PROCESSING, BOTTOM-UP AND TOP-DOWN

"Bottom-up" vs. "top-down," "forward" vs. "backward," and "data-driven" vs. "goal-directed" are three pairs of modifiers for terms such as "chaining," "inference," "parsing," "processing," "reasoning," and "search." They express essentially the same distinction, their difference lying in different metaphors drawn from different subareas of computer science and AI. The bottom-up–top-down distinction comes from parsing (qv); the forward–backward chaining distinction comes from rule-based systems (qv); goal-directed comes from problem solving (qv) and search (qv), and data-directed comes from discussions of control structures (qv). Now, however, they are virtually interchangeable.

One general way to consider the distinction is from within the paradigm of search. The basic issue of all search is to find a way to get from where you are to where you want to be. If you organize this by starting from where you are and search until you find yourself where you want to be, you are doing forward, data-directed, or bottom-up search. If you think about where you want to be, and plan how to get there by working backward to where you are now, you are doing backward, goal-directed, or top-down search. Notice that, having found the route during backward search, you still have to get to your goal. Although you are now moving in the forward direction, this is not forward search because all the search was already done in the backward direction.

Another general way to consider the distinction is from within the paradigm of rule-based systems (qv). A generic rule can be thought of as having a set of antecedents and a set of consequents. When the rule-based system notices that all the antecedents of a rule are satisfied, the rule is triggered and may fire (whether all triggered rules actually fire depends on the specifics of the rule-based system). When the rule fires, the consequent propositions are added to the knowledge base and the consequent actions are performed. These steps of triggering and firing happen as just described regardless of whether the rule-based system is using forward (or data-directed or bottom-up) reasoning or backward (or goal-directed or top-down) reasoning. To make the distinction, it is useful to isolate the step of rule activation. Only activated rules are subject to being triggered. In forward (or data-directed, or bottom-up) reasoning, whenever new data is added to the system, the data is matched against all antecedents of all rules (actual systems are more efficient than this sounds). If the data matches an antecedent of a rule, that rule is activated (if it is not already activated), and if all antecedents of the rule are now satisfied, the rule triggers. When a rule fires, the consequent propositions that are added to the knowledge base are treated like new data, matched against antecedents, and may cause additional rules to be activated and to be triggered. In backward (or goal-directed, or top-down) reasoning, rules are not activated when data is added. Rather, when a query is asked of the system, or the system is asked to do something, the query (or goal) is matched against all consequents of all rules (again, actual systems are more efficient than this sounds). If the query matches a consequent of a rule, the rule is activated, all its antecedents are treated as new queries or goals (now called subqueries, or subgoals), and may activate additional rules. Whenever a query or subquery matches an unconditional proposition in the knowledge base, it is answered, and if it came from an antecedent, the antecedent is now known to be satisfied. As soon as all antecedents of some rule are known to be satisfied, the rule triggers and may fire. When a rule fires, the queries that activated it are answered, and now other antecedents may be known to be satisfied, and their rules might be triggered. Notice that the triggering and firing of a rule always seems to happen in a "forward" direction, due to the significance of "antecedents" vs. "consequents," but what distinguishes forward from backward chaining is when the rule is activated.

Some of the history of these terms is discussed below, followed by explanations of the distinctions via examples of parsing, rule-based systems, and search. Then there will be some comparative comments and, finally, some discussion of mixed strategies.

History of Terms

Bottom-Up versus Top-Down. The earliest published occurrence of the phrase "top-down vs. bottom-up" seems to have been in a 1964 paper by Cheatham and Sattley (1), although the term "top-down," at least, seems to already have been in use:

The Analyzer described in this paper is of the sort known as "top down," the appellation referring to the order in which the Analyzer sets its goals. . . . The order in which a "bottom up" Analyzer sets its goals is much more difficult to describe. . . . (2)

It took a while, however, for these terms to become fully accepted. In 1965 Griffiths and Petrick (3) used the terms "bottom-to-top" and "top-to-bottom":

There are many ways by which the typology of recognition algorithms for [Context Free] grammars can be approached. For example, one means of classification is related to the general directions in which creation of a structural description tree proceeds: top-to-bottom, bottom-to-top, left-to-right and right-to-left.

