Computational Intelligence, Volume 14, Number 2, 1998

USING DIAGRAMS TO UNDERSTAND GEOMETRY

ROBERT K. LINDSAY

Mental Health Research Institute, University of Michigan, Ann Arbor

This paper describes ARCHIMEDES-STUDENT, a computer program that constructs and modifies its own representations of diagrams from instructions supplied by a human who is demonstrating a theorem of geometry. The program's representation permits it to make inferences from its constructions and to find a justification for the conclusion of the theorem. It is argued that the sort of perceptual reasoning displayed by this program represents one important aspect of understanding because it relates the abstract mathematical theorem to knowledge of spatial relations. For humans this approach grounds abstraction in experience and thus provides a more compelling demonstration than a formal proof. Because ARCHIMEDES-STUDENT is a well-defined computer program, it provides a precise suggestion of how this aspect of understanding can be achieved.

Key words: diagrammatic reasoning, imagery, geometry, understanding, spatial reasoning, knowledge representation.

1. INTRODUCTION

Theorems of plane geometry are generally illustrated in textbooks by diagrams. For most students a diagram facilitates following or discovering a proof. The diagram also appears to be valuable, perhaps essential, for literally and figuratively seeing the meaning of the theorem and understanding why it is interesting. Indeed formal proofs often add little to the understanding afforded by diagrams coupled with informal explanation, particularly for people not trained in mathematics. This paper is an exploration of the computational processes that underlie such use of diagrams as aids to understanding geometrical propositions.

A proof in the modern, post-Hilbert sense is a sequence of statements in a precisely defined formal language composed of strings of characters from a finite alphabet. Each statement is justified by applying precise rules of deduction to preceding statements, such as axioms, previously proved lemmas, and theorems. Diagrams can have no essential place in this *strict* sense of theorem *proving* because they have no accepted syntax or semantic theory specified in the standard logical formalism. This is not an inherent limitation of diagrams and several recent efforts have attempted to fill this vacuum; see Barwise and Etchemendy (1990, 1992), Kaufman (1991), Shin (1991, 1992, 1995), Stenning and Oberlander (1992), Barker-Plummer and Bailin (1992), and Wang (1995), whose work is discussed further below. Although the several formal theories of diagrams that have been thus proposed are not fully general, it is fairly certain that no substantive obstacle stands in the way of further developments.

Nonetheless, diagrams, even without a formal theory, clearly serve for humans a useful heuristic and pedagogical purpose that extends well beyond geometry textbooks, and includes the understanding of nongeometric mathematics, of physical principles, and even of informal biological and social theories. Diagrams and other graphical illustrations are ubiquitous conveyers of meaning and intuition even without a formal theory. There is much anecdotal evidence for this claim, as well as a number of systematic studies of the use of visual images in scientific discovery (e.g., Miller 1984; Thagard & Hardy 1992).

In contrast to the formal proofs alluded to above, geometry is generally introduced to students by a combination of diagrams and quasi-formal "textbook proofs." Although textbook proofs lack the rigor of fully axiomatized formal proofs, they maintain the sentential, sequential proof structure while permitting more flexible use of natural language. Thus they

Address correspondence to the author at the Mental Health Research Institute, University of Michigan, 205 Zina Pitcher Place, Ann Arbor, MI 48109; e-mail: lindsay@umich.edu.

employ flexible forms of reference (e.g., "the preceding," "see example 3," "similarly," etc.). They also may omit explicit justification of facts that are true in the accompanying diagram, such as that a construction must yield a particular ordering of objects, that a point must lie within a figure rather than outside it, and so forth. The sorts of arguments that have been found pedagogically most useful are a hybrid of less than rigorous verbal arguments and less than rigorous diagrammatic illustrations.

Some mathematicians and teachers have attempted to produce textbook proofs that use a bare minimum of sentential representation and argument. Nelson (1993) calls these "proofs without words" although they are not proofs in the strict sense. Furthermore, they are not "without words" for most contain some verbal symbols; those that do not are seldom comprehensible without additional verbal cues from a teacher. Because the emphasis of such exercises is both to demonstrate to the student that a conjecture is correct and to give it some intuitive interpretation, they are better called "diagrammatic demonstrations." They are compelling presumably because we are adapted to a world where geometric and physical laws prevail, and thus our nervous systems provide computational mechanisms that are efficient at reflecting these laws. A diagram that illustrates a mathematical relation by showing in essence that it is an accurate reflection of the behavior of objects in space provides an understanding because it grounds the abstraction in our customary experience. If the abstractions, thus interpreted, are consistent with that experience, they are understood and believed. If not, they are either disbelieved or not understood, or else other means, generally much more abstract and indirect, must be brought to bear. The latter would be the case, for example, with quantum electrodynamics for which there may be no experience-based model that can play the role of understanding mediator.

Rather than discuss the general case of experience-based cognition, understanding, and learning, this paper addresses the more narrow issue of the role of two-dimensional diagrams in expressing mathematical concepts. Specifically, the paper is limited to a particular investigation of the role of geometric diagrams in the *understanding* of geometric propositions, such as theorems. The proposal will be illustrated by a detailed description of how a particular computer program addresses this particular problem. The ARCHIMEDES-STUDENT program to be described "understands" a demonstration in the sense that it can construct an appropriate diagram from propositional instructions and can verify that the conclusion of the demonstration is true because it follows from knowledge the program has about certain types of objects and their spatial relations.

When observing plane geometric diagrams for the purpose of thinking about geometry, we generally focus on their spatial properties and choose to ignore such features as texture, color, perspective, depth, and most other features of pictures and images. Of course, the spatial properties of real physical objects are not inherently visual; unlike visual properties such as color, they are accessible through tactile or auditory channels even to the blind. Thus, although the full range of visual features are not fully exploited in geometric reasoning, diagrammatic representations are in another important sense more general than visual images.

2. RELATED WORK

Artificial intelligence research on theorem proving began with the Logic Theorist of Newell, Shaw, and Simon (1957). In their work theorem proving was characterized as a search for a sequence of syntactic transformations that will convert givens into a conclusion. Other early work on automatic theorem proving also concentrated on nongeometric mathematics and was thus restricted to the discovery of rule sequences without the assistance of a diagrammatic interpretation. Some of this work departed from the Logic Theorist model of heuristic search

of a problem graph, and explored the use of syntactic manipulations that offered little intuitive assistance to a human, such as the resolution method (Robinson 1965) and model-elimination (Loveland 1968).

A variety of means have been used to provide for computers an understanding of geometry with the aid of diagrams. One was the use of diagrammatic information to aid the discovery of sentential proofs. The Geometry Machine (Gelernter 1959b; Gelernter, Hansen, & Loveland 1960) attempted to discover axiomatized, formal proofs by searching a problem graph of possible deduction rule applications, but augmented this search with a heuristic that rejected attempts to prove any statement that was false in a diagram supplied (as a set of point coordinates) to the program. Additionally, it employed a rigorous method of detecting "symmetries" in the set of statements. That is, it could detect statements that were equivalent under a substitution of variable names (Gelernter 1959a). The resulting propositional-diagram hybrid succeeded in discovering proofs, but clearly failed to use the full range of assistance available from diagrams.

Others, following in the tradition of Gelernter, have also coupled a heuristic search of statements and rules with a diagram representation that can be used to guide the search. Barker-Plummer and Bailin (1992) constructed a theorem-proving system for set theory that extracts information from its representation for diagrams, provides a decomposition of the theorem into lemmas, and orders them into an overall proof plan. The Koedinger and Anderson (1990) model embodies diagrammatic information and they find that this substantially improves performance, and also makes it more similar to the problem-solving strategy of experts. They explicitly provide (rather than produce by a program) a set of prototypical diagrams and associated information that serve as a classification of problems. This classification aids theorem proving by focusing the program's attention to the appropriate category with which specific hints are associated. Diagrams per se are not available to the programs, and cannot be modified or used to form new conjectures, nor are geometric constraints used to draw inferences.

Still other projects have employed diagrams and diagrammatic information in a variety of ways in attempts to embody in computer models some of the methods that intuitively seem to be those employed by humans, as opposed to those that seem unlikely to be (for example, see Larkin & Simon 1987; Novak 1977; McDougal 1993; Narayanan 1992; and Glasgow & Papadias 1992). Although some empirical work has been done to test the claim of psychological validity, generally the models at present can make only limited experimental predictions.

Another way to bring diagrams into a programmed model of geometry is to formalize diagrams in a style similar to the style of formalized language, as mentioned above. A number of recent efforts have been directed toward this goal with some success. Kaufman (1991) devised a formal theory that can represent both physical and geometric properties of some simple objects and can be used to prove such things as that a string can be used to pull but not to push. Shin (1995) has developed a formalization whose rules refer to canonical diagram elements and define manipulations of them. Stenning and Oberlander (1991, 1992, 1995) also introduced algorithms for manipulating Venn diagrams. Barwise and Etchemendy (1990, 1992) have developed a programmed system that helps students reason about both first-order logic and diagrams by integrating propositional statements and graphical interpretations in a common system, accessible to the students by a graphical interface and based on precise rules for manipulating the representations. The system has been pedagogically useful, suggesting again the value and perhaps necessity of diagrammatic interpretations in the understanding of propositions. Wang (1995) has introduced a more general descriptive language for diagrams and has devised a semantics for the language that maintains important logical properties such as consistency and soundness. This permits the representation of diagrams in a formal

calculus, although the diagram elements are not perceptual objects. None of these theories specify precisely how objects can be recognized in a real diagram or a digitized picture, but work from a description of a diagram.

The work of Chou (1988) (building on work of Wu 1978 and Ritt 1938) introduced a combination of diagram and algebra to produce a theorem prover of scope and power. A diagram for a proposition is drawn and one point is assigned specific numerical coordinates (typically the origin) to fix the location of the diagram, and a line through that point is selected as determining the x-axis so that points on it have y-coordinates of zero, thereby fixing the orientation of the diagram. Variables are introduced for the x-coordinates of the other points on that line and for both coordinates for those other points that can be chosen arbitrarily. Finally, other variables are introduced for point coordinates that are dependent on the original assignments and variables. The hypotheses and conclusion of the theorem are stated in terms of a set of simultaneous equations that relate all of these variables. An algebraic algorithm is used to determine if the system of equations has a solution; if so, the proposition is a theorem. An important side effect of this procedure is that it identifies special conditions that must hold for the theorem to be valid. Matsuyama and Nitta (1995) also extended Wu's algebraic method by integrating it with logical problem graph reasoning methods to widen the scope of applicability beyond Chou's system. These methods use a diagram in a substantive way, but the algebraic inference procedures do not embody an intuitive spatial semantics that makes the deduction clear, at least to the nonalgebraist.

Still another approach to exploring the cognitive role of diagrams is to manipulate representations of diagrams directly, rather than work from descriptions of classes of diagrams. One version of this approach is the work of Furnas (1992) who defined problem solving as a series of applications of rules that find patterns in bitmaps and transform them into other patterns, thereby requiring no sentential reasoning at all. Edwards (1996) introduced a similar idea. Anderson and McCartney (1995, 1996) defined a system of picture element manipulation processes that can be combined according to explicit rules into programs that solve problems strictly by manipulating pixel arrays.

A different variation is the use of simulation processes that operate on bitmap representations of diagrams. This is an intuitively natural approach that has been discussed informally for many years, and has been the basis of some specialized models (e.g., Funt 1976, 1977, 1980; Larkin 1983; Gardin & Meltzer 1989; and Shrager 1990). There is a body of evidence supporting the intuitive notion that people reason, at least in some cases, by "running" a mental model of a situation, even though the model may not be veridical (Johnson-Laird 1983; Yates et al. 1988; Hegarty 1992; Hegarty & Ferguson 1993; Hegarty & Just 1993; Clement 1994).

Imagining an event and watching it unfold in the mind's eye is a common experience of most people. These commonplace observations have frequently been cited as the basis for at least some aspects of human thought. Many have suggested a motion picture analogy, wherein one observes a movie in the head. However, the experience is not limited to the recall of specific past events, but can be used to run thought experiments that envision how an event might play out, or to plan a method of accomplishing something, such as building a house or going on a trip. Such imaginations can in fact be counterfactual, as in dreams or when subjects incorrectly imagine how a physical event would unfold. For example, many people image that a stone swung in a circle on the end of a string would fly directly away from the center of rotation if the string would break. This and similar errors have been experimentally documented (see McCloskey, Caramazza, & Green 1980; Caramazza, McCloskey, & Green 1981; McCloskey & Kohl 1983; and McCloskey, Washburn, & Felch 1983). These observations make clear that the imagination is not simply a replay, but is under cognitive control, that is, it is *cognitively penetrable*, to use the terminology of Pylyshyn (1980, 1984).

