "Processing translational motion sequences," *Comput. Vis. Graph. Img. Proc.* **22,** 116–144 (1983).
7. B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artif. Intell.* **17,** 185–203 (1981).
8. K. Nakayama, "Biological image motion processing: A review," *Vis. Res.* **25,** 625–660 (1985).
9. S. Ullman, Recent Computational Studies in the Interpretation of Structure and Motion, in J. Beck, B. Hope, and A. Rosenfeld (eds.), *Human and Machine Vision*, Academic, New York, 1983, pp. 459–480.
10. H. C. Longuet-Higgins and K. Prazdny. The Interpretation of a Moving Retinal Image in S. Ullman and W. Richards (eds.), *Image Understanding 1984*, Ablex, Norwood, NJ, 1984, pp. 179–193.
11. R. Y. Tsai and T. S. Huang, Uniqueness and Estimation of 3-D

Motion Parameters and Surface Structures of Rigid Objects. in S. Ullman and W. Richards (eds.), *Image Understanding 1984,* Ablex, Norwood, NJ, 1984, pp. 135–171.

12. K. M. Mutch and W. B. Thompson, "Analysis of accretion and deletion at boundaries in dynamic scenes," *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-7,** 133–138 (1985).
13. H. Wallach and D. N. O'Connell, "The kinetic depth effect," *J. Exper. Psychol.* **45,** 205–217 (1953).
14. G. Johansson, "Visual motion perception," *Sci. Am.* **232,** 76–88 (1975).

E. Hildreth
MIT

This document was produced using the facilities of the MIT Artificial Intelligence Laboratory, supported in part by DARPA contract N00014-80-C-0505.

P

PAM

A goal-based story-understanding system (see Story analysis), PAM was written in 1978 by Wilensky at Yale University (see R. Wilensky, PAM, in R. C. Schank and C. K. Riesbeck (eds.), *Inside Computer Understanding: Five Programs Plus Miniatures,* Lawrence Erlbaum, Hillsdale, NJ, pp. 136–179; 1981).

M. Taie
SUNY at Buffalo

PANDEMONIUM

Developed in 1959 by O. G. Selfridge, PANDEMONIUM was one of the earliest attempts at machine learning (qv). It exemplified the approach to learning at the time, anthropomorphic neural networks with partially random initial structure. Research on learning has moved away from this paradigm (see O. G. Selfridge, Pandemonium: A Paradigm of Learning, in D. Blake and A. Uttley (eds.), *Proceedings of the Symposium on Mechanization of Thought Processes,* Her Majesty's Stationery Office, London, 1959).

K. S. Arora
SUNY at Buffalo

PARRY

PARRY was one of the earliest attempts at belief modeling (see Belief systems). Designed by Colby around 1971 at Stanford University, PARRY simulated the conversational behavior of a paranoid person. The system integrates inferences with affects and intentions to produce behavior that has been classified as paranoid by several psychologists who conversed with the program (see K. Colby, *Artificial Paranoia,* Pergamon, New York, 1975).

K. S. Arora
SUNY at Buffalo

PARSING

Parsing a sentence of a natural language such as English is the process of determining if it is syntactically well formed (grammatical) and, if so, of finding one or more structures (structural descriptions) that encode useful information of some kind about it. The word is derived from the Latin *pars orationis* (part of speech) and reflects a process that has been carried out by human beings from medieval times to the present. This activity traditionally took the form of assigning a part of speech to every word in a given sentence, of determining the grammatical categories of words and phrases, and of enumerating the grammatical relations between words. Its purpose was pedagogical, to help students of a language increase their mastery of it.

In modern times developments in linguistics and computer science have led to a somewhat different set of activities being associated with the term *parsing*. The availability of computers was one of the factors that led to the replacement of vague, partially specified procedures that were carried out by humans by well-specified algorithms that were carried out by machines. Also, the change in purpose of the activity led to a corresponding change in the nature of the structural descriptions produced. Pedagogical concerns were replaced by a requirement that structural descriptions reflect the meaning(s) of the sentences. This is of special importance in AI applications, in which the intent of input sentences must be understood and acted upon in an appropriate manner.