However, in 1968, they use the terms "bottom-up" and "top-down" in a title (4).

Also in 1968, Knuth (5) used the term "bottom-up" to describe a function "which should be evaluated at the sons of a node before it is evaluated at the node" and "top-down" to describe a function *f* as "one in which the value of *f* at node *x* depends only on *x* and the value of *f* at the *father* of *x*." Knuth encloses these terms in quotes.

By 1970 Early (6) used the phrase "the familiar top-down algorithm" without attribution in the abstract of an article, and by 1973 "The Top-down Parse" and "The Bottom-up Parse" appear as section titles in a text (7).

Bottom-up and top-down parsing are referred to as the "morsel" and the "target" strategies, respectively, in Ref. 8, p. 87, where the Predictive Analyzer of Kuno and Oettinger (9) is cited as an early example of the target strategy, and "the algorithm due to John Cocke and used in Robinson's PARSE program (10) was the earliest published example of a program using a morsel strategy." Calingaert (11) cites Lucas (12) as the first described use of recursive descent parsing, which is a form of top-down parsing.

Forward versus Backward Chaining. The terms "forward chaining" and "backward chaining" almost surely come from Newell, Shaw, and Simon's Logic Theory Machine (LT) paper, first published in 1957 (13). They discuss four methods used by LT to help find a proof of a formula in propositional calculus (qv). The last two methods discussed are called "the chaining methods":

These methods use the transitivity of the relation of implication to create a new subproblem which, if solved, will provide a proof for the problem expression. Thus, if the problem expression is "a implies c," the method of forward chaining searches for an axiom or theorem of the form "a implies b." If one is found, "b implies c" is set up as a new subproblem. Chaining backward works analogously: it seeks a theorem of the form "b implies c," and if one is found, "a implies b" is set up as a new subproblem (14).

This is not exactly the characterization of forward and backward chaining given above, but the essential idea is there. In both methods if a certain theorem is found, an appropriate subproblem is set up. In forward chaining the theorem is found by matching its antecedent, and its consequent is involved in the new subproblem, whereas in backward chaining the theorem is found by matching its consequent, and its antecedent is involved in the new subproblem. Finding a theorem is analogous to activating a rule in the characterization described at the beginning of this entry.

The rule-based system version of forward and backward chaining grew out of the LT version via the production system architecture of problem-solving systems promulgated in Ref. 15.

Goal-Directed Processing. The notion of goal-directed behavior surely comes from psychology. In a 1958 psychology text the following description of problem solving occurs:

We may have a choice between starting with where we wish to end, or starting with where we are at the moment. In the first instance we start by analyzing the goal. We ask, "Suppose we did achieve the goal, how would things be different—what subproblems would we have solved, etc.?" This in turn would determine the sequence of problems, and we would work back to the beginning. In the second instance we start by analyzing the present situation, *see the implications of the given conditions and lay-out, and attack the various subproblems in a "forward direction"* (16).

Also, goals and subgoals are discussed in the section on motivation:

The person perceives in his surroundings goals *capable of removing his needs and fulfilling his desires. . . . And there is the important phenomenon of emergence of* subgoals. *The pathways to goals are often perceived as organized into a number of subparts, each of which constitutes an intermediate subgoal to be attained on the way to the ultimate goal. (17)*

Cheatham and Sattley, who are mentioned above as publishing probably the first use of "bottom up vs. top down," also compared top-down parsing to goal-directed behavior:

In our opinion, the fundamental idea—perhaps "germinal" would be a better word—which makes syntax-directed analysis by computer possible is that of goals*: a Syntactic Type is construed as a goal for the Analyzer to achieve, and the Definiens of a Defined Type is construed as a recipe for achieving the goal of the Type it defines. . . . Needless to say, this use of the term "goal" is not to be confused with the "goal-seeking behavior" of "artificial intelligence" programs or "self-organizing systems"* (18).

Needless to say, the several uses of "goal" indeed have much in common.

