DOCUMENT RESUME

ED 199 061 SE 034 283

AUTHOR	Riley, Mary S.; Greeno, James G.
TITLE	Details of Programming a Model of Children's Counting in ACTP.
INSTITUTION	Pittsburgh Univ., Pa. Learning Research and Development Center.
SPONS AGENCY	National Inst. of Education (DHEW), Washington, D.C.
REPORT NO	LRDC-1980/6
PUB DATE	80
NOTE	123p.; Contains occasional small print in Figures.
EDRS PRICE	MF01/PC05 Plus Postage.
DESCRIPTORS	Artificial Intelligence; Cognitive Processes; *Computer Programs; Computer Science; *Educational Research; Learning Theories; *Mathematics Education; *Programing Languages; *Research Tools
IDENTIFIERS	*Computer Models; Computer Simulation; Mathematics Education Research

ABSTRACT

Presented is an introduction to the operation and mechanics of the ACTP production system, a version of Anderson's (1976) ACT system. ACTP is already in use modeling geometry theorem proving and counting of a set of objects, and has been identified as a potentially useful programing framework for developing models of the cognitive processes used in other tasks. The ACTP system is introduced in the context of COUNTER, a model of counting. Section one of this report presents a general overview of the model, including COUNTER's performance on a sample problem, to provide a general idea of how a production system operates. The second section discusses the mechanics of the model, including data structures, schemata, and single productions. The final section follows the sequence of testing and executing productions involved in counting a set of objects. A list of selected references and five appendices are included in this report. (MP)

ED199061

LEARNING RESEARCH AND DEVELOPMENT CENTER

DETAILS OF PROGRAMMING A MODEL OF CHILDREN'S COUNTING IN ACTP

1980/6

MARY S. RILEY AND JAMES G. GREENO

SE 034 283

University of Pittsburgh

2

DETAILS OF PROGRAMMING A MODEL OF CHILDREN'S COUNTING IN ACTP

Mary S. Riley and James G. Greeno

Learning Research and Development Center
University of Pittsburgh

1980

Preparation of this paper was supported by the Learning Research and Development Center, supported in part by funds from the National Institute of Education (NIE), United States Department of Health, Education, and Welfare. The opinions expressed do not necessarily reflect the position or policy of NIE, and no official endorsement should be inferred.

Table of Contents

DETAILS OF PROGRAMMING A MODEL OF CHILDREN'S COUNTING IN ACTP

Mary S. Riley and James G. Greeno

Learning Research and Development Center
University of Pittsburgh

This paper is intended as an introduction to the operation and mechanics of the ACTP production system, a version of Anderson's (1976) ACT system. Its preparation was motivated by the following considerations. ACTP is already being used by Greeno (1978) to model geometry theorem proving and by Greeno, Riley, and Gelman (1979) to model the elementary knowledge required to count a set of objects. ACTP has also been identified as a potentially useful programming framework for developing models of the cognitive processes involved in other tasks, such as answering questions about a process. Together, these current and projected uses of ACTP suggested that more people would need to become familiar with the system; this in turn suggested a need for ACTP documentation specifically directed towards developing that familiarity. It is hoped that this documentation will be useful for those just beginning to program in ACTP as well as for those who simply wish to understand the production system models developed by others in more detail. The interested reader is also referred to Greeno's (1978) discussion of the more general features of ACTP and its use in his work on geometry.

The ACTP system is introduced in the context of COUNTER, a model of counting developed by Greeno, Riley, and Gelman (1979). The first section of this paper presents a general overview of the model, including a sketch of COUNTER's performance on a sample problem, to provide a general idea of how a production system operates. Section 2 discusses the mechanics of the model, including data structures, schemata, and single productions. The last section, Section 3, follows in detail the sequence of testing and executing productions involved in counting a set of objects.

Section 1: Overview of COUNTER

The formal structure used in writing COUNTER is a production system with a sequential, first-match application discipline. This means that each element of knowledge is represented as an "if-then" rule, or production, containing a condition and an action. When the program is running, the process involves a series of cycles through a set of productions. On each cycle, the conditions specified in various productions are tested in order. Eventually, the condition of one of the productions is found to be true. Then the action of that production is performed. Performance of an action completes a cycle. On the next cycle, the conditions of the various productions are tested again until one of them is found true. The action of that production is performed, and so on. The formalism of a production system model is a useful one for constructing psychological theory, since the components of the process are easily identified in the elementary productions, and there must be a relatively explicit specification of the way in which different parts of the process interact. A running program is evidence that the components are sufficient for the tasks that the model is able to perform and that they are mutually compatible so that they can be integrated into a single functioning system. General discussions of production systems as models of psychological processes have been given by Anderson (1976), Anderson, Kline, and Beasley (1978, 1979), Davis and King, (1976), Hunt and Poltrock (1974), Klahr and Wallace (1976), Newell (1972, 1973a, 1973b), Newell and Simon (1972), and Simon (1975).

A Simple Production System for Counting

An example of a simple production system is given in Table 1.

Table 1

A Simplified Production System for Counting

	Condition		Action
P1.	Have NEXT–OBJECT Have NEXT–NUMBER	→	Point to NEXT–OBJECT Say NEXT–NUMBER Change NEXT–OBJECT to CURRENT–OBJECT Change NEXT–NUMBER to CURRENT–NUMBER
P2.	Have CURRENT–OBJECT Have CURRENT–NUMBER CURRENT–OBJECT is not the last object	→	Get NEXT–OBJECT Get NEXT–NUMBER
P3.	Have no CURRENT–OBJECT Have no CURRENT–NUMBER	→	Point to first object Say first number Make the first object CURRENT–OBJECT Make the first number CURRENT–NUMBER
P4.	Else	→	Say CURRENT–NUMBER Say Finish!

A B C D

In each production the condition is stated, and an arrow separates the condition from the action of that production. The term CURRENT-OBJECT simply refers to the most recently counted object. Thus CURRENT-OBJECT will at one time refer to object A, at another time to object B, and so on, as counting proceeds. NEXT-OBJECT refers to the object that is next to the CURRENT-OBJECT in the line of count. Since in the example counting will proceed from left to right, the NEXT-OBJECT will always be the object to the immediate right of the CURRENT-OBJECT. Thus when CURRENT-OBJECT is object B, NEXT-OBJECT is object C. Similarly, CURRENT-NUMBER and NEXT-

NUMBER refer to the most recently used number and the number following it in the list of counting names (e.g., TWO and THREE), respectively. Counting, then, consists of an iterative process of getting the NEXT-OBJECT and NEXT-NUMBER, counting that object with that number, making them the CURRENT-OBJECT and CURRENT-NUMBER, respectively, getting the NEXT-OBJECT and NEXT-NUMBER, and so on until there are no more objects to count. For example, suppose that COUNTER has just been told to count the objects and wishes to begin. Initially there are no CURRENT- (and therefore no NEXT) OBJECTs and NUMBERs so the test of the conditions of P1 and P2 will fail. P3's condition is tested next and is found to be true, causing the action of that production to be performed. Here the action consists of five parts: get the first object, which is in this case A; get the first number (ONE); point to the first object and say the first number (i.e., point to object A and say "ONE"); make the first object the CURRENT-OBJECT because it has just been counted; make the first number the CURRENT-NUMBER because it has just been used. Once this action has been performed, the first cycle is complete and everything starts over again from the top (notice that P4 was never tested on this cycle. On the second cycle, the condition of P1 fails but the condition of P2 is found to be true because there now exists a CURRENT-OBJECT and a CURRENT-NUMBER. This leads to the action of getting the NEXT-OBJECT (B) and the NEXT-NUMBER (TWO). On the third cycle, the condition of the first production is true, causing the action to be performed: COUNTER points to object B and says "TWO," then changes B to CURRENT-OBJECT and TWO to CURRENT-NUMBER (i.e., they are no longer identified as NEXT-OBJECT and NEXT-NUMBER). On the fourth cycle, P1's condition is therefore false, but the condition of P2 is true again, so the action of getting the NEXT-OBJECT (C) and the NEXT-NUMBER (THREE) is performed. On the fifth cycle, the condition of P1 is true so the action is performed: COUNTER points to C, says "THREE," then changes C and THREE to the CURRENT-OBJECT and

CURRENT-NUMBER, respectively. On the sixth cycle, P2's condition is true, so the action of getting the NEXT-OBJECT and the NEXT-NUMBER is performed again. On the seventh cycle, the condition of P1 is true so COUNTER points to D, says "FOUR," then changes D and FOUR to CURRENT-OBJECT and CURRENT-NUMBER. On the eighth cycle, the conditions of P1, P2, and P3 all fail. The reason P2's condition fails is because the CURRENT-OBJECT (D) is also the last object. P4's condition is always true since it is a default condition, so the action of repeating the most recently used number, FOUR, is performed (this is intended to symbolize COUNTER identifying the cardinality of the set of objects); COUNTER then says it is finished.

Notice that this production system takes appropriate account of a variety of details. For example, the productions whose conditions test for the presence of a CURRENT-, or NEXT-, OBJECT and NUMBER (P1 and P2) precede P3 even though P3 is always the first production executed during any counting sequence. This is actually a very efficient ordering since after the first cycle COUNTER will not go through the unnecessary steps of checking to see if it has begun counting yet, as it would if P3 were ordered first in the list. It is also psychologically appealing in that it seems unlikely that children would go through such unnecessary checking each time before they counted the CURRENT-OBJECT or got the NEXT-OBJECT and NEXT-NUMBER. On the other hand, the example is deliberately sketchy and incomplete. A serious psychological theory of the knowledge used in counting would involve detailed representations of procedures for scanning an array of objects, knowledge about the number and cardinality, and other components.

Evidence for Counting Principles

The model of counting that Greeno, Riley, and Gelman developed represents a formal investigation of children's understanding of counting that includes these more detailed representations. This work is based on previous investigations by Gelman and Gallistel (1978) from which they concluded that even very young children (3-, 4-, and 5-year-olds) understand more about counting than just pointing to objects and calling out numbers; they understand general principles of counting as well. The principles referred to are:

1. Stable ordering. Counting requires a set of symbols ordered in a fixed sequence. Gelman and Gallistel called these counting symbols numerons, a convention we will follow through the remainder of this paper.

2. One-to-one correspondence. Counting requires that each object to be counted is paired with exactly one numeron, and no two objects are paired with the same numeron.

3. Cardinality. The last numeron used in counting is the symbol for the number of items in the counted set.

4. Abstraction. Sets of objects need not be homogeneous for them to be counted.

5. "Doesn't matter." It doesn't matter what order the objects in a set are counted (also referred to as the Order Invariance principle).

Gelman and Gallistel observed children's performance on a variety of counting tasks and then related this performance to children's understanding of the above principles. For example, that children understand the stable ordering principle was inferred from the occurrence of idiosyncratic counting lists (e.g., "One, two, three, six, ten" or "A, B, C, D, . . ."). Children who used their own lists did so consistently, each time uttering the list elements in the same sequential order. This suggested to Gelman and Gallistel that children appreciate that whatever the list is, its elements should occur in a fixed order.

Evidence for children's understanding of the one-to-one correspondence principle came from the observation that most children attempted to pair each object with a unique numeron and almost never used the same numeron twice or skipped a numeron. The occasional failures that did occur seemed to result from simple mechanical failures in keeping track of just what objects had already been counted.

Gelman and Gallistel cited two sources of evidence for children's understanding of the cardinality principle. First are Gelman's (1972a, 1972b) magic experiments in which children were presented with two sets of objects and on each trial instructed to choose the set with the greater number of objects. Most children had no difficulty choosing the larger set in spite of differences in types of objects and/or arrangement of objects between the two sets. This suggested that these children were using cardinality as the relevant property for choosing a set. The second source of evidence came from observations that children frequently repeated the last numeron used in counting a set of objects, often with considerable emphasis. Repetition of the final numeron suggested that these children appreciated that it signifies something special.

Evidence for understanding of abstraction came from observations that children's counting behavior is unaffected by presenting them with nonhomogeneous sets of objects.

Evidence that children understand the "doesn't matter" principle came primarily from performance on a task that consisted of presenting the child with a set of five objects and asking him/her to count the objects. Then a constraint was imposed on the child's counting procedure by pointing to one of the objects and specifying a number that is to be assigned to it. The use of constraint here refers to a restriction on the way a particular procedure can be carried out. For example, the experimenter might point to the second object and instruct the child to "make that the four." Making the second object the four is a constraint in the sense that the child normally would have counted it as two. Some children performed counting with this additional constraint by changing the order in which the objects were counted. This involved counting the first object as one, temporarily skipping the second object, counting the third and fourth objects as two and three, respectively, returning to the second object to count it four, and finally counting the fifth object as five. These children seemed to understand the order invariance principle in the sense that they all reassigned numerons to specific objects in the set. Other children, however, adopted the procedure of counting the objects as usual until they arrived at the designated object, at which point they would continue to say the numerons in order until the designated numeron came up, then proceeded again as usual. Thus, in the above example, they counted the first object, saying, "One," then said, "Two, three," then counted the remaining objects, saying, "Four, five, six, seven." These children also satisfied the constraint but sacrificed a basic principle of counting that requires each object in the set be put into one-to-one correspondence with exactly one numeron.

Thus Gelman and Gallistel provided some interesting evidence that children understand the principles of counting. In fact, the simple production system for counting (Table 1) has the surface characteristics required by Gelman and Gallistel's five principles. The constraints of stable ordering and one-to-one correspondence are satisfied by using numerons in a fixed order and applying numeron tags to objects only when the objects have not yet been tagged, but continuing until all objects have been tagged. The principle of cardinality is used because the system assigns the final numeron as the quantity of the set. Abstraction is satisfied because the system does not distinguish whether the objects in the counted set are all the same. Finally, order invariance is satisfied because the system puts no restriction on the sequence in which objects in the visual representation are counted. That is, after counting the array of objects "A, B, C, D" from left to right, COUNTER would not hesitate to count the array "C, D, B, A," also from left to right, even though this would involve changing the assignments of numerons to specific objects in the second array as compared to the first (i.e., the numeron TWO would be assigned to object B in the first array, but to D in the second).

Since the simple production model performs in agreement with Gelman and Gallistel's principles, there is at least a limited sense in which it represents understanding of those principles. However, Greeno, Riley, and Gelman believe that a stronger representation of understanding is achieved in a system that they developed and which will be described in the remainder of this section.

The sense in which they believe that their representation of understanding is stronger than the simple production model involves the generality of the knowledge structures that produce performance in agreement with principles that are understood. Gelman and Gallistel's argument that children understand general principles is based on observations of several kinds of performance. Greeno, Riley, and Gelman reasoned that if they could develop a model that would simulate a substantial part of the variety of performance that led Gelman and Gallistel to infer that children understand principles, then the knowledge in the model might constitute a plausible hypothesis about the nature of children's understanding of the principles.

COUNTER

The current version of the COUNTER model can count a set of objects arranged in an approximately linear array. Normally, when asked to count a set of objects, COUNTER first sets a goal of finding the size of the set. Next COUNTER uses spatial information it has about the objects to find an end of the array and determine the direction of counting. It then prints out the name of the first object, together with the first numeron in its ordered list of numerons. This pairing of object and numeron is intended to represent counting that object. Once an object has been counted, COUNTER identifies the next object in the set and counts it with the next numeron in its list. This process of finding the next object and pairing it with the next numeron continues until COUNTER finds no more objects along the directional path. After counting is complete, COUNTER retrieves the goal from memory to find the size of the group, causing it to relate the last numeron used in counting to an internal representation of the set of counted objects. COUNTER then repeats the last numeron used with emphasis, assigning it as the cardinality of the set. COUNTER can also modify its normal counting procedure to simulate performance on the constrained counting task designed to test understanding of the order invariance principle.

The knowledge COUNTER uses to count is represented in two forms, semantic networks and productions. Semantic networks represent general factual knowledge and are similar to the network representations proposed elsewhere (Anderson & Bower, 1973; Norman & Rumelhart, 1975; Quillian, 1969). They consist of (a) nodes that denote ideas or elements of the task situation, and (b) labeled links that connect those nodes to denote the relations among them. In the model, semantic networks are used to represent both COUNTER's ordered list of counting names and the visual information COUNTER has about a set of objects. These can be thought of as the model's data structures.

For example, Figure 1 represents COUNTER's short list of numerons. It should be pointed out that the terminology, as well as the form, of the networks and productions discussed in this section are slightly simplified compared to those that actually appear in the ACTP model. This was done to familiarize the reader with the more general aspects of networks, productions, and how they interact, without becoming involved in confusing details. The details will be discussed in Section 2. With this in mind, the node CLIST stands for "counting list"; the links labeled <u>ispart</u> between CLIST and the nodes ONE, TWO, THREE, and FOUR identify each of these numerons as a member of the same list. A fixed ordering is imposed on the numerons by a simple pattern, or schema, for NEXT relations. Consider, for example, the nodes TWO and THREE. These nodes are linked to the node N2, which in turn is linked through a token relation to NEXT. The token relation simply identifies this pattern as a specific instance (or token) of the NEXT relation, to be distinguished from other instances of the NEXT relation. The NEXT relation between THREE and FOUR is identified by the token node N3. The links labeled arga and argb are used here to define the direction of the relation. So although TWO shares a NEXT relation with both ONE and THREE, TWO is linked to ONE through an argb link and to THREE through an arga link. This means that TWO is next to and after ONE, but next to and before THREE. Similarly, THREE is next to and after TWO (argb link), but next to and before FOUR (arga link).

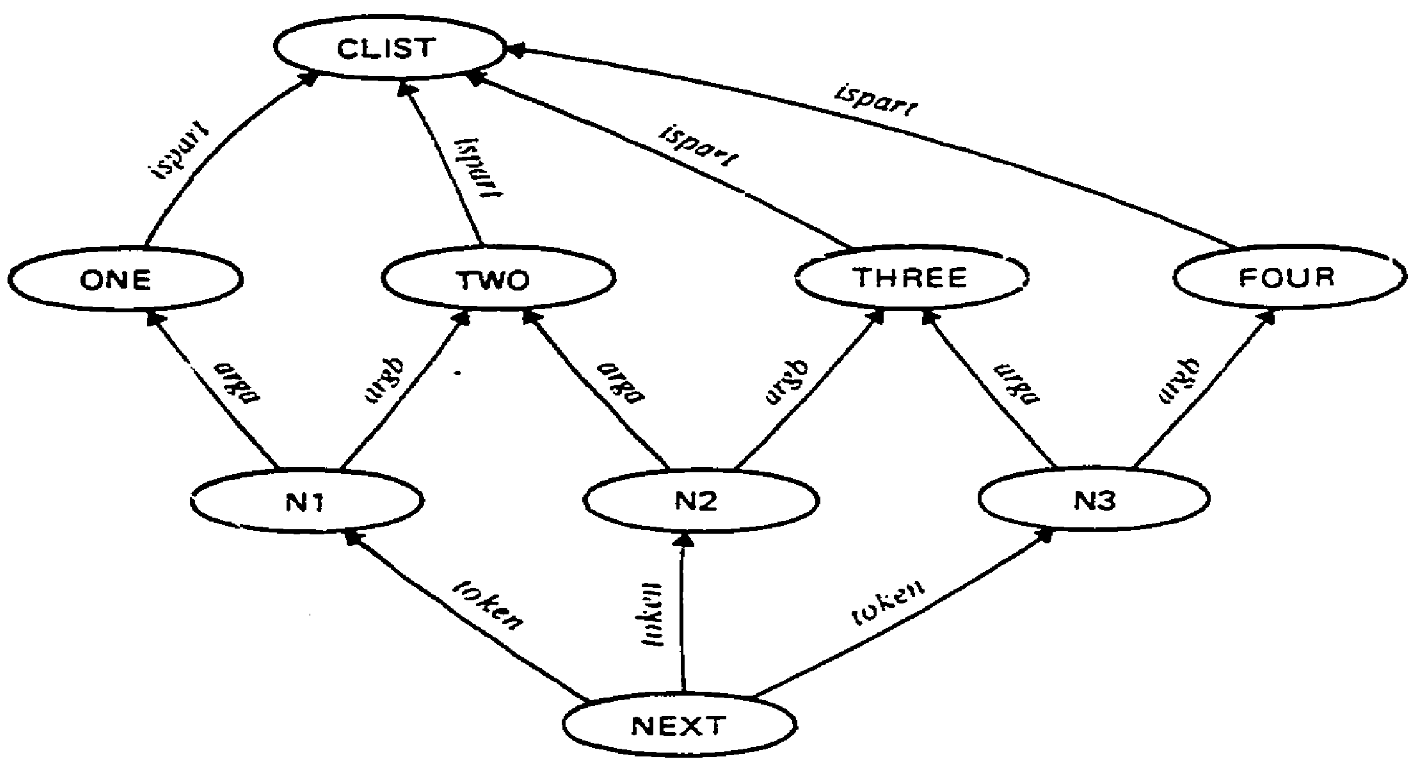

Figure 1. Semantic network representation of COUNTER's ordered list of numerons.

There are two main reasons why the number of elements in CLIST is limited. One reason was to capture the fact that young children simply do not have an unlimited resource of numerons. The second reason was that properly extending COUNTER's CLIST would involve more than just adding on numeron after numeron; it would depend on COUNTER acquiring the base-ten rule. However, since the ability to count large groups of objects is not central to the main issues addressed in the current model, the choice was made not to elaborate the acquisition process. There is, however, a production called ADDTAG which provides a means of extending CLIST to include up to ten numerons. The details of this production will be discussed in the last part of Section 2 under Schemata.

In addition to factual knowledge, COUNTER also has procedures in the form of the productions themselves. Everything COUNTER knows about how to count (i.e., getting the first object and numeron, pairing them, getting the next object and numeron, and so on) is represented as a set of productions, each of which contains a condition and an action. The condition specifies a particular interconnection of nodes and links, called patterns, that must be present in the semantic network in order for that condition to be true.

Figure 2 represents a simple production for getting the NEXT-NUMERON from the ordered counting list (CLIST). The condition consists of a single pattern; ORDASSIGN identifies the particular form of the pattern, shown in Figure 3.

The prefixes *C* and *V* define the types of nodes in the data base that can be matched to this pattern. *C* stands for "constant" which means that this part of the pattern can only be matched to a particular node in the data base which has the identical name (i.e., CURRENT). *V*, on the other hand, stands for "variable" which means that any node in the data base can qualify as a match so long as it has an <u>ida</u> link to the node *C*CURRENT (<u>ida</u> is simply one of the names used for

Condition	Action
((ORDASSIGN ☆C☆CURRENT ☆V☆CURRENT-NUMERON)	(ASYMREL ☆C☆NEXT ☆V☆TOKEN ☆V☆CURRENT-NUMERON ☆V☆NEXT-NUMERON))

Figure 2. Production for getting NEXT-NUMERON.

13

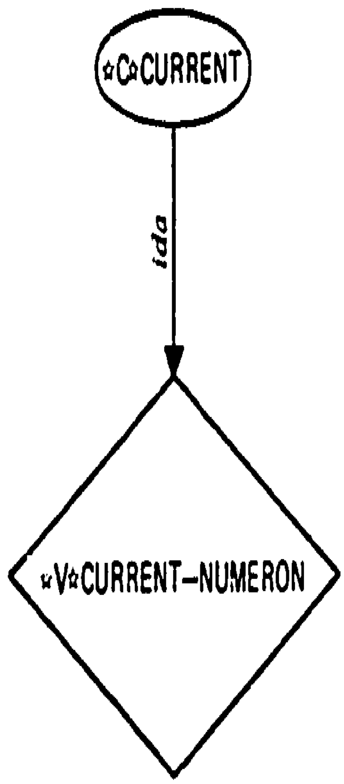

Figure 3. Pattern for retrieving the CURRENT–NUMERON from the data base.

links in this particular pattern). This means that at one time during counting, *V*CURRENT-NUMBER will match to ONE, at another time to TWO, and so on as counting proceeds. To make this a little clearer, assume that COUNTER has been told to count the array of objects "A, B, C, D" and has just finished counting B as TWO. This means that TWO is now the current numeron. COUNTER remembers this information by creating a temporary data structure that has the following pattern (Figure 4):

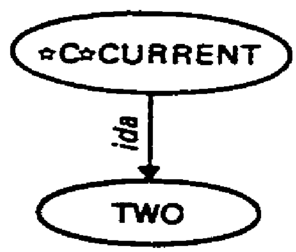

Figure 4. Data structure specifying two as the current numeron.

Assuming that the condition pattern of the production in Figure 2 is tested on the next cycle, it will match to the pattern in Figure 4 with *C*CURRENT matching to CURRENT as required and *V*CURRENT-NUMERON matching to TWO, since TWO is connected through an ida link to CURRENT. A successful match means the condition is true, so the action of the production is taken. In this case the action also consists of a single pattern and, just as ORDASSIGN tested for a particular pattern, ASYMREL tests for the pattern shown in Figure 5. *C*NEXT is a constant, *V*TOKEN and *V*NEXT-NUMERON are variables. *V*CURRENT-NUMERON is also a variable, but since it has already been matched to TWO during the condition test, it must remain matched to TWO for the remainder of the cycle. When this pattern is tested against the data base, a match is found: *V*CURRENT-NUMERON is matched to the TWO node in

Figure 1, *C*NEXT is matched to the NEXT node, *V*TOKEN is matched to the N2 node, and *V*NEXT-NUMERON is matched to THREE.

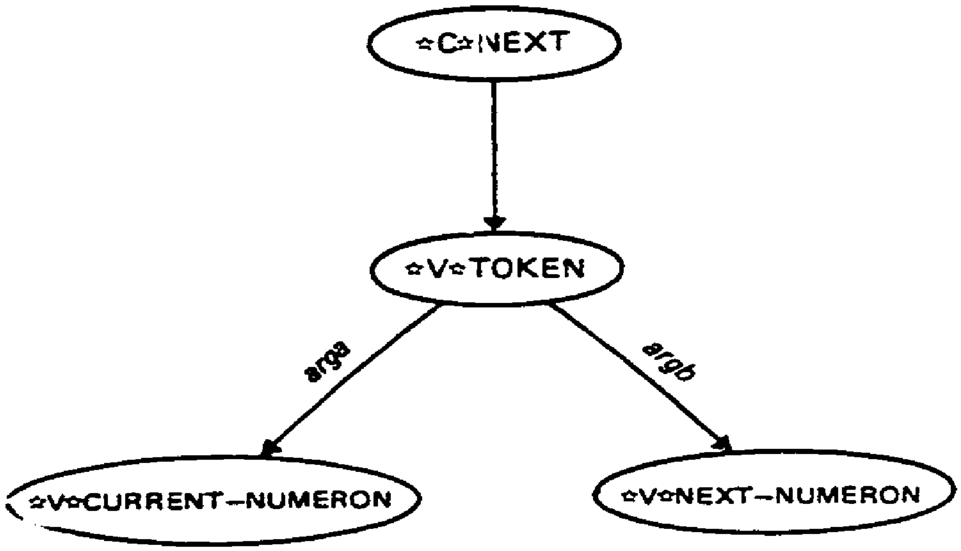

Figure 5. Pattern for retrieving NEXT–NUMERON from the data base.

Counting involves a series of such cycles through a set of productions. On each cycle, the conditions of various productions are tested in order until one of them is found to be true. This causes the action of that production to be executed, usually adding some new relations to the data base, and the cycle is complete. Cycling continues in this way until no more conditions are true.

Development of this counting model provided a specific set of hypotheses about the knowledge structures and procedures which together constitute understanding of the various counting principles. Briefly:

1. Stable ordering. Stable ordering is achieved through (a) the simple schema for NEXT relations which links each numeron in the counting list to its immediate successor, and (b) a

corresponding successor function--similar to the production in Figure 2--for accessing this ordered list.

2. <u>One-to-one correspondence</u>. Underlying one-to-one correspondence is a simple coordination between the procedures for choosing the next object and retrieving the next numeron. This coordination is achieved by the control structure of the counting procedure itself (similar in structure to, but slightly more complicated than the control structure in Table 1) and requires no additional knowledge structures.

3. <u>Cardinality.</u> Gelman and Gallistel's evidence for understanding of cardinality includes children's repetition of the final numeron, often with emphasis, and their performance in the magic experiments which apparently involves associating a quantity with the set of objects. COUNTER does this in a very simple way that depends on storing a goal in memory at the beginning of the counting sequence and maintaining that goal in memory during counting. (The details of goal storage and retrieval are discussed in Section 2.) The goal represents the intent to assign a numerical quantity to the set of counted objects. After counting is complete, COUNTER retrieves this goal from memory and adds to the data base the relational structure shown in Figure 6.

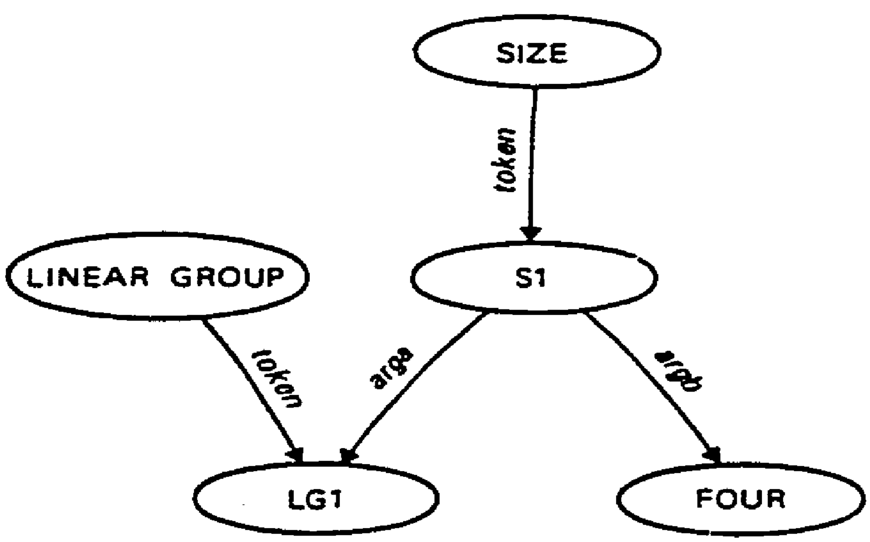

Figure 6. Data structure identifying the last numeron used in counting as the size of the group of counted objects.