Still another change stemmed from the use of formal systems to model aspects of natural languages. In particular, many different types of formal, generative grammars have been devised to specify the sentences of a language and to pair each sentence with a corresponding set of structural descriptions. The nature of these grammars, however, usually does not provide an obvious algorithm for computing structural descriptions from sentences. In this respect they are similar to systems of logic, which implicitly specify a set of provable theorems but do not explicitly tell how a particular theorem is to be proved. Just as proof procedures must be devised for systems of logic, so must parsing procedures be devised for formal grammars of natural languages. Parsing a given sentence with respect to a given grammar, then, is the process of determining whether the sentence belongs to the language specified by the grammar and, if so, finding all the structures that the grammar pairs with the sentence.

The most common type of grammar used within computer science to syntactically specify the sentences of a particular programming language and to assign structure to them is the BNF (Backus–Naur form) grammar. This is a notational variant of a class of grammars called *context-free* (CF) grammars, which play a prominent role in many computational and AI models of natural language. It is worth remarking, however, that there is a central difference between the use of CF grammars in computer science and in computational linguistics. In the former, subclasses of CF grammars are utilized that are both unambiguous (i.e., they assign at most one structural description to a sentence) and parsable in time linearly proportional to the length of the sentence parsed. In the latter, however, use is made of a parsing algorithm either for the general class of (ambiguous) CF grammars or for an even more general class of grammars, which often makes some use of CF grammars. For this reason we shall treat in some detail the parsing of CF grammars. Readers who are familiar with the BNF specification of programming languages may wish to skip the next section, which contains introductory material on phrase-structure grammars. The subsequent sections return to the central problem of parsing.

Phrase-Structure Grammars

The four components of a phrase-structure grammar are a set of symbols from a terminal vocabulary V_T (terminals), another disjoint set of symbols from a nonterminal vocabulary V_N (nonterminals), a distinguished elements of V_N called the start symbol, and a set of rules or productions P. By suitably replacing restrictions on the allowable productions, different types of phrase-structure grammars may be specified. The previously mentioned CF grammar is one such type. All of its rules are of the form $A \rightarrow A_1A_2 \cdots A_n$ where A belongs to V_N, and the A_i belong either to V_T or V_N. The CF right member of a production is the empty string, and such a production is called an erasing rule.

A *derivation* with respect to a CF grammar (V_N, V_T, S, P) is a sequence of strings, the first of which is the start symbol S, and each subsequent member (*sentential form*) is producible from its predecessor by replacing one nonterminal symbol A by a string of terminal and nonterminal symbols $A_1A_2 \cdots A_n$ where $A \rightarrow A_1A_2 \cdots A_n$ is a production of P. Sentential forms consisting entirely of terminal symbols can have no successors in a derivation, and the set of all such terminal sentential forms is said to constitute the language specified by the given grammar (V_N, V_T, S, P).

By way of illustration, consider the CF grammar G1 = (V_N, V_T, S, P) where V_N = (S, NP, VP, DET, N, V, PP, PREP),

$$V_T = (\text{I, the, a, man, park, telescope, saw, in, with})$$

(Terminals such as "telescope" and nonterminals such as PREP are to be regarded as atomic symbols), and P is the set of productions:

S → NP VP	N → man/park/telescope
VP → V NP/VP PP	DET → the/a
NP → I/NP PP/DET N	V → saw
PP → PREP NP	PREP → in/with

The BNF abbreviatory convention is used for writing

$$VP \rightarrow V\ NP/VP\ PP$$

to indicate the two productions VP → V NP and VP → VP PP. A sample derivation is the sequence of sentential forms:

S, NP VP, I VP, I VP PP, I V NP PP, I saw NP PP,
I saw DET N PP, I saw the N PP,
I saw the man PP, I saw the man PREP NP,
I saw the man in NP, I saw the man in DET N,
I saw the man in the N, I saw the man in the park.