Data-Driven Processing. The term "data-driven" seems to have been introduced by Bobrow and Norman in a paper on the processing of memory schemata:

Consider the human information processing system. Sensory data arrive through the sense organs to be processed. Low-level computational structures perform the first stages of analysis and then the results are passed to other processing structures. . . . The processing system can be driven either conceptually or by events. *Conceptually driven processing tends to be top-down, driven by motives and goals, and fitting input to expectations; event driven processing tends to be bottom-up, finding structures in which to embed the input* (19).

They go on to use "conceptually driven" and "top-down" as synonymous and "event-driven," and "data-driven" interchangeably and synonymously with "bottom-up."

Examples

Parsing. The simple grammar of Figure 1 and the sentence "They are flying planes" will be used to illustrate and explain the difference between top-down and bottom-up parsing. The grammar is presented as a set of rules, numbered for the purposes of this discussion. The direction of the arrows is traditionally shown as if the grammar were being used for generation of sentences. For parsing, the rules are backward—the antecedents on the right side and the consequent on the left side, as in PROLOG (see Logic programming). For example, rule 1 can be read "If a string consists of a noun phrase (NP) followed by a verb phrase (VP), then the string is a sentence (S)."

Top-down parsing begins with S, the initial symbol, which will be the root of the parse tree. This is equivalent to establishing the goal of finding that the string of words is a sentence. Rule 1 says that every sentence will consist of a noun phrase (NP) followed by a verb phrase (VP). Whenever there is a choice, the lowest numbered rule is tried first, and the rule is expanded left to right. Therefore, the next subgoal generated is that of finding an initial string of the sentence that is a NP. Rule 2 is activated, followed by rule 8. This situation is shown in Figure 2*a*. Since "planes" does not match "they," the algorithm backs up, and in place of rule 2, rule 3 is activated followed by rule 9, which succeeds. Next, the algorithm returns to rule 1 and generates the subgoal of finding a VP. Rules 5, 6, and 12 are activated, and rule 12 succeeds. This stage is shown in Figure 2*b*. The rest of the top-down parse is shown in stages in the rest of Figure 2.

Bottom-up parsing begins with the words in the sentence.

1. S → NP VP
2. NP → N
3. NP → PRO
4. NP → ADJ N
5. VP → VT NP
6. VT → V
7. VT → AUX V
8. N → planes
9. PRO → they
10. ADJ → flying
11. AUX → are
12. V → are
13. V → flying

Figure 1. An example grammar.

Again, additional ordering decisions must be made, so the leftmost possibility is tried first, as is the lowest numbered rule when there is a choice. So, the first thing that happens is that the first word of the sentence, "they," matches the antecedent of rule 9, which fires, analyzing "they" as a pronoun (PRO). Then rule 3 fires, analyzing "they" as a NP. NP matches antecedents in rules 1 and 5, but neither of these is triggered yet, and the parse moves on to "are." This causes rule 11 to fire. (Although rule 12 is also triggered, it does not fire due to the ordering rules.) Then rule 10 fires, followed by rules 8 and 2. This stage is shown in Figure 3*a*. At this point, nothing more can be done with the string NP+AUX+ADJ+NP, so there is backtracking (qv) to the most recently triggered but unfired rule, which is rule 4. Nothing can be done with the string NP+AUX+NP, so again the most recently triggered unfired rule fires, which now is rule 13, which analyzes "flying" as a V. Then rule 6 fires, followed by rule 5. This is shown in Figure 3*c*. Nothing can be done with the string NP+AUX+VP, so rule 7 fires, analyzing "are flying" as a single VT. Then rule 5 fires again, followed by rule 1, and the parse, shown in Figure 3*d*, is complete.

The purpose of this example was to compare top-down with bottom-up parsing. The use of left-to-right order, numerical order of the rules, and chronological backtracking was merely to keep the two algorithms as similar as possible although causing them to result in different parses.

Rule-Based Systems. Forward vs. backward chaining and data-directed vs. goal-directed processing will be compared using some made-up rules for how to spend the evening. These are shown in Figure 4 in the usual rule order, with the antecedents to the left of the arrow and the consequent to the right. For example, rule 1 says that if there is a good movie on TV and I have no early appointment the next morning, then I enter "Late-Movie-Mode."