This structure involves the relational property SIZE, a token node (S1) for the relation, and argument links (arga and argb) to the node representing the group of counted objects (LG1--for Linear Group) and to the numeron used last in counting (in this case, FOUR). The model has also been programmed to print the final numeron again, along with an exclamation point.

Gelman and Gallistel observed that children were less likely to repeat the final numeron when they counted larger sets of objects. In the model, repetition of the final numeron and formation of a relational data structure assigning size occur because the goal of finding the size is retrieved from memory. It is reasonable to interpret the observed lower frequency of repeating the final numeron as a result of forgetting, in which the longer process of counting included more opportunities for interference with retention of the goal of finding the set's size.

4. Abstraction. Representation of the understanding of abstraction occurs by simply omitting tests for the kind of object chosen at each step of counting.

5. Doesn't matter. Simulation of children's performance on the "Doesn't matter" (constrained counting) task requires (a) procedural knowledge about the preconditions and consequences of actions; (b) a procedure for checking the consequences of one action against the preconditions of another action; and (c) a procedure for planning action sequences such that early actions do not violate the preconditions needed for later actions. So, in addition to having a procedure for counting in the form of productions, COUNTER has knowledge about the preconditions and consequences of that procedure in the form of the semantic network shown in Figure 7. Given a set of objects to count, the COUNTER knows that the preconditions for counting any one of the objects with any one of the numerons are that the object has not yet been tagged with another numeron and the numeron has not yet been assigned to another object. (This is simply another

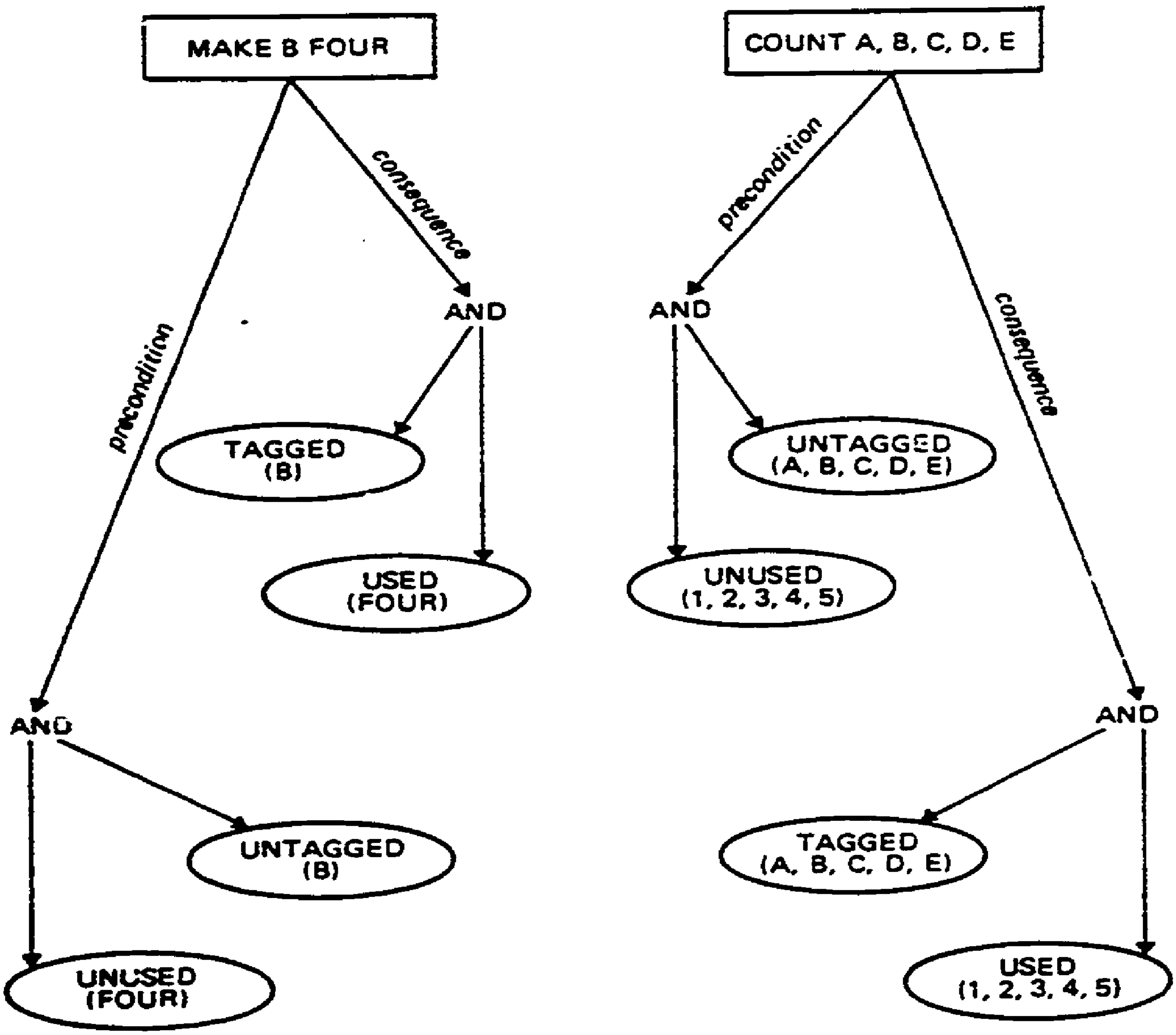

Figure 7. Data structure containing information about preconditions and consequences relevant to satisfying the special constraint.

way of saying that an object can be tagged only once, and a numeron can be assigned only once.) Counting the object with the numeron has the consequence that the object is then tagged and the numeron has been assigned. Similarly, COUNTER knows that satisfying the constraint of assigning a designated numeron to some object has the precondition that the object has not yet been tagged and the designated numeron has not yet been assigned; the consequences are that the object is tagged and the numeron is assigned. Therefore, when COUNTER is given the instruction, "Make that (the second object) the four," a procedure is executed that checks the consequences of this action (i.e., the second object is not tagged and the numeron is not assigned). If the consequences of carrying out one action violate the preconditions of another action, special checks for those preconditions are inserted in the normal counting procedure. In the example there is a violation: Given a set of five objects, if normal counting is allowed to proceed first, then the constraint can no longer be satisfied since its preconditions have been violated (i.e., the second object has been tagged TWO and FOUR has been assigned to the fourth object); similarly, if the constraint is satisfied first, then counting can no longer proceed as normal (i.e., in this case the second object is already tagged and FOUR is already assigned). These violations cause special checks for preconditions to be inserted in the normal procedure such that each time an object is chosen it is checked to determine whether it has already been tagged with a numeron (is it the constrained object?), and each time a numeron is chosen it is checked to determine whether it has already been assigned to an object (is the constrained numeron FOUR?). Normally the counting procedure simply omits these checks. Whenever one of the special checks determines that either an object or a numeron has already been tagged or assigned, respectively, a planning procedure is executed. The planning procedure modifies the sequence in which either the objects or labels are used to ensure that the preconditions of the constraint as well as of normal counting have not been violated when counting is complete. Depending upon whether any additional constraints (e.g., stable ordering, one-to-one correspondence) are imposed upon the planning procedure, a number of different sequences can be generated. Two possible modified sequences have already been discussed (page 7). The first sequence, which modifies the order in which the objects are counted, satisfies the additional constraint of maintaining one-to-one correspondence between the set of objects and the set of assigned numerons. According to the model, the second sequence does not take this constraint into consideration during planning with the result that numerons are skipped and one-to-one correspondence is not maintained.

A more thorough discussion of the model of counting and its theoretical implications can be found in Greeno, Riley, and Gelman (1979). The primary reason for mentioning it here is to provide the necessary background for discussing some of the productions in the next sections.

Section 2: Mechanics of the Model

The previous section presented a general overview of how a production system works, including an introduction to the form of COUNTER's knowledge structures and productions. However, before we can follow COUNTER through an entire counting episode in ACTP, the reader needs to become familiar with some additional features of ACTP. This section includes more detailed descriptions of COUNTER's knowledge structures together with a discussion of the mechanics of individual productions and their interactions with the data structures. Also included is a description of the schemata that comprise the condition and action patterns of the productions.

Knowledge Structures

There are two primary knowledge stuctures represented in the data base: (a) COUNTER's ordered list of numerons (CLIST), and (b) spatial information about the array of objects to be counted.

A semantic network representation of CLIST is shown in Figure 8. Notice that this is the same basic structure discussed in the last section, with the addition of a few more nodes and links. The elements of CLIST are the symbols ONE, TWO, and THREE. This is indicated by the links labeled ispart between the symbols and CLIST. The symbols are also members of the category NUMERON, as indicated by the links labeled isa between them and the NUMERON category name. The purpose of the list membership relation is to identify a numeron as a member of a list of numerons. This allows numerons in the list to be distinguished from other words COUNTER recognizes as numerons but does not yet use to count. For example, a child may know that EIGHT is a numeron before the child has learned to count to EIGHT.

The other important relation is the NEXT relation which is needed to provide a fixed order between the numerons in CLIST as required by the stable ordering principle. The ordered relation NEXT links the symbols ONE and TWO to show that TWO immediately follows ONE in the counting list. This linkage includes a token node (G0197) in the diagram) and links labeled arga and argb, indicating a specific instance of the relation NEXT in which ONE and TWO are the first and second arguments. The symbol ONE is linked through hasprop to the property name FIRST, representing that ONE has the property of being FIRST. This property allows COUNTER to identify ONE as the beginning of CLIST. The FOLLOWED property, on the other hand, allows COUNTER to find the end of CLIST. It was included because the relevant condition test for finding the end of the list is a test for the ABSENCE of a NEXT relation. However, ABSENCE tests can only be performed for single-link relations. For example, (ABSENCE OBJPROP TWO FIRST) would test for the absence of a hasprop link from TWO to FIRST. As shown in Figure 8, NEXT is a multi-link relation and so the ABSENCE test cannot be used. If it were not for this technicality, it would do just as well to search CLIST for a member A that was not connected through NEXT to another

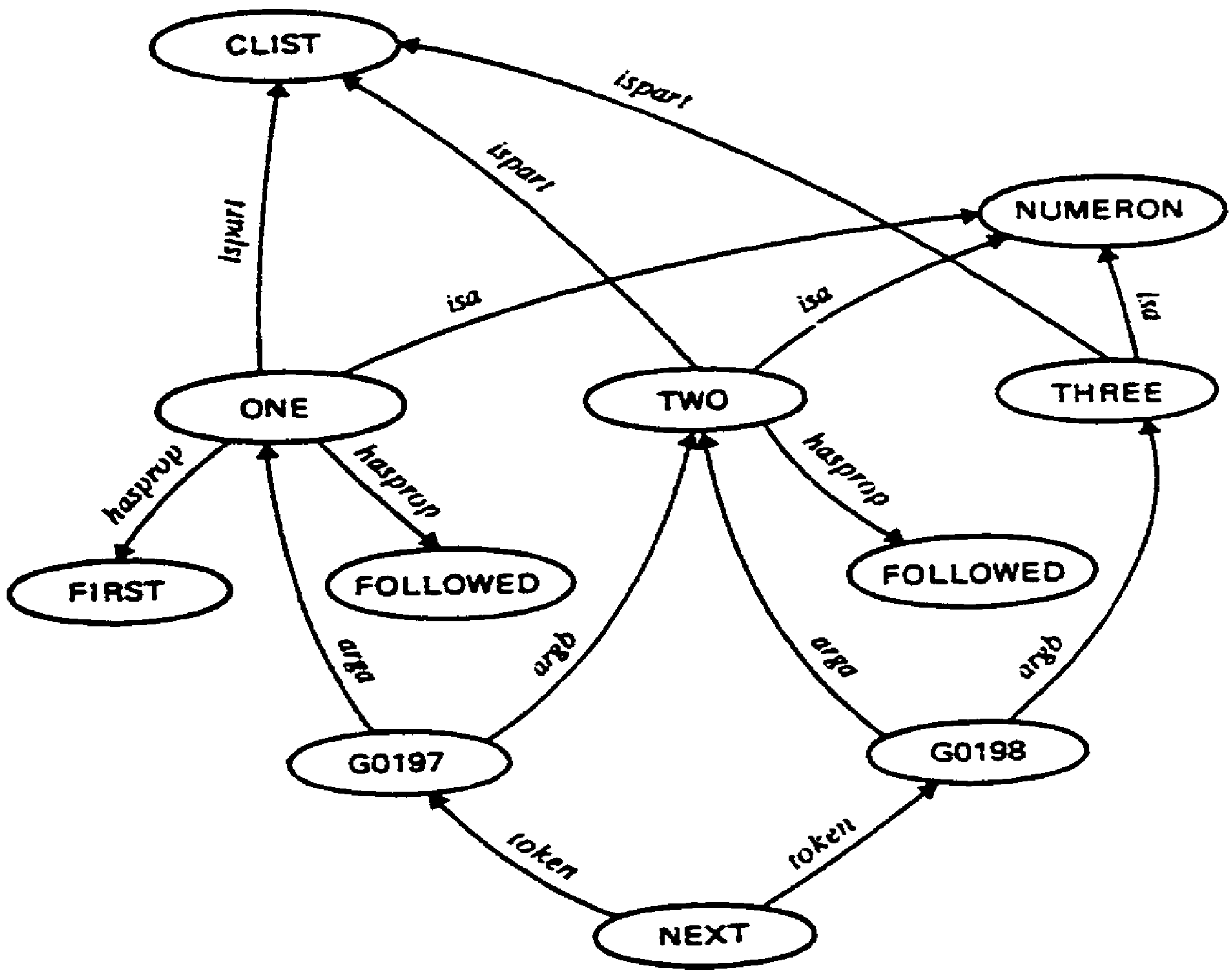

Figure 8. Data structure representing COUNTER's ordered list of numerons.

member B, such that member B came after member A in the list. If a member A were found that satisfied this constraint, then it would follow from the properties of an ordered list that member A was the last member of the list. However, in the current model, it is necessary to search instead for a member that has the ABSENCE of the single-line FOLLOWED relation; this member is then identified as the last member of CLIST. The ability to identify the last member of CLIST is a prerequisite to extending the list to include additional numerons.

Visual Information

The other data structure represents COUNTER's visual information about a set of objects and includes: (a) each object's X- and Y-coordinates; (b) the difference between the X-coordinates and the Y-coordinates of adjacent objects; and (c) the measure of the slope defined by each pair of adjacent objects. This quantitative information is used by a spatial scanning procedure for choosing the next object to count without skipping uncounted objects or repeating already counted objects. The scanning procedure is based on spatial relations that are used in forming perceptual groupings and has been shown to play an important role in counting (Beckwith & Restle, 1966). Although the current model can only form perceptual groupings for linear arrays, it seems reasonable that this scanning ability could be extended to other spatial configurations in a psychologically plausible way by including other relevant Gestalt grouping principles.

In the examples discussed in this paper, COUNTER counts four objects arranged in an approximately straight line such as the following:

A B C D

The data structure containing some of the visual information about these objects is shown in Figure 9. Objects A, B, C, and D are represented by the nodes OB0178, OB0171, OB0164,

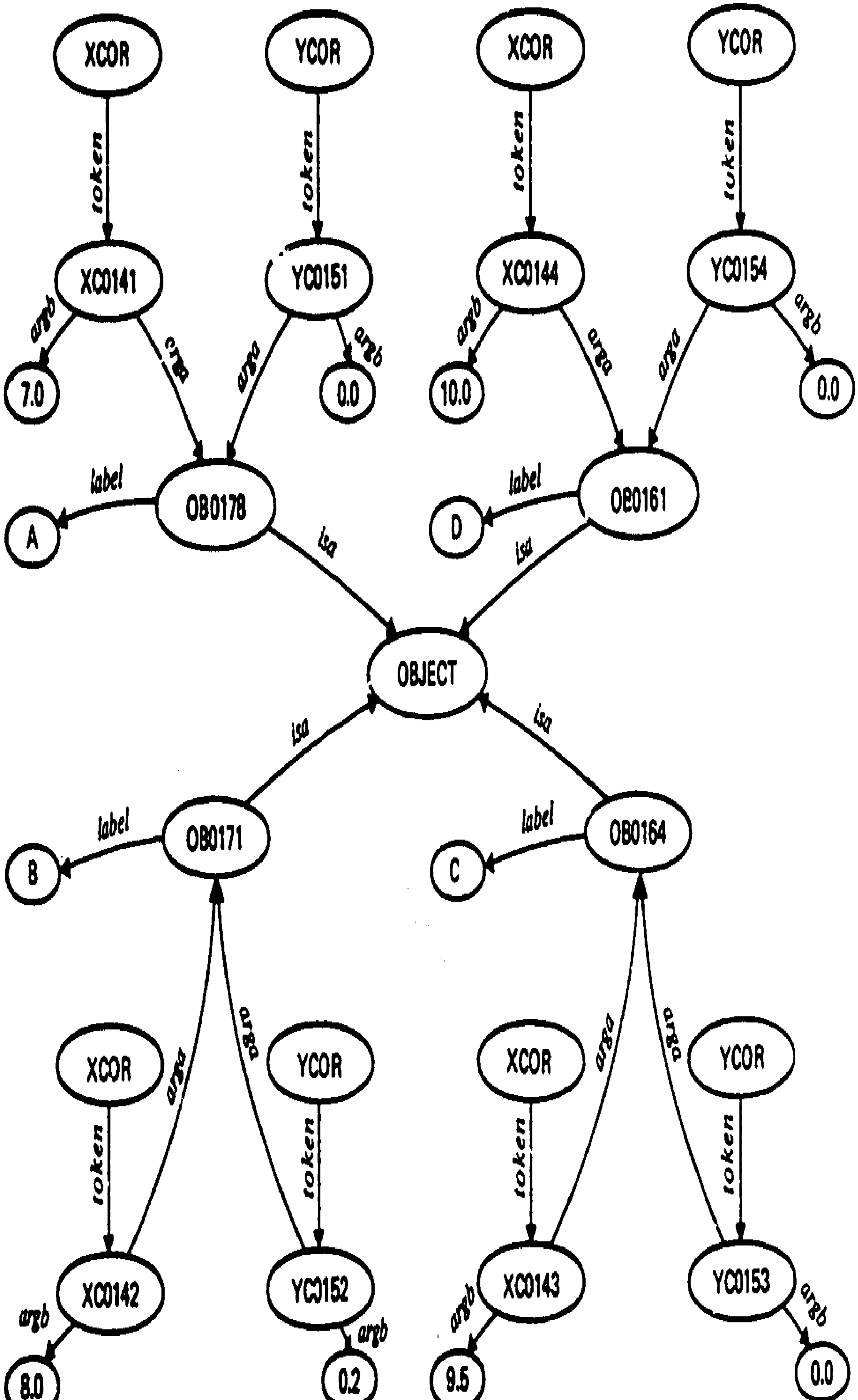

Figure 9. Data structure representing visual information about the array of objects.

and OB0161, respectively, and each of these nodes is linked through isa to the OBJECT category node to indicate category membership. Each object is also linked to its corresponding X- and Y-coordinates through relations that are tokens of XCOR and YCOR, respectively.

Information about the differences between the X- (or Y-) coordinates of adjacent object pairs, though not shown in Figure 9, is part of the same data structure and is represented in Figure 10. This particular example contains XDIF and YDIF information for objects A and B. Object A's X- and Y-coordinates are connected to their respective relation nodes through arga links; object B's X- and Y-coordinates are connected through argb links; and the value of the differences between the two X-, and two Y-, coordinates is connected by argc links.

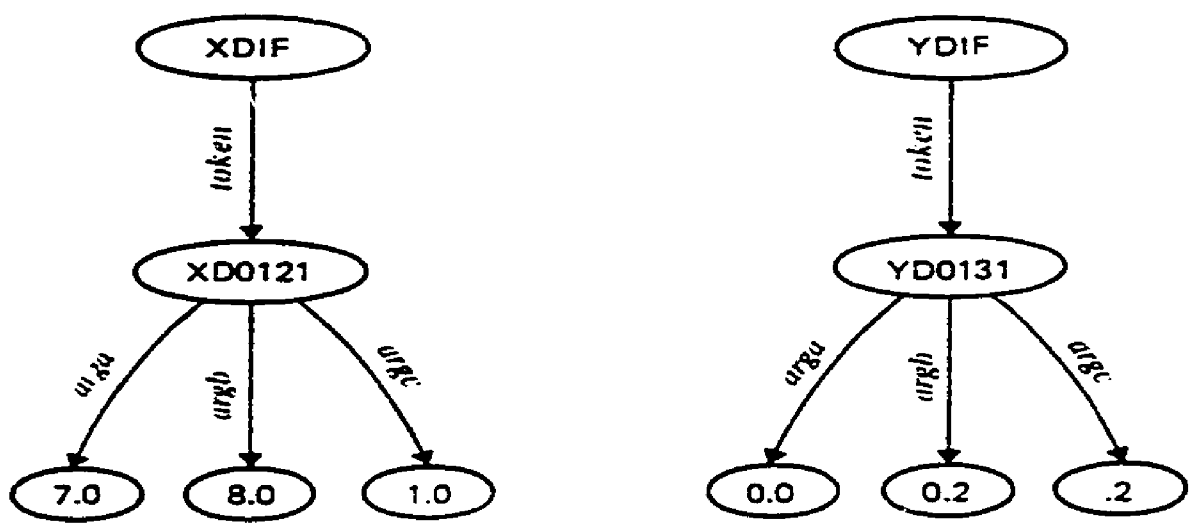

Figure 10. Data structures representing differences in X— and Y— coordinates for objects A and B.

Finally, the slope defined by two adjacent objects is represented in Figure 11. Notice there are two slope measures given for each pair of objects in the data structure.

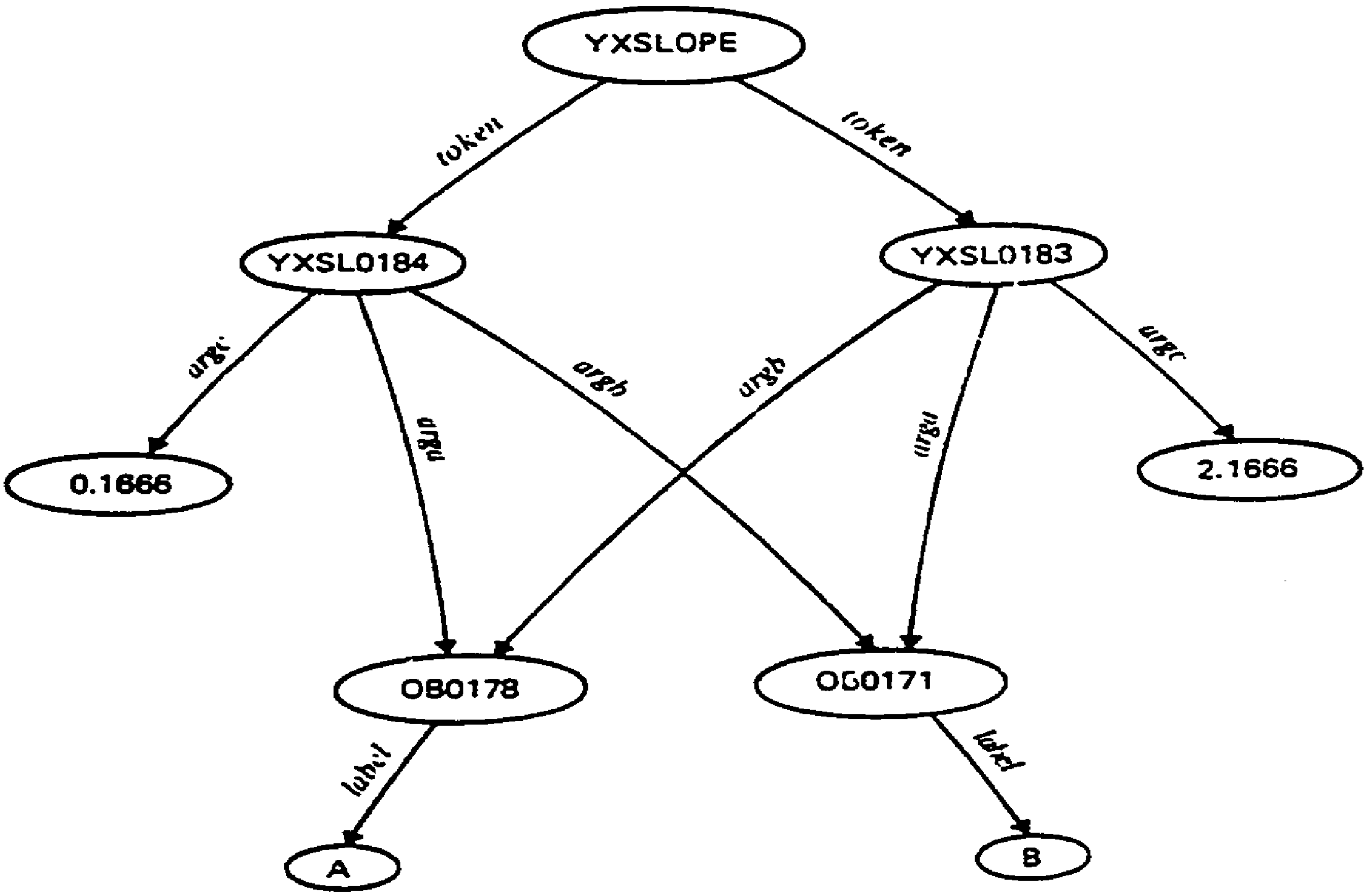

Figure 11. Data structure representing the slope defined by objects A and B.

This is because the slopes are defined in a system of linearized polar coordinates where 0.0 is horizontal and pointing to the right, 1.0 is vertical pointing upward, 2.0 is horizontal pointing left, 3.0 is vertical pointing downward, and intermediate directions are given intermediate values. Therefore, for any array of objects, the slope defined by any two of those objects can have one of two values, depending on the direction of counting. In the above example, the slope defined by objects A and B is 0.16666 when counting in a left to right direction, but 2.1666 when moving from right to left.

COUNTER uses all this information when counting an array of objects. After setting a goal to find the size of the array, it begins counting by forming an initial perceptual grouping which includes some objects at one end or the other of the array. After counting the end object as ONE, COUNTER uses the scanning process to count the remaining members of this initial grouping in order. Once all these objects have been counted, COUNTER determines if there are still objects to be counted that are not yet part of the perceptual grouping. If there are, the same scanning process is used to extend the group to include some additional objects which are then counted in turn. This process of extending the perceptual grouping and counting continues until all the objects have been incorporated into the grouping and counted. Compared to the successor function for finding the next numeron in the ordered CLIST, the procedure for finding the next object to count is relatively complicated. This suggests a plausible explanation for Gelman and Gallistel's finding that children almost never used the same numeron twice or skipped a numeron, yet they experienced occasional difficulty in keeping track of just what objects had already been counted.

Although the current version of the model limits the size of the initial perceptual grouping to three objects and extends the grouping by only a single object each time, this is not intended to mean that these numbers must remain fixed; they could be adjusted for particular spatial configurations with the addition of other Gestalt grouping principles. However, these additions would not alter the basic combination of perceptual grouping and scanning described above and in Section 3.

Single Productions

General form. Figure 12 shows an ACTP production. As with the other productions we have seen, it consists of a condition and an action. A grammar specifying the general form of productions is given in Table A-1 of Appendix A.

```
Condition:
    ((GNUMCHK ( (ASYMREL ONEONE VX11 V2 V22))) →
Action:
    (PRINT V2 V22) (POPSTACK) (UNBASE) ((ORDASSIGN BASE V22))
    GNEXTN V1 V10 V11 V12)
```

Figure 12. Production #48 from the current version of the counting model listed in Appendix B.

Condition. In ACTP the condition of a production consists of a control node and an optional set of pattern specifications. In the above example, the control node is GNUMCHK and the pattern specification is (ASYMREL ONEONE VX11 V2 V22). The control node of a production has to be active (i.e., has to be the current focus of COUNTER's counting procedure) for the condition test to succeed. If the control node is active, then the ACTP system searches for a set of links in its current semantic network corresponding to the pattern specification in the condition. For example, the pattern specification in Figure 12 is an ASYMREL structure containing four nodes (see Figure 13):

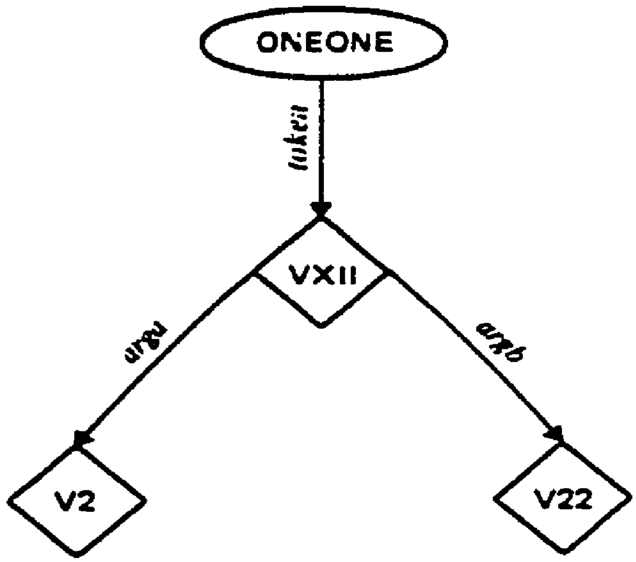

Figure 13: ASYMREL pattern specifying a ONEONE relation between nodes V2 and V22.