It does not follow from these facts, however, that mental imagery is *entirely* penetrable. There may well be certain aspects of it that cannot be altered by choice or the application of tacit knowledge (see Pinker 1985, p. 44). Furthermore there may be aspects of images that provide important inferences if allowed to proceed "automatically" without being explicitly altered, even though they are alterable. This is the case for nonpictorial representations as well. Consider a chess player generating a tree of possible moves. Clearly the direction of growth of this tree is under the cognitive control of the player, but it is constrained to be a tree of possibilities consistent with the rules of the game. There are well-studied perceptual phenomena, such as apparent motion, which show that the perceptual system is primed to interpret ambiguous stimulus conditions in certain ways, corresponding to the "most likely" interpretation, even when that interpretation is not correct (Shepard & Zare 1983; Shepard 1994). Conceivably there may be similar biases in favor of certain imaginal manipulation of mental contents, even in those cases where a variety of results could be imagined. Determining which aspects of images are and are not penetrable and what these biases are is an important but difficult theoretical and empirical problem.

Without a precise model of how images are represented and how they may be controlled cognitively, the motion picture or thought-experiment model can contribute little to a scientific understanding of thinking. The present work attempts to supply such a model by exploring this approach in the context of employing diagrams to understand mathematical ideas. Although the simulation approach differs from heuristic search in a space of propositions and from translation to a logical calculus, there is no claim that simulation processes are antithetical to other uses of diagrams, or that propositional reasoning is not involved in understanding geometry. Rather it seems likely that humans employ a wide variety of methods acting in concert to achieve their ends.

Closely related to the work described in this paper are THINGLAB (Borning 1981) and the geometric constraint engine of Kramer (1992). Although it did not employ a general simulation method, THINGLAB provided means for (1) defining classes of geometric (and other) objects, (2) specifying constraints among them and among their parts, and (3) defining special methods for maintaining each constraint as a user made alterations to the diagram through a graphical interface. In THINGLAB the emphasis was on supplying a user with an extensible system that could be used to explore the results of change subject to constraints. The present work also employs constraint maintenance, but does so within a simulation paradigm. In the work described here the emphasis is on devising a program that can itself recognize and record the effects of actions prescribed by a human instructor. Although the goals of the THINGLAB project differed from the present work, the constraint maintenance functionality required is similar. In a loose sense, the present work attempts to model what the THINGLAB *user* does, but only in the context of geometric problems.

Kramer's method establishes a series of movements of objects in three-dimensional space that will bring them into conformity with a given set of constraints among points, lines, and figures (for example that two points have zero distance between them, or that a line is tangent to a circle). The method does not rely on the solution of algebraic equations, but analyzes the degrees of freedom each component has and systematically chooses a plan that will positionally fix objects in sequence.

3. DIAGRAMMATIC DEMONSTRATIONS

A diagrammatic demonstration is a series of steps that involves in a substantive way the construction and observation of a diagram to illustrate a proposition. For example to demonstrate that the area of a parallelogram is equal to the product of the lengths of its base

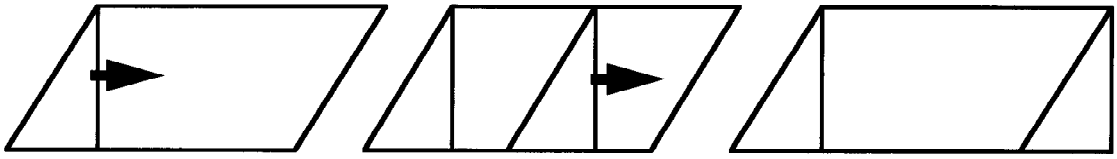

FIGURE 1. The area of a parallelogram is the same as the area of a rectangle of the same base and altitude.

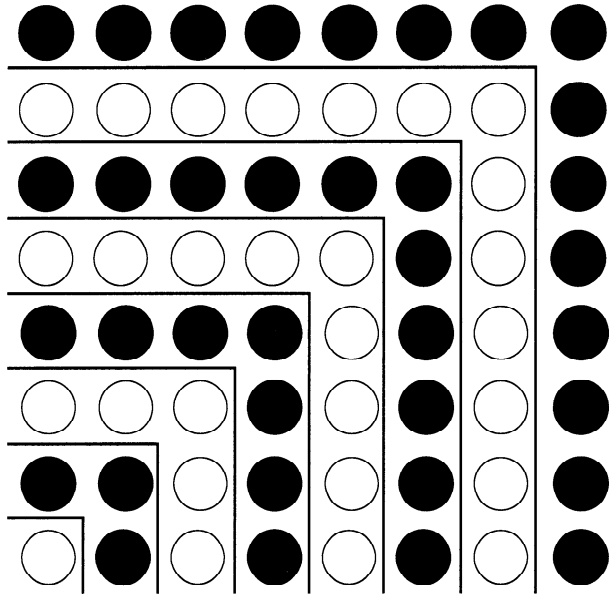

FIGURE 2. $1 + 3 + 5 + \cdots + (2n - 1) = n^2$ (after Nelson 1993).

and altitude (see Figure 1), a teacher might draw a parallelogram, construct a perpendicular from one vertex to the opposite side as shown, "slide" the resulting triangle to the other side, point out that a rectangle has now been formed, that the rectangle has the same area as the parallelogram because the rigid translation of the triangle did not alter area, and if the student knows the formula for the area of a rectangle, the correctness of the target proposition will be apparent.

Such demonstrations are not limited to geometry; for example number theory is a favorite topic as well. Figure 2 is one of many examples found in Nelson (1993). It illustrates the arithmetic proposition that the sum of the first n odd integers is n^2. This demonstration is slightly more subtle, and it may be necessary for the teacher to provide additional hints. In this example shading is used to illustrate that increasing n by 1 adds a half "border" to the area of the previous square, and the number of additional circles is always the next odd number. The demonstration turns on the geometric property that the area of a square with sides of length n is n^2, with the circles standing proxy for length units (small squares would perhaps be more precise, but circles usually suffice and make their *numerosity* more prominent).

Diagrammatic demonstrations in most cases employ verbal symbols to state the con-

jecture, as names for components of the diagram needed to specify construction sites and make other references, and as algebraic or verbal statements of intermediate propositions. Nonetheless, the nonverbal aspects of the diagram play substantive roles. In general the diagram makes use of the spatial relations inherent in the paper on which they are drawn. For example, placing A to the left of B and B to the left of C necessarily results in A being to the left of C; this transitivity of "to the left of" is captured by the topological properties of the paper. Similarly, geometric properties such as lengths of line segments are recorded and "enforced" by the properties of the paper.

It will be noticed that most of the creative work of a demonstration is done by the teacher. By limiting the task to understanding demonstrations provided by others, the additional complications of theorem invention and proof discovery are avoided. The remaining task however is by no means trivial. The constructions must actually be performed and recorded, the student must understand how to perform specified alterations (rigid translation in the example of Figure 1) in such a way that geometric constraints are maintained, new facts that result from constructions and alterations (such as the materialization of a rectangle) must be noticed, it must be confirmed that some objects are composed of others (in Figure 1 that the original parallelogram is partitioned into a quadrilateral and a triangle and that the final rectangle is partitioned into the same quadrilateral and an equivalent but displaced triangle), and a search may need to be conducted through known general algebraic or geometric knowledge (such as transitivity of equivalence relations) to find support for the putative conclusion.

4. THE ARCHIMEDES REPRESENTATION

Any representation system must specify three things: the syntax of the basic data structure formats in which information is stored, precise definitions of the processes that may be used to create specific models, and precise definitions of the processes used to access a model. The first component of a representation has been given various names by those who attempt a clear distinction between it and the processes. I have used *map* (Lindsay 1961), Larkin and Simon (1987) use *data structure*, and Tabachneck, Leonardo, and Simon (1994) use *format*, but none of these terms clearly embodies the distinction sought and none has been widely adopted. I have more recently used *representation-proper* (Lindsay 1988) and will continue to do so in this paper, although it is perhaps awkward.

Failure to specify the process components of a representation invariably leads to confusion. This is amply illustrated in the "imagery debates" that have been pursued for years (Kosslyn 1976, 1993; Kosslyn & Hatfield 1984; Kosslyn & Pomerantz 1977; Pylyshyn 1973, 1981; Shepard & Cooper 1982; Glasgow 1993). In the present work an attempt has been made to identify precisely a set of *construction processes* that form a basis set from which all constructions are made by composition. The set of construction processes therefore provides a functional description of some of the abilities of ARCHIMEDES-STUDENT. This is an attempt to divorce the functional description from the computational implementation, in the sense that if the underlying representation-proper is altered and a different computational architecture is used (such as a neural net or cellular automaton) the behavior of the system might become more (or less) efficient, but the analysis of the representation problem would be unchanged. Correspondingly, there is a set of functionally described *retrieval processes* that access the representation-proper to determine properties of the instance currently represented.

A specific physical situation or event is represented by a *model*, whose structure obeys the formation rules of the representation-proper and which can be altered and accessed by the construction and retrieval processes. For example, architectural drawings are a knowledge representation system whose construction processes use T squares and pencils and whose

retrieval processes use human vision, supported by knowledge of the notational conventions of the system. We of course do not have a computational description of these processes. The representation-proper is a set of certain configurations of graphite on pieces of paper (those that contain lines, text, etc. in certain but not all arrangements). The object at 1600 Pennsylvania Avenue in Washington, D. C. could be modeled on a specific set of pieces of paper with specific placements of graphite, called "plans and elevations of the White House."

ARCHIMEDES-STUDENT is an application of a more general system called ARCHIMEDES that defines a representation system for diagrams. Thus, for example, the White House plans might be represented by ARCHIMEDES. The construction and retrieval processes in this case would be List Processor (LISP) functions, and the representation-proper would be the abstract set of LISP structures that could in principle be created with these functions. An ARCHIMEDES model of the White House plans (not of the White House) would be a specific set of LISP expressions that preserve some of the information from the architectural drawings, which in turn preserve some of the information from the physical structure in Washington.

ARCHIMEDES-STUDENT combines ARCHIMEDES with procedures for following a sequence of steps provided by someone attempting to demonstrate a geometric proposition such as a theorem. A demonstration is presented to the ARCHIMEDES-STUDENT program as a series of demonstration steps consisting of *constructions* intermingled with *propositional-notations*, *instructions-to-observe*, and a *goal statement*. The goal statement is a statement of the theorem/relation to be demonstrated. The constructions are instructions on how to construct and alter the diagram to illustrate the theorem/relation. One type of construction specifies that a new segment or other component of the diagram is to be created with a specified relation to already created components. A second type of construction specifies *situation constraints*. These are properties that the program is to attribute to already existing components, such as that a specific segment must remain of fixed length or that a specific object is rigid. Situation constraints are in contrast to the constraints that follow from spatial properties (topological and geometric relations) that are inherent in the representation scheme itself and are always enforced. A third type of construction is a *simulation construction*, in which one or more objects is moved with the remaining components being altered as required by the spatial and situation constraints of the problem. Propositional-notations permit the explicit introduction of algebraic and geometric knowledge that has not been previously available to the system, for example, a formula for the area of a figure. Instructions-to-observe are attention-focusing directions that limit the program's search in cases where that search is costly.

Although the program can work from a script, it is generally more convenient to use it interactively. After each construction step the program creates or alters its representation and in the process creates names for the points and other components that it defines (the user could supply these names if desired), and these names (single characters for points, for example) can then be used to refer to the points when specifying future steps. The program does not allow the user to refer to objects by pointing, although this could be readily implemented.

The ARCHIMEDES-STUDENT program represents a diagram by means of three structures in computer memory: a two-dimensional array of points with integer-valued coordinates, an explicitly connected network of these points, and a set of frames encoded as LISP structures. Each point (pixel) constructed is placed at a specific location in a two-dimensional array by assigning it specific integer indices (coordinates). Each named segment is represented as a list of the points lying between its endpoints. Because the indices are integers, the pixel array is quantized. This means that line segments are not idealized smooth lines except when they run along a principal horizontal, vertical, or diagonal axis. The segment point indices are computed by the Bresenham/Pittway MidpointLine algorithm (Foley et al. 1990). "Anti-aliasing"—the addition of extra pixels to make a display of the segment less jagged in appearance to a human viewer—is not used. The algorithm determines a unique but not

necessarily straight sequence of pixels between any two points, but the movement of one end point by even one pixel, even in the direction that best approximates extending the length of the line, could cause other segment points to move as well. The representation is also used to produce a display for the user to observe, but the display is not in any sense directly observed by the program, which can only access its representation of the diagram.

The points of the array are also explicitly linked, using the usual LISP conventions, by associating each point with a list of its neighbors. This information is redundant in that it can be derived from coordinate information, but is included to make some connectivity computations possible without arithmetic operations.