For the examples of forward and backward chaining, it is assumed that all triggered rules fire and that processing is in parallel.

Suppose a forward-chaining system with these rules is first told that I have no early appointment. Rules 1 and 2 are activated. Suppose the system is then told that I need to work. Rule 3 is activated, and rule 2 is triggered and fired, concluding that I am in Late-Work-Mode. This activates, triggers, and fires rules 5 and 6, concluding that I should return to the office and stay up late.

To do the same problem using backward chaining, suppose that the system was first told that I had no early appointment and had work to do and then was asked whether I should return to the office. This query would activate rules 6 and 7, which would generate the subgoals Late-Work-Mode? and Work-at-Office-Mode? These would activate rules 2 and 3, generating the subgoals No-Early-Appointment?, Need-to-Work?, and Need-References? The first two would be satisfied, resulting in rule 2 triggering and firing. This would cause Late-Work-Mode to be satisfied, triggering and firing rule 6, and concluding that I should return to the office.

Notice that in forward inference more conclusions are generated, whereas in backward inference more subgoals are generated. Since the data were the same for the two examples, the same rules were fired, but different rules were activated.

Search. Forward and backward search can be illustrated with a water-jug problem. For this problem there are two jugs, one capable of holding 3 gallons and one capable of holding 4

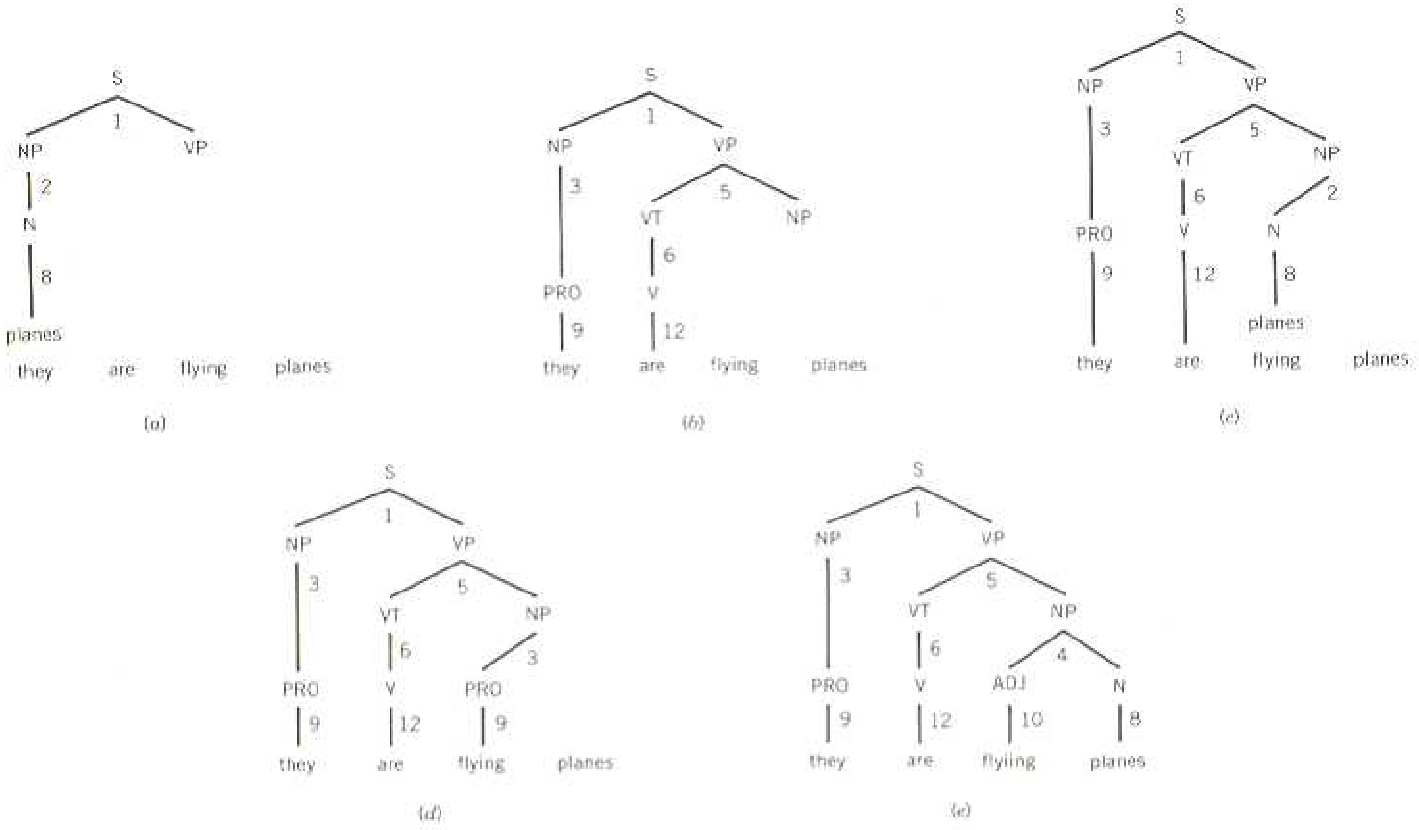

Figure 2. Top-down parsing.

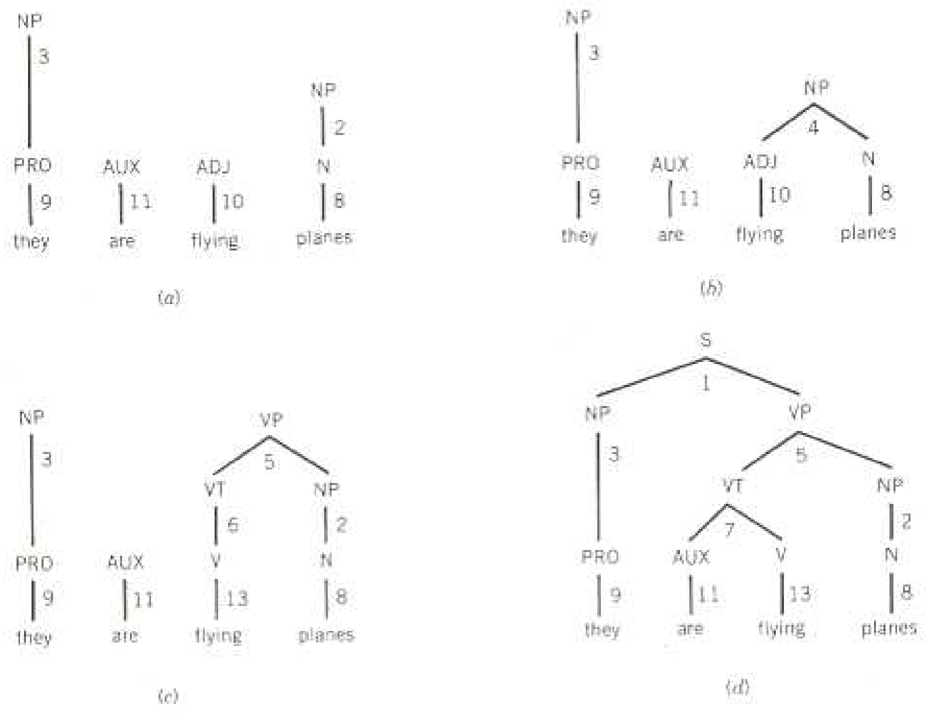

Figure 3. Bottom-up parsing.

1. Good-Movie-on-TV & No-Early-Appointment → Late-Movie-Mode
2. No-Early-Appointment & Need-to-Work → Late-Work-Mode
3. Need-to-Work & Need-References → Work-at-Office-Mode
4. Late-Movie-Mode → Stay-Up-Late
5. Late-Work-Mode → Stay-Up-Late
6. Late-Work-Mode → Return-to-Office
7. Work-at-Office-Mode → Return-to-Office

Figure 4. Rules for the evening.