In pattern specifications, nodes are designated as either constants or variables. Constants are represented as oval-shaped nodes which always keep the same value, whereas the diamond-shaped variable nodes can change their values from time to time as the system is running. In the example, ONEONE is a constant. VX11, V2, V22 are variables. When ACTP is running, some variables already have values. These are called bound variables, in contrast to free variables which have no current values. In searching for a pattern, ACTP has to use the values it has for bound variables just as it has to use the constants in the specifications. Thus, a pattern search starts with the constants and values of bound variables. ACTP then searches for nodes it can fill in for the free variables.

The production in Figure 12 is relevant to the special checking procedure for the constrained counting task. When COUNTER is presented with an array of four objects, A, B, C, and D, and told, for instance, to "make C the two," a different production causes ACTP to construct a pattern involving the object C and the numeron TWO linked together through a relation that is a token of ONEONE (see Figure 14):

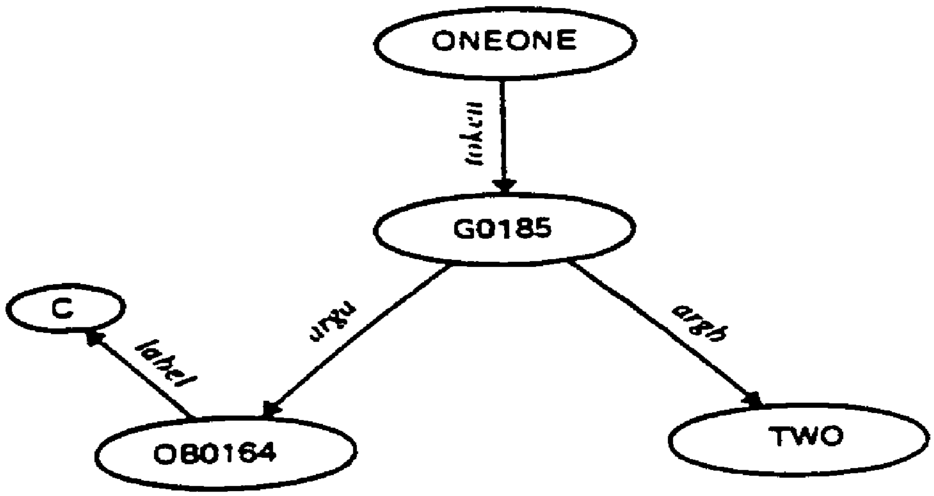

Figure 14. Data structure representing a ONEONE relation between object C and TWO.

This pattern allows COUNTER to remember that C and TWO are the object and numeron that are to be placed in one-to-one correspondence to satisfy the constraint. Then, when COUNTER is counting, each time a numeron is retrieved from the counting list, the condition of the production in Figure 12 checks to determine if it is the same numeron that is linked to C in the stored ONEONE pattern. If it is the same numeron, it is used to count C; if it is not the same numeron, then COUNTER knows it can go ahead and use it to count the other uncounted object it is current attending to.

Assume that COUNTER has been told to count the objects and "make C the two." COUNTER scans the array of objects, forms a perceptual grouping with A as the end object, and therefore ~ · ~ount A with the first numeron in its counting lis_, ONE. However, before COUNTER can proceed, it must first check if ONE is the constrained numeron. The production in Figure 12 is the relevant production for checking the numeron and so it is tested. V22 is bound to ONE for this test, and so ONE replaces V22 in the diagram (see Figure 15):

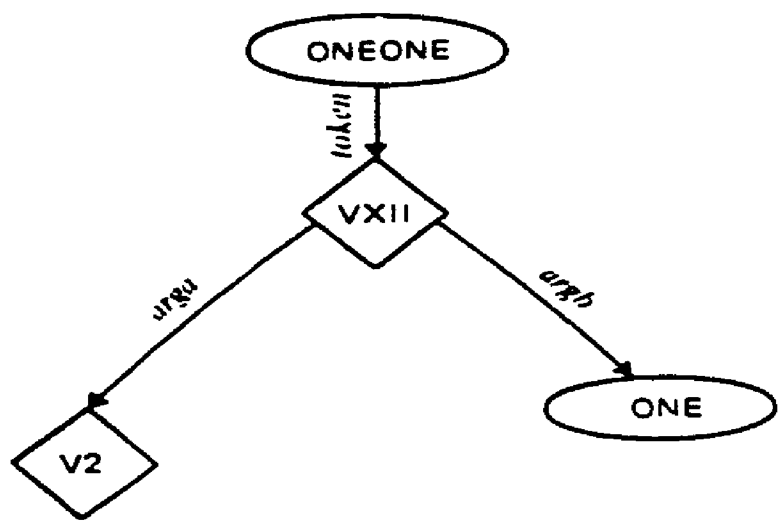

Figure 15. Pattern for retrieving a node related to ONE through a ONEONE relation.

ONEONE is always bound, of course, since it is a constant. Therefore, the pattern matches if there is a node related to ONEONE as a token, and to ONE through an <u>argb</u> relation, that can fit in as VX11; and another node related to the node found for VX11 through an <u>arga</u> relation. Since ONE is not linked to any other node through a relation that is a token of ONE-ONE, pattern matching fails and COUNTER goes ahead and counts A as ONE. COUNTER then selects the next object, B, and retrieves the next numeron, TWO. Again the procution in Figure 12 is relevant and so it is tested, this time with V22 bound to TWO (see Figure 16):

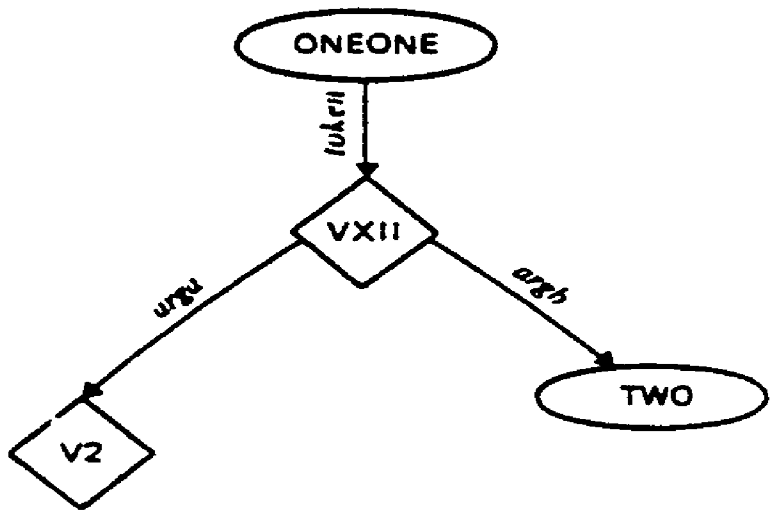

Figure 16. Pattern for retrieving a node related to TWO through a ONEONE relation.

This time the pattern matches to the structure stored in the data base which has the node G0185 related to ONEONE as a token and to TWO through the argb relation, and another node C related to G0185 through the arga relation. When pattern matching succeeds, two things happen: (a) the nodes that are found are assigned as the values of the variables mentioned in the pattern specification, and (b) the action of that production is performed. In the example, this means that C is now assigned to V2 and G0185 is assigned to VX11. The action of this production is discussed below.

Action. Three kinds of things happen in actions: (a) (a) executing special functions which include such operations as printing output to the terminal, (b) building patterns by adding new relations and nodes to the data base, and (c) remembering and activating nodes. The latter refers to the fact that an action can contain a list of constants and variables that will be activated and remembered on the next cycle of tests. Any constant on the list will be active for the next cycle; all other constants will be inactive. Any variable on the list will have its value remembered and thus will be a bound variable on the next cycle. All unmentioned variables will have their values forgotten and thus will be free variables on the next cycle.

The production in Figure 12 has five action components: (PRINT V2 V22), (POPSTACK), (UNBASE), ((ORDASSIGN BASE V22)), and GNEXTN V1 V10 V11 V12. PRINT is the special function for printing output; thus (PRINT V2 V22) is the action of printing the current values of the specified variables. In the example, V22 is bound to the numeron TWO, and V2 has just been bound to C. This piece of action, then, prints a node designating the object C along with the numeron TWO. The intention is to represent pointing to an object and saying a numeron.

(POPSTACK), another special function, is involved in removing goals from the data base once they have been satisfied. However, before we describe exactly how this is done, a brief discussion is in order concerning what goals are used in ACTP, and why goals are used in the first place.

Goals in ACTP are of two kinds. Simple goals are set by activating control nodes, such as GNUMCHK. Control nodes function as goals that produce selection of productions whose patterns will be tested. For example, the production shown in Figure 12 is one of two productions that may be tested when the control node GNUMCHK is active. Having GNUMCHK active corresponds to COUNTER having the goal of checking whether a numeron that has been retrieved is the specially constrained numeron in a counting task. Simple goals are set on a cycle-by-cycle basis and are set and removed without changing the network structure that represents the situation.

Complex goals are used when it is necessary to store information about a goal in memory. This happens whenever a goal cannot be achieved immediately and will need to be retrieved later after another goal has been set and achieved. Complex goals are stored in a pushdown memory stack. Whenever a new complex goal is adopted, the previously current goal is stored by placing it in the goal stack. Whenever the current goal is achieved, that goal is removed from the memory stack, and the next previous goal reaches the top of the goal stack.

Three functions in ACTP are involved in management of complex goals. A schema, GOALX, creates data structures that represent goal information. An example is in Figure 17. The structure shown there is constructed at the beginning of counting; the goal is to find the size of the group of objects presented to COUNTER. When this stucture is in memory, a pattern such as (GOALX GOAL VXI XFIND SIZE V2 V3) would be matched, so the system is able to retrieve information about what it needs to do next, after it h· ɜ completed a part of the task.

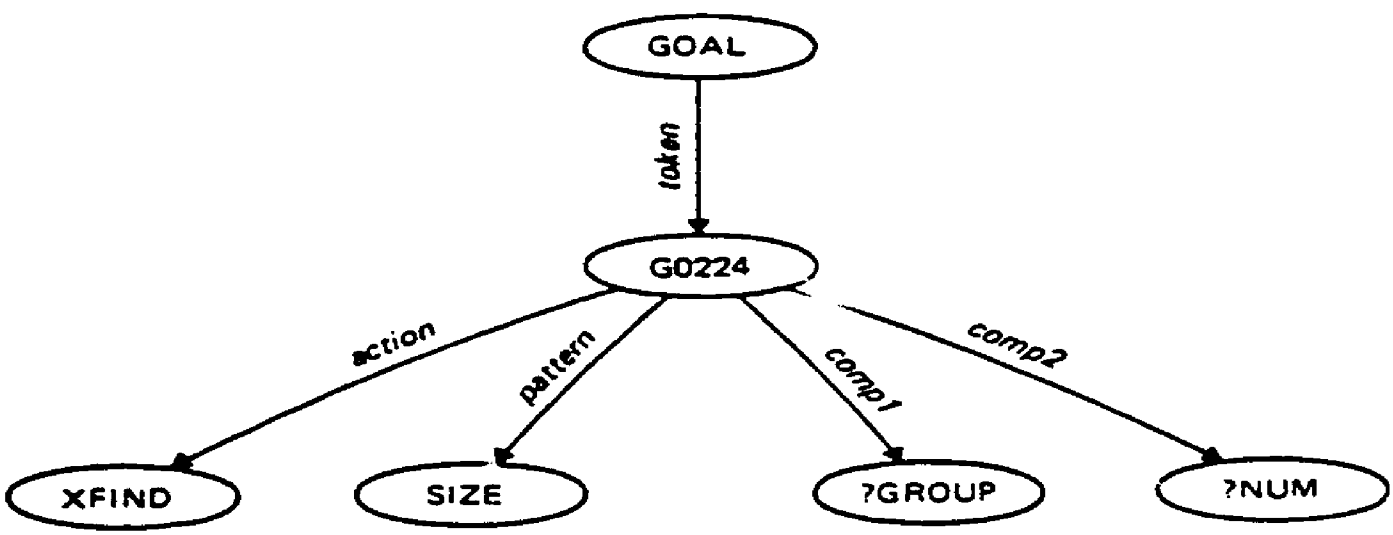

Figure 17. Data structure representing the goal of finding the size of the group of objects to be counted.

Goal structures are formed in ACTP by a function SETGOAL, which creates a structure in the form shown in Figure 17, using the GOALX schema. SETGOAL also modifies the goal stack. Before creating a new goal structure, SETGOAL adds the current goal to the stack of prior goals in memory. Another function

POPSTACK, is used in an action when a goal has been accomplished. POPSTACK removes the current goal from the data structure and changes the goal stack by removing the top entry from the stack and making it the current goal.

The structure shown in Figure 17 is formed when COUNTER is asked how many objects there are in a set. This identifies G0224 as the current goal. If in addition to determining how many objects are in an array, COUNTER is also told to "make C TWO," G0238 is added to the goalstack, and a second GOALX pattern is built in the data base, identifying G0238 as the new current goal (see Figure 18).

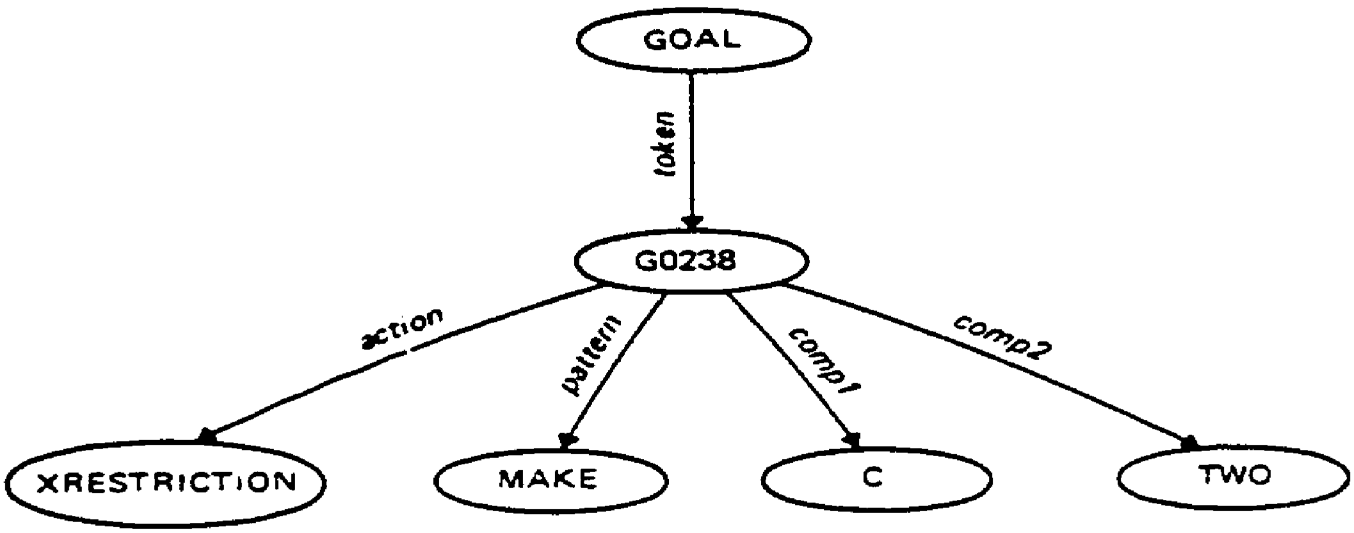

Figure 18. Data structure representing the goal of "Making C TWO"

Figure 19, then, represents the goal stack at the time the production in Figure 12 is relevant.

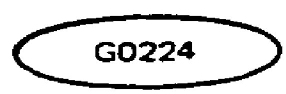

Figure 19. Goal stack containing one goal.

However, the current goal of satisfying the special constraint was achieved when the action (PRINT V2 V22) was performed. So (POPSTACK) removes G0238 from the data base and checks the goal stack to see if there are any more goals. This is equivalent to COUNTER asking itself, "Now that I've satisfied the constraint of making C the TWO, is there anything else I need to take care of?" In this case, there is another goal in the stack--the goal of assigning the last numeron used in counting as the cardinality of the set. POPSTACK removes G0224 from the stack and makes it the current goal once again.

Referring again to Figure 12, the special function (UNBASE) removes the structure that represents the current problem base. This data structure has the node BASE connected to one or more nodes in the data base by links ida, idb, idc, and so on. BASE's main function is to provide eraseable memory that is not easily handled with bound variables in the ACTP system. BASE puts a node into memory so that it can be retrieved and assigned as the value of a variable during some later cycle. In the example, the current problem base at the time the action of the production is taken is shown in Figure 20.

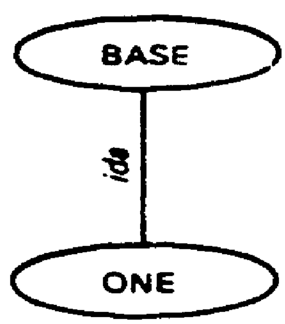

Figure 20. Data structure representing the current problem BASE.

This structure was used to store the most recently used numeron so that it would be available to COUNTER when it came time to retrieve the next numeron from the ordered list of numerons. That is, when it came time to count object B, COUNTER needed to remember that ONE was the last numeron it used in order to retrieve the appropriate next numeron (i.e., TWO) from CLIST. However, now that TWO has just been used to count C, ONE is no longer the most recently used numeron and so UNBASE removes it from the data base. The next action component, ((ORDASSIGN BASE V22)), creates a new pattern in the data base, making the current value of V22 (which in this case is TWO) the base of the problem, as shown in Figure 21.

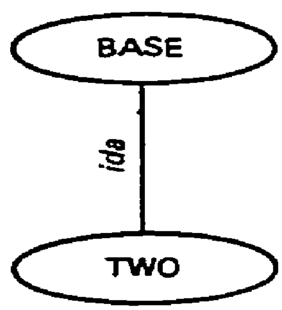

Figure 21. Data structure representing the current problem BASE.

This means that the next time COUNTER needs a numeron to count an object, it will remember that TWO was the last numeron it used and therefore choose THREE as the next unused numeron. THREE will be used to count the object and will then replace TWO as the current base, and so on. So in the example, UNBASE and ORDASSIGN are part of an iterative procedure that allows COUNTER to proceed systematically through its ordered CLIST without skipping or repeating numerons.

The last action component consists of GNEXTN and some variables. GNEXTN is a constant, and its being mentioned at the end of the production causes the control node GNEXTN to be active on the next cycle, in turn causing the condition patterns of a different production (or productions) to be tested on the next cycle. The reason GNEXTN, and not some other constant, is mentioned here is because it is the constant used as the control node for the productions that are relevant to the next step in COUNTER's procedure. In this case, the next step is to retrieve the next unused numeron from CLIST. This is because COUNTER had retrieved TWO, intending to use it to count B, but discovered that TWO was the constrained numeron and had to use it to count C, the constrained object, instead. This means that COUNTER still needs a numeron to count B. Since productions with the control node GNEXTN are designed to retrieve the next unused numeron from CLIST, their control node is activated for the next cycle.

Finally, V1, V10, V11, and V12 are variables; mentioning them in the action of the production causes their current values to remain bound on the next cycle. In the example, object B is the current value of V1 and must remain assigned so COUNTER can remember that B is the object it intends to count next. V10, V11, and V12 are related to COUNTER's perceptual scanning and grouping procedures which will be discussed in the next section.

Steps in matching and executing a production. The syntax of a production is easier to understand if one has a clear understanding of the procedures used in attempting to match the conditions and executing the actions of productions.

When ACTP tests a production, the question is whether the condition can be matched in the data structure. Typically, there are two parts of a condition: a control node and a pattern specifiction. Neither of these is required, and productions written for ACTP often have only a control node. More than one control node or more than one pattern specification can be included, but that has not been done in any models that have been programmed.

ACTP proceeds through the elements in the condition of a production. If an element is an atom, ACTP tests whether it is an active node. If it is not, then further testing is omitted. If an element is a list, ACTP assumes that the element is a pattern specification. This will be a list of concept schemata, each of which is a list beginning with a schema name, such as ASYMREL or ORDASSIGN. ACTP assembles a list of links that correspond to the concept schemata in the pattern, noting the nodes that consist of constants, bound variables, and free variables. ACTP then tests whether the pattern can be matched in the current data structure. If it fails, it proceeds to the next production. If a match is found, the free variables in the pattern are bound to the nodes that they matched, and ACTP goes on to the next element in the condition, if there is one. If all the control nodes are active and all the patterns can be matched, ACTP goes on to execute the action of that production.

Actions have three kinds of components: atoms, which must be variables or constants; special functions, which are included in single parentheses; and pattern specifications, which are doubly parenthesized, i.e., lists of lists. In executing an action, ACTP proceeds through the elements of the action. If ACTP encounters an atom that it recognizes as a constant, it places the atom on the list of active nodes. If the element is a variable, it places the value of the variable and the variable on the list of bound variables for the next cycle and makes the value an active node. If ACTP encounters the name of a special function in a list, then the function will be executed by LISP. If ACTP encounters a list that is not a special function, then it assumes a pattern specification. It assembles the list of links that correspond to the pattern specification, using the values of all variables that were either bound initially in the cycle or that were matched in testing the condition of the production. Any variables that do not have values are given values in the form of unique symbols generated by LISP. The links in this set are added to the data structure.

Comments on the use of parentheses in productions. The typical form of a production is shown in Figure 22. The entire production, consisting of a condition-action pair, is inside a set of parentheses. Then the condition is also enclosed in parentheses to separate it from the action. Within the condition, the list of pattern specifications is contained in yet another pair of parentheses, and each individual pattern specification is also in parentheses. Sometimes a condition contains no pattern specifications:

```
((GCOUNT) (PRINT V1 V22) (UNBASE) ((ORDASSIGN BASE
V22)) GNEXTOB V10 V11 V12)
```

Here the entire condition is the control node GCOUNT, closed off by a single parenthesis; everything else is the action.

In the action, pattern specifications are inside double parentheses. Special functions are inside single parentheses. Nodes and variables that are to be kept active for the next cycle are just mentioned, with no parentheses.

Thus, when reading a production from left to right:

1. A production starts with two left parentheses.

2. There is a single symbol at the beginning. This is the control node for the production.

3. If there is a right parenthesis after the control node, that completes the condition of the production.

4. If there is no right parenthesis after the control node, there should be two left parentheses. This is the beginning of a list of one or more pattern specifications. The list of pattern specifications ends with three right parentheses, and this completes the condition.

5. The action may contain one or more pattern specifications. Each pattern specification (or each list of pattern specifications) begins with two left parentheses and ends with two right parentheses.

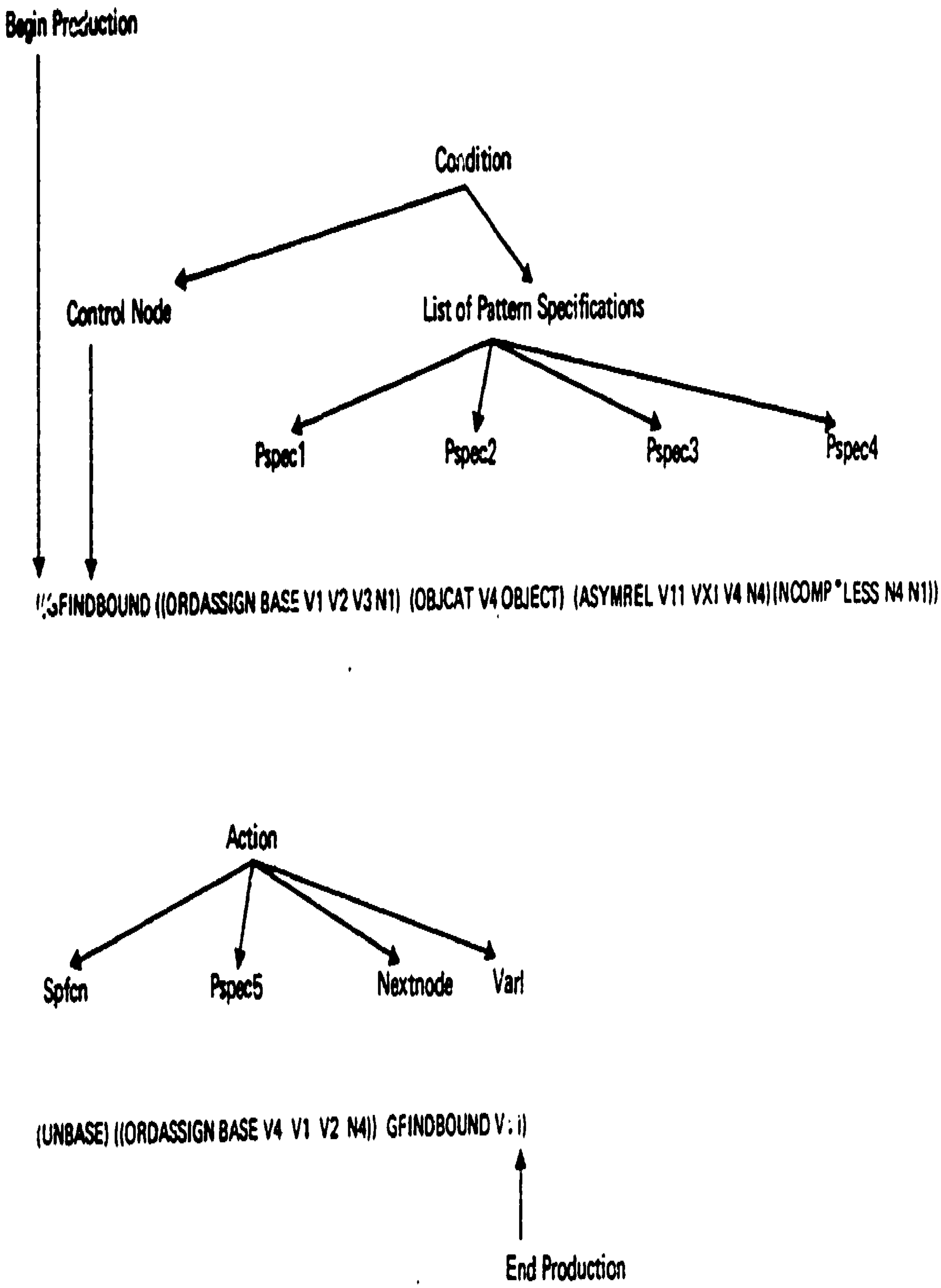

Figure 22. Typical form of a production (Production #16).

6. The action may contain one or more special functions. Each special function begins with one left parenthesis and ends with one right parenthesis.

7. The action may contain a control node to be active on the next cycle. This will be mentioned with no parentheses around it.

8. The action may contain one or more variables whose values will be remembered (bound) on the next cycle. These will be mentioned with no parentheses around them.

9. The production ends with a single right parenthesis. If the last thing in the production is a control node or a variable, the terminating parenthesis will be by itself. If the last thing is a special function, its right parenthesis will be with the terminating parenthesis, so that the production will end with two right parentheses. If the last thing is a pattern specification, its two right parentheses will be there, so the production will end with three right parentheses.

A flowchart for writing a production is shown in Figure 23. Although it assumes a rigid order for the action side of productions, this is not mandatory and experienced users may prefer different orders.

<u>A special note on conditions with no pattern specifications</u>. When the condition of a production consists only of a control node, the action of that production will be taken if (a) the control node is active during a given cycle, and (b) no preceding production has already been tested as true on that cycle. This can be made clearer by the following example.