The program can represent other geometric objects in addition to points and line segments, namely angles, triangles, squares, rectangles, parallelograms, quadrilaterals, and polygons. Each geometric-object type has an associated LISP-structure type. These are data types of Common LISP that define classes of representations with given properties (slots) that can be assigned specific values. Each specific geometric object of which the program is aware in a particular diagram has a *name*, and that name is associated with a specific instance of a LISP structure for its type. The LISP structure contains other information (via slots and values) about any situation constraints that have been imposed on the geometric object. Associated with the LISP structure of the name is an instance of another LISP structure (the *gridobject*) that contains information about the specific components of the geometric object; these components are similarly described and ultimately they refer to specific point coordinates. To illustrate, the LISP-structure formats for a square are shown in Figure 3 along with a specific instance. Status and constraint slots record situation constraints. Status slots contain one of a set of symbols, such as *fixed* or *arbitrary*. Situation slots contain an arbitrary LISP predicate; a new constraint is added by combining it by conjunction to the old one, thus the default (always true) predicate is "(and t)." LISP structures for other geometric objects are defined in an analogous manner.

Each of the geometric objects is defined in terms of its component points and segments, and segments are defined in terms of their endpoints. Each endpoint of a segment, and hence each vertex of another geometric object, is named. The collection of LISP structures that are part of a particular representation (model) determine the set of geometric objects of which the program is "aware" and for which it has names. This collection includes every geometric object whose construction the user has instructed. It also includes other geometric objects that may arise indirectly by construction. For example constructing the diagonals of a parallelogram indirectly creates eight triangles. Humans generally, but not always, *notice* these indirectly constructed geometric objects as a matter of course. The program does likewise by examining its representation after each demonstration step and augmenting its inventory of geometric objects. However, unlike for humans whose visual perception is extremely efficient at such noticing, this step is sometimes computationally expensive for the present implementation of ARCHIMEDES, and can produce a profusion of objects that are irrelevant to the demonstration. Consequently it is possible for the user to instruct the program not to notice everything automatically. In that case it is necessary for the user to use an instruction to *observe*—that is, confirm the existence of—those geometric objects that will turn out to be essential to the demonstration.

For many computations, most notably the simulation constructions, the points of a segment between its endpoints are not directly involved. All simulations are movements of named points, with segments and other geometric objects "going along for the ride." By this I mean that as the endpoints are moved the segment points are reconstructed by the algorithm when needed. Thus the jaggedness due to the coarseness of quantization has no direct effect on these computations. On the other hand, some computations involve edge following and marking. For these computations each specific connecting point is visited.

A **squarename** is a structure recording information about a named square; it has the following features with values of the indicated type:
(**displayform** of type string)
(**gridsquare** of type gridsquare)
(**sizestatus** of type symbol)
(**sidelengthconstraints** of type predicate)
(**orientationstatus** of type symbol)
(**orientationconstraints** of type predicate))

A **gridsquare** is a structure recording information about a square; it has the following features with values of the indicated type:
(**names** of type list-of-squarnames)
(**vertex-1** of type pointname)
(**vertex-2** of type pointname)
(**vertex-3** of type pointname)
(**vertex-4** of type pointname)
(**side-1** of type segmentname)
(**side-2** of type segmentname)
(**side-3** of type segmentname)
(**side-4** of type segmentname)
(**sidelength** of type real-number)
(**side-1**-bearing of type real-number))

Example: Lisp symbol abcd might be bound to the following structure
(displayform "abcd")
(gridsquare alpha)
(sizestatus 'fixed)
(sidelengthconstraints (and t))
(orientationstatus 'arbitrary)
(orientationconstraints (and t))

Example: gridsquare alpha might be bound to the following structure:
(names (abcd))
(vertex-1 a)
(vertex-2 b)
(vertex-3 c)
(vertex-4 d)
(side-1 ab)
(side-2 bc)
(side-3 cd)
(side-4 ad)
(sidelength 12)
(side-1-bearing 30))

FIGURE 3. Examples of name and grid structure representations.

The object types such as triangles are devices that permit expressing certain generalizations. For example, a right triangle object will be associated with any right triangle, independent of position, orientation, and scale. This is important to enable the discovery and expression of generalizations, as described in Section 9.

5. CONSTRUCTION AND RETRIEVAL PROCESSES

Construction processes modify the representation of a diagram and retrieval processes access information about a diagram from its representation. These sets of processes are intertwined, in that performing a construction often requires obtaining knowledge about the current diagram and construction processes employ retrieval processes to obtain this information. In writing the ARCHIMEDES program, an attempt has been made to distinguish these two classes of process and to identify them by name. Each process is a LISP function, although in general they are executed for their side effects (altering the memory state) rather than their value.

The names of the functions have been chosen to be descriptive. By convention, most construction process names begin with "construct" and most retrieval process names begin with "retrieve." However, to avoid excessive awkwardness in names these conventions have been relaxed as follows. Retrieval functions that "observe" (examine the representation to verify a statement provided by the demonstrator) begin with that prefix rather than "retrieve." Processes that "erase" (by removing parts of a representation) begin with that prefix rather than with "construct." Composite processes that "notice" (both examining the representation and constructing newly found geometric objects) begin with that prefix. Construction processes that simply record propositional information use the prefix "remember." Additionally, retrieval processes that test for a condition and evaluate to *true* or *false* (that is, processes that are predicates) use the prefix "p-" reversing the LISP convention of ending predicate names with the suffix "-p."

If a demonstrator uses the letter *a* as a parameter, this evaluates to pointname LISP structure, to which the function is applied. This is the standard form of reference to geometric objects named by the program. A second form of reference is a gridobject. For example, the pointname that is the value of the symbol *a* has an associated LISP structure (the "gridpoint") that contains information about the array representation of the geometric object, for example its integer coordinates. The demonstrator can refer to this geometric object by writing (*locus a*), but this is in general unnecessary as the pixel structures are used only internally.

For each object type there is a process that creates an instance, another that copies an instance, and another that erases an instance. For example, (construct-and-name-point *integer integer*) creates a pointname using the next unused character as a name and associates it with a newly constructed gridpoint associated with the indices given as arguments. The process (construct-straight-path *gridpoint gridpoint*) finds and marks the pixels between the arguments and associates the list with the appropriate segment name. There are processes that create segments perpendicular or parallel to a given segment through a given point; these use retrieval processes that test for the appropriate relationship. For example, (construct-and-name-perpendicular-from-pointname *segmentname pointname direction stop-predicate*) creates a segment that begins at *pointname* (which is on *segmentname*) and continues in *direction* until *stop-predicate* evaluates to true. The process (construct-square-on-segment *segmentname direction*) uses the preceding to construct a square with *segmentname* as one side, where *direction* determines on which side of *segmentname* the square will lie. There is also a set of construction functions that appropriately record situation constraints, such as (construct-unfix-pointname *pointname*).

The function (simulation-construction) initiates attempted alterations using the method described below. The alteration to be attempted is initialized by one of a number of functions that can be evaluated before calling (simulation-construction). For example, (rotate-rigid-triangle *trianglename pointname clock-direction stop-predicate*) establishes a process that will apply a movement to one vertex of *trianglename* in the direction that will rotate the triangle about *pointname* in either the clockwise or counterclockwise direction until *stop-predicate* is true. Note that the direction of displacement may change as the triangle rotates. There are similar setup functions for other types of movements.

Other construction functions handle propositional information. For example, (remember-goal-proposition) records the goal of the demonstration, and (remember-area-equivalence *list-of-objects list-of-objects*) records the fact that the total areas of the objects on the two lists are equal.

The noticing functions make use of a process that finds all closed paths, that is, lists of segments that begin at one point and tie together into a sequence that ends at that point. Other functions use this inventory of closed paths to find segments, triangles, etc. by testing to see which paths meet the requirements of that type; if no LISP object exists for such discovered objects, they are created and named appropriately.

Observing functions are supplied with lists of pointnames. If these are connected in a closed path, the appropriate LISP objects defining that object are created (unless they already exist); if they are not so connected, the function reports a failure to observe. There are also observing functions for propositional information. These may involve making inferences based on the previously recorded set of facts. For example, if area equivalences between A and B and between B and C have been recorded, this will confirm the area equivalence between A and C and record it. If the function is unable to confirm the equivalence, it will report a failure to observe.

These functions are those that are required by the demonstration to be explained in this paper. More details about how they operate are given in the illustration. There are other functions in the system that are not involved in the examples and are not further described in this paper.

6. IMPLEMENTATION OF COMPUTATIONS

Varieties of skill and knowledge have been represented in a variety of ways in order to facilitate different styles of computation for the different skills. For example, translations and rotations that preserve metric properties are readily done arithmetically because arithmetic is computationally efficient on conventional computer hardware and a model of the Cartesian plane combined with the conventional distance and angle metrics is a good representation of geometric properties. Other computations are less efficiently done by arithmetic. Some of these, such as discovering enclosures, can be done by path following and by using the point network representation. Other knowledge requires that information must be associated with particular geometric objects in a diagram, such as properties that define its type (e.g., triangle), naming conventions, hypothesized properties that must be maintained (such as the equilateral property), and hypothesized relations to other geometric objects (such as congruency). This information is efficiently represented with the LISP structures.

Even with this variety of representations there are certain useful processes that could better be done by other means, means that could be much more efficiently carried out on another type of computer, such as an array processor. For example, one way to infer that a boundary is closed would be to use a "visual routine" of the sort suggested by Ullman (1985). Here is one such procedure, based on "color spreading." Find three adjacent boundary (black)

pixels and select two nonboundary (white) pixels adjacent to the central black one but not immediate neighbors of one another; color one red and the other green. Then iterate, coloring the neighbors of any red pixel red if they are white or red, but not if they are black, and similarly for green pixels. Stop if a red pixel is adjacent to a green pixel in the north, east, south, or west direction; conclude "not closed." Else, stop when no pixel changes to red or green during an iteration step or when all pixels are black, red, or green; conclude "closed." Note that the structure of the Cartesian plane is exploited by the adjacency calculation, but not arithmetically. This process would be very fast on a parallel array processor, whereas a path tracing process would not be aided by parallel operations.

The construction and retrieval processes have been defined in terms of what they accomplish with respect to a diagram. For example, the predicate p-equal-length is to respond affirmatively if and only if the two segments have the numerically same "length." There are any number of ways in which the process might be implemented. The implementation of course depends on the representation-proper. If a segment's "length" is stored as a digitized real number in a memory location pointed to by the segment's name, then the two lengths could be retrieved and compared numerically. If segments have associated endpoints that in turn have associated coordinates (as with ARCHIMEDES), then the lengths can be computed by a metric formula (such as Euclidean distance) and compared numerically. Another representation might require the counting of pixels. However, these options do not exhaust the possibilities. For example, lengths could be represented in such a way that the segment representations could be "moved" to one another and 1–1 correspondence tested; this computation would also work with nonstraight curves. Still other representations might employ nonstandard digital hardware or nondigital hardware. Of course neural representation must be possible, although there is at present little knowledge of what form it might take.

These different representations and different physical implementations will in general lead to quite different computational speeds. Short computational time is essential to the success of a model of diagrammatic reasoning. The computations must be sufficiently fast that the potential efficiency of constraint-based diagrammatic representation is realized if this idea is to offer any assistance with the frame problem (Lindsay 1988). Furthermore, the relative time as well as the overall time of certain computations, and how time increases with problem complexity, must be appropriate if a model is to serve as a detailed and useful psychological theory rather than only as an instance of artificial intelligence or as a general cognitive theory. The ARCHIMEDES model does not directly address these efficiency issues. Indeed there are certain processes in their present implementation on a conventional computer that are clearly much less efficient than human perception and cognition. For example, noticing a triangle is based on a search algorithm that is slower and less direct than human perception. This does not automatically invalidate the psychological relevance of this work however. A parallel array processor or an efficient machine vision program may someday make this noticing process efficient and bring the speeds of the various processes into proper registration with human abilities. In the meanwhile, the model proposes that certain processes are sufficient for its diagrammatic understanding tasks, and that they are necessary if the representation-proper is composed of the postulated geometric objects and relations. This is possible because the processes have been specified at the level of geometric objects.

7. SIMULATION CONSTRUCTIONS

As construed here, simulation has four aspects: (1) a *particular* instance of a situation, rather than an abstract description of a class of situations, is represented; (2) the parts of

the situation interact according to explicit causal laws; (3) the behavior is restricted to obey certain constraints; and (4) the process is incremental.

The most straightforward implementation of this process would be to simulate physical situations by employing the classical methods of mathematical physics. After all, the real number line is a model of one-dimensional space, including properties such as continuity, density, order, and direction. Vector algebra introduces dimensionality and permits the definition of geometric concepts, such as distance, as functions of vectors. The laws of Newtonian mechanics supplement this mathematical description of space with laws that describe the acceleration of mass under the influence of forces.