gallons. The legal operations are filling the 3-gallon jug from a water tap (symbolized as f3); filling the 4-gallon jug from the water tap (f4); emptying the 3-gallon jug by pouring out all its contents (e3); emptying the 4-gallon jug by pouring out all its contents (e4); pouring the contents of the 3-gallon jug into the 4-gallon jug until either the 3-gallon jug is empty or the 4-gallon jug is full, whichever happens first (p34); or pouring the contents of the 4-gallon jug into the 3-gallon jug until either the 4-gallon jug is empty or the 3-gallon jug is full, whichever happens first (p43). Each state of the problem is represented by a pair of integers showing the contents of the 3-gallon jug and then the contents of the 4-gallon jug. For example, ⟨1, 4⟩ represents the state in which there is 1 gallon in the 3-gallon jug and 4 gallons in the 4-gallon jug. If operator p43 is applied to this state, the resulting state is ⟨3, 2⟩. The particular problem under consideration is that of getting from state ⟨0, 0⟩ to state ⟨0, 2⟩.

Figure 5 shows the state-space representation of this problem, assuming a parallel breadth-first search that stops as soon as the goal is found. No operator is shown that would move from a state to a state at the same or an earlier level of the search tree. For example, from state ⟨3, 1⟩ operator f4 would go to state ⟨3, 4⟩, but that is on the same level as ⟨3, 1⟩, and operator e4 would go to state ⟨3, 0⟩, but that is on an earlier level. What level a state is on, of course, depends on where the search started.

Figure 5*a* shows a forward search from ⟨0, 0⟩ until ⟨0, 2⟩ is found. Figure 5*b* shows a backward search from ⟨0, 2⟩ to ⟨0, 0⟩. Notice that, in this example, the same states are explored, but in a slightly different order. Notice also the difference between searching backward to find a way of getting from ⟨0, 0⟩ to ⟨0, 2⟩ and searching forward to find a way of getting from ⟨0, 2⟩ to ⟨0, 0⟩. In the latter case, one operator, namely e4, would suffice.

Comparisons

Efficiency. Whether bottom-up (or forward, or data-driven) processing is more efficient than top-down (or backward, or goal-directed) processing depends on the way the search space branches. If the average state has more successors than predecessors, backward search will be more efficient. If it has more predecessors than successors, forward search will be more efficient. To consider an extreme, if the search space forms a tree