Compare Productions 23 and 24 from Appendix B:

<u>P23</u>

((GOBJCHK ((ABSENCE OBJPROP V1 SPECIAL)) GNEXTN V10 V11 V12 V1)

<u>P24</u>

((GOBJCHK) GNEXTOB V10 V11 V12)

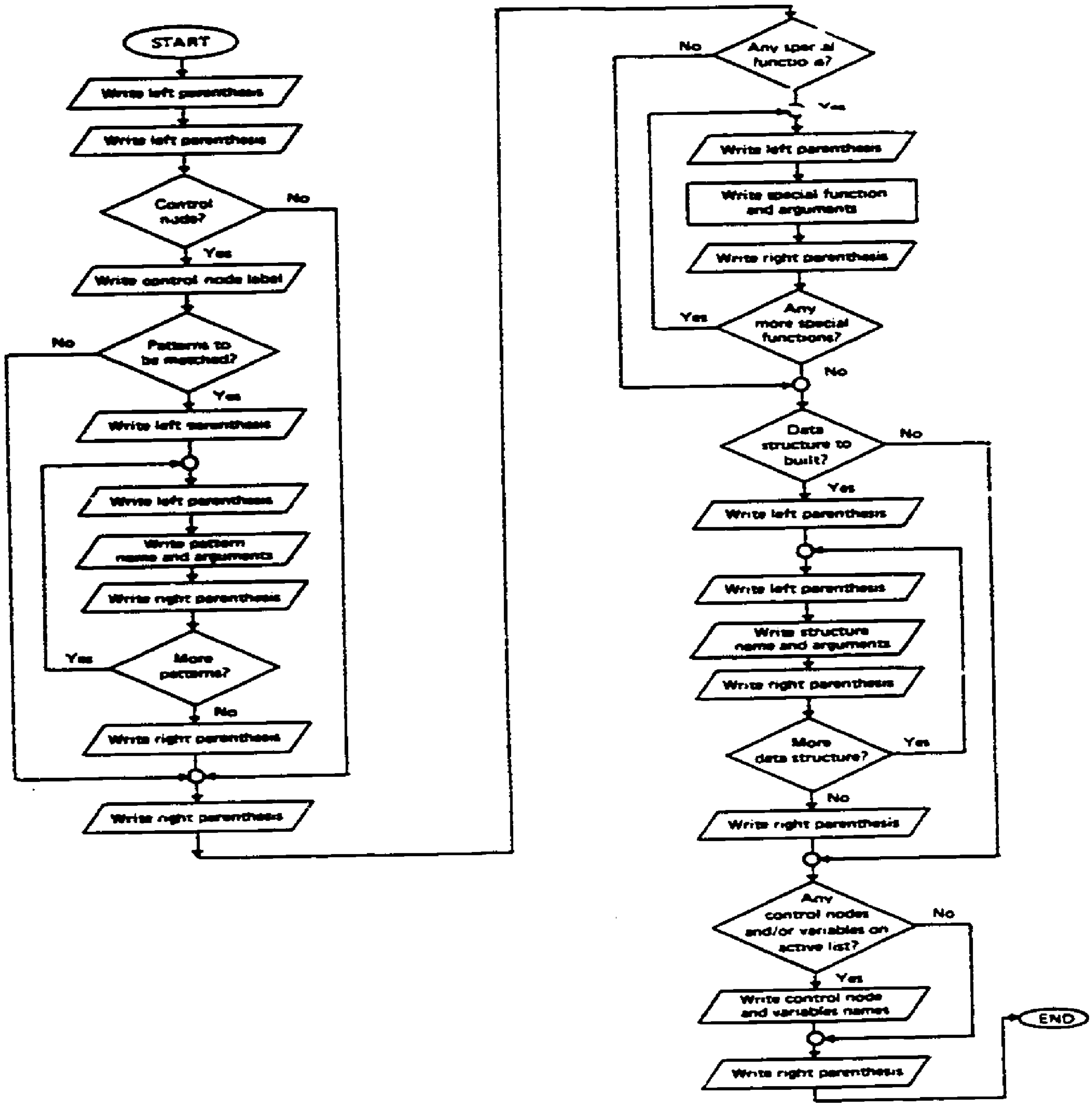

Figure 23. Flowchart for writing a single production.

These productions are relevant to the special checking involved in the constrained counting task. In constrained counting, each time COUNTER selects a new object to count, it checks the object to determine if it is the one involved in the constraint. If it is not the constrained object, COUNTER goes ahead and gets the next unused numeron from CLIST. If it is the constrained object, COUNTER skips it and selects the next object to count from the array. This is because COUNTER intends to count the constrained object whenever the constrained numeron shows up as the "next" numeron in the course of counting the array; skipping the constrained object here, then, ensures that it will not be counted twice.

Production 23 checks the object assigned as the value of V1 to make sure that it does not have the property SPECIAL ("SPECIAL" here means "constrained"). If the object in question is not a special (constrained) object, then the condition of this production is true and its action is taken. This causes the control node GNEXTN to be active on the next cycle as well as the variables V10, V11, V12, and V1 to remain bound with their current values. GNEXTN controls the productions responsible for getting the next counting name in the model's ordered counting list.

Production 24, on the other hand, has no condition patterns to be matched. However, while it is true that there are no <u>explicit</u> condition patterns, consider the following situation. On any cycle when GOBJCHK is active, both Productions 23 and 24 are possible candidates for testing because they both have the same control node. But remember that only one production is fired during any single cycle and this production will be the first production whose control node is active and whose condition pattern (if any) matches successfully to the data base. Furthermore, ACTP productions are always tested in order. This means that whenever GOBJCHK is active, Production 23 is always tested before Production 24. If Production 23's condition pattern matches, its action changes the active control node to GNEXTN for the next cycle; in this case Production 24 is never tested at all. In fact, the only time Production 24 is tested is when Production 23's condition pattern does not match. So, taken together, Productions 23 and 24 say, "If the object attached as the value of V1 is not special (i.e., if this is not the constrained object), then go ahead and get the next numeron in the counting list; otherwise (i.e., if this is the constrained object) find another object to count (this is accomplished by activating the control node GNEXTOB)." Production 24 does, therefore, have an implicit condition in this case by virtue of following another production having the same control node. This will often be true of other "conditionless" productions in the model and is important to keep in mind when interpreting Appendix B.

Schemata

A schema is a concept represented as a set of links that go together to make a recognizable configuration. Each schema has a name which is used to identify that schema in productions. A schema also has some slots that are filled in with variables or constants when the schema is used in a production. Some schemata in ACTP correspond to a single link; other schemata contain several links. Three of the single-link schemata--OBJTYPE, OBJPROP, and PARTOBJ--are shown in Figure 24. OBJTYPE (also referred to as OBJCAT for "object category") has its arguments linked by isa, OBJPROP has its arguments linked by hasprop, and PARTOBJ has its arguments linked by ispart.

A fourth schema that is used in this system is ASYMREL, a generic structure involving an asymmetric relation with any number of arguments. For example, the ASYMREL schema shown in Figure 25 has two arguments for a total of four nodes. The name of the relation (for example, NEXT, ONEONE, YXSLOPE, SDIF or YDIF) is the node at the top. A unique symbol is a token of that relation. This symbol can be any convenient

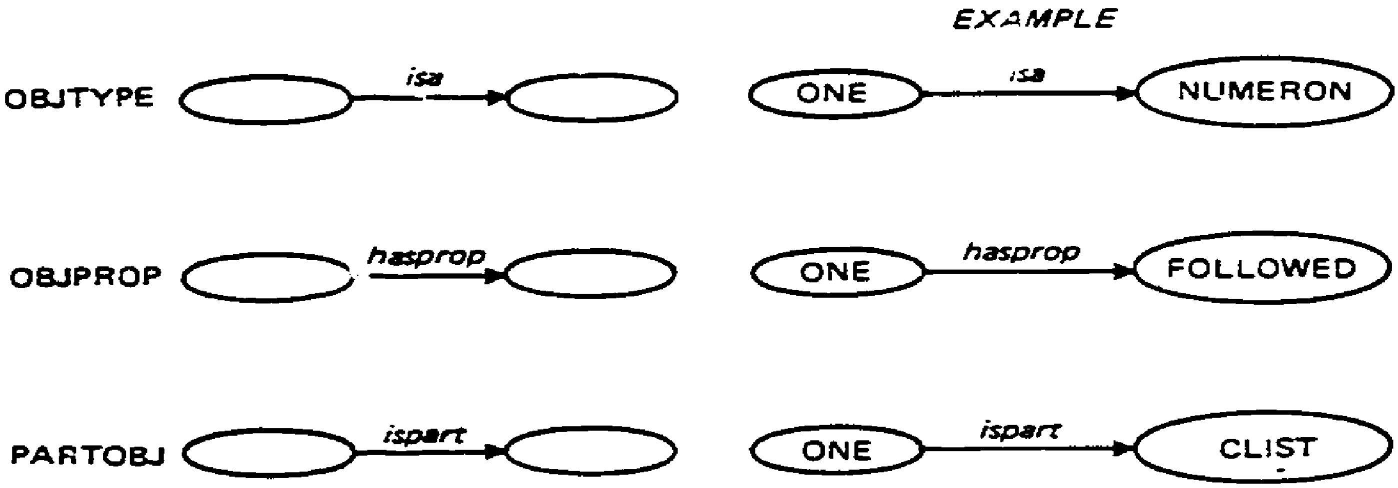

Figure 24. Single—link schemata.

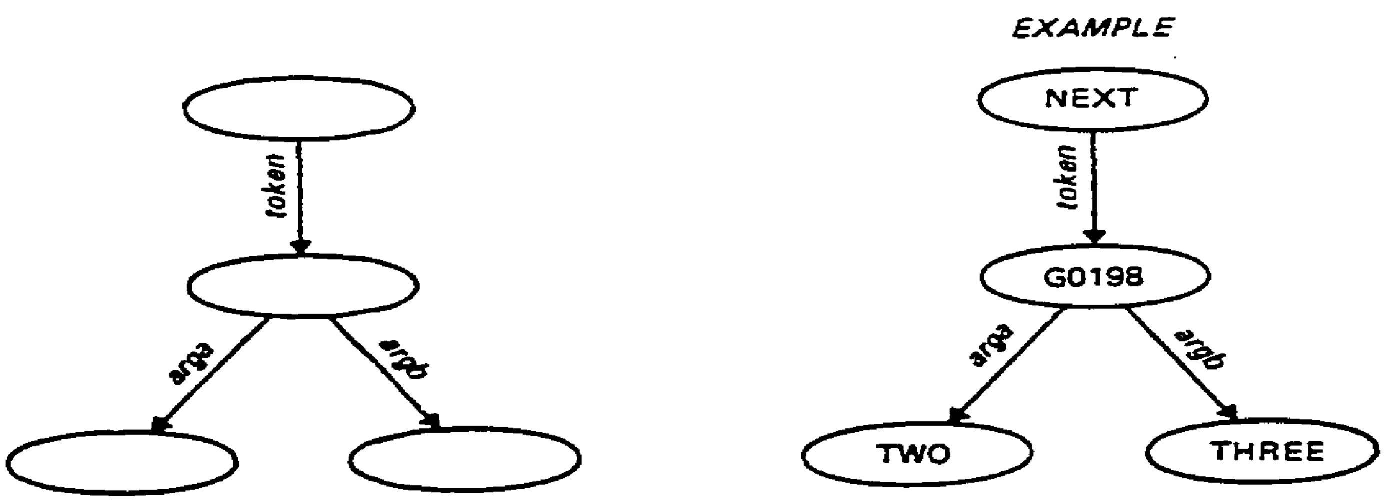

Figure 25. ASYMREL schema.

identifier such as NXT or YX, followed by a number; when an identifier is not specified, then the unique symbol is G followed by a number. The arguments are included as the remaining nodes in the structure. For example, suppose a production has the pattern (ASYMREL NEXT VX1 V21, V22), and V21 is bound with the value TWO (refer to the CLIST diagram in Figure 8). This would match with VX1 bound to G0198, and V22 bound to THREE. Consider another example involving the same pattern, but suppose that V22 is bound with the value TWO when pattern matching occurs. Then the pattern will match with VX1 bound to G0197, and V21 bound to ONE.

Finally, the ORDASSIGN schema tests for the pattern shown in Figure 26:

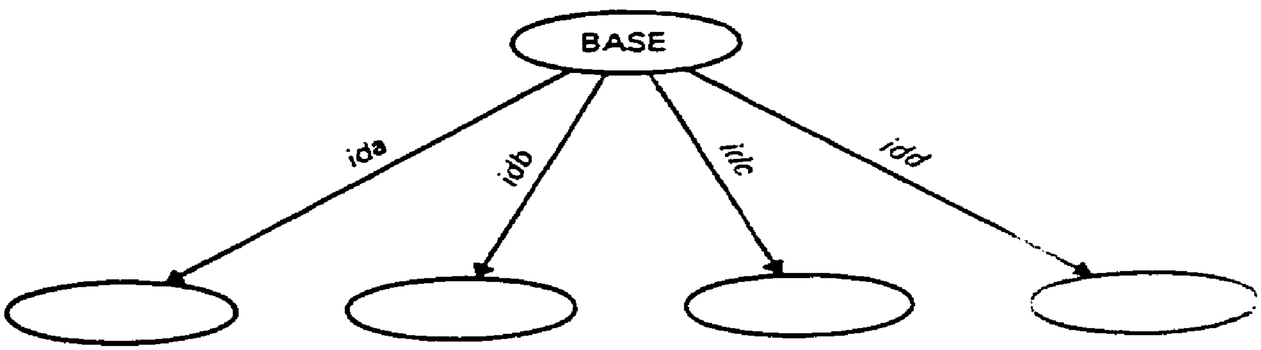

Figure 26. ORDASSIGN schema.

The top node is generally BASE, though it can be anything. Although there are four arguments linked to BASE in the above example, ORDASSIGN, like ASYMREL, can take an unlimited number of arguments. For example, in Production 48 ORDASSIGN took only one argument: ((ORDASSIGN BASE V22)). The ORDASSIGN pattern can be removed from the data base by using the function UNBASE. This means that the base of the process can be altered from time to time during the running of the production system. The utility of this schema will become apparent in the discussion of the actual counting model.

The uses of schemata involve matching and generating patterns. In pattern matching, a schema will be part of a structure that has some of its arguments already fixed as constants or bound variables. If all the arguments are fixed, pattern matching is simply a check to see whether the links in the data base agree with those specified in the schemata. If some of the arguments are not fixed, then pattern matching involves a search to see whether there are nodes in the data base that will fit into the specified structure.

For example, suppose a production has the pattern (ASYMREL NEXT VX1 V21, V22). NEXT is a constant; suppose VX1, V21, and V22 are not bound. The pattern matches if there is a node related to NEXT as a token, to another node through the arga relation, and to yet another node through the argb relation. The diagram of the CLIST structure indicates two sets of nodes that qualify: either G0197, ONE, TWO, or G0198, TWO, THREE. One of the token nodes (either G0197 or G0198) will become bound as the value of VX1, and its corresponding arga and argb relation nodes will become bound as the values of V21 and V22, respectively. Since there are several valid possibilities, it is not clear which of the sets of nodes will actually be found.

Now consider an example of how different schemata are combined in a complex pattern specification. The example is taken from the condition pattern of the ADDTAG production mentioned earlier. Basically, this production extends the ordered CLIST by (a) identifying an element known to be a numeron but not yet a number of CLIST, and (b) linking this element through NEXT to the last numeron in CLIST:

```
((ADDTAG ((ASYMREL NEXT VX1 V21 V22) (ABSENCE OBJPROP V22
FOLLOWED)
    (OBJCAT V23 NUMERON) (ABSENCE OBJPROP V23 FOLLOWED)))
((PARTOBJ V23 CLIST) (ASYMREL NEXT VX2 V22 V23) (OBJPROP V22
FOLLOWED))
```

The condition consists of the control node ADDTAG and a pattern specification composed of four schemata. This means that the pattern to be found cannot have a different node in the ASYMREL schema than in the OBJPROP schema. Thus the two schemata are combined into the following single pattern in Figure 27:

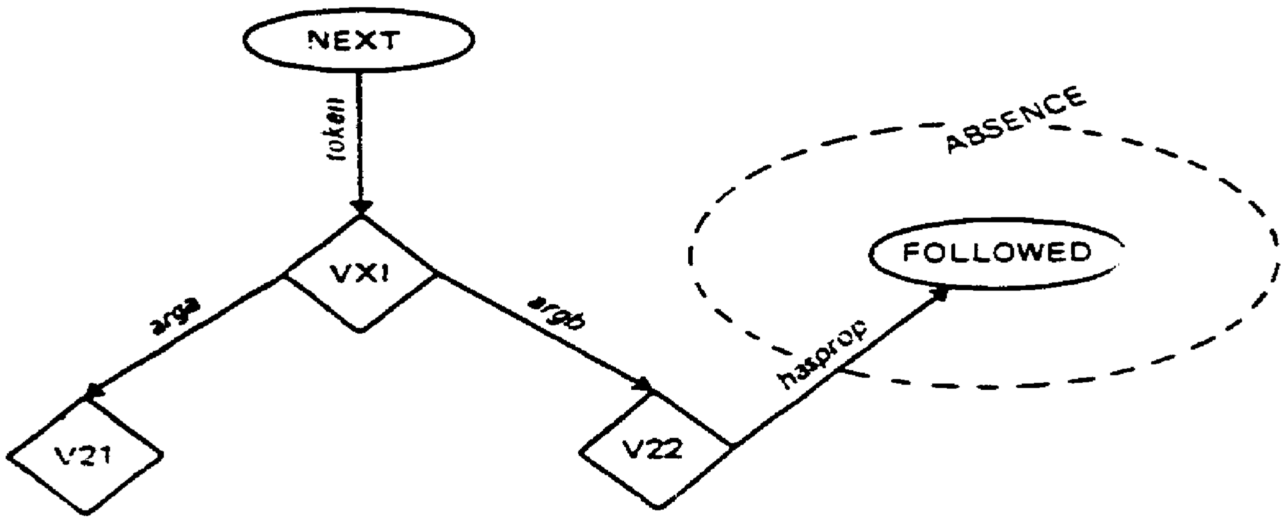

Figure 27. Pattern for retrieving the last numeron in CLIST.

Together they allow the model to identify the last numeron in CLIST. That is, the only nodes involved in NEXT relations in the current data base are the nodes representing the numerons in CLIST and the only one of these nodes lacking the property FOLLOWED is the last one. Indeed, referring back to Figure 3, node THREE is the only node in the data base that has the right relation with both the consta· NEXT and the constant FOLLOWED. The pattern matches, THR. becomes the value of V22, and TWO and G0198 are assigned as the values of V21 and VX1, respectively, as shown in Figure 28.

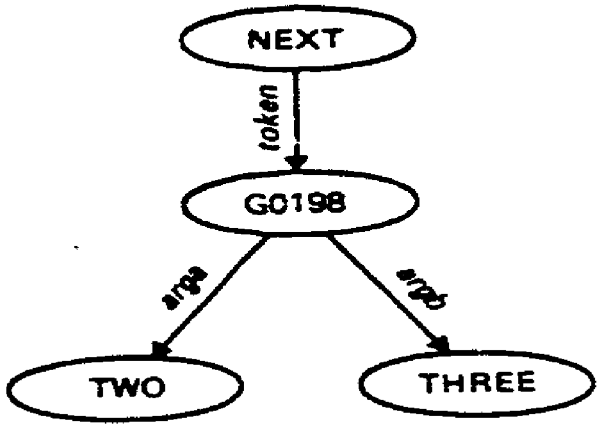

Figure 28. Data structure representing the NEXT relation between TWO and THREE.

Similarly, the last two schemata of the condition pattern mention a single variable name, V23. Together, these two schemata allow the model to find an element in its data base that is a NUMERON but is not yet a member of CLIST. (OBJCAT V23 NUMERON) specifies that the node assigned as the value of V23 must be a NUMERON. The following diagram (Figure 29) shows all the nodes that qualify:

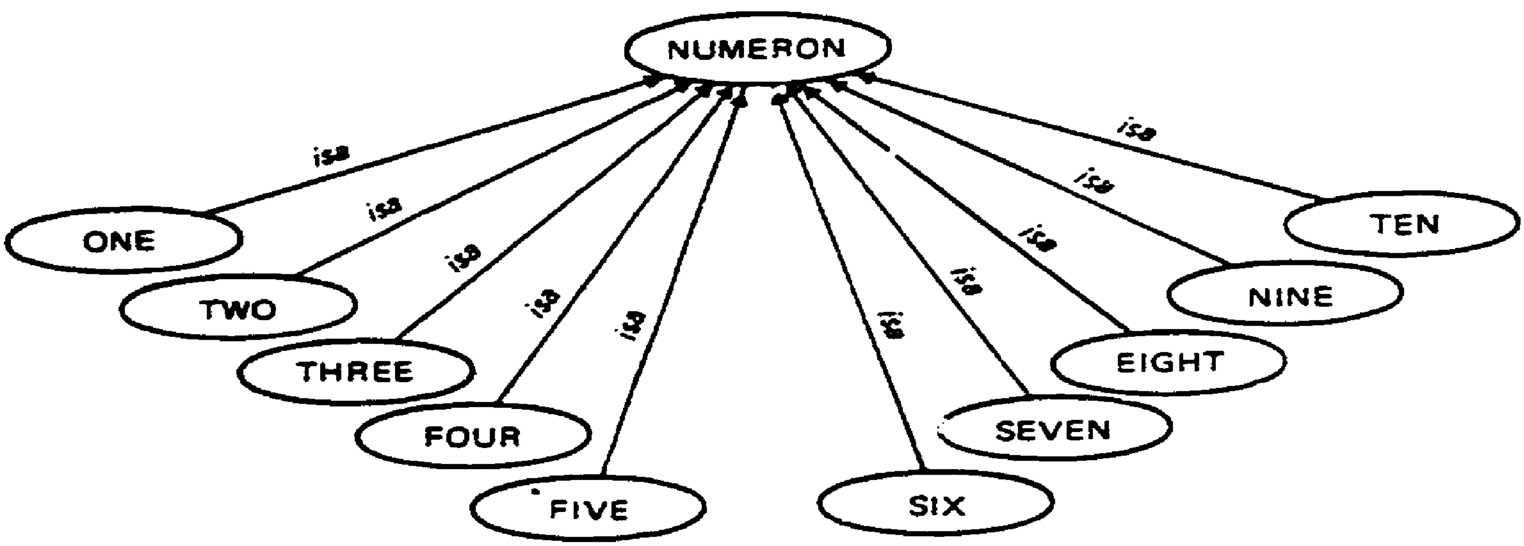

Figure 29. Data structure showing the members of COUNTER's numeron category.

However, (ABSENCE OBJPROP V23 FOLLOWED) further specifies that the node assigned as the value of V23 cannot have a hasprop link to FOLLOWED. This additional requirement eliminates ONE and TWO as possible candidates for the value of V23 since these three nodes were linked to FOLLOWED when they became members of CLIST. Although the node THREE does satisfy both requirements, it has already been assigned as the value of V22 and cannot be assigned again in the same condition.

Considering all four schemata together, then, the condition of the above production says, "Find the end of the current CLIST and then find an element that is known to be a numeron but is not already in the ordered CLIST." Assuming that this condition can be met, the action of the production is to make this unordered numeron part of CLIST, link it with THREE through a relation that is a token of NEXT, and link THREE through hasprop to FOLLOWED. This results in the expanded version of CLIST shown in Figure 30.

Notice that the FOUR was the numeron chosen to be bound as the value of V23. As mentioned earlier, numerons FOUR to TEN, inclusive, were all possible candidates. FOUR was bound simply because the pattern matcher found this node first when evaluating the elements of NUMERON. A more elegant version of ADDTAG would perhaps assign varying "strengths" to the as-yet-unordered numerons; a numeron's strength would then determine its probability of being bound to V23 when ADDTAG was active (as opposed to leaving it up to the built-in "whims" of the pattern matcher). The relative strengths of numerons could conceivably be a function of such things as watching Sesame Street and seeing SIX, or hearing a poem with TEN in it.

A general point about the system is that most link-types have inverses, and the system is indifferent to which direction is specified in a schema. The inverse of isa is memb, the inverse of hasprop is isprop, the inverse of ispart is haspart, the inverse of token is type. Arga's inverse is argal, and argb's inverse is argbl. This is not an issue of any substantive importance, but there will be times when the inverse

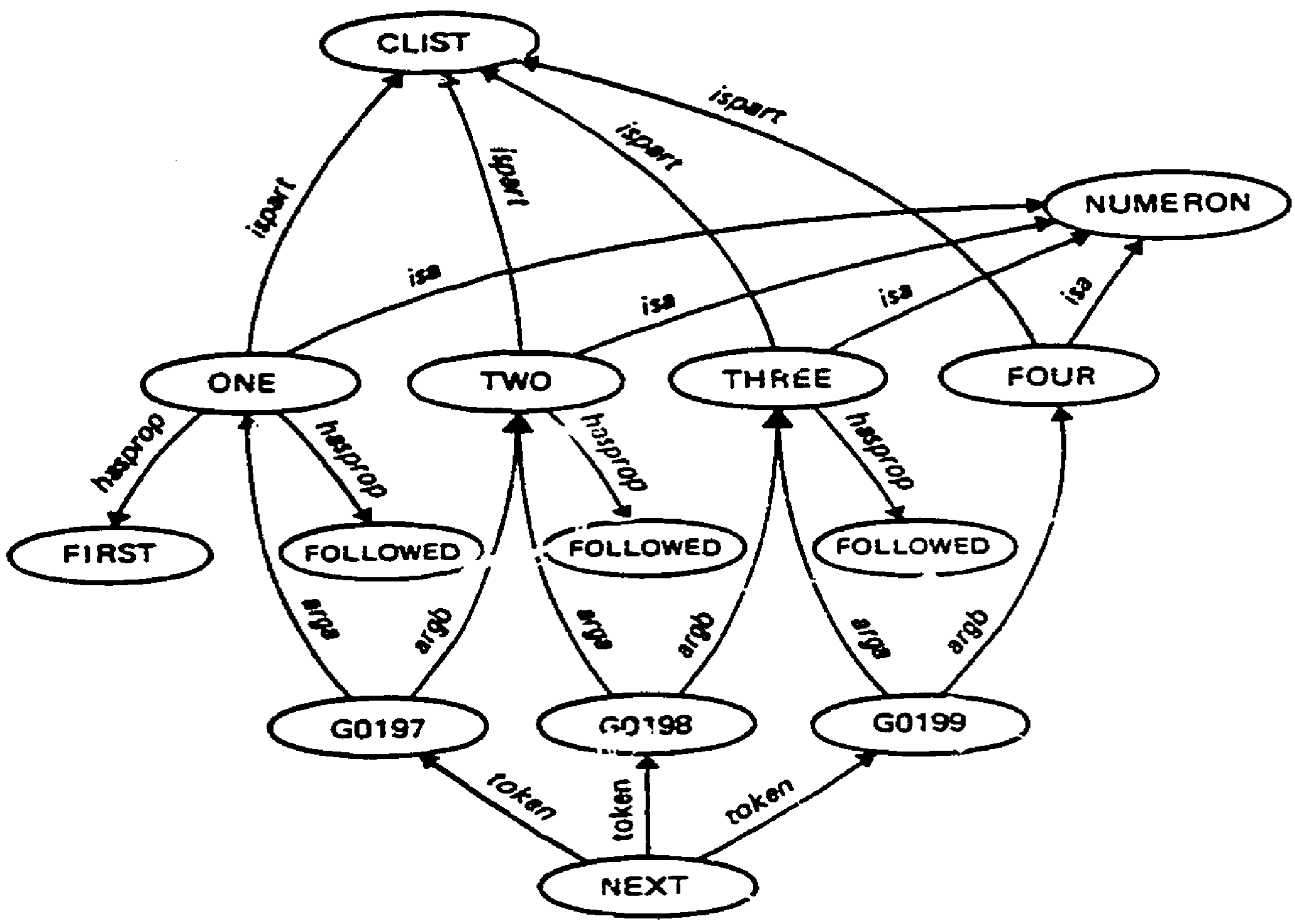

Figure 30. COUNTER'S extended T.

relations will be specified, and it is confusing if "A isa B" is not recognized as identical to "B memb A," and so on.

Section 3: A Detailed Look at How the Model Counts

This section follows COUNTER in detail as it counts a group of four objects. It describes the components of the counting procedure as a series of actual cycles through COUNTER's set of productions. The output from these cycles is shown in Appendix C and will be referred to throughout the discussion.

Some Preliminaries

Before COUNTER can begin to count, it must be "started up" as shown at the beginning of Appendix C. STARTUP is a LISP function that informs the ACTP system running the model of the variable names, constants, links, and so on, that will be used in a particular set of productions. Without this information, the system cannot distinguish variables from constants, for example, and therefore cannot operate. STARTUP also builds COUNTER's ordered list of numerons, CLIST, into the data base. Initially CLIST consists of only the numerons ONE, TWO, and THREE (see Figure 8). During the first few cycles, the function ADDTAG will be used to extend the list to six numerons. A listing of the STARTUP relevant to the current version of the counting model is given in Appendix D along with a discussion of the main STARTUP functions.

(Incidentally, a "*" in the printout indicates that anything following it on the same line was typed in from the terminal.)

Still referring to Appendix C, the next line after (STARTUP) to be typed in from the terminal is (GENSET OBJECTS). GENSET is another LISP function that sets up an initial data structure; something like GENSET is always needed to define a model's initial knowledge state. In the counting model, GENSET sets up the representation of the visual information in the display o.

objects. The reason this was not done in STARTUP is because GENSET takes different arguments depending on the number of objects to be counted, and their location; its arguments could not be easily changed in STARTUP. GENSET takes as its arguments the names of objects and the values of their respective X- and Y-coordinates. On the basis of this information, it computes the difference between the X- (and Y-) coordinates of adjacent object pairs as well as the slope defined by those pairs. The output of this function is the data structure representing the spatial information the model uses to scan and count the objects. In the example shown in Appendix 2, the variable OBJECTS had as its value the list:

```
(OBJECTS (A 7,0 0.0)  (B 8.0 0.20000000)  (C 9.5 0.0)
(D 10.0 0.0)
```

Operating on this list, GENSET generated the data structure discussed in the last section and shown in Figure 9. Some of the nodes in this structure were printed out in response to the YES reply to SHOW-STRUCTURE?

TRACE is a LISP function that takes the names of other LISP functions as its arguments, e.g., PREQPLAN and PREQCHK. These other functions are then "traced" whenever they are called during a cycle. A trace is a detailed report of a function's execution within a program and is primarily used as a debugging device.