The final sentential form, "I saw the man in the park," is one of the sentences in the language generated by G1.

G1 has been made simple to aid in illustrating certain parsing algorithms, but it is deficient in generating such sentences as "a park in I saw I." A much more complicated set of productions is required to produce reasonable coverage of a natural language without generating such unwanted strings of terminals.

Requiring that the structural descriptions reflect meaning makes the task of producing adequate grammars much more difficult. The use of derivations can be extended to provide for structural descriptions by replacing each production $A \rightarrow A_1A_2 \cdots A_n$ by a corresponding production $A \rightarrow (_A A_1A_2 \cdots A_n)$ where $(_A$ and) are two new terminal symbols.

The result of such systematic replacement of productions P and augmentation of V_T for G1 is another CF grammar, G2. For every derivation of G1 there is a derivation of G2 in which corresponding productions are invoked. The structural description of a sentence generated by G1 is the sentence generated by the corresponding derivation with respect to G2. For the example, that derivation is

$$\mathrm{S},\ (_{\mathrm{S}}\mathrm{NP\ VP}),\ (_{\mathrm{S}}(_{\mathrm{NP}}\mathrm{I})\ \mathrm{VP}),\ \ldots,$$
$$(_{\mathrm{S}}(_{\mathrm{NP}}\mathrm{I})\ (_{\mathrm{VP}}(_{\mathrm{VP}}(_{\mathrm{V}}\mathrm{saw})\ (_{\mathrm{NP}}(_{\mathrm{DET}}\mathrm{the})\ (_{\mathrm{N}}\mathrm{man})))$$
$$(_{\mathrm{PP}}(_{\mathrm{PREP}}\mathrm{in})\ (_{\mathrm{NP}}(_{\mathrm{DET}}\mathrm{the})\ (_{\mathrm{N}}\mathrm{park})))))$$

This last sentential form of the derivation with respect to G2 is the structural description of "I saw the man in the park." It is a labeled bracketing that is one notation for expressing the tree structure shown in Figure 1.

This representation is easier for humans to assimilate but takes more space, and henceforth the labeled bracketing format will be used to represent trees, further simplifying it to

(S (NP I) (VP (VP (V saw) (NP (DET the) (N man)))
(PP (PREP in) (NP (DET the) (N park)))))

Note that there is a second structural description of "I saw the man in the park":

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man))
(PP (PREP in) (NP (DET the) (N park))))))

which groups the words in such a way that the string "the man in the park" is a single constituent, an NP, leading naturally to the interpretation that the man was in the park when he was seen. The first structural description, however, does not

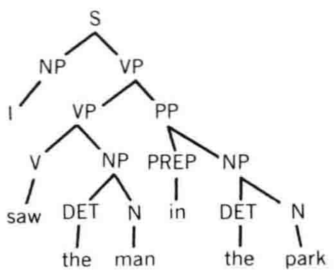


Figure 1.

group the string "the man in the park" as a single constituent. Instead, the VP "saw the man" is a sister to the PP "in the park," indicating the interpretation that the location of the seeing of the man was in the park.

Other types of phrase structures such as finite-state grammars, context-sensitive grammars, and unrestricted rewriting systems (1–3) are definable by placing suitable restrictions on the type of productions allowed. They play a lesser role in specifying natural languages and hence are not treated here.

A topic of some importance in parsing is that of equivalence between two grammars. Grammars that generate the same language are said to be weakly equivalent. The term *strong equivalence* has been used to imply at least the existence of a 1 : 1 correspondence between the structural descriptions of two grammars, and in some definitions of strong equivalence, the existence of a trivial homeomorphism from the structures of one grammar to those of another has been required. The importance of equivalent grammars to parsing is that there are several circumstances in which it is preferable not to parse a given grammar directly but rather to construct from it an equivalent grammar and to parse with respect to it instead. In certain cases the structural descriptions assigned by the equivalent grammar are just as useful as those assigned by the given grammar. In other cases structural descriptions with respect to the given grammar are needed, but it still may be more efficient to obtain an equivalent grammar, parse with respect to it, and convert the resulting structural descriptions to those of the given grammar than to parse directly with respect to that given grammar. There is another reason for using equivalent grammars in parsing. Sometimes a particular parsing algorithm is only valid for grammars with productions satisfying some restrictions, and it can be shown that for an arbitrary grammar, an equivalent grammar that satisfies those restrictions can be constructed. Often such restrictions are not essential for a particular parsing algorithm but greatly simplify its exposition and hence are useful for pedagogical purposes.