The problem of computational complexity, however, makes this approach, at least in its most general form, impractical. However, we can restrict the analysis to "critical points," such as endpoints of line segments and intersections of curves, and interpolate the connecting lines and curves. This immensely simplifies the simulation of the behavior of points interacting under the effects of connectivity. In this way, basic spatial relations may be captured by a simulation on contemporary hardware and are conceivably done in brain circuitry.

However, the imposition of additional situation constraints, such as requiring certain pairs of distances to remain equal or requiring that a point remain on a given curve, adds new computational burdens. Such constraints are essential in the statement of theorems and the representation of problems and puzzles. As noted, these situation constraints are in addition to the constraints of space and physics that the representation is supposed to embody. In the general case, adding constraints within the mathematical framework discussed will lead to sets of nonlinear equations for which there are no general solution methods that are polynomial time in complexity. There are alternatives to finding closed algebraic solutions, however. Kramer (1992) uses degree of freedom analysis to create a system that is able to plan a sequence of movements that will bring a set of objects into compliance with a collection of constraints of the sort employed here, even though they begin in an arbitrary configuration of noncompliance. His system is computationally efficient.

The present problem, however, is even easier because it is computationally much less expensive to *maintain* constraints under incremental changes than it is to discover a configuration that obeys them. Thus, once a diagram has been constructed from a propositional description, it is possible to forgo repeating that task while generating future legal states with a high degree of accuracy and much less computation. Modifying diagrams incrementally provides an envisionment of the geometric system in the sense that qualitative simulations of kinematics (Forbus 1984) and dynamics (de Kleer & Brown 1984) do for simulated physical systems. The envisionment may then be used in problem solving and conjecturing by achieving or failing to achieve certain states.

The simulation-construction function of ARCHIMEDES works in an incremental fashion by making prescribed movements of specified points and then checking to see if the situation constraints are violated. If no violation has occurred, the program tests to see if a prescribed stopping condition has been achieved and, if so, returns the resulting diagram and halts with success. If the stopping condition has not been achieved and no situation constraints are violated, the program repeats this *cycle* by again making the next alteration that is called for.

If one or more constraints is violated as a result of movement during a cycle, the program enters an inner iteration loop, at each *step* of which the violated constraints are examined. Each violation yields a stress on certain points. For example, if a segment is too long, its endpoints are stressed in the directions that move them closer. After all violated constraints add stresses, net stresses are computed by vector addition and movements are proposed in those grid directions that could reduce the net stress. Generally there are a number of such movements and combinations of movements possible, and the one leading to the least overall stress in the diagram is chosen. This may or may not result in the satisfaction of all constraints;

if not, the others are tried in turn until one is found that produces a constraint-satisfied diagram (at which point the inner iteration ends and the next cycle is begun), or all have failed. If all fail, the best is chosen and another step taken to resolve the impasse. If success is not achieved in this manner, and the stopping condition has not been achieved, the simulation ends with failure.

8. DIAGRAMMATIC DEMONSTRATION OF THE PYTHAGOREAN THEOREM

The program has been applied successfully to several demonstrations, one of which will be described in detail here. The example is a diagrammatic demonstration of the Pythagorean theorem: The square of the length of the hypotenuse of a right triangle is equal to the sum of the squares of the lengths of the other two sides. This demonstration is one of many such to be found in Loomis (1940). The diagrams for the complete demonstration are given in Figure 4; the reader may wish to examine this figure before reading the explanation in order to understand better what is involved and to see why the demonstration is usually compelling to those who understand it.

Understanding the relation between the diagram and the algebraic statement that $C^2 = A^2 + B^2$ involves identifying the algebraic squaring of a number representing the length of a segment with the geometric square constructed with that segment as one side. This identification is merely implicit in the program, which has no independent representation of the algebraic relation, nor indeed any substantial knowledge of algebra. For the program, the task is simply to find a justification for each of the prescribed demonstration steps, such that the area of the hypotenuse square (which the program knows only as a particular square named abde) is rearranged into two smaller squares that can be superimposed respectively on the other two initial squares. This is the only sense in which the program understands the demonstration. Note that this understanding is limited to the particular diagram. Understanding that the relation is true of right triangles in general is not automatically achieved by following this one case, but requires additional methods discussed later.

The construction steps for this demonstration are given in Figure 5 and are cross-referenced to the diagrams of Figure 4. Names of points appear in Figure 4 at the steps when they are constructed. To keep the figure uncluttered a name is not repeated unless the point moves. To make the transcript of steps easier to follow, Figure 6 shows in one drawing most but not all of the constructed lines and point names. The following describes more specifically what computations underlie each of these steps. Each demonstration step is sequentially passed to a loop that executes its functions, then notices changes (new geometric objects) that have arisen, then displays the results.

Step 1 sets a parameter *compliance to 1. This parameter determines the coarseness with which the program observes its representation. For example, a retrieval function that is asked to determine if two points are at the same location will reply affirmatively if and only if the distance between them is less than or equal to the compliance. A *compliance of 1 means that this will occur only if the points are identical or one point apart in a principal (above, below, left, or right, but not diagonal) direction.

Step 1 creates three point objects with names a, b, and c and assigns them to the coordinates specified. It then constructs three segment objects and assigns them names ab, bc, and ac; it also makes ba an alias for ab, etc. It then constructs the list of segment points for each of these segments and marks all these points "black." Finally, it constructs a triangle object whose component sides and vertices are appropriately referenced, and names the object abc with aliases acb, bac, bca, cab, and cba. The aliases are defined merely so that the user does not need to know the details of a naming convention in order to reference an object. Other

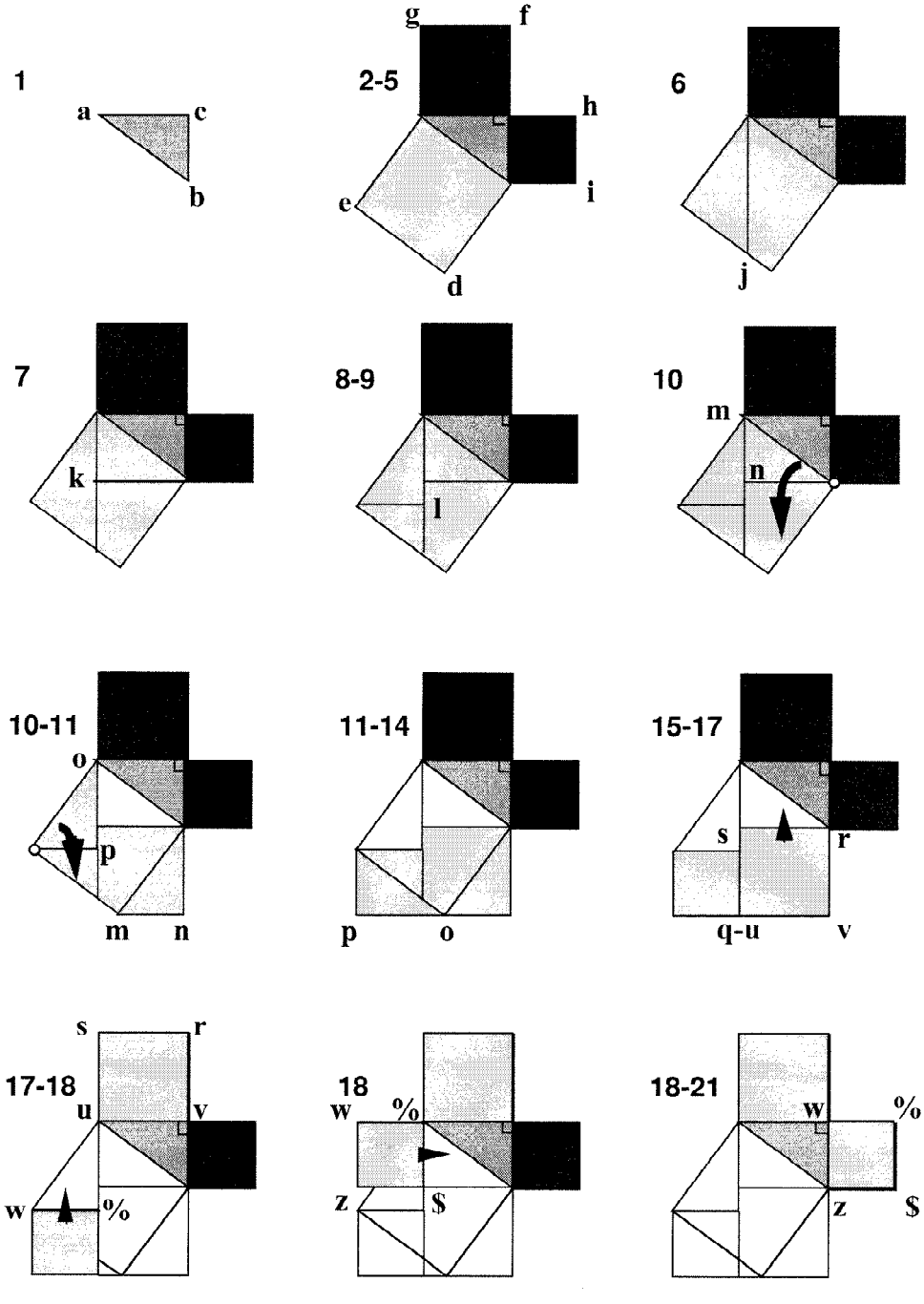

FIGURE 4. Diagrammatic demonstration of the Pythagorean theorem.

Pythagorean Proof Demonstration for ARCHIMEDES-STUDENT

This is a list of steps that is sequentially executed by the construct-notice loop

```
;STEP 1-----------------------------------------------
  ;Constructing right triangle
  (setf *compliance 1)
  (construct-and-name-point  200 200)
  (construct-and-name-point  240 170)
  (construct-and-name-point  240 200)
  (construct-and-name-triangle-from-names a b c)
;STEP 2-----------------------------------------------
  ;Constructing square on the hypotenuse
  (construct-and-name-square-on-segment ab :clock-direction t)
;STEP 3-----------------------------------------------
  ;Constructing square on the long-leg
  (construct-and-name-square-on-segment ca :clock-direction nil)
;STEP 4-----------------------------------------------
  ;Constructing square on the short-leg
  (construct-and-name-square-on-segment bc :clock-direction t)
;STEP 5-----------------------------------------------
  ;Set target of demonstration: abde = acfg + cbih
  (remember-goal-proposition (list 'equal-area (list 'abde) (list 'acfg 'cbih)))
  (setf *notice-flag nil)
;STEP 6-----------------------------------------------
  ;Constructing extension of long-leg-square side
  (construct-and-name-segment-extension-to-segment ag a de)
;STEP 7-----------------------------------------------
  ;Constructing extension of short side leg
  (construct-and-name-segment-extension-to-segment bi b aj)
;STEP 8-----------------------------------------------
  ;Constructing perpendicular to first construction
  (construct-and-name-perpendicular-from-pointname aj e)
;STEP 9-----------------------------------------------
  ;Observe Decomposition: abde = akb + ael + lej + bkjd
  (observe-decomposition abde (list  'akb  'ael  'lej  'bkjd))
;STEP 10----------------------------------------------
  ;Constructing and rotating triangle akb/mnb
  (construct-copy-of-point a) ;this is m
  (construct-copy-of-point k) ;this is n
  (construct-and-name-triangle-from-names m n b)
  (remember-area-equivalence (list 'akb) (list 'mnb))
  ;The following sets goal (stopping-condition) and also defines the
  ;perturbation that will be made at the start of each simulation cycle
  (rotate-rigid-triangle 'mnb 'b '(p-retrieve-colinear (locus bm)(locus bd))
                         :clock-direction nil)
  (construct-unfix-points m n) ;This tells simulate what is moveable
  (simulation-construction :namedpoints '(m n b) :namedsegments '(mn nb bm))
```