CYCLE tells the system to begin the process of cycling through its set of productions. A YES response to THINK-ALOUD? causes the names of all currently bound variables and control nodes to be printed out at the beginning of each cycle. If a NO response is given, only the number of the current cycle is typed each time, except when the action of a production includes the PRINT function.

(SETQ DEBUG NIL) is a LISP signal telling the system not to print out debugging information during the cycles that follow it. (This could just as well have been typed in before starting CYCLE.) Similarly, if at any time debugging information is needed, (SETQ DEBUG T) can be typed in, causing this information to be printed out on subsequent cycles. In fact, LISP signals can be given as input to any cycle, after which ACTP will execute the functions that are specified.

The output from each cycle includes a list of the active nodes and a list of the variables that have bound values. So when the system now returns:

```
NIL
NIL
1
>>> *
```

NIL indicates that at the beginning of the first cycle there were no active control nodes and no currently bound variables; therefore, no productions were tested and no action taken.

As already mentioned, LISP signals can be given as input to a cycle. Also permitted are inputs that begin with any one of the words on ACTP's list of titles that is defined in STARTUP (see Appendix D). These inputs then become active during that cycle in the same sense that constants mentioned in the actions of productions become active for the next cycle. For example, in Appendix C, the next input from the terminal is (ADDTAG) which causes this control node to become active as shown at the beginning of Cycle 2:

```
(ADDTAG)
NIL
2
```

ADDTAG is the control node of the production relevant to extending COUNTER's ordered list of numerons (discussed under <u>Schemata</u>). The NIL under the (ADDTAG) says that there are still no bound variables. ADDTAG is active on this cycle and since it matches the control node of Production 1, the action of the production is taken. Notice that whenever an action is taken, the corresponding action patterns are printed out at the end of the cycle along with a temporary data structure identifying any permanent new additions to the data base. These structure descriptions start with a symbol beginning with ST and give a list of the nodes included in the structure. For example, at the end of Cycle 2 in the appendix, the action patterns of Production 1 are printed out and ST0200 identifies the new additions to the data base:

```
2
(((PARTOBJ  CLIST)  (ASYMREL  NEXT  VX2  V22  V23)
(OBJPROP  V22  FOLLOWED)))
(ST0200  (FOUR CLIST NEXT G0199 THREE FOLLOWED))
```

This says that at the end of Cycle 2, FOUR (assigned as the value of V23) has become a part of CLIST: the relational node NEXT has been assigned a new token node (G0199) that takes THREE and FOUR as its ordered arguments; and THREE is now connected through a hasprop link to FOLLOWED. The data structure containing these new additions to the ordered list of counting names is shown in Figure 30. FIVE and SIX are added to CLIST in the same way by activating (ADDTAG) for Cycles 3 and 4.

On Cycle 5, COUNTER prepeares to count by setting the goal of finding the cardinality of the set of objects. (HOWMANY) is typed in from the terminal, causing this control node to be active on this cycle. (HOWMANY is one of the words on ACTP's list of titles.) Production 4 is the only production with this control node. It is tested during Cycle 5 and there are no condition patterns to be matched so the action is taken. This causes a new goal, represented by the token node G0224, to be added to the top of the goal stack and the structure in Figure 19 to be added to the data base. This structure represents a goal for finding the numerical size of a group of objects (i.e., its cardinality).

COUNTER is now ready to count. This time NIL is typed in from the terminal which indicates that no input is provided for the next cycle. However, part of the action of P4 was to activate the constant GSEE so there is an active control node at the beginning of Cycle 6.

COUNTER Counting

The following discussion of the counting procedure skips over the initial scanning and perceptual grouping of the objects (Cycles 6-11) and begins with Cycle 12. By this time, COUNTER has already scanned the array A B C D and has formed a perceptual grouping of the three leftmost objects (i.e., A, B, and C). Then using information about which end of the array it found and the direction of the path between the end object and another object in the group, COUNTER formed the following relational structure (Figure 31) which indicates that the array is approximately

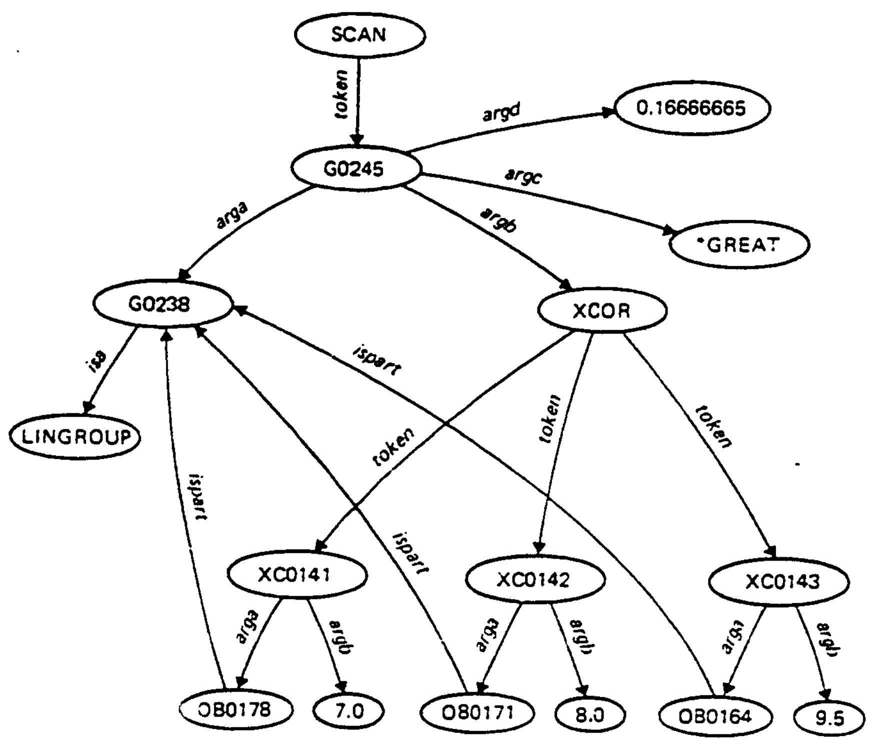

Figure 31. Perceptual grouping of objects A, B, and C.

horizontal, scanning is to occur in a left-to-right direction, and the slope of the path that will be used to extend the group if it becomes necessary. A perceptual linear grouping, designated by the node G0238, has been formed and consists of three objects denoted OB0178, OB0171, and OB0164 (objects A, B, and C, respectively). A structure involving a relation called SCAN has also been formed. The arguments of the scan relation are the group of objects (G0238), the dimension of scanning (XCOR for horizontal, YCOR for vertical), the direction of scanning (*GREAT for left-to-right or bottom-up, *LESS for the opposite directions), and the slope defined by the first two objects in the array (0.16666665).

<u>Cycle 12</u>. During this cycle, COUNTER first focuses on the perceptual grouping of objects it has just formed and identifies the object at the left end of this grouping as the first object to be counted. This object (i.e., object A) is then tagged with the property of being the current bound of the set, an operation equivalent to placing a tag on each object as it is counted.

At the beginning of Cycle 12, GCOMPACT2 is active. Following GCOMPACT2, on the same line, are the bindings of the variables that were held over from the previous cycle. The next line pairs these bindings with their respective variables:

```
(*GREAT V12 XCOR V11 OB0178 V1 7.0 N1 OB0171 V2
8.0 N2 OB0164 V3 9.5 N3)
```

Thus, *GREAT is currently assigned as the value of V12, XCOR is assigned to V11, OB0178 is assigned to V1, 7.0 to N1, and so on.

P22 is the only production whose control node is active on this cycle, and so it is the only production that gets tested. The condition of this production tries to match the following pattern (Figure 32) to the data base:

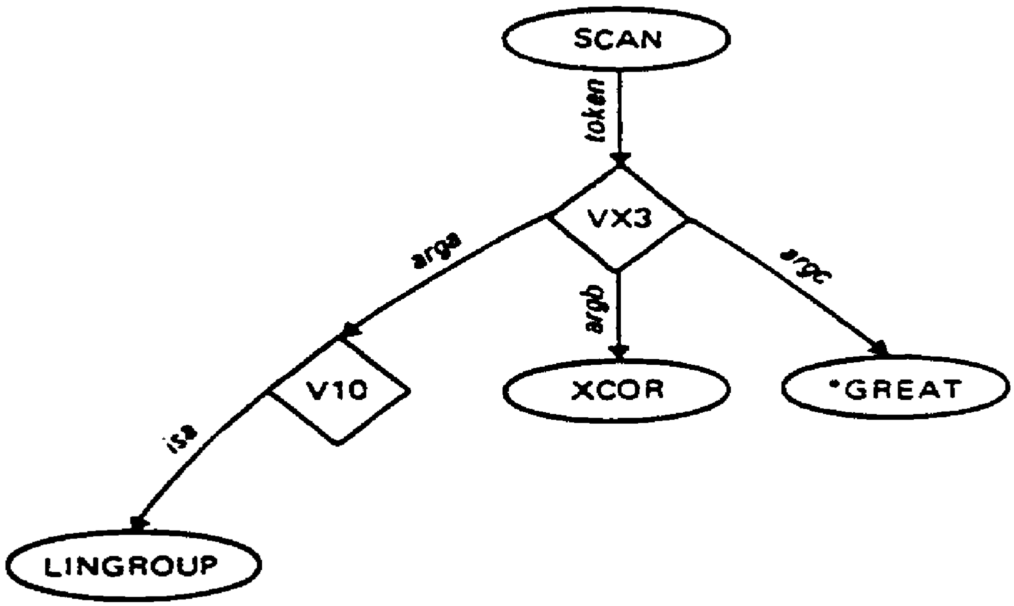

Figure 32. Pattern for retrieving visual information about the group of objects.

The pattern matches to the data structure shown in Figure 31, and the free variables V10 and VX3 are bound to G0238 and G0245, respectively. Since the condition matches, the action is performed. This action assigns V1 (object A) the property BOUND. (Note: This refers to the lower bound of the set of uncounted objects and is not to be confused with the bound value of a variable.) The second action component, (UNBASE), removes all arguments from the temporary data structure, BASE. Finally, the constant GNEXTN (NEXT Number) is activated because it is the control node of the production relevant to the next step in the model's counting procedure, and the bindings of V10, V11, V12, and V1 are held for the next cycle. The next step is to get a numeron from the ordered CLIST.

Cycle 13. On Cycle 13, COUNTER retrieves a numeron from its ordered list of counting names so it can count the first object in the array. In this case the successor function is not yet applicable since getting the "next" numeron from CLIST requires that there already be a current numeron. This means that COUNTER must begin with the first numeron; this involves simply identifying the numeron in CLIST that has the property first.

At the beginning of this cycle, GNEXTN is active and G0238, XCOR, *GREAT, and OB0178 are already assigned as the values of V10, V11, V12, and V1, respectively. P45 and P46 are both candidates for testing since they both have the active control node. P45 is the production that retrieves the next numeron in CLIST. It comes before P46 in the model's set of productions and so it is tested first. The first part of its condition requires finding the following ORDASSIGN pattern (Figure 33) in the data base:

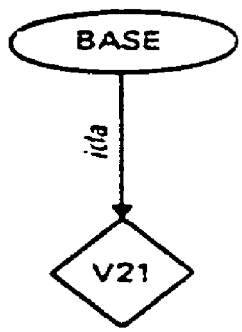

Figure 33. Pattern for retrieving the current problem BASE.

Throughout the counting procedure, V21 is the variable that gets bound to the most recently used numeron. However, since there are no used numerons on this cycle, this pattern fails to match and P45's action is not taken.

P46 is tested next. Its function is to retrieve the first numeron from CLIST. The condition requires finding the pattern shown in Figure 34.

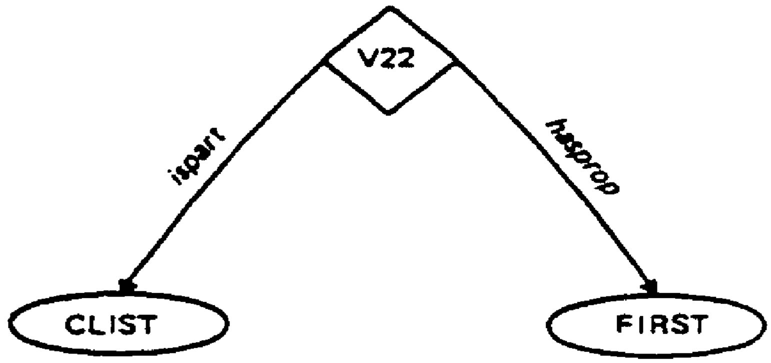

Figure 34. Pattern for retrieving the first numeron from CLIST.

This pattern is matched to the CLIST structure in the data base (see Figure 30) and V22 is bound to ONE. The action of P46 activates the control node GCOUNT and holds the bindings of V1, V22, V10, V11, and V12 for the next cycle.

Cycle 14. This cycle involves COUNTER counting the object A with the numeron ONE. P55 is the only production whose control node is active. There are no condition patterns to be tested and so the action is performed. The values of V1 and V22 are printed out at the terminal.

***** (OB0178 ONE)

This is intended to represent COUNTER counting object OB0178 with the numeron ONE. The rest of the action removes the current BASE (on this particular cycle there is none to remove) and reassigns the numeron that was just used as the current BASE. Figure 35 shows ONE as the current problem BASE, signifying that ONE is the most recently used numeron.

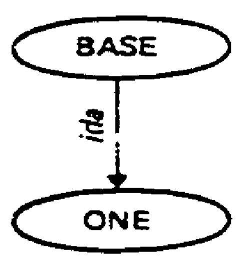

Figure 35. Data structure specifying one as the current problem BASE.

GNEXTOB (NEXT OBject) is then activated and V10, V11, and V12 hold their bindings for the next cycle.

Cycle 15. Having counted object A, COUNTER now scans its initial perceptual grouping, LINGROUP, to determine if there are objects that have not yet been assigned the property BOUND, (i.e., have not yet been counted).

P31 is the relevant production here. In order for its condition to be true, there must exist a node in the data base that is a member of the current LINGROUP (LINGROUP is the value of V10) but does not have the property BOUND. The current data base contains the following information (Figure 36) on LINGROUP which is represented by the token node G0238:

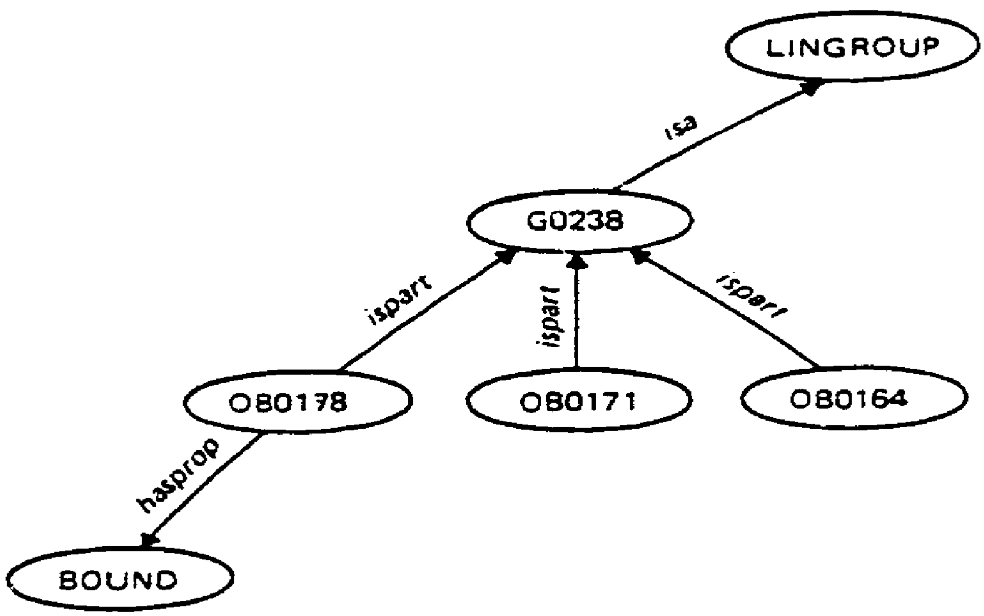

Figure 36. Data structure containing information about the set of objects.

There are two candidates for a match: OB0178 and OB0164 which represent the objects B and C, respectively. OB0164 happens to be chosen and is assigned as the value of V1. Before this object can be counted, however, COUNTER must check to see if there are any other objects between OB0164 and the most recently counted object (OB0178). So GCHBETWEEN (CHeck BETWEEN) is activated and the mentioned variables keep their values for the next cycle.

Cycle 16. Before counting the object it just found, COUNTER checks to make sure that there are no other untagged (i.e., uncounted) objects between this object (object OB0164) and the last object it counted (object OB0178). Since the direction of counting in the example is from left to right, this is equivalent to checking for the existence of another uncounted object whose X-coordinate is less than OB0164's X-coordinate. If there does exist an object with a smaller X-coordinate, this object replaces OB0164 as COUNTER's current focus and the checking procedure is repeated until the uncounted object with the smallest X-coordinate is found. Thus, in the same way that applying the successor function to CLIST ensures that the retrieved numeron is the one immediately next to the most recently used numeron, this perceptual checking procedure ensures that the object selected for counting is the one closest to the object that is currently the upper bound of the array.

At the beginning of this cycle, GCHBETWEEN is active and so P41 is tested. The condition requires first finding the following pattern (Figure 37) in the data base.

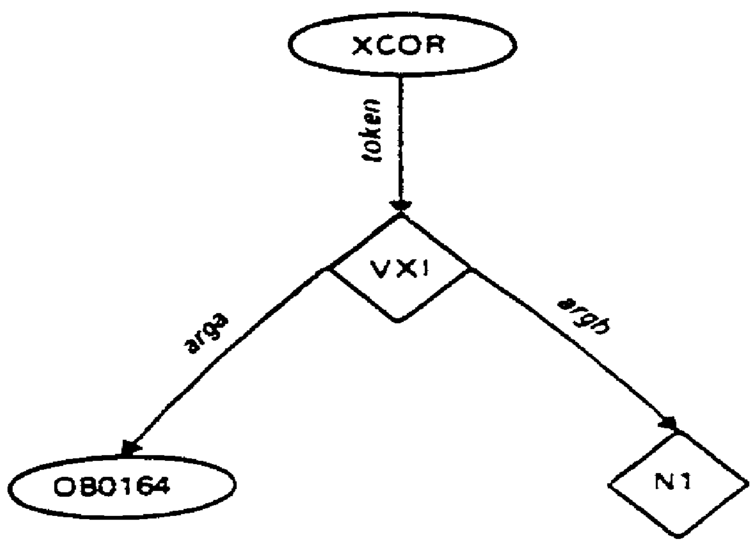

Figure 37. Pattern for retrieving the value of object C's X-coordinate.

Next a second pattern must be found, constrained so that the node assigned as the value of V2 does not have the property BOUND, as shown in Figure 38.

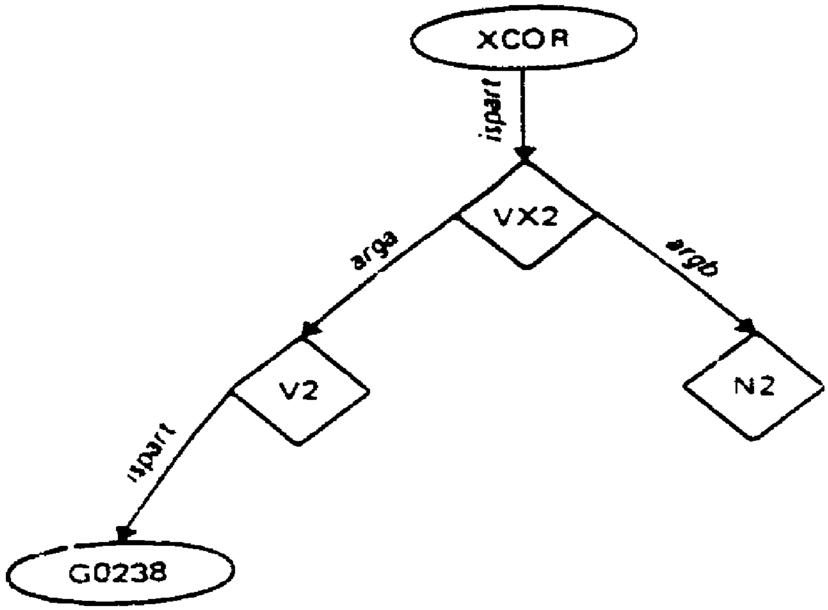

Figure 38. Pattern for retrieving an object in LINGROUP together with its X-coordinate

The first pattern is matched to the data base and XC0143 and 9.5 are assigned as the values of VX1 and N1. These nodes cannot be assigned again to different variables during the same cycle, so the second pattern must be matched to a different set of nodes. Again the match is successful and SC0142, OB0171, and 8.0 are assigned as the values of VX2, V2, and N2, respectively. This means that COUNTER has identified another uncounted object in LINGROUP: it does not yet know, however, if this object is closer than OB0164 to the last counted object. So the next condition test involves a comparison of N1 and N2 to determine if the relation bound to V12 is true with respect to them. In this case, V12 has the value *GREAT and, since 9.5 (OB0164's X-coordinate) is greater than 8.0 (OB1071's X-coordinate), the relation is indeed true. This indicates that OB0171 is closer than OB0164 to the last counted object (OB0178), casuing COUNTER to change its current focus. However, before it can go ahead and count OB0171, the model must repeat the checking procedure to determine if there exists yet another object closer than OB0171 to OB0178. Therefore, the last condition component and the entire action of the current production prepare COUNTER to refocus its checking procedure on OB0171.

The refocusing process is somewhat confusing here because the current production, P41, is also the production that will be used to check OB0171. This requires that before checking can proceed, OB0171 must be assigned as the value of V1, the variable that P41 considers to be bound to COUNTER's current focus. Without this reassignment, the next time P41 is executed V1 would still be bound to OB0164 and the model would be caught in an infinite loop. OB0171 cannot be assigned as the value of V1 on this cycle, however, because OB0171 is already assigned to V2, and V1 is already bound to OB0164. The strategy here is therefore the following: (1) remove the bindings of V1 and N1 by failing to mention V1 and N1 with the other variables on the action side of the production; (2) hold OB0171 and its X-coordinate (8.0) in temporary memory until they can be assigned on the next cycle to the then free variables V1 and N1, respectively. The second part of this strategy requires the use of the ORDASSIGN schema. Simply mentioning the variable bound to OB0171 in the action of the production is not appropriate here because this would mean that OB0171 would already be assigned on the next cycle and could therefore still not be assigned to V1, even though V1 would then be a free variable. The ORDASSIGN schema, on the other hand, allows nodes to be remembered without assigning them to variables.

COUNTER has already used the ORDASSIGN schema to store the node representing the last numeron used in counting and therefore already has the following pattern (Figure 39) in memory:

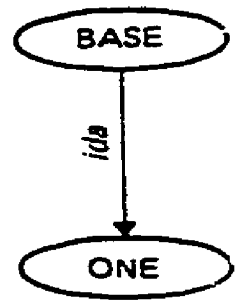

Figure 39. Data structure specifying ONE as the current problem BASE.

Creating a new problem BASE requires that this existing BASE structure first be removed from memory. Before this is done, however, COUNTER needs some way of remembering the last numeron since it will need this information when it comes time to retrieve the next numeron in CLIST. The last condition component serves this function. The pattern created by (ORDASSIGN BASE V21) is matched to the above structure in the data base with the result that V32 is now bound to ONE. Thus, even after the special function UNBASE removes the existing BASE structure, ONE is temporarily remembered as a variable binding. Once the old BASE is removed, ((ORDASSIGN BASE V21 V2 N2)) creates a structure that becomes the new problem BASE (Figure 40):

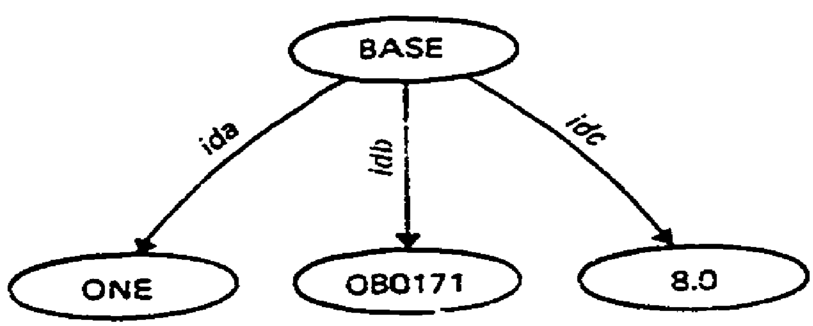

Figure 40. Data structure specifying the current problem BASE.

This allows OB0171 and 8.0 to be remembered for the next cycle without being held as specific variable bindings. Finally, V1 is not mentioned in the action of the production and so it becomes a free variable for the next cycle.

Cycle 17. Cycle 17 simply serves to assign OB0178 and 8.0 as the values of V1 and N1, respectively. This permits COUNTER to refocus its checking procedure to determine if OB0171 is, in fact, the object closest to the last counted object.

GCHB2 (CHeck Between) is active, causing P44 to be tested. The condition pattern matches to the BASE structure created during the previous cycle. Notice, however, that two of the arguments in the BASE structure have been assigned to new variables. During the last cycle, OB0171 and 8.0 were assigned to new variables. During the last cycle, OB0171 and 8.0 were assigned to V2 and N2; the action taken during this cycle reassigns them to the variables V1 and N1 and holds these new bindings for the next cycle. In this way, V1 is now bound with COUNTER's most recent candidate for the object closest to the last object counted. So together P41 and P44 allow COUNTER to scan the uncounted objects in the direction of the last object counted until it finally finds the one that is the closest.

Note--in a slightly newer version of ACTP rebinding variables can be accomplished within a single production with a special function called REBIND. The action of P41 could be rewritten as:

(REBIND V1 V2) (REBIND N1 N2) GCHBETWEEN V10 V11 V12

V1 is rebound with the value of V2 and N1 is rebound with the value of N2. V21 is not mentioned in the action because it is to keep its current binding, which it will since it is the current base of the problem; for this reason (UNBASE) is omitted from the action as well. Written this way, P41 also accomplishes what use to require P44. P44 is therefore no longer necessary and P41 can simply call itself.

Cycle 18. During this cycle, the same checking procedure that was applied to OB0164 is applied to OB0171. This time there exist no uncounted objects between the current focus of the checking procedure and the last counted object. OB0171 is then identified as the next uncounted object in the array. COUNTER then prepares to count OB0171 by activating the control node of the production relevant to retrieving the next numeron from CLIST.

GCHBETWEEN is active again but this time V1 is bound with OB0171 instead of OB0164, changing the assignment of nodes to condition patterns. Now the first condition pattern matches to the structure in the data base containing OB0171 and its X-coordinate, as shown in Figure 41.

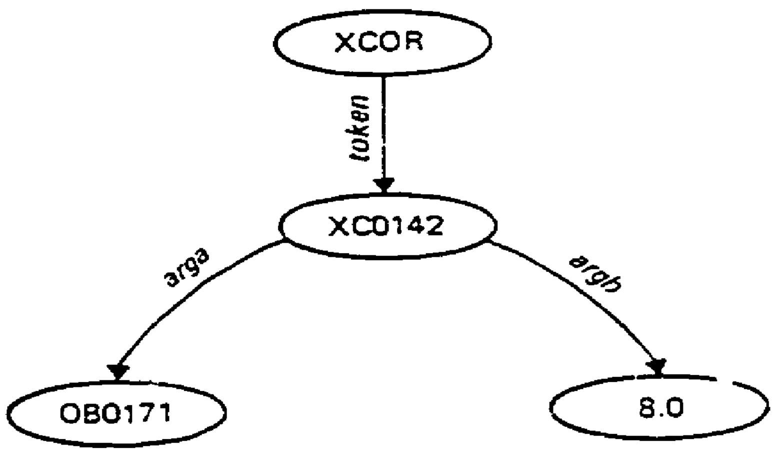

Figure 41. Data structure specifying an X-coordinate relation between object B and 8.0.

Since the only other uncounted member of LINGROUP is OB0164, it gets assigned as the value of V2 when the next part of the condition pattern is matched. Ths problem arises when the values of N1 and N2 are compared. N1 has the value of 8.0 and is therefore not greater than the value of N2 which is 9.5 and the condition fails to match.

P42 is tested next and since there are no condition patterns to be matched, the action is taken. This results in OB0171 being linked to the property BOUND, indicating that it is the next object to be counted. GNEXT is activated again and variables V10, V11, V12, and V1 keep their bindings for the next cycle.

Cycle 19. COUNTER has just found another object to count and this requires retrieving another numeron from CLIST. This time, COUNTER has a last used numeron in memory and so it can apply a simple successor function to CLIST to retrieve the next unused numeron.