Three types of equivalent grammars for CF grammars much used in parsing are those with no erasing rules, Chomsky normal-form grammars, and Greibach normal-form grammars (1,2). Using capital letters to denote nonterminals and lowercase letters to denote terminals, Chomsky normal-form grammars are those with rules $A \rightarrow BC$ or $A \rightarrow a$, and Greibach normal-form grammars are those with rules $A \rightarrow aA_1A_2 \cdots A_n$ where $A_1A_2 \cdots A_n$ is a string of zero or more nonterminals.

Another topic to be considered in this section relates to the encoding of complex information in the nodes of structural description trees and to generalizing productions so as to require that the nodes they involve have prescribed information. This information usually takes the form of *features* and their *values*. Feature names are atomic entities such as ANIMATE, NUMBER, and GENDER and feature values are sometimes binary (+ or −), sometimes *n*-valued but atomic such as MASCULINE, FEMININE, and NEUTER, and sometimes complex such as a set of recursively used feature-value pairs. Often, the use of such features can enormously simplify the complexity of the set of productions required to specify a particular language. Also, it can be shown that such use of features can be made in ways that do not affect the class of languages definable. Thus, for example, it is possible to extend CF grammars with feature restrictions in such ways that the languages definable are precisely those definable by ordinary CF grammars. This use of features has played an important role in restoring to favor the use of CF languages in natural-language specification and parsing. They are discussed further in a subsequent section.

The final subject to be discussed in this section involves the equivalence between phrase-structure grammars and corresponding automata. All of the types of phrase-structure grammars mentioned have as counterparts corresponding types of automata. Finite-state automata correspond to finite-state grammars, pushdown automata correspond to CF grammars, linear bounded automata to CS grammars, and Turing machines to unrestricted rewriting systems. By correspondence is meant the existence of constructions from grammars to automata and vice versa that preserve the languages specified and the structural descriptions assigned to their sentences.

There are several ways in which this correspondence can be exploited; the one of most concern regards the use of automata to model particular parsing algorithms and to model ways of implementing those algorithms. The space available here allows only informally describing parsing algorithms rather than specifying them precisely by means of automata, but the reader is directed to other sources for instances of such usage of equivalent automata (2,4). In the next three sections, three of the most commonly used CF parsing algorithms are examined: recursive-descent, left-corner, and chart.

Recursive-Descent Parsing

Recursive-descent parsing, sometimes just called top-to-bottom CF parsing (1,3,4), systematically pieces together structural description trees from top to bottom and from left to right. At each stage of parsing the leftmost unexpanded nonterminal is identified, and its daughter nodes are attached using one of the productions that rewrite that nonterminal. If there is more than one such production, the parser tries them all, following a separate continuation path in each case. Such a process is called *nondeterministic*. Its implementation is often achieved by using a pushdown list to store continuations that are subsequently retrieved and followed.

Terminal symbols thus incorporated into a structural description are matched against the next symbols of the string being parsed. Failure to match causes the continuation in question to fail or block. A continuation also fails if there are remaining input string symbols after the last nonterminal has been expanded.