FIGURE 5. Steps of the diagrammatic proof of the Pythagorean theorem. Continued following pages.

```
;STEP 11-----------------------------------------------------
  ;Constructing and rotating triangle ael/oep
  (construct-copy-of-point a) ;this is o
  (construct-copy-of-point l) ;this is p
  (construct-and-name-triangle-from-pointnames o e p)
  (remember-area-equivalence (list 'ael) (list 'oep))
  (rotate-rigid-triangle 'oep 'e '(p-retrieve-equal-locus o d)
                         :clock-direction t)
  (construct-unfix-points o p) ;This tells simulate what is moveable
  (simulation-construction :namedpoints '(o e p) :namedsegments '(oe ep po))
;STEP 12-----------------------------------------------------
  ;Observe Area-equivalence abde = mnb + oep + ejl + bkjd
  (observe-area-equivalence (list 'abde)
                            (list 'mnb 'oep 'ejl 'bkjd))
  (setf *notice-flag nil)
;STEP 13-----------------------------------------------------
  ;Observe Polygon
  (observe-polygon b k l e p n)
  (setf *notice-flag nil)
 ;STEP 14-----------------------------------------------------
  ;Observe Decomposition: bklepn = mnb + oep + ejl + bkjd
   (observe-decomposition bklepn (list 'mnb  'oep  'ejl  'bkjd))
   (setf *notice-flag nil)
;STEP 15-----------------------------------------------------
  ;Constructing segment jq
  (construct-and-name-segment-extension-to-segment 'lj j 'pn)
;STEP 16-----------------------------------------------------
  ;Observe Decomposition
  (observe-decomposition bklepn (list 'lepq 'bkqn))
  (setf *notice-flag nil)
;STEP 17-----------------------------------------------------
  ;Constructing and translating square rsuv/bkqn
  (setf *compliance 0.5)
  (construct-copy-of-square (locus 'bkqn))
  (remember-area-equivalence (list 'bkqn) (list 'rsuv))
  (translate-rigid-square 'rsuv 'north '(p-retrieve-equal-locus r f))
  (construct-unfix-points r s u v) ;This tells simulate what is moveable
  (simulation-construction :namedpoints '(r s u v)
                                        :namedsegments '(rs su uv vr))
;STEP 18-----------------------------------------------------
  ;Constructing and translating square wz$%/lepq north, then east
  (construct-copy-of-square (locus 'lepq))
  (construct-remember-area-equivalence (list 'lepq) (list 'wz$%))
  (translate-rigid-square 'wz$% 'north '(p-retrieve-equal-locus % a))
  (construct-unfix-points w z $ %) ;This tells simulate what is moveable
  (simulation-construction :namedpoints '(w z $ %)
                                        :namedsegments '(wz z$ $% %w))
  (translate-rigid-square 'wz$% 'east '(p-retrieve-equal-locus % h))
  (construct-unfix-points w z $ %) ;This tells simulate what is moveable
  (simulation-construction :namedpoints '(w z $ %)
                                        :namedsegments '(wz z$ $% %w))
```

```
;STEP 19-----------------------------------------------
  ;Observe Decompositions rsuv=acfg & wz$% = cbih
  (observe-decomposition rsuv (list 'acfg))
  (observe-decomposition wz$% (list 'cbih))
  (setf *notice-flag nil)
;STEP 20-----------------------------------------------
  ;Observe Area-equivalences bkqn=acfg & lepq=cbih
  (observe-area-equivalence (list 'bkqn)(list 'acfg))
  (observe-area-equivalence (list 'lepq)(list 'cbih))
  (setf *notice-flag nil)
;STEP 21-----------------------------------------------
  ;Verify target proposition
  (observe-target)
  (setf *notice-flag nil)
;End of Demonstration
```

FIGURE 5. Continued.

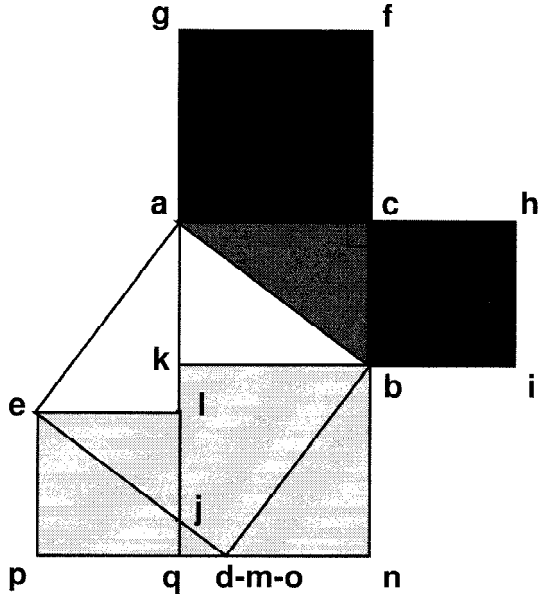

FIGURE 6. Names of points in the Pythagorean theorem demonstration.

polygons can be referred to by any sequence of vertex names that follows the perimeter in either direction.

Steps 2, 3, and 4 construct squares on each side of the triangle. This entails the construction and naming of new point and segment objects, and all of these are thus known to the program. The actual construction of a square requires the construction of perpendiculars from its endpoints, and then the connection of the new points to form the fourth side. Notice

that it is necessary to specify a parameter *clock direction* that determines on which side of the segment the square is to be constructed. By convention, the primary name of a segment is the concatenation of the names of its endpoints in alphabetical order. Thus if the user refers to a segment as ba, its primary name is still ab and the "direction" of the segment is from point a to point b. The *clock-direction* parameter when set to *t* means that the square is "to the right of" the directed segment, that is, to follow the perimeter one moves from a to b and then turns clockwise 90 degrees.

Step 5 specifies the statement that is to be demonstrated. In this instance it is introduced as soon as the diagram contains the figures to which it refers, but it could be introduced at a later time as well. The statement is simply stored until the program is instructed to try to verify it. Note again that the statement is specific to this particular instance, and thus is not an accurate statement of the universal generalization that applies to all right triangles.

Step 5 also sets *notice-flag to *nil*. By default this parameter is set to *t* at the start of each demonstration step, and when that is the case the program makes a systematic scan of its representation at the completion of each demonstration step and notices any objects that it has not previously noticed, that is, for which it has no LISP-structure. As noted above, this can be computationally costly in a complex diagram, primarily because there are many such objects. For example, if two named points b and c are identified on a segment ad, there are six segments identified: ab, ac, ad, bc, bd, and cd. Furthermore, any three of these might form a triangle; in fact none do because all such triangles are degenerate, but this must be determined. This again is a process that human perception appears to do effortlessly, but which requires extensive arithmetic computation with the present representation. A human can readily identify all of these segments and "triangles," but is likely not to do so unless they become important. Thus in some cases the program may notice more than a human normally will.

The noticing program considers only named points—those that have been specifically identified by the demonstrator either by direct reference or because a construction requires them. If a construction results in the crossing of two segments and the intersection is not specifically pointed out, neither that point nor the four new segments containing it will be recognized automatically, unless they are found in the process of looking for new figures. In this case the program will see fewer objects than a human perceiver presumably will normally see. Thus the program makes specific predictions about which newly emergent diagram features will be noticed, and these predictions could be compared experimentally to typical human performance.

A similar issue arises in the case of relations among objects. Again, new relations such as congruency or equality can arise by construction. These can be many and varied, and no attempt has been made to have the program automatically *notice* them. Therefore, critical relations that arise must be pointed out to the program by the demonstrator with an instruction to *observe* them.

Steps 6, 7, and 8 perform basic construction processes at the designated places in the diagram. The procedure for doing these particular constructions is to identify a target and mark each point of the target a nonblack "color." Points are then added to the segment being constructed until a point of the target color is encountered, then all points are repainted black. Line segments are extended one point at a time by moving to the next point that best approximates the direction of the original segment, or, in the case of constructing perpendiculars, in the direction of the negative reciprocal of the target's slope. In general, directions are determined arithmetically.

In the example, the demonstrator avoids some problems by specifying the end from which an extension is to be made (note the middle argument in the construction functions), but a program that does more than follow a demonstration will need to be equipped with the ability

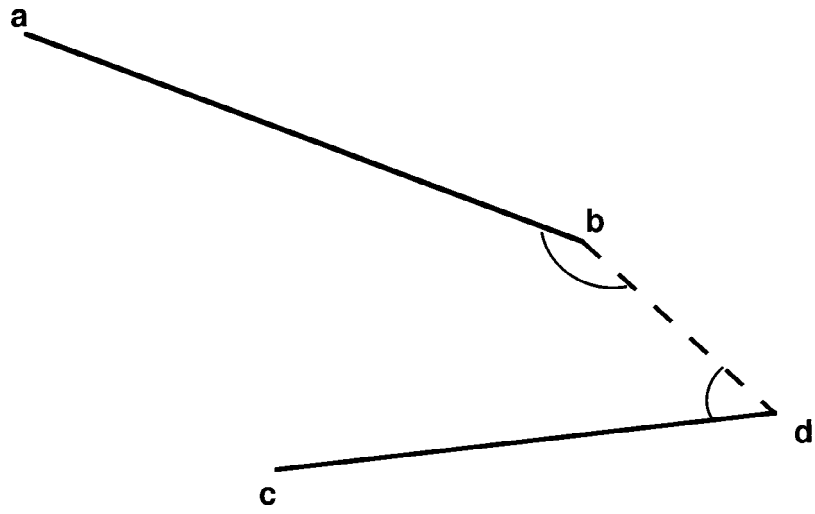

FIGURE 7. Constructing the intersection of two segments.

to determine the direction from a point to a line (and the related ability of determining on which side of a line a point lies). One way of doing this has been implemented in ARCHIMEDES; it is based on a form of Euclid's parallel postulate. If an end of one segment is connected to the end of the other it forms a transversal (say from b to d in Figure 7). The measures of the corresponding interior angles then indicate whether the lines converge in the directions of the connected ends (when the sum exceeds 180 degrees) or of the opposite ends (when the sum is less than 180 degrees), or if the lines are parallel and hence do not converge (when the sum is exactly 180 degrees). The actual computation is slightly more complex because the program has no direct way of knowing that the points are at "the same end" and so must consider the case where, say, b and c were connected. This can be detected, however, because one or both interior angles is greater than 180 degrees. Thus it is possible to write a general function that extends a pair of nonparallel segments until they intersect. This function is based on arithmetic computations and is rapid, but it does not seem to be as direct as human visual perception.

Step 9 is an instruction to observe that the hypotenuse square has now been decomposed into four objects, that is, that the four objects are entirely contained within the boundary of the square, and every point within the square is contained in exactly one of the four objects. The program verifies that this is the case. Such a computation could be done by arithmetic on coordinate points, but this would be very complex and in the general case of arbitrary shapes might be intractable. One alternative would be to use a "color spreading" visual routine such as that described earlier. This method does not well match the current implementation and hardware, so a boundary following algorithm is used instead. To begin, each point on the boundary of the putative container is marked red. Next each point on the boundary of each putatively contained object is considered in turn. If a boundary point is black (that is, it has never been visited), it is marked green. If it is red (that is, it lies on the container boundary) it is marked blue. If it is green (that is, it has already been found as a noncontainer boundary of a contained object) it is marked yellow. If it is blue or yellow, the function immediately returns nil because this is a third tracing of an exterior or interior boundary, indicating that two contained objects are superimposed. After each object has been boundary colored, it

remains only to determine that the container boundary is entirely blue and that there are no green (unvisited) points on contained boundaries.

However, there is an additional difficulty that arises because of the way segment points are constructed. For example, even if segments ab and ac are collinear, unless they lie along a principal direction or principal diagonal, their segment points may differ in a few cases because of the Bresenham/Pittway algorithm. In all such cases, the segment points of the overlapping portions will be within one pixel of one another. Therefore, before a final color check each boundary point on the container is again examined to detect near misses. If a boundary point is red and it has a red neighbor, both are colored blue. The contained object boundaries are also reexamined, and if a boundary point is green its neighbors are examined. If any is green, both are colored yellow; if any is red, both are colored blue. After this, if the claim is true, the container boundary is entirely blue and there are no green (unvisited) points on contained boundaries.

Steps 10 and 11 are the first simulation constructions in this example. Each rotates a triangle about one of its vertices until a stopping condition is met. Each involves making a copy of two vertices of a triangle (exclusive of the pivot vertex) and defining the new triangle formed by the pivot vertex and these two new named points. The program is also told to make note of the fact (without confirming it) that the original triangle and its copy have the same area. This is necessary so that the program can later trace through the history of area equivalences to establish the goal statement. The function rotate-rigid-triangle does not do the actual simulation, but sets the stage for it. This function specifies that the triangle is rigid (that is, its sides are of fixed length) and that the vertex closest to the pivot vertex is to be repeatedly subjected to a displacement in a direction that produces rotation in the indicated direction. (The direction of the vertex displacement will change automatically as the triangle rotation proceeds.) Again, note that the demonstrator selects whether the rotation is clockwise or counterclockwise, and he will presumably pick the direction that results in the lesser amount of rotation to the stop condition. Finally, all points in the diagram save the two vertices that will move are marked fixed in location in order to avoid any attempts to move them during the simulation. Should the simulation fail to be possible under these conditions, it would stop and report the failure. In this example, the simulation will succeed.

Finally, the function (simulation-construction) is called. It is told which objects are involved so that it can focus its processing on a single component of the diagram. The simulation is an iteration that repeatedly moves one or more points and checks for constraint violations, as described earlier.

Step 12 is an instruction to observe an area equivalence. This is confirmed by substituting equivalences from the remembered list of known equivalences until a match to the target is achieved. This is straightforward, but the program does not attempt an exhaustive check of all possible substitutions. Rather, it considers them in the order presented. This means that the demonstrator can make the problem harder or even impossible by pointing things out in a different order. The order chosen in this example makes the task easy, as it does for human students who also could be confused by less straightforward teaching. Noticing is turned off after the demonstration step because nothing has been constructed (although the program does not directly know this; it could find out, but at some effort).