GNEXTN is active and P45 is tested. The condition requires finding the following pattern (Figure 42) in the data base:

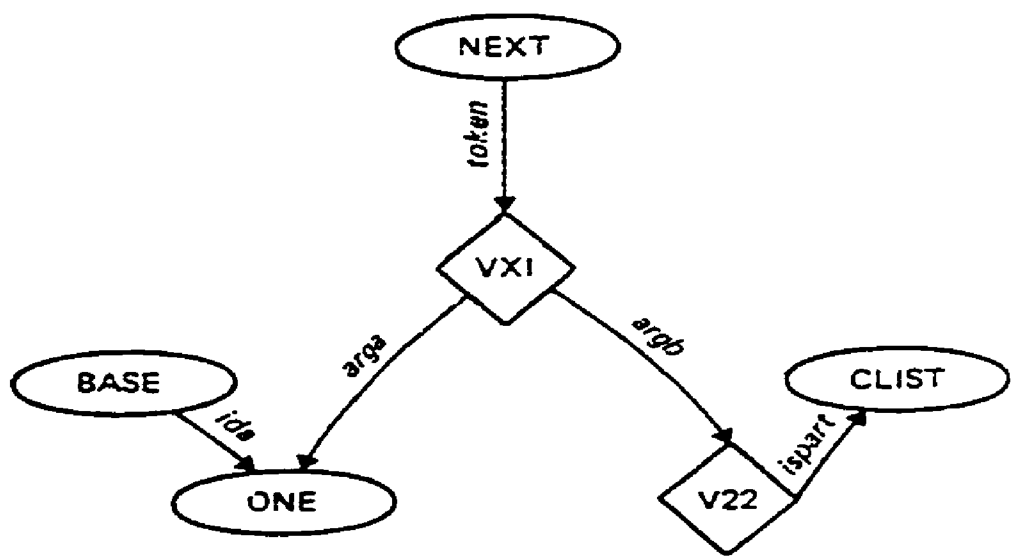

Figure 42. Pattern for retrieving the numeron in CLIST that is next to ONE.

A match is found (see Figure 30) with the result that TWO and G0197 get assigned as the values of V22 and VX1, respectively. Thus TWO has been identified as the member of CLIST that immediately follows the last used numeron ONE. The results is that GCOUNT is activated and COUNTER prepares to count OB0171.

Cycle 20. Having just retrieved the next unused numeron from its ordered list of counting names, COUNTER goes ahead and counts OB0171 (object B) as TWO.

Since GCOUNT is active, P55 is the only production to be tested on this cycle. There are not condition patterns to be matched so the action of printing out the values of V1 and V22 is taken:

***** (OB0171 TWO)

This represents COUNTER counting the second object in the array, OB0171, with the numeron TWO.

The next piece of action removes the current BASE (which contains the node ONE) and reassigns it with the value of V22, which is TWO in this case. In this way, COUNTER can "remember" that TWO is now the last numeron it used. This information is retrieved when it comes time to get the next counting word in CLIST.

GNEXTOB is activated and V10, V11, and V12 keep their bindings.

Cycle 21. COUNTER is again in search of the next object to count. It first scans the perceptual grouping, LINGROUP, and identifies OB0164 (object C) as a possible candidate.

GNEXTOB is active and so P31 is tested. This production determines if there are any more uncounted objects in the current LINGROUP by trying to match the pattern shown in Figure 43

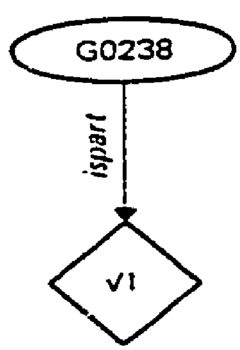

Figure 43. Pattern for retrieving an object from LINGROUP.

under the additional constraint that V1 cannot be linked to the property BOUND. (G0238 is the current member of LINGROUP.) The pattern matches and OB0164 is assigned as the value of V1. OB0164 was the only choice possible since OB0178 and OB0171, although also members of LINGROUP, both have the property BOUND. The only action taken is to activate control node GCHBETWEEN and keep the mentioned variables bound with their values.

Cycle 22. Before counting OB0164, COUNTER scans the array for any other uncounted objects in LINGROUP that are between OB0164 and the last object it counted. Since the only other objects in LINGROUP (OB0178 and OB0171) have already been counted, COUNTER identifies OB0164 as the next object to be counted and prepares to get the next unused numeron from CLIST.

GCHBETWEEN is active. P41 is tested and the first condition pattern is matched, assigning XC0143 as the value of VX1 and 9.5 as the value of N1. The next part of the condition tests for the pattern (shown in Figure 44) where the node chosen as the value of V2 cannot be linked to the property

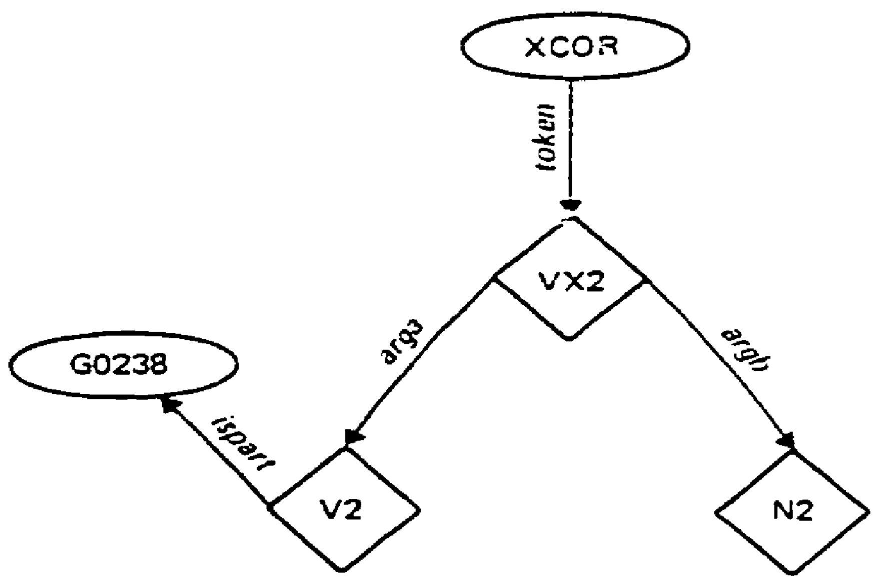

Figure 44. Pattern for retrieving an object in LINGROUP together with its X-coordinate.

BOUND. Since the only two candidates for V2 (i.e., OB0178 and OB0171) also have the property of being BOUND, the condition fails and P42 is tested. This production has no pattern specifications to be matched and so its action is taken: OB0164 is assigned the property BOUND; GNEXTN is activated for the next cycle; and the mentioned variables keep their current bindings.

Cycle 23. Again COUNTER applies a simple successor function to CLIST to retrieve the next unused numeron.

The active control node GNEXTN causes P45 to be tested on this cycle. The first part of the condition pattern causes COUNTER to recall its current problem BASE (which is TWO in this case) and binds it as the value of V1. The second part of the condition pattern tests for the node that comes after V1 in CLIST, shown in Figure 45. The pattern matches and THREE is assigned as the value of V22. The action of this production activates control node GCOUNT and holds the bindings of the mentioned variables.

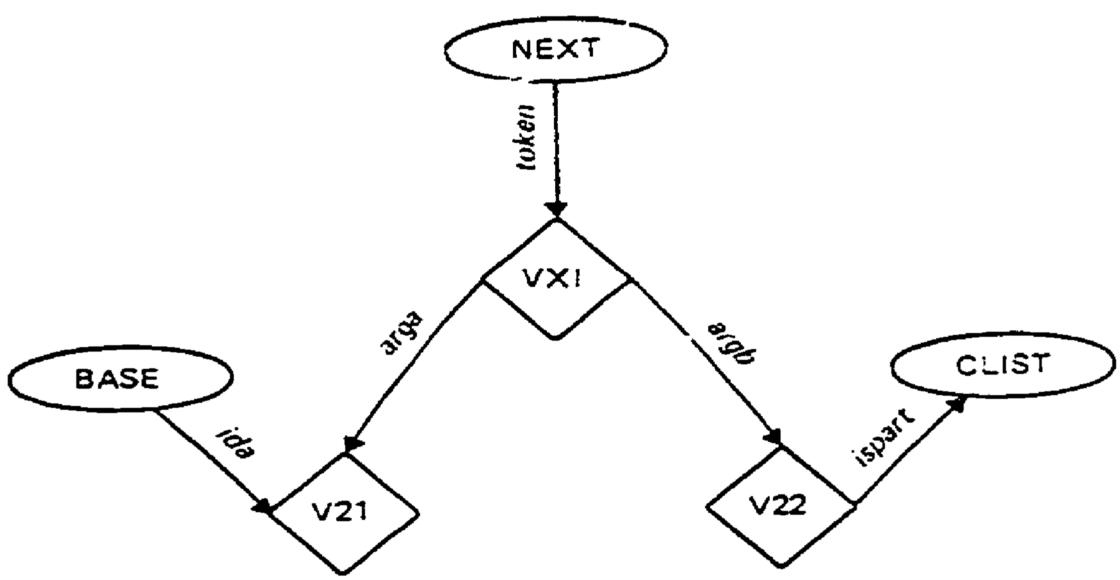

Figure 45. Pattern for retrieving a next numeron from CLIST.

Cycle 24. On this cycle, COUNTER counts OB0164 (object C) as THREE.

GCOUNT is active and P55 gets tested. Since there is no condition pattern, the action of printing out the current values of V1 and V22 is taken:

***** (OB0164 THREE)

The rest of the action removes the current BASE and links it to the value of V22. GNEXTOB is then activated and V10, V11, and V12 keep their bindings for the next cycle.

Cycle 25. This time when COUNTER tries to find another object to count, it discovers that there are no more uncounted objects in LINGROUP. It therefore prepares to execute a production that extends the perceptual group to include new objects.

Since GNEXTOB is active, P31 is tested. V10 is currently bound to G0238 which is the symbol for LINGROUP, the perceptual subset formed during the initial scanning of the array. V1 is a free variable and the condition of this production requires that it be matched to a node in the data base that has an ispart link to G0238 but at the same time lacks a hasprop link to BOUND. The only members of LINGROUP are OB0178, OB0171, and OB0164; since all these nodes have the property BOUND, the condition fails to match.

P32 also has GNEXTOB as a control node and it is tested next. There are no condition patterns to be matched so the action of this production is taken, causing GEXTEND to be active on the next cycle.

Cycle 26. On this cycle COUNTER extends the perceptual grouping it just finished counting to include other objects in the array that have not yet been counted. In the current version of the model, this is a simple procedure that extends the group a single object at a time, proceeding along the same line as the scanning path of the current perceptual group. Extending the group, then, requires first retrieving the relevant perceptual information and so the first pattern that must be matched to the data base is shown in Figure 46 (see P33):

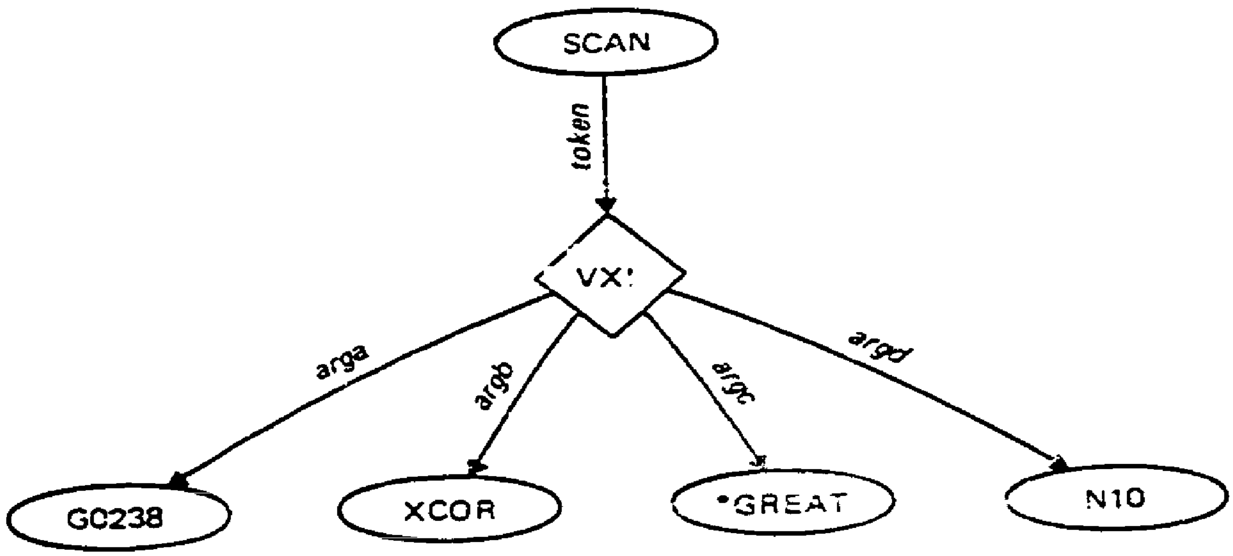

Figure 46. Pattern for retrieving visual information about the objects.

It matches to the data structure shown in Figure 7, binding G0145 to VX 1 and 0.16666665, the slope of the scanning path, to N10.

Next COUNTER checks to see if there are any objects in the array that are not yet part of LINGROUP. It tries to match the pattern (Figure 47) with the restriction that the node assigned as the value of V1 cannot be linked through <u>ispart</u> to G0238 (the node symbolizing LINGROUP). In this case, OB0164 is the only node that qualifies and so it is bound to V1, its X-coordinate is bound to N1, and the relational node XC is assigned to VX2.

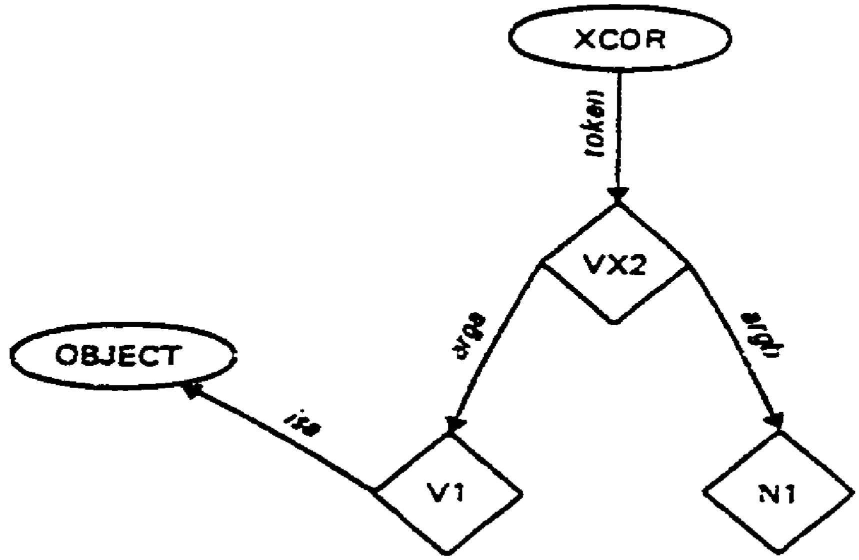

Figure 47. Pattern for retrieving an object and corresponding X-coo· ·nate.

Now COUNTER determines the slope defined by this new object and one of the counted objects by matching the pattern shown in Figure 48:

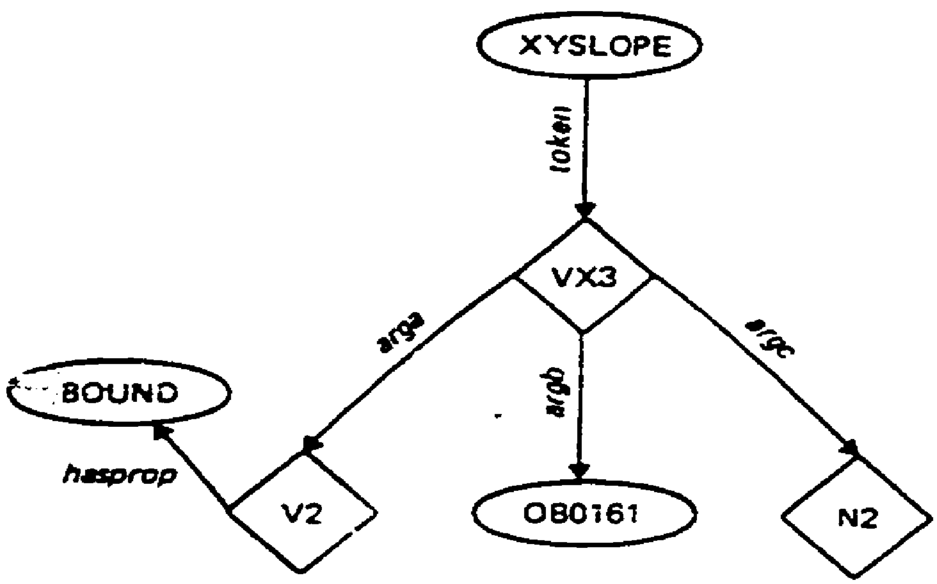

Figure 48. Pattern for retrieving the slope defined by object D and an adjacent object.

The match is successful and OB0164 is bound as the value of V2, and 0.0 and YXCL0170 are found as the values of N2 and VX3, respectively.

The next condition requirement is that the values of N1 and N2 must be approximately equal (i.e., the new object must form an approximately linear array with the other objects). This requirement is also satisfied, making the entire condition of this production true and so the action is taken.

Now COUNTER must check if there are any other objects closer than OB0164 to the most recently counted object. So GCHCLOSER is activated and the mentioned variables are kept bound with their values.

Cycle 27. COUNTER must make sure that the new object is the object closest to the already counted group (i.e., closest to LINGROUP). OB0161 (object D) is the only remaining object and therefore has to be the closest. COUNTER identifies this object as the next object to count.

GCHCLOSED is active so P39 is tested first on this cycle and tries to match the pattern constrained so that V3 cannot be a part of LINGROUP (Figure 49):

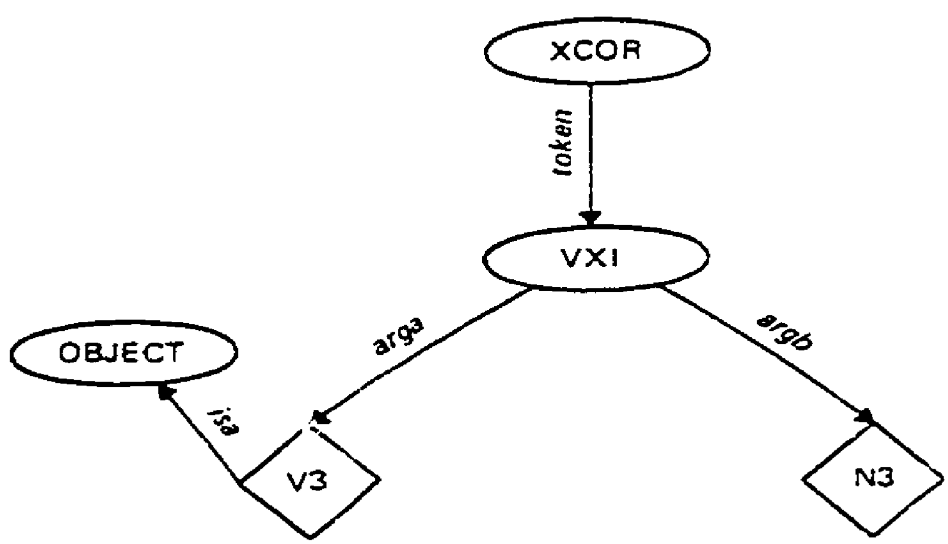

Figure 49. Pattern for retrieving two nodes linked through an XCOR relation.

The only possible candidate for V3 is OB0161 but since it is still assigned as the value of V1 from the last cycle, the condition fails to match (i.e., there are no other uncounted objects closer to the already counted group than OB0161).

P40 is tested next. There are no condition patterns to be matched so the action is taken. The action adds the following structure (Figure 50) to the data base making OB0161 part of LINGROUP and assigning it the property BOUND.

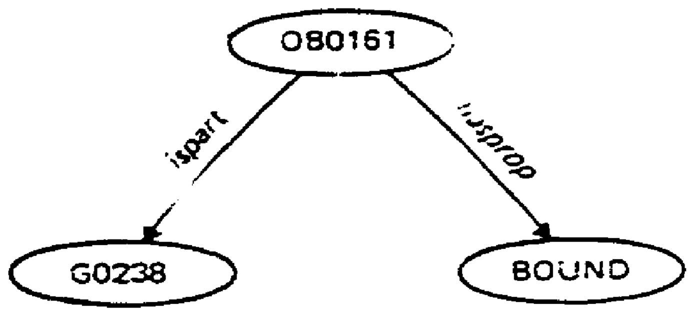

Figure 50. Data structure identifying D as a part of LINGROUP.

Cycle 28. Once again COUNTER retrieves the next numeron from CLIST. GNEXTN is active. P45 is tested and the following condition pattern (Figure 51) is successfully matched and FOUR is bound as the value of V22.

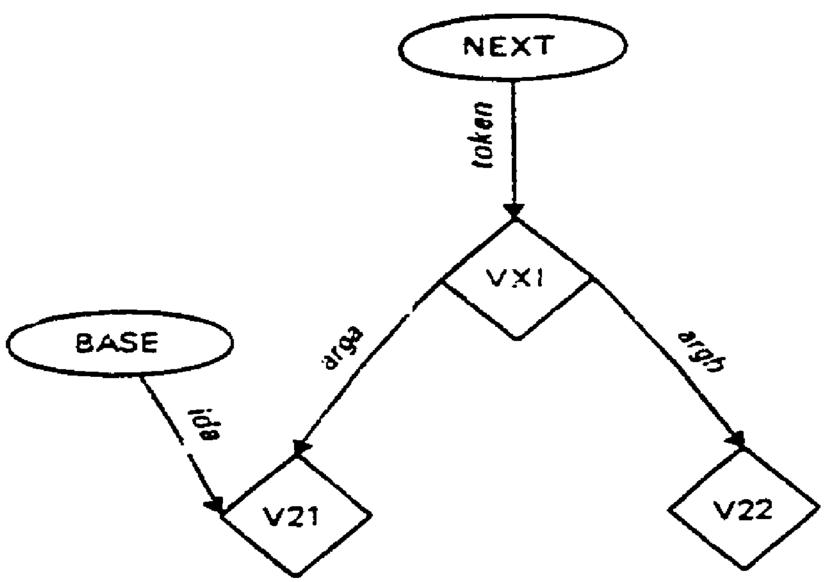

Figure 51. Pattern for retrieving the next numeron from CLIST.

Cycle 29. At this point, COUNTER "sees" an object it wants to count and has a counting name ready to use. The active control node GCOUNT causes P55 to be tested on this cycle. Since there are no condition patterns to match, the action of this production is taken with the result that the object gets counted:

**** (OB0161 FOUR)

Cycle 30. Having just counted an object, COUNTER attempts to find another object to count. GNEXTOB is active, causing P31 to be tested. However, since there are no more objects in LINGROUP which do not also have the property BOUND, the condition pattern fails to match.

P32 is tested next. There are no condition patterns to match so the action is taken to activate GEXTEND and keep the mentioned variables bound with their values.

Cycle 31. The control node GEXTEND is active and once again COUNTER attempts to extend LINGROUP to include any objects in the array that have not yet been counted. P33 is tested and the first condition pattern is successfully matched to the data base (this is intended to represent COUNTER recalling the relevant perceptual information about the array, including the direction of counting, slope of the array, and so on). However, since there exist no more objects in the array that have not already been made part of LINGROUP, the next three condition patterns fail to find a match for V1.

P34 is tested next on this cycle. There are no condition patterns to be matched and so the action of activating the control node RECALL is taken. However, the only variable to keep its binding for the next cycle is V10 (its binding is G0238, the symbol node for LINGROUP). This is because now that COUNTER has already counted all the objects, it need no longer remember the information it used to determine the direction and slope of the counting path.

Cycle 32. COUNTER has not more objects to count so it checks to see if there is anything else it wanted to do; that is, are there any current goals on the goal stack. In ACTP goal retrieval is accomplished with the GOALX schema which generates a pattern identical to the pattern generated by the SETGOAL schema. ACTP then tries to match this pattern against the pattern at the top of the current goal stack. A successful match indicates a current goal has been identified.

In the example, COUNTER has a single goal pattern, shown in Figure 17 (page 34). This represents the goal of assigning the cardinality of the set of counted objects. Retrieving this goal, then, requires generating the identical pattern with the GOALX schema.

RECALL is active on this cycle and so the model first tries to match the pattern specified in the condition of P35 to the pattern at the top of the goal stack. This particular pattern is only relevant to the constrained counting task and fails to match COUNTER's current goal. P36 is tested next and this time the pattern matches. This represents COUNTER recalling that it is to find the cardinality of the set and causes the control node GCARDINAL to be activated.

Cycle 33. On this cycle, COUNTER satisfies its current goal by assigning FOUR, the last numeron it used, as the cardinality of the set of objects it just counted.

The active control node, GCARDINAL, causes P38 to be tested on this cycle. The condition pattern of this production requires that COUNTER first remember the last numeron it used before it can assign it as the cardinality of the set of objects represented by G0238 (currently bound as the value of V10). The pattern matches and FOUR is assigned as the value of 21 (Figure 52):

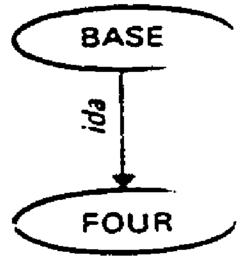

Figure 52. Data structure represented current problem BASE.

The action creates a structure that links FOUR to G0238 through a relation node that is a token of SIZE (Figure 53)

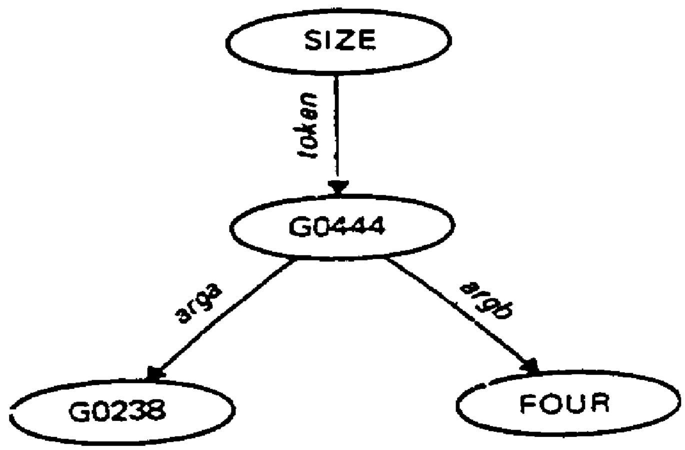

Figure 53. Data structure identifying FOUR as the cardinality of the group of counted objects.

and print[illegible] value of V21, followed by an exclamation point, identifying this numeron as the cardinality of the set:

***** (FOUR!)

Since the cardinality goal is now satisfied, the action (POPSTACK) removes the corresponding structure from COUNTER's goal stack.

Cycle 34. On this cycle, COUNTER checks again to see if there is anything else it intended to do. RECALL is active, but since there are no more goals left in the stack, the condition patterns of both P35 and P36 fail to match. The action of P38 indicates that COUNTER has completed its counting procedure:

((FINISH))

References

Anderson, J. R. Language, memory and thought. Hillsdale, N. J. Lawrence Erlbaum Associates, 1978.

Anderson, J. R., & Bower, G. H. Human associative memory. Washington, D. C.: Winston and Sons, 1973.

Anderson, J. R., Kline, P. J., & Beasley, C. M. A general learning theory and its application to schema abstraction (ONR Technical Report 78-2). Pittsburgh, Pa.: Carnegie-Mellon University, 1978.

Anderson, J. R., Kline, P. J., & Beasley, C. M. Complex learning processes. In R. E. Snow, P. A. Federico, W. E. Montague (Eds.), Aptitude, learning, and instruction: Cognitive process analyses. Hillsdale, N. J.: Lawrence Erlbaum Associates, 1980.

Beckwith, M., & Restle, F. Process of enumeration. Psychological Review, 1966, 73(5), 437-444.

Davis, R., & King, J. An overview of production systems. In Machine representations of knowledge. Dordrecht: D. Reidel Publishing Company, 1976.

Gelman, R. The nature and development of early number concepts. In H. W. Reese (Ed.), Advances in child development and behavior (Vol. 7). New York: Academic Press, 1972. (a)

Gelman, R. Logical capacity of very young children: Number invariance rules. Child Development, 1972, 43, 75-90. (b)

Gelman, R., & Gallistel, C. R. The child's understanding of number. Cambridge, Mass.: Harvard University Press, 1978.

Greeno, J. B. A study of problem solving. In R. Glaser (Ed.), Advances in instructional psychology (Vol. 1). Hillsdale, N. J.: Lawrence Erlbaum Associates, 1978.

Greeno, J. G., Riley, M. S., & Gelman, R. Young children's counting and understanding of principles (Working Paper). Pittsburgh: University of Pittsburgh, Learning Research and Development Center, 1979.

Hunt, E. B., & Poltrock, S. E. The mechanics of thought. In B. H. Kantowitz (Ed.), Human information processing: Tutorials in performance and cognition. Hillsdale, N. J.: Lawrence Erlbaum Associates, 1974.

Klahr, D., & Wallace, J. G. Cognitive development: An information-processing view. Hillsdale, N. J.: Lawrence Erlbaum Associates, 1976.

Newell, A. A theoretical exploration of mechanisms for coding the stimulus. In A. M. Melton & E. Martin (Eds.), Coding processes in human memory. Washington, D. C.: Winston, 1972.

Newell, A. You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), Visual information processing. New York: Academic Press, 1973. (a)

Newell, A. Production systems: Models of control structures. In W. G. Chase (Ed.), Visual information processing. Academic Press, 1973. (b)

Newell, A., & Simon, H. A. Human problem solving. Prentice-Hall, 1972.

Norman, D. A., & Rumelhart, D. E. Explorations in cognition. San Francisco: Freeman, 1975.

Quillian, M. R. The teachable language comprehender. Communications of the ACM, 1969, 12, 459-476.