One successful parse path is given by:

S

(S NP VP)

(S (NP I) VP)

(S (NP I) (VP V NP))

(S (NP I) (VP (V saw) NP))

(S (NP I) (VP (V saw) (NP NP PP)))

(S (NP I) (VP (V saw) (NP (NP DET N) PP)))

(S (NP I) (VP (V saw) (NP (NP (DET the) N) PP)))

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man)) PP)))

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man))
 (PP PREP NP))))

(S (NP I)(VP (V saw)(NP (NP (DET the)(N man))
 (PP (PREP in) NP))))

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man))
 (PP (PREP in) (NP DET N)))))

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man))
 (PP (PREP in) (NP (DET the) N)))))

(S (NP I) (VP (V saw) (NP (NP (DET the) (N man))
 (PP (PREP in) (NP (DET the) (N park)))))

Another successful parse path leads to the structural description

(S (NP I) (VP (VP (V saw) (NP (DET the) (N man)))
 (PP (PREP in) (NP (DET the) (N park)))))

whose structure indicates the sequence of continuations involved in producing it.

Note that whenever the production NP → NP PP is used to expand a leftmost nonterminal, the resulting structure continues to have NP as its leftmost unexpanded nonterminal. Hence, expansion via this rule takes place indefinitely, and it is seen that the algorithm does not terminate. More generally, it is observed that nontermination of the recursive descent algorithm will occur whenever it is applied to a grammar that admits recursive left branching, i.e., a leftmost derivation from some nonterminal A to a string beginning with A.

There are several ways of ensuring termination of this algorithm. First, the grammar to be parsed can be required to disallow recursive left branching. This is not as serious a limitation as might at first be expected because it has been shown that there are constructions that map a given CF grammar into an equivalent CF grammar that is not left recursive. Constructions to Greibach normal form can be used for this purpose. One such construction is due to Rosenkrantz (5). A second way of ensuring termination of the algorithm is applicable to grammars that contain no erasing rules. For such grammars a continuation can be blocked whenever the number of nonterminals that remain to be expanded exceeds the number of still unmatched terminal symbols in the input string.

An obvious improvement that can be made to recursive-descent parsing involves the use of a left-branching reachability matrix R with elements R(A,B) whose arguments A and B range over the union of V_T and V_N. R(A,B) is T (true) or F (false) depending, if B belongs to V_N, on whether it is possible to left branch down from B to A, and, if B belongs to V_T, depending on whether A = B. That is, R(A,B) is T if the grammar in question has a derivation from B to a string beginning with A or if A = B. Note that R depends only on a given CF grammar, not on any particular string to be parsed with respect to it. Thus, R can be computed once and for all for a given grammar of interest. Warshall's algorithm (6) for computing R has been proved optimal and is therefore recommended.

Use of matrix R is illustrated by the following continuation:

S

(S NP VP)

(S (NP DET N) VP)

The leftmost unexpanded nonterminal DET cannot left branch down to the next input string terminal "I" because R(I,DET) = F. Hence, this continuation can be blocked at this point without having to consider further continuations that result from expanding DET.

Left-Corner Parsing

As the name implies, left-corner parsing (called SBT parsing in Ref. 4) builds sentence structures in a left-to-right, bottom-to-top fashion, piecing together the left corner of a structural description tree first. It is not the only parsing algorithm that builds structure from bottom to top. Shift-and-reduce parsing is one of the more commonly encountered cases in point (1,3,4). At each step in left-corner parsing, having determined a left-corner subtree of a structural description tree, it attempts to extend that subtree by scanning the productions for those whose right members begin with the root node of the left-corner subtree. Substituting that subtree for the first constituent of the right member of such a production gives a larger left-corner subtree; all of the daughter nodes of its root node except the first remain to be replaced by appropriate structure, this being accomplished in left-to-right order, recursively using this same left-corner parsing algorithm.

Once again, this algorithm is nondeterministic. There can be more than one production with a right member beginning with a given constituent, leading to one type of nondeterminism. Another source of nondeterminism arises whenever a subtree is successfully built up to replace a constituent other than the first one in the right member of some production. In addition to making the replacement, it is also necessary to attempt to build the subtree up to a larger subtree with the same root node.