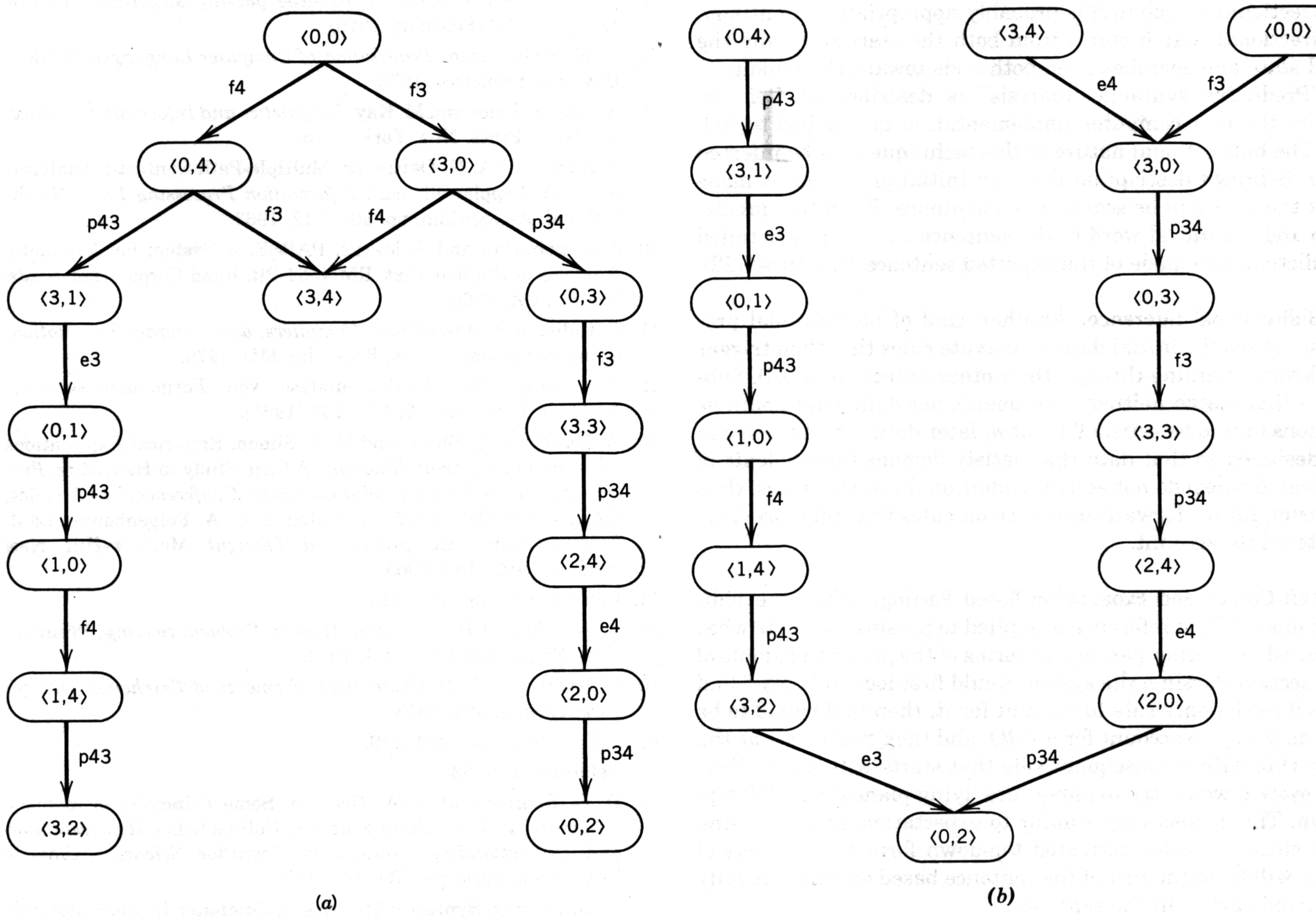

Figure 5. (*a*) Forward search. (*b*) Backward search.

rooted in the start state, a forward search will have to search a large part of the tree, whereas a backward search will only have to search up a linear branch.

Pattern Matching and Unification. In rule-based systems or reasoning systems, the choice of forward vs. backward chaining affects the difficulty of the required pattern-matching (qv) routines. In forward chaining, or data-driven reasoning, one is always asserting new facts to the system, and these have no free variables. Similarly, when rules fire, the newly inferred facts have no free variables. Therefore, one is always matching antecedents that may have variables against facts with no variables. Pattern matching two symbol structures when only one might have variables is a fairly simple routine.

On the other hand, in backward-chaining systems one often asks "wh" questions, such as "What shall I do this evening?" or "What organism is infecting this patient?" If the rules are represented in predicate logic (qv) rather than in propositional logic (qv), this involves matching a question with a variable against consequents with variables. Subgoals may also have variables, so in general, back-chaining systems must be written to match two symbol structures, both of which may have variables, and this requires the unification algorithm, which is considerably more involved than simple pattern matching.

Mixed Strategies

Bidirectional Search. If it is not clear whether forward or backward search would be better for a particular application, bidirectional search (qv) is probably appropriate. Essentially, bidirectional search starts from both the start state and the goal state and searches from both ends toward the middle.

"Predictive syntactic analysis" is described in Ref. 20, where the first computer implementation is ascribed to Ref. 21. The bidirectional nature of this technique may be inferred from Bobrow's description that "an initial prediction is made that the string to be scanned is a sentence. From this prediction and the initial word in the sentence . . . . more detailed predictions are made of the expected sentence structure" (22).

Bidirectional Inference. Another kind of bidirectional processing uses the initial data to activate rules that then trigger backward chaining through their other antecedents (23). Subgoals that match neither consequents nor data can remain as demons (qv) to be satisfied by new, later data. The system can be designed so that data that satisfy demons (antecedents of activated rules) do not activate additional inactive rules, thus focusing future forward inference on rules that take previous context into account.