Simon, H. A. The functional equivalence of problem solving skills. Cognitive Psychology, 1975, 7, 268-288.

APPENDIX A

Formal Description of ACTP

A model in ACTP has the following components:

1. a set of constants,
2. a set of variables,
3. a set of relations,
4. a set of concept schemata,
5. a set of numerical relations,
6. a network of categorical and other background knowledge, including a set of input types,
7. a network of information constituting the initial task situation, and
8. a set of productions.

Constants are defined in STARTUP. They include symbols to be used as control nodes, such as GNEXTOB and GNEXTN, as well as symbols that will be included in the data structure and referred to in productions, such as NUMERON, ZERO, ONE, and so on.

Variables are also defined in STARTUP, for example, V1, V2, and so on.

Relations are listed in STARTUP in order to define pairs of inverse relations, for example, (LABL CNPT) and (ISA MEMB).

A concept schema is a name, a set of arguments, and a list of relations between pairs of the arguments. In a network where there are nodes and relational links, a subnetwork matches a schema if the nodes in the subnetwork correspond to the arguments of the schema, so that all the relations in the schema correspond to links between pairs of nodes that are determined by the argument-node correspondence.

Numerical relations are the standard binary relations, such as greater than, equal, and so on.

Background knowledge includes categories defined in STARTUP, consisting of isa relations between category members and category names. One of the categories is TITLE, and the members of this category can be used in providing input information during operation of the model. Names of categories and members of categories are automatically defined as constants by ACTP. Information other than categories can also be included in background information. An example is the set of successor relations involving NEXT that are provided among the numerons that are in the initial CLIST for COUNTER.

The network representing the initial task situation contains nodes that represent objects that are present in the situation as well as relations among the objects. The situation presented to COUNTER has objects that are to be counted and spatial relations among the objects. Formally, the network for the situation and the network of background knowledge are indistinguishable in ACTP, but the two networks typically have information that differs significantly in the psychological interpretation of the model.

Productions have been described informally in considerable detail in this report. A grammar specifying the syntax of ACTP productions is given in Table A-1. The first rule says that a production has a condition and an action, with the condition first. The second rule says that a condition can have one or more control nodes and one or more pattern specifications. In practice, there is always a single control node and either one pattern specification or no pattern specification. The third rule says that the control node is a constant. "Constant" is not a terminal, but the terminals that are written for constants are defined for specific models. Figure A-1 shows a fragment of the derivation tree for the production that was discussed initially, shown in Figure 12 (page 28). First, Rule 1 rewrites production as condition and action. Rule 2 is used to rewrite condition as a single control node

Table A-1

Grammar for ACTP Production

1. Production → condition + action
2. Condition → (control node)* , (pattern specification)*
3. Control node → *constant*
4. Pattern specification → (concept schema)* , (numerical constraint)*
5. Concept schema → *schema name* + (argument)*
6. Argument → *constant*
7. Argument → *variable*
8. Numerical constraint → NCOMP + *numerical relation* + n–argument + n–argument
9. N–argument → *variable*
10. N-argument → *number*
11. Action → (special function)* , (pattern specification)* , (*variable*)* , (*constant*)*
12. Special function → PRINT + (argument)*
13. Special function → POPSTACK
14. Special function → UNBASE
15. Special function → FINISH
16. Special function → REBIND + *variable* + argument

Note. x + y means order is mandatory, x , y means order is optional; lower case means nonterminal; italicized means terminal nodes defined for specific models; upper case means terminal; (x) means x is optional; x* means x can be repeated.

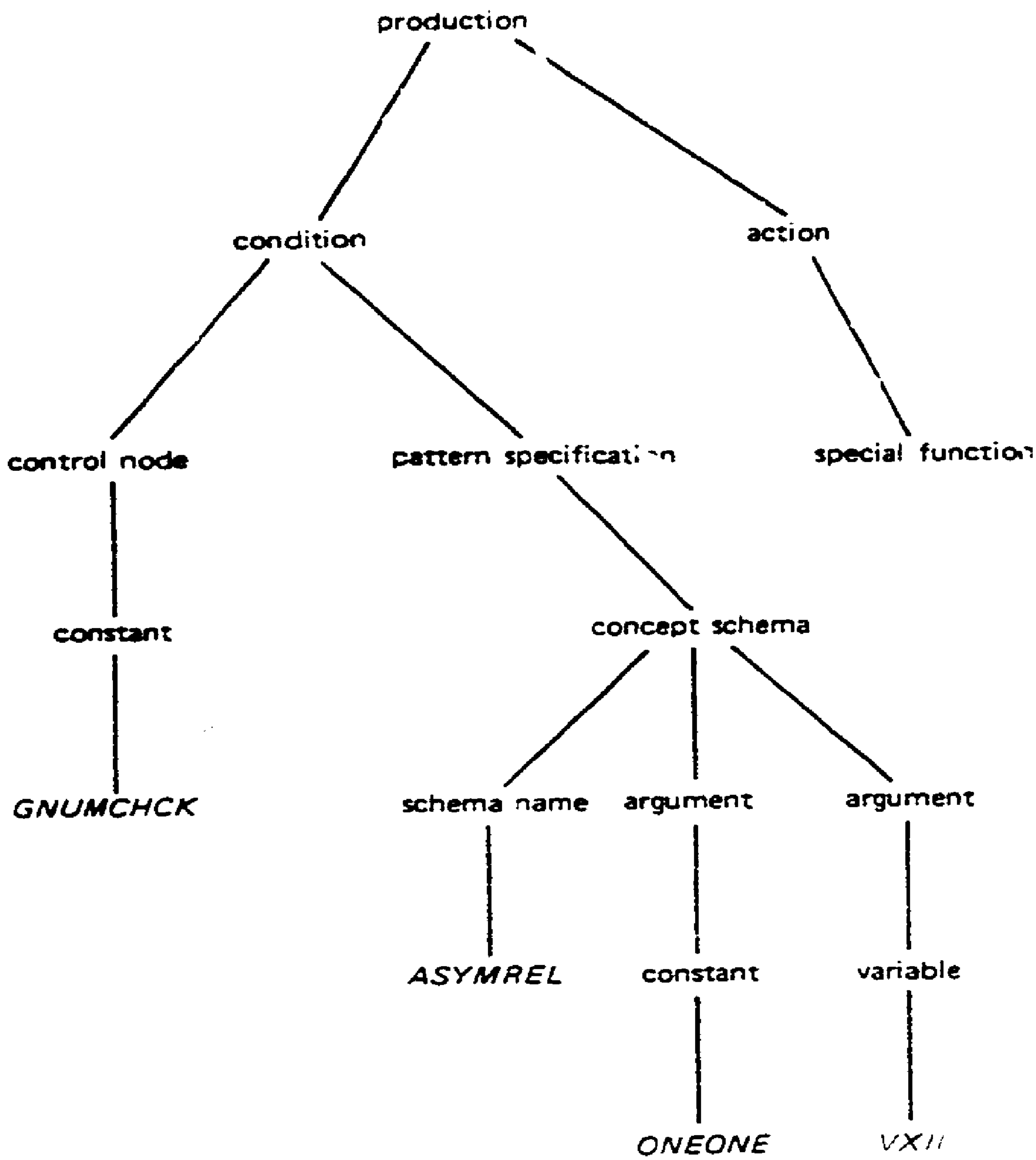

Figure A-1. Fragment of derivation tree for production #48.

and a single pattern specification. Rule 3 is used to rewrite control node as constant. Constant is rewritten as GNUMCHK, one of the constants defined in the COUNTER model.

The fourth rule in Table A-1 says that a pattern specificat can have one or more concept schemata and one or more num ical constraints. The fifth rule says that a concept schema has a schema name at the beginning followed by one or more arguments. While the number of arguments is syntactically optional, most concept schemata have the number of arguments fixed, and the production must have the required numer of arguments.

In Figure A-1 the pattern specification is rewritten according to Rule 4 as a single concept schema. The concept schema is rewritten as schema name plus four arguments (two are shown) using Rule 5. The schema name is ASYMREL. The first argument is a constant, ON NE. The second argument is a variable, VX11.

Rules 8, 9, and 10 specify the syntax of numerical constraints. They must begin with the symbol OMP, then have the name of a relation (for example, *GREAT), then have two arguments. These arguments can be either variables or literal numbers. Variables would be assumed to have numerical values.

Rule 11 specifies the syntax of actions, which can have one or more special functions, one or more pattern specifications, variables, and constants. The production in Figure 12 (page 28) has three special functions, a pattern specification, a constant, and three variables. A fragment of the derivation is shown in Figure A-1 where Rule 11 is used to rewrite the action.

Rules 12 to 15 specify the special functions that are available in ACTP at present. REBIND was not available when COUNTER was programmed. It is used to bind the variable listed first either to the constant or the value of the variable that is listed second.

Figure A-2 presents a formal description of the ACTP program in brief form. The program has three states: input, match condition, and execute action. When the system starts, there is a data structure and a list of productions. The system initially goes to its input state. If an input is received, the system responds by activating nodes and forming network structure that is added to the data. The system then goes to State 2, with the first production available to be tested.

In State 2, the condition of a production is tested. If a match is not found, the next production is made available and the system remains in State 2. If a match is found, the free variables that were matched are bound to the nodes that were found in the match, and the system goes to State 3.

State 3 executes the action of the production that was matched. If the action includes the special function FINISH, the system will halt. The components of the action are executed: special functions are performed, new network structure is added to the data, variables that are mentioned are retained with their values for the next cycle, and constants that are mentioned are made active nodes for the next cycle.

We now present a more detailed formal description of ACTP's operation. The data structure is a graph, with a set of nodes:

$$X = \{x_1, . . . , x_N\}.$$

Links in the graph are distinguished; there is a set of relations:

$$R = \{r_1, . . . , r_M\}.$$

In the usual way, each r_i defines a set of ordered pairs on the graph. Each member of the set is a pair that is linked by relation r_i; i.e.,

$$R_i = \{(x_j, x_k): r_i (x_j, x_k)\}.$$

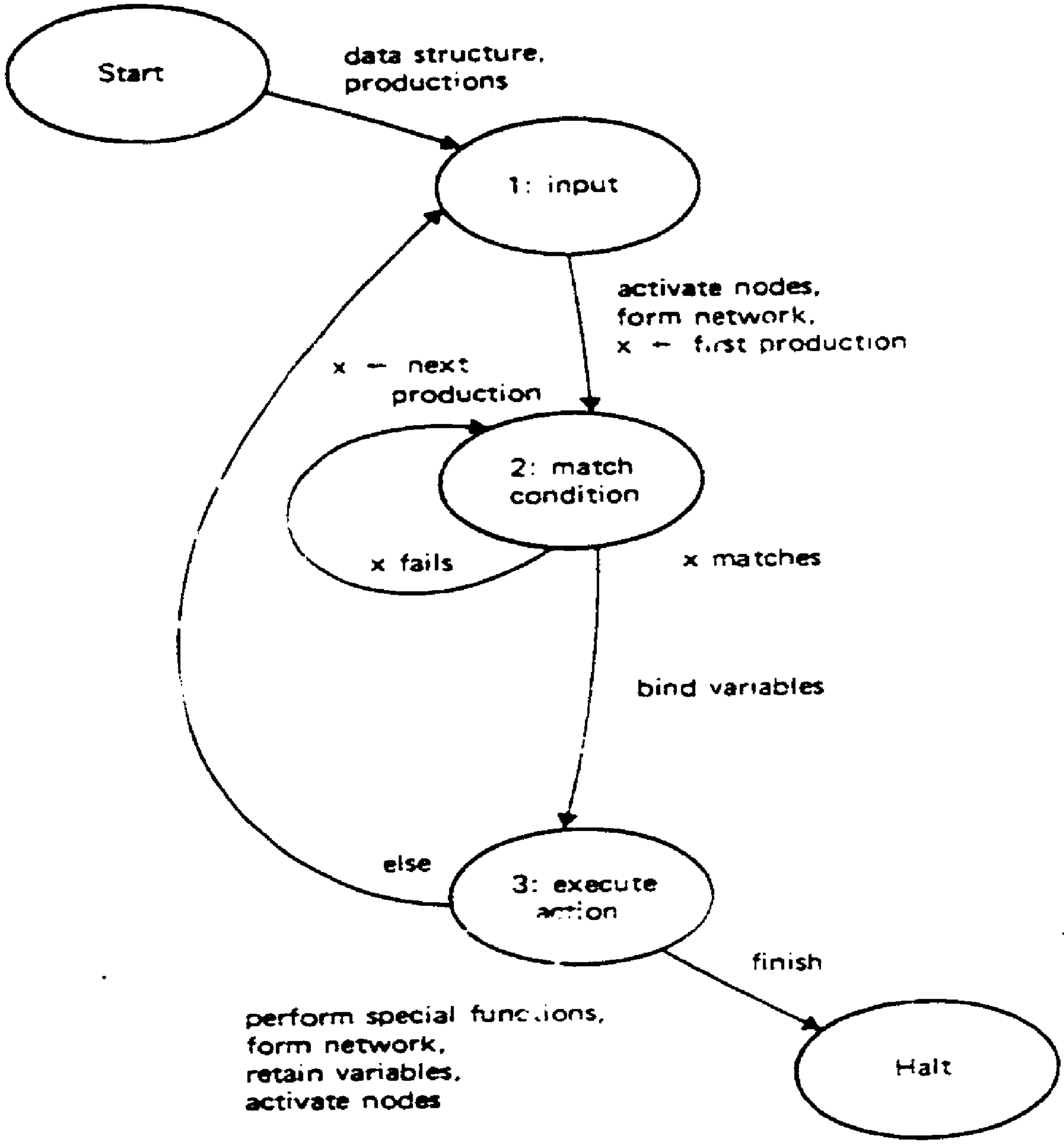

Figure A-2. Formal description of the ACTP program.

A subset of the nodes in the graph are designated as constants; that is, they have a property that permits them to be referred to directly in productions.

$$C = \{x: x \in X, c(x)\}.$$

Another property is applied to a changing set of nodes during operation of a model. This is membership in the set of active nodes.

$$A = \{x: x \in X, a(x)\}.$$

Finally, nodes are the values of variables. Each variable defined in the model defines a set V_i consisting of the element to which the variable is bound. If the variable is not bound, V_i is empty.

$$V_i = \{x: x \in X, v_i(x)\}.$$

Now, consider Figure A-2. At the start, the initial data structure is a graph of the form specified above. Initially A is empty. The input can cause elements to be placed in A and can cause additional relations to be applied in the graph. It is possible for new nodes to be added, although this is typically not done from input.

In State 2, ACTP attempts to match the condition of a production. The condition is a formula whose terms are constants, bound variables, and free variables of the form:

$$(\exists x_i) \ . \ . \ . \ (\exists x_q) \ (F(b_1 \ . \ . \ . \ b_p \ x_1 \ . \ . \ . \ x_q)).$$

where each b_i is either a constant or a bound variable, and each x_j is a free variable. F is a conjunction of terms, each of which is one of the following:

$$a(b_i), \ r_i(b_j, b_k), \ r_i(b_j, x_k), \ r_i(x_j, x_k).$$

For the condition to be matched, the formula must be true in the data structure. ACTP attempts to verify F by testing the assertions about constants and bound variables and then searching for a set of elements that satisfy the constraints on the free variables. If the search succeeds, then the elements found to correspond to the free variables become the values of those variables as the system moves to State 3.

Ordinarily the pattern must be matched with distinct values for all the different free variables, and no free variable can be matched to the value of any bound variable or constant in the pattern, although this restriction does not apply to variables that have numbers as values. To state this additional constraint formally, let

$$W = \{x_1, \ldots, x_q\},\ B = \{b_1, \ldots, b_p\};$$

that is, B is the set of values of the bound variables, and W is the set of values of free variables that satisfies the pattern match. Let $N(x_i)$ mean that x_i is a number. For a pattern to match, the formula F must be true with the following constraints:

$$\forall x_i, x_j (x_i, x_j \in W \rightarrow x_i = x_j \rightarrow N(x_i), N(x_j));$$

$$\forall x_i, b_j (x_i \in W, b_j \in B \rightarrow x_i = b_j \rightarrow N(x_i)).$$

There is a facility in ACTP for relaxing the constraint of distinct values. This facility was not used in COUNTER and is not described in the body of this report. In writing ACTP pattern specifications, one can specify subpatterns of variables and constants. This is done by placing the terms in parentheses, along with subpattern tags, which may be any distinctive symbols. For example, the condition of Production P6 in Appendix B is ((OBJCAT V1 OBJECT) (OBJECT V2 OBJECT)). Since V1 and V2 are different variable names, they must be matched to different nodes in the data structure. To relax that restriction, the pattern specification would be stated as ((OBJCAT (V1.A) OBJECT) (OBJCAT (V2.B) OBJECT)). This specifies two subpatterns, tagged A and B.

When subpatterns are designated, the values of variables in different subpatterns are allowed to overlap. In the example mentioned above, V1 and V2 could be matched to the same value since they are in different subpatterns. Variables that are not in subpatterns can be called global variables. All the global variables must have distinct values, and all variables in subpatterns must be distinct from all the global variables. Further, all the different variables in each subpattern must be distinct. (Again the restrictions to distinct values do not apply if the values are numbers.)

In the execution state, special functions are performed, some of which alter the data structure by removing links. New network structure is added, including addition of new nodes in the graph. If a variable v_i is mentioned, its value is put into V_i for the next cycle, and if a constant b_j is mentioned, it is a member of A for the next cycle.

APPENDIX B

A List of COUNTER's Productions

```
   PROLIST

1. (((ADDTAG
      ((ASYMREL NEXT VX1 V21 V22)
       (ABSENCE OBJPROP V22 FOLLOWED)
       (OBJCAT V23 NUMERON)
       (ABSENCE OBJPROP V23 FOLLOWED)))
     ((PARTOBJ V23 CLIST)
      (ASYMREL NEXT VX2 V22 V23)
      (OBJPROP V22 FOLLOWED)))
2.  ((MAKE ((ORDASSIGN BASE V1 V21)))
     ((OBJPROP V1 SPECIAL)
      (OBJPROP V21 SPECIAL)
      (SETGOAL GOAL VX12 XRESTRICTION MAKE V1 V21)
      (ASYMREL ONEONE VX11 V1 V21))
     (PREQPLAN)
     GSEE)
3.  ((MAKE2 ((ORDASSIGN BASE V1 V21)))
     ((OBJPROP V1 SPECIAL)
      (OBJPROP V21 SPECIAL)
      (SETGOAL GOAL VX12 XRESTRICTION MAKE V1 V21))
     (PREQPLAN)
     GSEE)
4.  ((HOWMANY) ((SETGOAL GOAL VX1 XFIND SIZE ?GROUP ?NUM)) GSEE)
5.  ((GSEE
      ((OBJCAT V1 OBJECT)
       (OBJCAT V2 OBJECT)
       (OBJCAT V3 OBJECT)
       (ASYMREL YXSLOPE VX1 V1 V2 N1)
       (ASYMREL YXSLOPE VX2 V2 V3 N2)
       (NCOMP APXEQ N1 N2)))
     ((OBJCAT V10 LINGROUP)
      (PARTOBJ V1 V10)
      (PARTOBJ V2 V10)
      (PARTOBJ V3 V10))
     GDIMEN
     V10
     N1)
6.  ((GSEE ((OBJCAT V1 OBJECT) (OBJCAT V2 OBJECT)))
     ((OBJCAT V10 PAIR) (PARTOBJ V1 PAIR) (PARTOBJ V2 PAIR))
     GCPAIR
     V10
     V1
     V2)
7.  ((GSEE ((OBJCAT V1 OBJECT))) ((OBJCAT V10 SINGLE)) GCSINGLE V10 V1
```

Appendix B (Cont'd)

```
 8.   ((GDIMEN ((NCOMP APXEQ N1 1.0)))
       ((ASYMREL SCAN VX1 V10 YCOR))
       GALIGN
       V10)
 9.   ((GDIMEN ((NCOMP APXEQ N1 3.0)))
       ((ASYMREL SCAN VX1 V10 YCOR))
       GALIGN
       V10)
10.   ((GDIMEN) ((ASYMREL SCAN VX1 V10 XCOR)) GALIGN V10)
11.   ((GALIGN
        ((ASYMREL SCAN VX1 V10 V11)
         (PARTOBJ V1 V10)
         (ASYMREL V11 VX2 V1 N1)
         (PARTOBJ V2 V10)
         (ASYMREL V11 VX3 V2 N2)
         (PARTOBJ V3 V10)
         (ASYMREL V11 VX4 V3 N3)
         (NCOMP *LESS N1 N2)
         (NCOMP *LESS N2 N3)))
       GCHBOUND
       V10
       V11
       V1
       V2
       V3
       N1
       N3)
12.   ((GCHBOUND
        ((OBJCAT V4 OBJECT) (ASYMREL V11 VX1 V4 N4) (NCOMP *LESS N4 N1)))
       GCHBOUND2
       V10
       V11
       V1
       V2
       V3
       N1
       N3)
13.   ((GCHBOUND
        ((ASYMREL SCAN VX1 V10 V11) (ASYMREL YXSLOPE VX2 V1 V2 N10)))
       ((ASYMREL SCAN VX1 V10 V11 *GREAT N10) (OBJPROP V1 BOUND))
       GDIRECT
       V10
       V11)
14.   ((GCHBOUND2
        ((OBJCAT V4 OBJECT) (ASYMREL V11 VX1 V4 N4) (NCOMP *GREAT N4 N3)))
       ((ORDASSIGN BASE V4 V1 V2 N4))
       GFINDBOUND)
15.   ((GCHBOUND2
        ((ASYMREL SCAN VX1 V10 V11) (ASYMREL YXSLOPE VX2 V3 V2 N10)))
       ((ASYMREL SCAN VX1 V10 V11 *LESS N10) (OBJPROP V3 BOUND))
       GDIRECT
       V10
       V11)
```

```
16.   ((GFINDBOUND
        ((ORDASSIGN BASE V1 V2 V3 N1)
         (OBJCAT V4 OBJECT)
         (ASYMREL V11 VX1 V4 N4)
         (NCOMP *LESS N4 N1)))
       (UNBASE)
       ((ORDASSIGN BASE V4 V1 V2 N4))
       GFINDBOUND
       V11)
17.   ((GFINDBOUND
        ((ASSIGN BASE V1 V2 V3) (ASYMREL YXSLOPE VX1 V1 V3 N10)))
       ((OBJCAT V10 LINGROUP)
        (PARTOBJ V1 V10)
        (PARTOBJ V2 V10)
        (PARTOBJ V3 V10)
        (OBJPROP V1 BOUND)
        (ASYMREL SCAN VX1 V10 V11 *GREAT N10))
       GDIRECT
       V10
       V11)
18.   ((GDIRECT ((ASYMREL SCAN VX1 V10 V11 V12))) GCOMPACT V10 V11 V12)
19.   ((GCOMPACT
        ((PARTOBJ V1 V10) (ASYMREL V11 VX1 V1 N1)
                          (OBJPROP V1 BOUND)
                          (PARTOBJ V2 V10)
                          (ASYMREL V11 VX2 V2 N2)
                          (PARTOBJ V3 V10)
                          (ASYMREL V11 VX3 V3 N3)
                          (NCOMP V12 N3 N2)))
       GCOMPACT2
       V1
       V2
       V3
       N1
       N2
       N3
       V11
       V12)
20.   ((GCOMPACT2
        ((OBJCAT V4 OBJECT)
         (ASYMREL V11 VX1 V4 N4)
         (NCOMP V12 N2 N4)
         (ASYMREL YXSLOPE VX2 V1 V2 N10)))
       ((OBJCAT V10 LINGROUP)
        (PARTOBJ V1 V10)
        (PARTOBJ V4 V10)
        (PARTOBJ V2 V10)
        (ASYMREL SCAN VX3 V10 V11 V12 N10))
       GCOMPACT
       V10
       V11)
```

```
21.  ((GCOMPACT2
       ((OBJCAT V4 OBJECT)
        (ASYMREL V11 VX1 V4 N4)
        (NCOMP V12 N3 N4)
        (ASYMREL YXSLOPE VX2 V1 V4 N10)))
      ((OBJCAT V10 LINGROUP)
       (PARTOBJ V1 V10)
       (PARTOBJ V2 V10)
       (PARTOBJ V4 V10)
       (ASYMREL SCAN VX3 V10 V11 V12 N10))
      GCOMPACT
      V10
      V11)
22.  ((GCOMPACT2 ((OBJCAT V10 LINGROUP) (ASYMREL SCAN VX3 V10 V11 V12)))
      ((OBJPROP V1 BOUND))
      (UNBASE)
      (* . GNEXTN)
      V10
      V11
      V12
      V1)
23.  ((GOBJCHK ((ABSENCE OBJPROP V1 SPECIAL))) GNEXTN V10 V11 V12 V1)
24.  ((GOBJCHK) GNEXTOB V10 V11 V12)
25.  ((GOBJCHK2 ((ABSENCE OBJPROP V1 SPECIAL))) GNEXTN V10 V11 V12 V1)
26.  ((GOBJCHK2) GETSPECIALNUMBER V10 V11 V12 V1)
27.  ((GOBJCHK3 ((ABSENCE OBJPROP V1 SPECIAL))) GNEXTN V10 V11 V12 V1)
28.  ((GOBJCHK3 ((ASYMREL ONEONE VX1 V1 V22)))
      (PRINT V1 V22)
      (POPSTACK)
      GNEXTOB
      V10
      V11
      V12)
29.  ((GETSPECIALNUMBER
       ((ORDASSIGN BASE V21)
        (PARTOBJ V22 CLIST)
        (ASYMREL NEXT VX1 V21 V22)
        (ABSENCE OBJPROP V22 SPECIAL)))
      (PRINT V22)
      (UNBASE)
      ((ORDASSIGN BASE V22))
      GETSPECIALNUMBER
      V1
      V10
      V11
      V12)
```

```
30.  ((GETSPECIALNUMBER
       ((ORDASSIGN BASE V21)
        (PARTOBJ V22 CLIST)
        (ASYMREL NEXT VX1 V21 V22)))
      (POPSTACK)
      GCOUNT
      V1
      V22
      V10
      V11
      V12)
31.  ((GNEXTOB ((PARTOBJ V1 V10) (ABSENCE OBJPROP V1 BOUND)))
      GCHBETWEEN
      V1
      V10
      V11
      V12)
32.  ((GNEXTOB) GEXTEND V10 V11 V12)
33.  ((GEXTEND
       ((ASYMREL SCAN VX1 V10 V11 V12 N10)
        (OBJCAT V1 OBJECT)
        (ABSENCE PARTOBJ V1 V10)
        (ASYMREL V11 VX2 V1 N1)
        (OBJPROP V2 BOUND)
        (ASYMREL YXSLOPE VX3 V2 V1 N2)
        (NCOMP APXEQ N10 N2)))
      GCHCLOSER
      V10
      V11
      V12
      N10
      V1
      N1
      V2)
34.  ((GEXTEND) RECALL V10)
35.  ((RECALL ((GOALX GOAL VX12 XRESTRICTION MAKE V1 V21)))
      (PRINT V1 V21)
      (UNBASE)
      ((ORDASSIGN BASE V21))
      (POPSTACK)
      RECALL
      V10)
36.  ((RECALL ((GOALX GOAL VX1 XFIND SIZE ?GROUP ?NUM))) GCARDINAL V
37.  ((RECALL) (FINISH))
38.  ((GCARDINAL ((ORDASSIGN BASE V21)))
      ((ASYMREL SIZE VX1 V10 V21))
      (PRINT V21 !)
      (POPSTACK)
      RECALL)
```

```
39.  ((GCHCLOSER
       ((OBJCAT V3 OBJECT)
        (ABSENCE PARTOBJ V3 V10)
        (ASYMREL V11 VX1 V3 N3)
        (NCOMP V12 N1 N3)
        (ASYMREL YXSLOPE VX2 V2 V3 N2)
        (NCOMP APXEQ N10 N2)
        (ORDASSIGN BASE V21)))
      (UNBASE)
      ((ORDASSIGN BASE V21 V3 N3))
      GCHC2
      V10
      V11
      V12
      N10
      V2)
40.  ((GCHCLOSER) ((PARTOBJ V1 V10))
                  ((OBJPROP V1 BOUND))
                  (* . GNEXTN)
                  V10
                  V11
                  V12
                  V1)
41.  ((GCHBETWEEN
       ((ASYMREL V11 VX1 V1 N1)
        (PARTOBJ V2 V10)
        (ABSENCE OBJPROP V2 BOUND)
        (ASYMREL V11 VX2 V2 N2)
        (NCOMP V12 N1 N2)
        (ORDASSIGN BASE V21)))
      (UNBASE)
      ((ORDASSIGN BASE V21 V2 N2))
      GCHB2
      V10
      V11
      V12)
42.  ((GCHBETWEEN) ((OBJPROP V1 BOUND)) (* . GNEXTN) V10 V11 V12 V1)
43.  ((GCHC2 ((ORDASSIGN BASE V21 V1 N1)))
      GCHCLOSER
      V10
      V11
      V12
      N10
      V1
      N1
      V2)
44.  ((GCHB2 ((ORDASSIGN BASE V21 V1 N1))) GCHBETWEEN V1 N1 V10 V11 V12)
```

```
45.  ((GNEXTN
       ((ORDASSIGN BASE V21)
        (PARTOBJ V22 CLIST)
        (ASYMREL NEXT VX1 V21 V22)))
      (* . GCOUNT)
      V1
      V22
      V10
      V11
      V12)
46.  ((GNEXTN ((PARTOBJ V22 CLIST) (OBJPROP V22 FIRST)))
      (* . GCOUNT)
      V1
      V22
      V10
      V11
      V12)
47.  ((GNUMCHK ((ABSENCE OBJPROP V22 SPECIAL)))
      GCOUNT
      V1
      V22
      V10
      V11
      V12)
48.  ((GNUMCHK ((ASYMREL ONEONE VX11 V2 V22)))
      (PRINT V2 V22)
      (POPSTACK)
      (UNBASE)
      ((ORDASSIGN BASE V22))
      GNEXTN
      V1
      V10
      V11
      V12)
49.  ((GNUMCHK2 ((ABSENCE OBJPROP V22 SPECIAL)))
      GCOUNT
      V1
      V22
      V10
      V11
      V12)
50.  ((GNUMCHK2) GETSPECIALOBJECT V1 V22 V10 V11 V12)
51.  ((GNUMCHK3 ((ABSENCE OBJPROP V22 SPECIAL)))
      GCOUNT
      V1
      V22
      V10
      V11
      V12)
52.  ((GNUMCHK3) (UNBASE) ((ORDASSIGN BASE V22)) GNEXTN V1 V10 V11 V1
```

```
53.   ((GETSPECIALOBJECT ((ABSENCE OBJPROP V1 SPECIAL)))
       (PRINT V1 SKIPPED)
       GNEXTOB
       V10
       V11
       V12)
54.   ((GETSPECIALOBJECT) (POPSTACK) GCOUNT V1 V22 V10 V11 V12)
55.   ((GCOUNT) (PRINT V1 V22)
                (UNBASE)
                ((ORDASSIGN BASE V22))
                GNEXTOB
                V10
                V11
                V12))
```