As with recursive-descent parsing, the recursive left-branching matrix R(A,B) can be used to curtail continuations that must eventually fail. At each point where a left-corner subtree has been built up, one knows the nonterminal that one is next attempting to satisfy (i.e., to replace). If that subtree has root A, and B is the nonterminal to be satisfied, then one should block if R(A,B) = F without attempting further left-corner building of this subtree.

The first few steps in one successful path for the left-corner parsing of the sentence "I saw the man in the park" with respect to the grammar G1 are the following: The only production whose right member begins with the first word in the sentence to be parsed, "I," is NP → I. This gives the left-corner subtree (NP I), and one of the productions whose right member begins with the root of this subtree is S → NP VP. Letting the left-corner subtree satisfy the NP node of this production gives the new (partially determined) subtree (S (NP I) VP). One must still parse the remaining string "saw the man in the park" up to a tree with root VP to satisfy the VP node in (S (NP I) VP). Proceeding in the same way, the left-corner subtree (V saw) and then the partially determined subtree (VP (V saw) NP) are obtained. The remaining input string at this point is "the man in the park." An initial substring of it must be left-corner-parsed up to a subtree with root NP. Suppressing the details of this, it turns out to be (NP (DET the) (N man)), with "in the park" left as the remaining portion of the input string. This subtree is used to replace the NP node in the previous structure (VP (V saw) NP), giving the new left-corner subtree (VP (V saw) (NP (DET the) (N man))). Next one of the productions is chosen whose right member begins with VP, VP $\rightarrow$ VP PP, and it is combined with the previously determined left-corner subtree to obtain (VP (VP (V saw) (NP (DET the) (N man))) PP). Finally, the remaining input string "in the park" is left-corner-parsed up to a subtree with root node PP, and this subtree is used to replace the PP in the previous structure. The result is one of the required structural descriptions, (S (NP I) (VP (VP (V saw) (NP (DET the) (N man))) (PP (PREP in) (NP (DET the) (N park))))).

The Rosenkrantz equivalent grammar construction was previously mentioned as a means of eliminating left branching. It is also of value in relating recursive-descent parsing and left-corner parsing. Griffiths and Petrick (7) have shown that the left-corner parsing of a given CF grammar is mimicked by the recursive-descent parsing of the corresponding Rosenkrantz equivalent grammar. This has been exploited in parsing efforts making use of the PROLOG programming language. The productions of a CF grammar can be directly transcribed into a PROLOG form that permits parsing without programming any parsing algorithm. PROLOG itself provides a top-down, depth-first search (qv) procedure, which has the effect of performing recursive-descent parsing (see Recursion). Although PROLOG does permit the easy implementation of a CF parser, its value is limited by the limitations of recursive-descent parsing, namely that it does not allow recursive left branching, and it is slower than such alternatives as left-corner parsing and chart parsing for most grammars of practical interest. To avoid both of these problems, the Rosenkrantz equivalent grammar construction (programmed in PROLOG) has been used to obtain a grammar that can be parsed via the PROLOG recursive-descent procedure, thus mimicking left-corner parsing of the original grammar.

Griffiths and Petrick also used the Rosenkrantz construction to prove that the time required for left-corner parsing is, at worst, a constant (depending on the grammar) multiple of the time required to parse the sentence with respect to the same grammar by recursive descent (8).

Chart Parsing

All of the parsing algorithms described to this point have worst-case exponential upper bounds. That is, there exist grammars and sentences whose parsing requires a number of steps proportional to a constant raised to the power of the number of words in the input string of words. The reason for this is that when two or more nondeterministic continuations arise, each of them can lead to the identical determination of some substructure common to them all. To avoid this, it is possible to store information as to which subtrees have been found to span substrings of the input string. This information can then be looked up to avoid computing it more than once.

There are a number of different chart-parsing algorithms. One variant, separately discovered by Cocke, Kasami, and Younger (9), is now usually referred to as CKY parsing. (See Parsing, Chart). Like most of the other chart-parsing algorithms, it has a worst-case upper bound proportional to the cube of the length of the input string. Other chart-parsing algorithms of note are due to Kay (10), Earley (11), Kaplan (12), and Ruzzo (13). The latter is both especially simple and efficient, having a worst-case upper bound proportional to the cube of the input string length with an attractively small constant of proportionality.