Step 13 is an instruction to observe that a new object, a 6-sided polygon, has emerged. To do this requires the discovery of some new segments and the creation of their LISP structures, as well as the verification that the figure is closed, and the construction of its LISP structure. Once this has been done, Step 14 instructs the program to observe that this new object is a composition of four other objects, and this is verified by the same algorithm used earlier. Again, noticing is turned off for these steps.

Step 15 constructs a short segment that partitions the polygon in a new way. Noticing,

turned on by default, results in the noticing of two new squares formed by the construction of this segment. Step 16 instructs the program to notice that the polygon is composed of these two squares.

Steps 17 and 18 are again simulation constructions, namely translations of the new squares. As before, copies of the squares are made so the old ones are left behind, they are marked rigid, and stopping conditions are specified. Step 18 is a sequence of two translations. Here note that *compliance is set to 0.5 meaning that points are considered to have identical position only if their coordinates are identical. This results in stopping conditions being exactly met and avoids premature stopping. This works because the translations are along principal directions. The tailoring of *compliance to the situation is straightforward and it avoids some difficult technical problems that arise when results are only approximate. However, the burden of choosing *compliance is clearly an unnatural one for constructing diagrammatic demonstrations, and is a problem that needs to be addressed.

Step 19 consists of two instructions to observe "decompositions" but in these cases the decompositions are the limiting case of the container and the contained being superimposed. Nonetheless, the check is made by the previously mentioned method rather than by checking superposition of vertices.

Step 20 has two parts, each of which involves checking, by substitution of previously remembered equivalences, that the components of the polygon are area equivalent to the original squares on the sides of the right triangle. Finally, Step 21 asks the program to verify the target proposition. It does this by transitive substitution of area equivalences.

9. GENERALIZATION AND LEARNING

Understanding must entail more than following isolated examples. One essential additional feature is the ability to remember what was understood, that is, to learn, and to generalize the specific instance to an appropriate class of other instances to, at least, allow recognition of other cases of the same sort to which the previous experience can be applied.

One of the often-cited limitations of diagrammatic reasoning is the lack of obvious ways to reach or even represent general conclusions. A specific diagram is of a specific size, location, orientation, and so forth. Thus an image of, say, a triangle, must be the image of a specific triangle: it may be an equilateral triangle or a scalene triangle, but it cannot be both at once. Natural and artificial languages, on the other hand, allow the specification of general classes of objects and general statements with syntactic constructs such as variables, quantifiers, and Boolean connectives.

One suggestion that addresses the limitation to specific instances is to break a problem into cases, to select a representative instance of each case, and to demonstrate that the conclusion holds for each of these instances. This is how Johnson-Laird (1983) proposed to handle generalization with his theory of mental models. For example, to demonstrate a general theorem for triangles by this method, one would choose an arbitrary length for one side and construct on it a set of specific but varied instances, each of which represents a class of triangles, such as acute, right and obtuse. For each specific instance it would then be determined by construction that the theorem held.

However, the argument underlying the generalization-by-case method requires knowledge of a number of facts that must somehow be known to the system. In this example, one fact is that only relative, not absolute, segment lengths are important. This justifies the choice of an arbitrary first side length, which merely sets the scale of the diagram. A second is the classification of angles into three categories; but there is no a priori reason to know that 45 degrees is not an important special case (it sometimes is). Furthermore, not all selections of

points and segment length are guaranteed to result in a triangle: the sides must satisfy the triangle inequality, and each angle must be less than 180 degrees. This information is not represented in a form that is available to the program. Therefore, either this knowledge is assumed as additional knowledge unrelated to the system's knowledge of space, or the method must "know" that repeated failure to construct even one triangle is not sufficient reason to give up trying other values. Thus, generalization by cases *presupposes* a way to construct an exhaustive set of cases, and the successful generation of an appropriate set of cases requires substantial geometric and other knowledge. I will say that such knowledge is *exogenous* to the simulation and representation model.

Simulation offers another approach for generalizing because it permits running "experiments" that show how some parts of a diagram co-vary with changes in others. By observing the interactions of diagram components as one property is varied, a person can frequently come to understand a geometric relation in a deeper sense. For example, one property of a diagram, say a segment length, may be altered through a continuous range of values without explicitly changing other properties, but allowing them to vary as required by the spatial and situation constraints. The diagram is observed to determine other properties that are altered in a way that correlates with the manipulation. For example, as the length of the side of a triangle varies, the size of the opposite angle will also vary. Such experiments also support understanding of algebraic as well as geometric relations. ARCHIMEDES-STUDENT can be instructed to perform these manipulations and observations.

This method can at times reveal certain critical properties. In the above demonstration of the Pythagorean theorem, no explicit mention is made that the triangle in question must be a right triangle. That limitation could be illustrated to the program by giving it non–right triangles and showing that the demonstration does not go through, and a program could be written to compare the successful examples with the nonsuccessful ones to discover the critical feature. However, such a program would not of itself provide the basis for any insight into the significance of the right angle.

We might, however, have ARCHIMEDES-STUDENT sweep the "right" angle from less than 90 degrees to more than 90 degrees while holding other components fixed to show that the "hypotenuse" varies with the "right" angle, and so does the area of its square. It can also be seen that the squares on the other legs remain unchanged. It has been previously demonstrated that the sum of those areas equals the area of the largest square when the altered angle is 90 degrees. Combining these observations shows that a right angle is a critical, or watershed, value for the relation between these areas. This is one way of seeing that the Pythagorean relation is false for non–right triangles without examining an unlimited set of specific cases or knowing how to select a complete set of representative cases.

An example of generalizing by simulation and cases is illustrated by some experiments in which ARCHIMEDES was applied to the triangle congruency theorems, e.g., two triangles are congruent if they have two sides and the included angle congruent (SAS), two angles and the included side congruent (ASA), or all three sides congruent (SSS), but are not necessarily congruent if they have three congruent angles (AAA), two sides and a nonincluded angle congruent (SSA), or other combinations of components in correspondence.

Here is one way the congruency theorems have been demonstrated to ARCHIMEDES. It, too, requires the assumption of exogenous knowledge and still runs into the generalization problems, but it is often revealing to a human. Choose three noncollinear points at random and connect them to form a triangle. Fix certain measures, such as two sides and the included angle. That is, annotate the representation of the constructed triangle (i.e., its situation constraints) to indicate that the measures must remain fixed. Instruct the program to attempt to alter those sides and angles that are not fixed, using its simulation algorithm. If the simulation is unable to alter the triangle, conclude that its shape is fully determined by the

specified features, else that it is not. For many students, this method provides an understanding of the congruency theorems that is lacking from a deductive proof because it demonstrates the interactions among sides and angles in terms of perceptual processes.

Again, the problem arises as to what is an *arbitrary* triangle. The SSA case is particularly instructive here, because if the constructed triangle happens be a right triangle, then it will indeed be unalterable, leading to a false conclusion. Furthermore, if the triangle is not a right triangle, then although there could be two possible solutions, they cannot be smoothly transformed into one another by the simulation algorithm without passing through a range of values that violate a constraint, and the algorithm does not permit this. One step toward generality is to apply the procedure to several randomly chosen sets of vertices for each case with the understanding that all must pass the test. This reduces the probability of being mislead by a special case, and is generally convincing to people, though it is not a mathematical certainty. However, the knowledge that using several cases increases the likelihood of a conclusion being correct is again *exogenous* knowledge.

These examples illustrate how simulation can be used to generalize beyond a single case by showing how spatial constraints interact to determine the relationship among diagrammatic features. Simulation could conceivably be used to demonstrate other generalizations as well, notably asymptotic behaviors, periodic relations, and some symmetric relations. In all of these cases, making substantive use of such information requires exogenous knowledge, that is, knowledge that is not explicitly embodied in the existing representation and simulation system. As noted above, either such knowledge must remain implicit in the use of the system, or it must be represented in ways that the program can manipulate. To achieve the latter, there is no alternative to a predicative representation of what appears to be inherently predicative knowledge. Thus generalization and understanding must involve verbal representations, although representations need not be exclusively verbal.

The use of object frames in ARCHIMEDES is a mechanism for introducing a form of quantification over variables and specifying classes of objects that are independent of location, scale, and orientation. ARCHIMEDES has been extended to employ the abstraction power of the frame system to permit it to learn lemmas. This will be illustrated by the following example, based on another proof of the Pythagorean theorem, this one attributed to Euclid; see Figure 8. Like the earlier example, it involves the partitioning of the hypotenuse square. However, the initial partition is of the pentagon formed by the hypotenuse square combined with the original right triangle. The partition components are triangles. There is a symmetry to the procedure that provides opportunity to exploit the diagram class abstraction method.

After construction of the triangle and squares, the next portion of the demonstration is the construction of segment CE, thus forming the triangle ACE, then construction of line EJ parallel to AC, thus forming parallelogram ACJE, with diagonal CE. The demonstration then proceeds to show that the two triangles composing the parallelogram are congruent. It does this by the (laborious) simulation method that rotates one of these triangles through 180 degrees around one of the common vertices (say C), and then translates it along its long side until the two triangles are superimposed, establishing that they are of equal area (since they are congruent). Another simulation step establishes that the smaller triangle ACK is of equal area to triangle EJL, from which it follows that the parallelogram ACJE is equal in area to AKLE (recall the example from Figure 1), the larger of the two rectangles into which the hypotenuse square is partitioned by segment KL. Thus it has been demonstrated that triangle ACE is half the area of that larger rectangle.

Segment FB is now constructed, thus forming triangle FBA. With the construction of FM parallel to AB, a new parallelogram FMBA is formed with diagonal FB. We could demonstrate that the area of this triangle is half the area of the square ACGF by simulation: a rotation and translation of FBA and a translation of FMG to superimpose ABC as before. However,

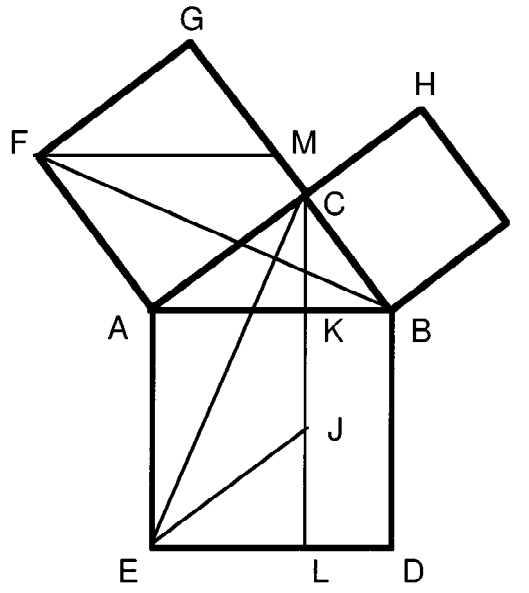

FIGURE 8. A proof by Euclid of the Pythagorean theorem.

if the system could recognize that this situation is "the same as" the previous demonstration, the simulation steps could be avoided and the conclusion drawn immediately.

To do so, a representation called a *signature* (following the terminology of Wang 1995) is introduced that characterizes a diagram component with which a conclusion is to be associated. The representation specifies the elementary figures which compose it (e.g., a parallelogram and two triangles), how they are related (for example the triangles share a side, which is the diagonal of the parallelogram), and any stated constraints on these components or their components that are extant in the generating instance (none in the present case), the default being that if no constraints are present in the generating figure, none are involved in the signature. The second process needed is one that can find in an arbitrary diagram representation any instance of this signature. Associated with the signature is a list of conclusions (e.g., that the triangles are congruent and of equal area). As presently implemented, it is necessary for the demonstrator to instruct the system when it should construct a new signature, what its components are (how they are related is determined by the program), and what conclusions are to be recorded.

The next demonstration step is the show that triangle ACE is congruent to triangle AFB; this can be verified by rotation of one of these triangles about the common vertex A until the two are superimposed. This then confirms that the larger partition AKLE of the hypotenuse square is equal in area to the square ACGF on the leg AC.

At this point, an additional abstraction is often seen by many people, namely that the other "half" of the proof—that the smaller rectangle KLDB is equal in area to the square CBIH on the leg BC—follows by "the same" argument. Indeed it is true that a formal proof of the two halves is identical except for a renaming of the points. Gelernter (1959a) devised a method of "syntactic symmetries" that could detect such cases. However, it should be possible to compute this relation from the diagram, as it were, rather than from the statements of a formal proof. That is the approach I have taken. To do so requires the formation of a

second signature based on the component triangles ACE and AFB. This signature also records information relating their sides, based on constraints on these sides because they are also sides of squares. The second half of the proof is then the sequential use of the two signatures.