APPENDIX C

Output from the Series of Cycles Involved in COUNTER Counting a Group of Four Objects

```
*(STARTUP)

NIL
*(GENSET OBJECTS)

SHOW-STRUCTURE? *YES

(OBO161 D 10.0 0.0)
(OBO164 C 9.5 0.0)
(YXSL0169 OBO161 2.0)
(YXSL0170 OBO161 0.0)
(OBO171 B 8.0 0.20000000)
(YXSL0176 OBO164 1.8823529)
(YXSL0177 OBO164 3.8823529)
(OBO178 A 7.0 0.0)
(YXSL0183 OBO171 2.1666666)
(YXSL0184 OBO171 0.16666665)
NIL
*(TRACE PREQPLAN PREQCHK)

(PREQCHK PREQPLAN)
*(CYCLE)

THINK-ALOUD? *YES

0
>>> *(LISP PROGN (SETQ DEBUG NIL) NIL)

NIL
NIL
1
>>> *(ADDTAG)

(ADDTAG)
NIL
2
(((PARTOBJ V23 CLIST) (ASYMREL NEXT VX2 V22 V23) (OBJPROP V22 FOLLOWE
D)))
(ST0200 (FOUR CLIST NEXT G0199 THREE FOLLOWED))
>>> *(ADDTAG)

(ADDTAG)
NIL
3
(((PARTOBJ V23 CLIST) (ASYMREL NEXT VX2 V22 V23) (OBJPROP V22 FOLLOWE
D)))
(ST0211 (FIVE CLIST NEXT G0210 FOUR FOLLOWED))
>>> *(ADDTAG)
```

```
(ADDTAG)
NIL
4
(((PARTOBJ V23 CLIST) (ASYMREL NEXT VX2 V22 V23) (OBJPROP V22 FOLLOWE
D)))
(ST0222 (SIX CLIST NEXT G0221 FIVE FOLLOWED))
>>> *(HOWMANY)

(HOWMANY)
NIL
5
(((SETGOAL GOAL VX1 XFIND SIZE ?GROUP ?NUM)) GSEE)
(STK GOAL (G0187) (HALT))
(ST0225 (GOAL G0224 XFIND SIZE ?GROUP ?NUM))
>>> *NIL

(GSEE)
NIL
6
(((OBJCAT V10 LINGROUP) (PARTOBJ V1 V10) (PARTOBJ V2 V10) (PARTOBJ V3
 V10)) GDIMEN V10 N1)
(ST0239 (G0238 LINGROUP OB0178 OB0171 OB0164))
>>> *NIL

(GDIMEN G0238 0.16666665)
(0.16666665 N1 G0238 V10)
7
(((ASYMREL SCAN VX1 V[illegible] XCOR)) GALIGN V10)
(ST0246 (SCAN G0245 G0238 XCOR))
>>> *NIL

(GALIGN G0238)
(G0238 V10)
8
(GCHBOUND V10 V11 V1 V2 V3 N1 N3)
>>> *NIL

(GCHBOUND G0238 XCOR OB0178 OB0171 OB0164 7.0 9.5)
(G0238 V10 XCOR V11 OB0178 V1 7.0 N1 OB0171 V2 OB0164 V3 9.5 N3)
9
(((ASYMREL SCAN VX1 V10 V11 *GREAT N10) (OBJPROP V1 BOUND)) GDIRECT V
10 V11)
(ST0275 (SCAN G0245 G0238 XCOR *GREAT 0.16666665 OB0178 BOUND))
>>> *NIL

(GDIRECT G0238 XCOR)
(XCOR V11 G0238 V10)
10
(GCOMPACT V10 V11 V12)
>>> *NIL
```

Appendix C (Cont'd)

```
(GCOMPACT G0238 XCOR *GREAT)
(G0238 V10 XCOR V11 *GREAT V12)
11
(GCOMPACT2 V1 V2 V3 N1 N2 N3 V11 V12)
>>> *NIL

(GCOMPACT2 OB0178 OB0171 OB0164 7.0 8.0 9.5 XCOR *GREAT)
(*GREAT V12 XCOR V11 OB0178 V1 7.0 N1 OB0171 V2 8.0 N2 OB0164 V3 9.5
N3)
12
(((OBJPROP V1 BOUND)) (UNBASE) (* . GNEXTN) V10 V11 V12 V1)
(ST0306 (OB0178 BOUND))
>>> *NIL

(GNEXTN G0238 XCOR *GREAT OB0178)
(OB0178 V1 XCOR V11 *GREAT V12 G0238 V10)
13
((* . GCOUNT) V1 V22 V10 V11 V12)
>>> *NIL

(GCOUNT OB0178 ONE G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0178 V1 ONE V22)
14
((PRINT V1 V22) (UNBASE) ((ORDASSIGN BASE V22)) GNEXTOB V10 V11 V12)
***** (OB0178 ONE)
(ST0315 (BASE ONE))
>>> *NIL

(GNEXTOB G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10)
15
(GCHBETWEEN V1 V10 V11 V12)
>>> *NIL

(GCHBETWEEN OB0164 G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0164 V1)
16
((UNBASE) ((ORDASSIGN BASE V21 V2 N2)) GCHB2 V10 V11 V12)
(ST0330 (BASE ONE OB0171 8.0))
>>> *NIL

(GCHB2 G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10)
17
(GCHBETWEEN V1 N1 V10 V11 V12)
>>> *NIL

(GCHBETWEEN OB0171 8.0 G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0171 V1 8.0 N1)
18
(((OBJPROP V1 BOUND)) (* . GNEXTN) V10 V11 V12 V1)
(ST0345 (OB0171 BOUND))
>>> *NIL
```

```
(GNEXTN G0238 XCOR *GREAT OB0171)
(OB0171 V1 XCOR V11 *GREAT V12 G0238 V10)
19
((* . GCOUNT) V1 V22 V10 V11 V12)
>>> *NIL

(GCOUNT OB0171 TWO G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0171 V1 TWO V22)
20
((PRINT V1 V22) (UNBASE) ((ORDASSIGN BASE V22)) GNEXTOB V10 V11 V12)
***** (OB0171 TWO)
(ST0352 (BASE TWO))
>>> *NIL

(GNEXTOB G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10)
21
(GCHBETWEEN V1 V10 V11 V12)
>>> *NIL

(GCHBETWEEN OB0164 G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0164 V1)
22
(((OBJPROP V1 BOUND)) (* . GNEXTN) V10 V11 V12 V1)
(ST0367 (OB0164 BOUND))
>>> *NIL

(GNEXTN G0238 XCOR *GREAT OB0164)
(OB0164 V1 XCOR V11 *GREAT V12 G0238 V10)
23
((* . GCOUNT) V1 V22 V10 V11 V12)
>>> *NIL

(GCOUNT OB0164 THREE G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11 OB0164 V1 THREE V22)
24
((PRINT V1 V22) (UNBASE) ((ORDASSIGN BASE V22)) GNEXTOB V10 V11 V12)
***** (OB0164 THREE)
(ST0374 (BASE THREE))
>>> *NIL

(GNEXTOB G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10)
25
(GEXTEND V10 V11 V12)
>>> *NIL

(GEXTEND G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11)
26
(GCHCLOSER V10 V11 V12 N10 V1 N1 V2)
>>> *NIL
```

```
(GCHCLOSER G0238 XCOR *GREAT 0.16666665 OB0161 10.0 OB0164)
(XCOR V11 *GREAT V12 G0238 V10 0.16666665 N10 OB0161 V1 10.0 N1 OB016
4 V2)
27
(((PARTOBJ V1 V10)) ((OBJPROP V1 BOUND)) (* . GNEXTN) V10 V11 V12 V1)

(ST0403 (OB0161 G0238))
(ST0405 (OB0161 BOUND))
>>> *NIL

(GNEXTN G0238 XCOR *GREAT OB0161)
(OB0161 V1 G0238 V10 *GREAT V12 XCOR V11)
28
((* . GCOUNT) V1 V22 V10 V11 V12)
>>> *NIL

(GCOUNT OB0161 FOUR G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10 OB0161 V1 FOUR V22)
29
((PRINT V1 V22) (UNBASE) ((ORDASSIGN BASE V22)) GNEXTOB V10 V11 V12)
***** (OB0161 FOUR)
(ST0412 (BASE FOUR))
>>> *NIL

(GNEXTOB G0238 XCOR *GREAT)
(G0238 V10 *GREAT V12 XCOR V11)
30
(GEXTEND V10 V11 V12)
>>> *NIL

(GEXTEND G0238 XCOR *GREAT)
(XCOR V11 *GREAT V12 G0238 V10)
31
(RECALL V10)
>>> *NIL

(RECALL G0238)
(G0238 V10)
32
(GCARDINAL V10)
>>> *NIL

(GCARDINAL G0238)
(G0238 V10)
33
(((ASYMREL SIZE VX1 V10 V21)) (PRINT V21 !) (POPSTACK) RECALL)
(ST0445 (SIZE G0444 G0238 FOUR))
***** (FOUR !)
>>> *NIL

(RECALL)
NIL
34
((FINISH))
FINISH
>>> *NIL
```

APPENDIX D

STARTUP

STARTUP is the LISP function that puts the terminology that will be used in a particular set of productions into ACTP's memory. ACTP's terminology includes three kinds of symbols: constants, variables, and links.

One way to memorize constants is with a function called MEMORIZE, for example:

```
(MEMORIZE (QUOTE (COUNTED USED FOLLOWED BOUND SKIPPED
SPECIAL)))
```

MEMORIZE simply tags constants so they will be usable either as control nodes or constant nodes in patterns. Another way of establishing constants is by the use of the function CATEGORY:

```
(CATEGORY (QUOTE (TITLE ADDTAG MAKE MAKE2 HOWMANY)))

(CATEGORY (QUOTE (NUMERON ZERO ONE TWO THREE TEN
. . . .)))
```

As with MEMORIZE, all the terms in these two lists become usable constants. However, two additional things are done by CATEGORY. A link is formed between the first term in the list and each of the other members. The link is <u>isa</u>, indicating category membership. For example:

ADDTAG	isa	TITLE
MAKE	isa	TITLE
THREE	isa	NUMERON

Finally, the CATEGORY function makes it possible to use the listed terms as input to ACTP during a cycle.

The second kind of term is a variable. Variable names are set up by a function called VARIABLE, for example:

```
(VARIABLE (QUOTE (V1 V2 V3 V4 . . . . V25 VX1 VX2
VX3 . . . . VX12)))
```

(VARIABLE (QUOTE (N1 N2 N3 N4 N5 N6 N7 N8 N9 N10)))

The third kind of term is a link. Link names are set up by a function called PUTINV:

(MAPC (QUOTE PUTINV) RPAIRS)

RPAIRS is a list of word pairs that gets defined before PUTINV is called:

(SETQ RPAIRS (QUOTE ((LABL CNPT) (ISA MEMB) (HASPROP ISPROP) . . . .)))

PUTINV then takes this list and makes each member of a pair the inverse of the other member. This is needed because the pattern matching system in ACTP assumes that each link goes in two directions, and the function that creates links in patterns as ACTP is running looks up the inverse of each link name and creates bidirectional links. For example, the inverse of isa is memb (for member). This means that when a link is made giving

ONE isa NUMERON

there is also a link giving

NUMERON memb ONE.

The following pages include the version of STARTUP that is in the model.

```
(DEFPROP STARTUP
 (LAMBDA NIL
  (SETQ IBASE (*PLUS IBASE 2))




  (SETQ BASE (*PLUS BASE 2))
  (SETQ *NOPOINT T)
  (SETQ EQTOL 1.0)
  (SETQ IDLIST (QUOTE (NIL IDA IDB IDC IDD IDE)))
  (SETQ ARGLIST (QUOTE (NIL NIL ARGA ARGB ARGC ARGD ARGE)))
  (SETQ VARLIST NIL)
  (SETQ RELATIONLIST NIL)
  (SETQ RPAIRS
        (QUOTE
         ((LABL CNPT) (ISA MEMB)
                      (HASPROP ISPROP)
                      (HASPART ISPART)
                      (IS IS1)
                      (IDA IDA1)
                      (IDB IDB1)
                      (IDC IDC1)
                      (IDD IDD1)
                      (IDE IDE1)
                      (IDF IDF1)
                      (IDG IDG1)
                      (ARGU ARGU1)
                      (ARGA ARGA1)
                      (ARGB ARGB1)
                      (ARGC ARGC1)
                      (ARGD ARGD1)
                      (ARGE ARGE1)
                      (ARGF ARGF1)
                      (ARGG ARGG1)
                      (TYPE TOKEN)
                      (PREQ PREQ1)
                      (*GREAT LEQ)
                      (GREQ *LESS)
                      (EQUAL EQUAL)
                      (APXEQ APXEQ)
                      (ACTION ACTION1)
                      (TARG TARG1)
                      (PATT PATT1)
                      (*IS *IS1))))
```

```
(MAPC (QUOTE PUTINV) RPAIRS)
(CATEGORY (QUOTE (TITLE ADDTAG MAKE MAKE2 HOWMANY)))
(CATEGORY (QUOTE (NUMREL APXEQ BETWEEN LEQ GREQ *LESS *GREAT EQUAL)))
(CATEGORY (QUOTE (NUMERON ZERO ONE TWO THREE TEN NINE EIGHT SEVEN SIX FIVE FOUR)))
(CATEGORY (QUOTE (THING SQUARE TRIANGLE STAR CIRCLE PENTAGON CROSS TRUCK FISH CORN FENCE DUCK PIG)))
(MEMORIZE
 (QUOTE
  (GNEXTOB GNEXTN
           GSEE
           GDIMEN
           GCPAIR
           GCSINGLE
           GALIGN
           GCHBOUND
           GCHBOUND2
           GDIRECT
           GCOMPACT
           GCOMPACT2
           GFINDBOUND
           GCLINGROUP
           GEXTEND
           GCHCLOSER
```



```
           GCARDINAL
           GCHBETWEEN
           GCHB2
           GOBJCHK
           GOBJCHK2
           GETSPECIALNUMBER
           GNUMCHK
           GNUMCHK2
           GETSPECIALOBJECT
           GCOUNT
           GCHC2
GOBJCHK3 GNUMCHK3
           RECALL)))
 (MEMORIZE (QUOTE (OBJECT NUMERON CLIST FIRST NEXT BASE LINGROUP XCOR YCOR YXSLOPE PAIR SINGLE SCAN)))
 (MEMORIZE (QUOTE (COUNTED USED FOLLOWED BOUND SKIPPED SPECIAL)))
 (MEMORIZE (QUOTE (GOAL XFIND SIZE ?GROUP ?NUM)))
 (MEMORIZE (QUOTE (XRESTRICTION)))
 (SETQ ACTIVE NIL)
```

```
(VARIABLE
 (QUOTE
  (V1 V2
      V3
      V4
      V5
      V6
      V7
      V8
      V9
      V10
      V11
      V12
      V13
      V14
      V15
      V16
      V17
      V18
      V19
      V20
      V21
      V22
      V23
      V24
      V25
      VX1
      VX2
      VX3
      VX4
      VX5
      VX6
      VX7
      VX8
      VX9
      VX10
      VX11
      VX12)))
  (VARIABLE (QUOTE (N1 N2 N3 N4 N5 N6 N7 N8 N9 N10)))
  (SETQ COUNTLIST (QUOTE (ONE TWO THREE)))
  (SETQ CON (CAR COUNTLIST))
  (LINK CON (QUOTE FIRST) (QUOTE HASPROP))
  (LINK CON (QUOTE CLIST) (QUOTE ISPART))
  (MAPC (QUOTE GENCLIST) (CDR COUNTLIST))
  (PUTPROP (QUOTE PREREQ) (QUOTE ((GNUMCHK . GCOUNT) (GOBJCHK . GNEXTN))) (QUOTE MEMB))
  (PUTPROP (QUOTE PREREQ2) (QUOTE ((GNUMCHK2 . GCOUNT) (GOBJCHK2 . GNEXTN))) (QUOTE MEMB))
(PUTPROP @PREREQ3 @((GNUMCHK3 . GCOUNT)(GOBJCHK3 . GNEXTN)) @MEMB)
  (SETQ EUTOL 0.30000000)
  (SETQ DEBUG NIL))
FXPR)
```

APPENDIX E

Glossary

ABSENCE. Function that tests for the absence of single-linked relations in the data base.
Example: (ABSENCE OBJPROP V22 FOLLOWED) (Production 1 in Appendix B).

Action. The "then" part of production rule which is executed when the condition of that production is true. Actions consist of (1) executing special functions, (2) building patterns by adding new relations to the data base, and (3) remembering and activating nodes.

action. Relation in the GOALX schema.
Example: G0224---action--->XFIND (Figure 17).
Inverse: action1.

action1. Relation in the GOALX schema.
Example: XFIND---action1--->G0224.
Inverse: action.

Active node. Constants that are (1) mentioned in the action of an executed production, or (2) typed in from terminal.

ACTP. A production system for developing simulation models. ACTP consists of (1) a set of production rules, (2) a set of terms and concepts needed for the production rules to be used, and (3) an executive program that is used to operate the productions.

APXEQ. Constant given as an argument to the function NCOMP to test for a single-linked quantitative relation of "approximate equality" between two nodes corresponding to numbers in the data base.
Example: (NCOMP APXEQ N1 3.0) (Production 8 in Appendix B).

apxeq. Relation in the NCOMP schema.
Example: N1---apxeq--->N2.
Inverse: apxeq

arga. Relation in the ASYMREL schema.
Example: G0197---arga--->ONE (Figure 1).
Inverse: arga1.

arga1. Relation in the ASYMREL schema.
Example: ONE---arga1--->N1.

Arguments. Nodes that are included in relational structures as the objects and elements that are related.

ASSIGN

Assigned node. A node that is remembered as the value of a variable.

ASYMREL. Generic schema for specifying an asymmetric relation with any number of arguments. ASYMREL consists of (1) a relation node (e.g., NEXT, ONEONE) linked through a token relation to (2) a token node represented by a unique symbol, which in turn is linked through arga, argb, argc, . . . relations to one or more arguments.

Example: (ASYMREL NEXT G0197 ONE TWO) specifies the pattern
NEXT---token--->G0197
G0197---arga--->ONE
G0197---argb--->TWO (Figure 8).

Atoms. Single words used as names for constants, variables, functions, etc.

Examples: FOLLOWED, V1, UNBASE.

BASE. Schema whose main function is to provide eraseable memory not easily handled with bound variables in ACTP. BASE consists of the node BASE linked through ida, idb, idc, . . . relations to one or more arguments.

Example: BASE---ida--->ONE (Figure 20).

Bound variables. Variables having a currently assigned value.

CATEGORY. LISP function in STARTUP that (1) tags constants so they will be usable either as control nodes or constant nodes in patterns, (2) forms an isa link between the first term in the category list and each of the other members, and (3) makes it possible to use the list members as input to ACTP during a cycle.

Example: (CATEGORY (QUOTE (TITLE ADDTAG MAKE MAKE2 HOWMANY))). (Appendix D).

cnpt. Single-link relation used to specify the name of a node in the data base.

Example: A---cnpt--->OB0178.
Inverse: label.

Concept schema. Pattern consisting of a name, set of arguments, and a list of relations between pairs of arguments.

Condition. The "if" part of a production rule. Conditions consist of (1) no, one, or more control nodes; and (2) no, one, or more pattern specifications.

Condition test. An attempt to match the condition pattern specification to the corresponding nodes and links in the data base.

Constant. Name of a specific node in the data base.

Control node. A constant in the condition of a production that must be active for that production to be tested. Control nodes function as goals that produce selection of productions whose patterns will be tested.

comp. Relation in the GOALX schema.
Example: G0224---comp1...>?GROUP.
Inverse: compl1

Complex goals. Goals which cannot be achieved immediately and will need to be retrieved at a later time. Complex goals are stored in the data base by the function SETGOAL.

Cycle. A single loop through a set of productions during which (1) conditions of productions are tested in order until one of them is found true; and (2) the action of that production is performed, ending the cycle.

CYCLE. Function that tells the ACTP system to begin the process of cycling through PROLIST.

Data structure. Semantic network representing the information upon which the production system works--on which actions operate and on which the conditions of productions can be determined true or false.

EQUAL. Constant given as an argument to the function NCOMP to test for a single-linked quantitative relation of "equal" between two nodes corresponding to numbers in the data base.
Example: (NCOMP EQUAL N1 N2).

equal. Relation in the EQUAL schema.
Example: N1---equal--->N2.
Inverse: EQUAL.

Execute. Performing the action of a production.

False condition. A condition whose pattern specifications cannot be matched to the data base.

Free variables. Variables having no currently assigned values.

GENSET. LISP function that sets up an initial data structure.
Example: (GENSET OBJECTS) (Appendix C).

Goal. Objective that motivates performance of an action.

GOALX. Schema specifying goal information. GOALX consists of (1) a relation node, GOAL, linked through a token relation to (2) a token node represented by a unique symbol, which in turn is linked through action, pattern, compl, comp2 relations to corresponding arguments.

Example: (GOALX GOAL G0244 X FIND SIZE ?GROUP ?NUM) specifies the pattern

```
GOAL---token--->G0224
G0224---action--->XFIND
G0224---pattern--->SIZE
G0224---compl--->?GROUP
G0224---comp2--->?NUM (Figure 17).
```

Goal stack. Memory device for storing the previously current goal whenever a new complex goal is adopted. Operates on a "first on the stack, last off the stack" basis.

*GREAT. Constant given as an argument to the function NCOMP to test for a single-linked quantitative relation of "greater than" between two nodes corresponding to numbers in the data base.

Example: (NCOMP *GREAT N1 N2).
Inverse: LEQ

GREQ. Constant given as an argument to the function NCOMP to test for a single-link quantitative relation of "greater than or equal to" between two nodes corresponding to numbers in the data base.

Inverse: *LESS.

haspart. Relation in the PARTOBJ schema.
Example: CLIST---haspart--->TWO.
Inverse: ispart.

hasprop. Relation in the OBJPROP schema.
Example: ONE---hasprop--->FOLLOWED (Figure 24).
Inverse: isprop.

Input. ACTP or LISP commands typed in from the terminal.

isa. Relation in the OBJTYPE schema.
Example: ONE---isa--->NUMERON (Figure 24).
Inverse: memb.

ispart. Relation in PARTOBJ schema.
Example: ONE---ispart--->CLIST (Figure 24).
Inverse: haspart.

isprop. Relation in OBJPROP schema.
Example: FOLLOWED---isprop--->ONE.
Inverse: hasprop.

label. Single-link relation used to specify the name of a node in the data base.
Example: OB0178---label--->A.
Inverse: cnpt.

*LESS. Constant given as an argument to the function NCOMP to test for a single-link quantitative relation of "less than" between two nodes corresponding to numbers in the data base.
Example: (NCOMP *LESS N1 N2).
Inverse: GREQ.

LEQ. Constant given as an argument to the function NCOMP to test for a single-link quantitative relation of "less than or equal to" between two nodes corresponding to numbers in the data base.
Example: (NCOMP LEQ N1 N2).
Inverse: *GREAT.

Link. Labeled connections between nodes in the data base that denote relations between them.

List. Atoms or lists enclosed in parentheses.
Example of a list of atoms: (OBJCAT V4 OBJECT) (Figure 22).
Example of a list of lists: The entire production in Figure 22.

Match. Attempt to find a configuration of nodes and links in the data base that correspond to the pattern specification in the condition.

memb. Relation in the OBJTYPE schema.
Example: NUMERON---memb--->ONE.
Inverse: isa.

MEMORIZE. LISP function in STARTUP which tags constants so they will be usable either as control nodes or constant nodes in patterns.
Example: (MEMORIZE (QUOTE (COUNTED USED FOLLOWED BOUND SKIPPED SPECIAL))) (Appendix D).

NCOMP. Function that tests for single-link quantitative relations in the data base. (See APXEQ, EQUAL, *GREAT, GREQ, *LESS, LEQ.)

Network. See semantic network.

Node. Symbol denoting ideas or elements in a task situation.

OBJCAT. See OBJTYPE.

OBJPROP. Schema used to represent property relations.

ORDASSIGN. Schema used as an eraseable memory structure which can be removed by the function UNBASE. Consists of a top node (usually BASE) linked through ida, idb, idc, . . . to one or more arguments.
Example: ((ORDASSIGN BASE V21)).

partobj. Relation in the PARTOBJ schema.
Example: CLIST---ispart--->ONE.
Inverse: haspart

Pattern. See pattern specification.

pattern. Relation in the GOALX schema.
Example: G0224---pattern--->SIZE (Figure 17).
Inverse: pattern1.

pattern1. Relation in the GOALX schema.
Example: SIZE---pattern1--->G0224.
Inverse: pattern.

Pattern construction. Building a list of links and nodes that correspond to a concept schema.

Pattern matching. Testing whether a particular configuration of nodes and links can be found in the data base.

Pattern specification. A particular configuration of nodes and links that is to be matched in the data base.

POPSTACK. One of three special ACTP functions involved in the management of complex goals. POPSTACK (1) removes the current goal from the data base once it has been achieved, then (2) removes the top goal from the goal stack and makes it the current goal.
Examples: (See Productions 35 and 38 in Appendix B.)

PRINT. Special function for printing output at the terminal.
Example: (PRINT V1 V22) Production 55 in Appendix B).

Production. Conditional ("if-then") statement used to represent elements of knowledge in a production system. Consists of (1) a condition, and (2) an action.

Production rule. See production.

Production system. See ACTP.

PROLIST. List containing all the productions in a particular system.

PUTINV. Function in STARTUP that takes as its argument a list of constant pairs and makes each member of a pair the inverse of the other member.

Example: (MAPC (QUOTE PUTINV) RPAIRS). where RPAIRS is equal to ((LABL CNPT) (ISA MEMB) (HASPROP ISPROP) . . . .))

REBIND. A function that replaces the value of A by the value of B; B retains its value. A must be a variable; B can be either a variable or a constant.

Relations. Labeled connections between nodes.

Schema. See concept schema.

Semantic network. Knowledge represented as an interconnection of nodes and links in the data base.

SETGOAL. One of three special ACTP functions involved in the management of complex goals. Whenever a new goal is set SETGOAL (1) adds the current goal to the top of the goal stack and (2) adds the new goal to the data base using the GOALX schema.
Example: (SETGOAL GOAL VX1 XFIND SIZE ?GROUP ?NUM) Production 4 in Appendix B).

Simple Goals. Goals which can be achieved immediately and which are set by activating control nodes.

Special functions. Functions used for purposes other then building patterns.
Examples: PRINT, POPSTACK.

STARTUP. LISP function that informs the ACTP system of the variable names, constants, links, and so on, that will be used in a particular set of productions. (See Appendix D.)

TITLE. Name of category defined in STARTUP whose members can be used in providing input information during operation of the system.
Example: (CATEGORY (QUOTE (TITLE ADDTAG MAKE MAKE2 HOWMANY)) (Appendix D).

token. Relation in the ASYMREL schema.
Example: NEXT---token--->G0197 (Figure 8).

TRACE. LISP function that takes the names of other LISP functions as its arguments. These other functions are then "traced" whenever they are called during a cycle.

Trace. Providing a detailed report (called a "trace") of a function execution within a program. Primarily used as a debugging device.

True condition. Condition whose control node(s) is active and whose pattern specification(s) (if any) can be matched to the data base.

<u>type</u>. Relation in the ASYMREL.
Example: G0197---type--->NEXT.
Inverse: <u>token</u>.

<u>UNBASE</u>. Special function that removes the ORDASSIGN structure representing the current problem base.
Example: (See Production 55 in Appendix B.)

<u>Variable</u>. Symbol that can be assigned the value of different nodes in the data base.