Ruzzo's parsing algorithm makes use of a chart or matrix whose elements $t_{i,j}$ $(0 \leq i \leq j \leq n)$ are determined during the course of parsing a string of length n. Each element $t_{i,j}$ consists of a set of items each of which is a production of the given grammar with a single dot located somewhere among the constituents of the right member. For example, PP $\rightarrow$ PREP DOT NP is a typical item. The positions between the terminals of the input string are numbered as in (0 I 1 saw 2 the 3 man 4 in 5 the 6 park 7). If element $t_{i,j}$ contains the item $A \rightarrow A_0 \cdots A_k$ DOT $A_{k+1} \cdots A_m$, this indicates that the input string between points i and j has been parsed up to a string of trees whose roots are $A_0\ A_1 \cdots A_k$, and if some string beginning at point j can be parsed up to a string of trees with roots $A_{k+1} \cdots A_m$, the concatenation of both those strings of trees can be parsed up to the parent node A to obtain a tree with root A.

For grammar G1 the items of $t_{0,0}$ are (S $\rightarrow$ DOT NP VP, NP $\rightarrow$ DOT I, NP $\rightarrow$ DOT NP PP, NP $\rightarrow$ DOT DET N, DET $\rightarrow$ DOT the, and DET $\rightarrow$ DOT a. They are obtained by taking the productions that begin with a constituent A such that R(A,S) = T, where R is the left-branching reachability matrix, and forming items in which the dot is located in front of the first constituent. These items indicate the initial possibilities. Three types of actions fill in the elements of the matrix $t_{i,j}$. Elements $t_{j,j}$ are filled in by generating items of the type we have seen for $t_{0,0}$ in much the same manner that has already been illustrated. Some of the items of elements $t_{i,j}$ are formed from those of $t_{i,j-1}$ by hopping the dot one position to the right if the constituent hopped over is either the input string terminal located between points $j - 1$ and j or else a nonterminal from which there is a derivation to that input terminal. The remaining items are filled in by considering, in the proper sequence, pairs of items, the first of which is of the form $A \rightarrow A_0 \cdots A_p$ DOT $A_{p+1} \cdots A_q$ and the second of which is the form $A_{p+1} \rightarrow \cdots$ DOT. If such a pair comes from corresponding elements $t_{i,j}$ and $t_{j,k}$, this indicates that the substring of terminals from points i to j has been parsed up to the string $A_0 \cdots A_p$ with a possible continuation of A_{p+1}, and the substring from points j to k has been parsed up to A_{p+1}, realizing that possibility. Hence, the dot is hopped over the constituent to its right in the first item, and the resulting item is included among the items of $t_{i,k}$.

Acceptance is indicated by the presence of an item of the form S $\rightarrow \cdots$ DOT in $t_{0,n}$. Structural descriptions are easily obtained by modifying the form of items described above to indicate the tree structure of the constituents to the left of the dot. Whenever the dot is hopped over a constituent, that constituent is replaced by any structure it dominates. The necessary information about this structure comes either from the other item involved or from information supplied by the grammar about a derivation of the next terminal symbol from the hopped-over nonterminal.

Complex Feature-Based Grammars

There are a number of types of grammars of current interest for specifying natural languages that make some central use of complex feature-augmented rules and structures. To various degrees, they also incorporate other formal devices in addition to their central phrase-structure components. They share a common requirement for matching, in a certain way, rules containing complex features with structural description trees or tree fragments containing complex features, and hence they are sometimes referred to as unification-based grammars. Examples of such grammars are generalized phrase-structure grammars (see Grammar, generalized-phrase-structure) (14), definite clause grammars (see Grammar, definite-clause) (15), functional unification grammars (16), head grammars and head-driven grammars (17), lexical functional grammars (18), and PATR-II-type grammars (19).