Signatures are similar in concept to the "diagram configurations" of the DC model of Koedinger and Anderson (1990). In that model, configurations were defined so that the system, which attempts to prove theorems, can apply similar methods to similar problems, although the program was unable to discover new configurations by itself. McDougal (1993) employed case-based reasoning (generalizing from previously solved cases) in his geometry proof system, POLYA, with similar purpose. In ARCHIMEDES, the signatures are to be generated by examination of specific figures, and are used essentially as lemmas to avoid repeating simulation steps, which remain the heart of the inference process.

10. DEPICTIONS AND DESCRIPTIONS

Representation has long been recognized to be a key issue in computational modeling but it remains the source of much confusion and controversy (for example, see Bickhard & Terveen 1995). This is particularly evident in the continuing discussions of the role of imagery in cognition. For example, Glasgow (1993) attempted to distinguish descriptions from depictions but ran into a flood of objections from those who commented on her paper.

The concept of representation is slippery for many reasons, but one of the greatest sources of difficulty is the still widespread habit of describing a representation merely by writing down an example, such as a map, and saying nothing about its source or how it is used, that is, the construction and retrieval processes. Advocates of depictional representations often ignore the retrieval processes. They choose examples and informally assess "representations" by looking at them, and thereby underestimate the complexity of retrieval processes because human perception makes certain observations so effortlessly.

On the other hand, advocates of descriptional representations are prone to ignore the construction processes needed for propositional representation because human perception and linguistic ability so readily translate pictures into verbal descriptions and thence to predicate-calculus-like notations. Thus a predicate calculus advocate may use ON(block, table) as an appropriate and natural representation of a situation where a block is on a table, while ignoring the processes that construct that representation. In fact, however, the creation of such sentences either from English statements or from video camera input is nontrivial and has not been solved except for the simplest cases. Novak (1993) wonders what a propositional description of an automobile power steering actuator would look like. Surely it would be enormously complex, but, more to the present point, the complexity of the construction process that creates it (from, say, a video input) would be much more so.

Furthermore, a detailed drawing may be reduced to "Gear A is to the left of gear B" thence to LEFT-OF(A, B). This overlooks the poverty of such descriptions as well as the difficulty of creating them. A propositional representation suitable for even very simple inferences about mechanical systems must be vastly more complex. Forbus, Faltings, and Nielsen (1991) and Forbus (1993) argue that most such inferences *must* depend on complete geometric descriptions. That argument suggests that propositional descriptions must be extremely detailed and lengthy for such problems. In fact the mechanical behavior even in the simple case of two gears depends critically on the exact shapes of the gear teeth, so an adequate propositional representation must be very detailed. In highly regular cases, such descriptions may be algebraic, but in general this is not possible. In cases involving mechanical couplings where metric information is fully known it is possible to represent the possible configurations of parts using "configuration spaces" (Joskowicz & Sacks 1991) and to abstract these into

qualitatively important classes (Stahovich, Davis, & Shrobe 1996), but this information does not readily translate into predicate representations that are parsimonious enough for logical deduction. Because we have no idea how to implement the construction processes that take diagrams into predicate calculus, the argument that propositional representations are in principle adequate for any depictionist example simply loses all force.

Thus the practice of ignoring the construction and retrieval processes is particularly dangerous when dealing with representations that are natural for humans, because what is seemingly effortless for a human "constructor" or "retriever" may hide enormously complex processes that are almost totally unknown to us at either the psychological or neurological levels of analysis. It should be clear, however, that even if a complete neurological, chemical, and physical description of the processes of human visual perception were at hand, their classification as depictional or descriptional would depend on what descriptive language was used to describe *them*. Thus the processes could perhaps truthfully be described as propositional, analog, digital, or statistical, among others, and the structures involved could be described in terms of abstractions like association of concepts, or synapse strengths and voltages, or energy patterns, or current flows, or quantum mechanical states—all with equal validity.

From the point of view of psychology or cognitive science, the first order of business is to discover a functional description of a virtual machine capable of exhibiting performance that is broadly similar to human performance as described in terms of goals and tasks. A proposed model of human imagery based on this virtual machine would then need to be implemented on a real machine in order to evaluate issues of computational efficiency and to compare the model to human performance on such things as chronometric measures and informational capacity. The present work is a limited beginning in this direction.

11. DISCUSSION

The work reported here, like most work in cognitive science, is based on the hypothesis that intelligent systems, including humans, have mental representations of the things they know and do. There have been two schools of thought about the nature of representations. One proposes the use of a general representation language, such as a symbolic calculus, that can express arbitrary propositions and imperatives. The other proposes a collection of representations, each of which is to some degree specialized to an ability. As noted in Section 10, the distinction does not depend on a pictorial versus verbal distinction. The two could coexist. The present work explores the specialist school.

The arguments for specialized representations go back to the early literature of artificial intelligence, and of course have roots much earlier in philosophy and psychology. Amarel (1962) illustrated the power of specialized representations that make use of mathematical structure to reflect the natural structure of a problem. Sloman (1971) argued that analogical representations are often computationally superior to predicate calculus representation, are not inherently less rigorous, and are appropriate for theories of minds that evolved in a physical world requiring perception and action. Lindsay (1963) argued that specialized mental models are appropriate for inference with natural language, and later (1973) pointed out that generality may be achieved by a general method for creating specialized, efficient representations.

Sloman (1993), Palmer (1978), Lindsay (1988), and others have characterized a class of representations, sometimes called "natural models," that cannot be altered in any way that violates certain constraints. This means that altering the representation leads to the representation of other knowledge that is consistent with the original, subject to the constraints. In contrast, predicate calculus has essentially no a priori constraints on what it can describe,

even though it has a highly constrained syntax. Moreover, it permits the use of variables and quantification. The structure of a particular domain of discourse is provided in a predicate calculus representation by axioms, and deduction is a general inference method for determining the consequences of the axiomatized structure. Thus predicate calculus is a very general form of representation. However, generality does not guarantee computational advantages, and may weaken them in particular domains in comparison to a restricted representation that is tailored to the domain (Lindsay 1973). If the constraints of the representation, whatever form it takes, can be efficiently maintained, and if they reflect the properties of the subject of thought, then the representation is a good one. However, some knowledge states cannot be represented by the specialized model: some things cannot be said. The interesting case is where the things that cannot be said are not interesting or useful and those that can be said are manipulable in computationally efficient ways. In the case of diagrams that are tailored to spatial relations, the domain of application may be large and important. This point has been made in a variety of terminologies by others as well, for example by Palmer (1978), Stenning (1993), and Sloman (1993). There are many examples of such representations that have been proposed and studied, although the general characterization has not always been drawn (see Johnson-Laird 1983).

The essence of a physical diagram is that it *necessarily* preserves topological and geometric properties of two-dimensional space—it cannot violate them. (Note that this is not true of a two-dimensional rendering of a three-dimensional object; recall as well the famous and amusing drawings of Escher.) Thus, for example, when two points are displaced by equal distances in the same direction, the length of the segment connecting them remains unchanged. The situation is different when we move from the manipulation of real objects in space or physical diagrams to the use of imagery in minds or data structures in computer memory. In the latter case we can impose whatever rules we choose. We may for example choose to impose rules that reflect the motion of rigid bodies. If points are represented as integer pairs (coordinates) and equal displacements are made by incrementing the coordinates of each pair identically, this procedure will assure that the equality-of-segment-length "fact" remains true in the representation (assuming length is computed as Euclidean distance). This follows from the laws of arithmetic and algebra, which are efficiently implemented on digital hardware. It could conceivably follow for other reasons in other implementations. For example, in the real world, translation of a rigid stick (at nonrelativistic velocities) maintains length by virtue of the laws of physics (not of mathematics). It is because the laws of physics and geometry may be veridically reflected (by design) in the mathematics of analytic geometry, and these may be veridically maintained by computation, that an array representation is able to make (correct) inferences about imaginary experiments. As noted above, whether these are characterized as descriptive or depictive representations is a matter of level of analysis and point of view. Both views are valid and each may be appropriate for different purposes.

On the other hand, we may employ rules (construction and retrieval processes) that implement nonrigid movement. ARCHIMEDES permits this because situation constraints that, for example, mark segments as rigid may be omitted. If an endpoint is fixed in location by a constraint and the other endpoint is moved, the segment length may change. This representation thus makes it possible to perform a variety of simulations in addition to rigid body movement.

When the discourse moves to human mental imagery the situation is by no means as clear. What is clear is that mental imagery is not limited to rigid body movement: we can imagine many things including objects and events that could not happen in the physical world. It does not follow that the mind can imagine anything whatsoever, or that it can imagine everything with equal facility. An assumption of this work is that the human mind is indeed predisposed to handle certain types of imagery and simulation better than others, presumably those kinds for

which evolution has best prepared us, especially the motion of rigid objects. As noted earlier, the determination of what is and is not cognitively penetrable in this sense will be difficult. I have also argued that the recognition, representation, and understanding of mathematical relations with the aid of diagrams and imagery manipulation requires the use of quantified predicative knowledge to both state the conditions and the conclusion. There is no such thing as purely diagrammatic mathematical understanding.

12. SUMMARY AND CONCLUSIONS

One way to view the objective of this research is to contrast it with work on theorem proving in geometry and other formal domains, where *proving* means discovering a sequence of syntactic transformations on propositions. In contrast, in the present research the transformations are not syntactic manipulations of linear expressions, but geometric transformations on two-dimensional diagrams. Thus the "proofs" themselves carry a semantic interpretation, which for humans relates them to familiar knowledge about the structure of real space and events.

Note that understanding as here construed does not amount to possessing a representation of an item of knowledge. Rather, understanding is a *process*. In particular, it is the process of confirming that transformations of representations are correct with respect to the system's underlying repertoire of permitted transformations, and thus that the situation, fact, or event that is understood is *consistent* with the system's "theory" of the subject. This falls well short of a full explanation of understanding as it is done by human minds, but is a fundamental component of full understanding.

The work described earlier on formalizing reasoning with Venn/Euler diagrams in logical calculus fashion applies to descriptions of diagrams, rather than to bitmapped representations. The general approach of these efforts has been to start with an abstracted description of a limited class of diagrams such as Venn diagrams (rather than, say, an image or bitmap representation) and construct a linear (predicate calculus) description language and rules of inference; that is, the efforts amount to expressing a theory of diagrams in a standard, linear formalism. Work that formalizes the use of Venn diagrams exploits the geometric and topological properties of diagrams in limited ways, and does so indirectly, by defining rules for combining diagrams which ensure the essential spatial property (transitivity of the "contained in" relation) not by actual rigid motion of closed curves, but by definitions that rely on human perception for recognition. The same is true of Wang (1995), although he is addressing a wider range of geometric objects. Similarly, the use of occupancy arrays by Glasgow (1993) employs only a limited set of spatial properties (e.g., adjacency and a coarse measure of direction) that are preserved and used to make inferences. In contrast, the present work employs general methods for manipulating bitmaps in such a way that all topological and metric properties of the plane are preserved. All of the approaches, including ARCHIMEDES, are complementary, not antithetical.

This work has empirical support in the general sense that it is a plausible explanation of important and still mysterious human abilities. Experimental tests could address certain aspects of the model. For example, the model exhibits greater success with some orderings of demonstration steps than with others, and these could be taken as empirical predictions of human abilities. Also, the model makes predictions about what objects and properties will be noticed and the order in which they will be noticed—predictions that could be compared to human performance. The model does not do exhaustive searching of all substitution patterns in algebraic expressions and will fail to see conclusions that were presented in certain sequences; it is clear that humans find certain orders of presentation easier to follow than others, and

thus there is another opportunity for confirmation of the program as a psychological model. Such experiments have not been done because it is felt they are premature.

The immediate challenge of this research, rather, is to devise methods that are computationally plausible and accomplish something *like* human spatial reasoning ability. It aims to construct not a performance model but a competence model in the sense of Chomsky (1965). Thus the research is intended to say something about what the *task* requires. Marr (1982) also suggested that this sort of task analysis should be the first step for either psychological modeling or artificial intelligence. The problem addressed may be characterized as a problem of cognitive science, or the scientific rather than the engineering thrust of artificial intelligence.

ACKNOWLEDGMENTS

This material is based on work supported by the United States National Science Foundation under Grant No. IRI-9203946 and Grant No. IRI-9526942. The author also wishes to thank Maija Kibens for assistance with this project.

REFERENCES

AMAREL, S. 1962. On the automatic formation of a computer program which represents a theory. *In* Self-Organizing Systems. *Edited by* M. C. Yovits, G. Jacobi, and G. Goldstein. Spartan Books, Washington, D.C., pp. 107–175.

ANDERSON, M., and R. MCCARTNEY. 1995. Inter-diagrammatic reasoning. *In* Proceedings of the International Joint Conference on Artificial Intelligence, Montreal. Morgan Kaufmann, San Mateo, CA, pp. 878–884.