Left-Corner and Expectation-Based Parsing. When the bidirectional style of inference is applied to parsing, one gets what is called left-corner parsing. In terms of the parsing example of the section Parsing, the system would first look at "they," find rule 9 as the only rule to account for it, then find rule 3 to be the only way to account for a PRO, and then find rule 1 as the only rule with a consequent side that starts with a NP. Next the system would try to parse "are flying planes" as a VP top-down. This is also very similar to expectation-driven parsing (qv) since the rules activated top-down form expectations of what will be in the rest of the sentence based on what actually occurred earlier in the sentence.

Conclusions

The control structure of an AI system that does reasoning, parsing, problem solving, or search is often organized into one of two basic approaches. One approach is called bottom-up, forward, or data-driven. The other is called top-down, backward, or goal-directed. The distinction is most easily understood as whether search is from goal to start or if rules are activated by their consequents or their antecedents.

Issues of efficiency or ease of implementation may decide which approach to take in a particular application, but mixed strategies are also possible.

BIBLIOGRAPHY

1. T. E. Cheatham and K. Sattley, Syntax-Directed Compiling, *Proceedings of the Spring Joint Computer Conference Washington, DC,* Spartan Books, Baltimore, MD, pp. 31–57, 1964.
2. Reference 1, p. 55.
3. T. V. Griffiths and S. R. Petrick, "On the relative efficiencies of context-free grammar recognizers," *CACM* 8(5), 289–300 (May 1965).
4. T. V. Griffiths and S. R. Petrick, Top-Down Versus Bottom-Up Analysis, in A. J. H. Morrell (ed.), *Information Processing 68: Proceedings of IFIP Congress 1968,* North-Holland, Amsterdam, pp. 437–442, 1969.
5. D. E. Knuth, *The Art of Computer Programming,* Vol. 1, *Fundamental Algorithms.* Addison-Wesley, Reading, MA, p. 362, 1968.
6. J. Early, "An efficient context-free parsing algorithm," *CACM* 13(2), 94–102 (February 1970).
7. F. W. Weingarten, *Translation of Computer Languages,* Holden-Day, San Francisco, 1973.
8. K. Sparck Jones and M. Kay, *Linguistics and Information Science,* Academic Press, New York, 1973.
9. S. Kuno and A. G. Oettinger, Multiple-Path Syntactic Analyzer, in C. M. Popplewell (ed.), *Information Processing-1962,* North-Holland, Amsterdam, pp. 306–312, 1963.
10. J. J. Robinson and S. Marks, PARSE: A System for Automatic Analysis of English Text, RM-4564-PR, Rand Corporation, Santa Monica, CA, 1965.
11. P. Calingaert, *Assemblers, Compilers, and Program Translation,* Computer Science Press, Rockville, MD, 1979.
12. P. Lucas, "Die Strukturanalyse von Formeluebersetzern," *Elektron. Rechenanl.* 3, 159–167 (1961).
13. A. Newell, J. C. Shaw, and H. A. Simon, Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics, *Proceedings of the Western Joint Computer Conference,* Los Angeles, CA, pp. 218–239, 1957. Reprinted in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought,* McGraw-Hill, New York, pp. 109–133, 1963.
14. Reference 13, pp. 117–118.
15. A. Newell and H. A. Simon, *Human Problem Solving,* Prentice-Hall, Englewood Cliffs, NJ, 1972.
16. D. Krech and R. S. Crutchfield, *Elements of Psychology,* Knopf, New York, p. 383, 1958.
17. Reference 16, pp. 218–219.
18. Reference 1, p. 33.
19. D. G. Bobrow and D. A. Norman, Some Principles of Memory Schemata, in D. G. Bobrow and A. Collins (eds.), *Representation and Understanding: Studies in Cognitive Science,* Academic Press, New York, pp. 138–140, 1975.
20. D. G. Bobrow, Syntactic Theories in Computer Implementations,

in H. Borko (ed.), *Automated Language Processing*, Wiley, New York, pp. 215–251, 1967.

21. I. Rhodes, "A new approach to the mechanical syntactic analysis of Russian," *Mechan. Transl.* 6, 33–50 (1961).

22. Reference 20, p. 235.

23. S. C. Shapiro, J. Martins, and D. McKay, Bidirectional Inference, *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, MI, pp. 90–93, 1982.

S. C. SHAPIRO
SUNY at Buffalo