It is beyond the scope of this entry to describe these formalisms sufficiently to describe their parsers. Note that they all incorporate the use of a suitably-modified phrase-structure grammar parser (see Grammar, phrase-structure), usually one of the more common types of parsers. See the sources cited for detailed information about these grammars and their parsers.

Two other types of grammars might also be included among those of this section because their rules and structures also make essential use of complex features. These two, transformational grammars and augmented transition network grammars, however, are treated separately in the subsequent sections.

Transformational Grammar Parsing

It should first be noted that there is no single theory that is agreed upon by all who use the term *transformational grammar* (TG) to label the syntactic theory they advocate (see Grammar, transformational). This is so because TG has evolved, splitting on occasion into distinct formal models of language with significant differences in the type of rules that are allowed, on the constraints imposed on rule application, and on the type of structural descriptions and intermediate structures that are advocated.

It is beyond the scope of this entry to discuss these diverse types of grammars and their parsers. Some of them are discussed in Transformational grammar. A few general remarks are given below.

TG makes central use of a base phrase-structure-grammar component (usually a CFG) to specify a set of base trees (deep structures) and a transformational component to map those trees into a set of surface-structure trees. A simple operation (usually just extracting the terminals) on a surface-structure tree yields one of the sentences in the language thus specified. The meaning of a sentence is encoded in either the base structures, the surface structures, or some combination of both, depending on the type of TG in question.

The transformational component consists of a set of one or more transformations, usually ordered in their application in a rather complex way. A transformation maps each of a class of trees that satisfies certain conditions into a corresponding tree.

Note that the normal direction in which transformations are formulated to operate is from deep structure to surface structure. In transformational parsing, however, one is required to find the corresponding deep and surface structures in a sentence. For some applications and some variants of TG it is sufficient to find only the surface structure or only the deep structure, but it is in general only possible to know one of them has been correctly determined if the other has also been determined and the transformational mapping between them has been verified.

One way to determine the deep and surface structures assigned by some TG to a given sentence is to limit the possible phrase-structure and transformational rules that might be applicable to the derivation of that sentence and then to try all combinations of these rules in the forward direction to see which paths terminate with the given sentence. This is called analysis by synthesis. It was suggested by Matthews (20) but was never implemented. It appears to be prohibitively inefficient.

Another way to determine deep and surface structure is to reverse the forward generative procedure, going from a sentence to its surface structures. Unfortunately, the machinery for forward generation is not directly convertible to a corresponding system for going in the other direction. The first step, determination of the surface structure corresponding to a given sentence, is complicated by the fact that it includes structure reflecting base phrase-structure productions as well as other structure of transformational origin. Petrick (21) has shown that it is possible to compute from a given TG meeting certain requirements a new CF grammar whose productions are a superset of the TG's base component productions and that generates a set of structural description trees that include all of those surface structures assigned to sentences of length not exceeding n by the TG. This augmented base-component CF grammar can be used to determine all of the surface structures a TG assigns to any sentence of length n or less, together with some possible spurious ones (given a particular sentence of some length, the augmented CF grammar valid for sentences up to that length can be found if one has not already been determined).

Petrick also presents several ways of inverting the effect of transformations. True inverse transformations are, in general, not computable, but pseudoinverse transforms can be mechanically computed from a given set of forward transforms, and they can be applied, together with some additional CF parsing, to obtain a set of structures that includes all of the deep structures assigned by the TG. These must be checked to ensure that they contain only base-component phrase structure and to ensure that they may be mapped into the previously determined surface structure using the transformational component of the TG in question.

This parsing algorithm is dependent on certain restrictions being placed on the class of TGs to which it is applied. Without such restrictions, classes of grammars such as those of Chomsky's aspects model have been proved to be equivalent to Turing machines (qv) (22), and hence it is known that no parser valid for the entire class is possible.

A final point to note with respect to transformational parsing is that most parsers labeled transformational are not constructed from a given TG in such a way as to guarantee their correctness. Neither are they usually constructed by hand and then proved to be valid parsers of normally formulated TGs.