ANDERSON, M., and R. MCCARTNEY. 1996. Diagrammatic reasoning and cases. *In* Proceedings of the Thirteenth National Conference on Artificial Intelligence, Portland, OR. AAAI Press, Menlo Park, CA, pp. 1004–1009.

BARKER-PLUMMER, D., and S. C. BAILIN. 1992. Proofs and pictures: Proving the diamond lemma with the GROVER theorem proving system. *In* Reasoning with Diagrammatic Representations. Technical Report SS-92-02. American Association for Artificial Intelligence, Menlo Park, CA, pp. 102–107.

BARWISE, J., and J. ETCHEMENDY. 1990. Visual information and valid reasoning. *In* Visualization in Mathematics. *Edited by* E. P. Glinert. Mathematical Association of America, Washington, DC.

BARWISE, J., and J. ETCHEMENDY. 1992. Hyperproof: Logical reasoning with diagrams. *In* Reasoning with Diagrammatic Representations. Technical Report SS-92-02. American Association for Artificial Intelligence, Menlo Park, CA.

BICKHARD, M., and L. TERVEEN. 1995. Foundational Issues in Artificial Intelligence and Cognitive Science: Impasse and Solution. North-Holland, Amsterdam.

BORNING, A. 1981. The programming language aspects of THINGLAB, a constraint-oriented simulation laboratory. ACM Transactions on Programming Languages and Systems, **3**(4):353–387.

CARAMAZZA, A., M. MCCLOSKEY, and B. GREEN. 1981. Naive beliefs in "sophisticated" subjects: Misconceptions about trajectories of objects. Cognition, **9**(2):117–123.

CHOMSKY, N. 1965. Aspects of the Theory of Syntax. MIT Press, Cambridge, MA.

CHOU, S.-C. 1988. Mechanical Geometry Theorem Proving. D. Reidel Publishing Company, Dordrecht.

CLEMENT, J. 1994. Imagistic simulation and physical intuition in expert problem solving. *In* Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Atlanta, GA. *Edited by* A. Ram and K. Eiselt. Lawrence Erlbaum, Hillsdale, NJ, pp. 201–206.

DE KLEER, J., and J. S. BROWN. 1984. A qualitative physics based on confluences. *In* Qualitative Reasoning about Physical Systems. *Edited by* D. G. Bobrow. Elsevier, Amsterdam, pp. 7–83.

EDWARDS, G. 1996. Geocognostics—A new paradigm for spatial information? *In* AAAI-96 Spring Symposium Series, Stanford University, Stanford, CA, pp. 6–14.

FOLEY, J. D., A. VAN DAM, S. K. FEINER, and J. F. HUGHES. 1990. Computer Graphics: Principles and Practice. Addison-Wesley, Reading, MA.

FORBUS, K. 1984. Qualitative process theory. *In* Qualitative Reasoning about Physical Systems. *Edited by* D. G. Bobrow. Elsevier, Amsterdam, pp. 85–168.

FORBUS, K. D. 1993. Image and substance. Computational Intelligence, **9**(4):377–378.

FORBUS, K. D., B. FALTINGS, and P. NIELSEN. 1991. Qualitative spatial reasoning: The CLOCK project. Artificial Intelligence, **51**(1):417–472.

FUNT, B. V. 1976. Whisper: A Computer Implementation Using Analogues in Reasoning. Ph.D. dissertation, Department of Computing Science, University of British Columbia, Vancouver.

FUNT, B. V. 1977. Whisper: A problem-solving system utilizing diagrams and a parallel processing retina. *In* Proceedings of the Fifth International Joint Conference on Artificial Intelligence, IJCAI-77, Cambridge, MA. Carnegie-Mellon University, Pittsburgh, pp. 459–464.

FUNT, B. V. 1980. Problem solving with diagrammatic representations. Artificial Intelligence, **13**(3):201–230. (*Reprinted in* R. Brachman and H. Levesque, eds. 1981. Readings in Knowledge Representation. Los Altos, CA, Morgan Kaufmann, pp. 441–456.

FURNAS, G. W. 1992. Reasoning with diagrams only. *In* Reasoning with Diagrammatic Representations. Technical Report SS-92-02. American Association for Artificial Intelligence, Menlo Park, CA, pp. 118–123.

GARDIN, F., and B. MELTZER. 1989. Analogical representations of naive physics. Artificial Intelligence, **38**:139–159.

GELERNTER, H. 1959a. A note on syntactic symmetry and the manipulation of formal systems by machine. Information and Control, **2**:80–89.

GELERNTER, H. 1959b. Realization of a geometry theorem proving machine. *In* International Conference on Information Processing. UNESCO House, Paris, pp. 273–282.

GELERNTER, H., J. R. HANSEN, and D. W. LOVELAND. 1960. Empirical explorations of the geometry theorem proving machine. *In* Western Joint Computer Conference, 17. New York: National Joint Computer Committee, pp. 143–147. *Reprinted in* E. Feigenbaum and J. Feldman, eds. 1963. Computers and thought. McGraw Hill, New York.

GLASGOW, J., and D. PAPADIAS. 1992. Computational imagery. Cognitive Science, **16**(3):355–394.

GLASGOW, J. I. 1993. The imagery debate revisited: A computational perspective. Computational Intelligence, **9**(4):309–333.

HEGARTY, M. 1992. Mental animation: inferring motion from static displays of mechanical systems. Journal of Experimental Psychology: Learning, Memory and Cognition, **18**(5):1084–1102.

HEGARTY, M., and J. M. FERGUSON. 1993. Strategy change with practice in a mental animation task. *In* Annual Meeting of the Psychonomic Society, Washington, D.C.

HEGARTY, M., and M. A. JUST. 1993. Constructing mental models of machines from text and diagrams. Journal of Memory and Language, **32**:717–742.

JOHNSON-LAIRD, P. N. 1983. Mental Models. Toward a Cognitive Science of Language, Inference, and Consciousness. Harvard University Press, Cambridge, MA.

JOSKOWICZ, L., and E. P. SACKS. 1991. Computational kinematics. Artificial Intelligence, **51**:381–416.

KAUFMAN, S. G. 1991. A formal theory of spatial reasoning. *In* Proceedings of the Second Conference on Knowledge Representation, pp. 347–356.

KOEDINGER, K. R., and J. R. ANDERSON. 1990. Abstract planning and perceptual chunks: Elements of expertise in geometry. Cognitive Science, **14**:511–550.

KOSSLYN, S. M. 1976. Can imagery be distinguished from other forms of internal representation? Evidence from studies of information retrieval times. Memory & Cognition, **4**(3):291–297.

KOSSLYN, S. M. 1993. Images in the computer and images in the brain. Computational Intelligence, **9**(4):340–342.

KOSSLYN, S. M., and G. HATFIELD. 1984. Representation without symbol systems. Social Research, **51**(4):1019–1044.

KOSSLYN, S. M., and J. R. POMERANTZ. 1977. Imagery, propositions, and the form of internal representations. Cognitive Psychology, **9**(1):52–76.

KRAMER, G. A. 1992. A geometric constraint engine. Artificial Intelligence, **58**:327–360.

LARKIN, J. H. 1983. The role of problem representation in physics. *In* Mental Models. *Edited by* D. Gentner and A. L. Stevens. Lawrence Erlbaum, Hillsdale, NJ, pp. 75–98.

LARKIN, J. H., and H. A. SIMON. 1987. Why a diagram is (sometimes) worth ten thousand words. Cognitive Science, **11**:65–100.

LINDSAY, R. K. 1961. Toward the Development of a Machine Which Comprehends. Doctoral dissertation, Carnegie-Mellon University, Pittsburgh, PA.

LINDSAY, R. K. 1963. Inferential memory as the basis of machines which understand natural language. *In* Computers and Thought. *Edited by* E. A. Feigenbaum and J. Feldman. McGraw-Hill, New York, NY, pp. 217–233.

LINDSAY, R. K. 1973. In defense of ad hoc systems. *In* Computer Models of Thought and Language. *Edited by* R. Schank and K. Colby. W. H. Freeman, San Francisco, pp. 372–395.

LINDSAY, R. K. 1988. Images and inference. Cognition, **29**:229–250. (*Reprinted in* J. I. Glasgow, N. H. Narayanan, and B. Chandrasekaran, eds. 1995. Diagrammatic Reasoning: Computational and Cognitive Perspectives. MIT Press, Cambridge, MA, pp. 111–135.

LOOMIS, E. S. 1940. Pythagorean Proposition: Its Proofs Analyzed and Classified and Bibliography of Sources for Data of the Four Kinds of "Proofs," 2nd ed. Edwards Brothers, Ann Arbor, MI.

LOVELAND, D. W. 1968. Mechanical theorem proving by model elimination. Journal of the Association for Computing Machinery, **15**(2):236–251.

MARR, D. 1982. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. Freeman, San Francisco.

MATSUYAMA, T., and T. NITTA. 1995. Geometric theorem proving by integrated logical and algebraic reasoning. Artificial Intelligence, **75**(1).

MCCLOSKEY, M., A. CARAMAZZA, and B. GREEN. 1980. Curvilinear motion in the absence of external forces: Naive beliefs about the motion of objects. Science, **210**(4474):1139–1141.

MCCLOSKEY, M., and D. KOHL. 1983. Naive physics: The curvilinear impetus principle and its role in interactions with moving objects. Journal of Experimental Psychology: Learning, Memory, & Cognition, **9**(1):146–156.

MCCLOSKEY, M., A. WASHBURN, and L. FELCH. 1983. Intuitive physics: The straight-down belief and its origin. Journal of Experimental Psychology: Learning, Memory and Cognition, **9**(4):636–649.

MCDOUGAL, T. F. 1993. Using case-based reasoning and situated activity to write geometry proofs. *In* Annual Meeting of the Cognitive Science Society. Lawrence Erlbaum, Hillsdale, NJ, pp. 711–716.

MILLER, A. I. 1984. Imagery in Scientific Thought. Birkhauser, Boston.

NARAYANAN, N. H. 1992. Imagery, Diagrams and Reasoning. Ph.D. dissertation, Department of Computer and Information Science, Ohio State University, Columbus.

NELSON, R. B. 1993. Proofs without Words. Exercises in Visual Thinking. The Mathematical Association of America, Washington, DC.

NEWELL, A., J. C. SHAW, and H. A. SIMON. 1957. Empirical explorations with the Logic Theory Machine. *In* Proceedings of the Western Joint Computer Conference, **15**, pp. 218–239. (*Reprinted in* E. Feigenbaum and J. Feldman. 1963. Computers and Thought. McGraw-Hill, New York.)

NOVAK, G. 1977. Representations of knowledge in a program for solving physics problems. *In* Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA. Carnegie-Mellon University, Pittsburgh, PA, pp. 286–291.

NOVAK, G. S., Jr. 1993. Computational and brain representations of imagery. Computational Intelligence, **9**(4):398–401.

PALMER, S. 1978. Fundamental aspects of cognitive representation. *In* Computing and Categorization. *Edited by* E. Rosch and B. B. Lloyd. Erlbaum, Hillsdale, NJ, pp. 259–303.

PINKER, S. 1985. Visual cognition: An introduction. *In* Visual Cognition. *Edited by* S. Pinker. MIT Press, Cambridge, MA, pp. 1–63. (*Reprinted from* Cognition: International Journal of Cognitive Psychology, **18**.)

PYLYSHYN, Z. W. 1973. What the mind's eye tells the mind's brain: A critique of mental imagery. Psychological Bulletin, **80**(1):1–24.

PYLYSHYN, Z. W. 1980. Computation and cognition: Issues in the foundations of cognitive science. Behavioral and Brain Sciences, **3**:111–133.

PYLYSHYN, Z. W. 1981. The imagery debate: Analogue media versus tacit knowledge. Psychological Review, **88**(1):16–45.

PYLYSHYN, Z. W. 1984. Computation and Cognition: Toward a Foundation for Cognitive Science. MIT Press, Cambridge, Massachusetts.

RITT, R. F. 1938. Differential equations from an algebraic standpoint. *In* AMS Colloquim Publications. American Mathematical Society, New York.

ROBINSON, J. A. 1965. A machine-oriented logic based on the resolution principle. Journal of the ACM, **12**(1):23–41.

SHEPARD, R. N. 1994. Perceptual-cognitive universals as reflections of the world. Psychonomic Bulletin and Review, **1**(1):2–28.

SHEPARD, R. N., and L. A. COOPER. 1982. Mental Images and their Transformations. MIT Press, Cambridge, MA.

SHEPARD, R. N., and S. L. ZARE. 1983. Path-guided apparent motion. Science, **220**(6 May):632–634.

SHIN, S.-J. 1991. A situation-theoretic account of valid reasoning with venn diagrams. *In* Situation Theory and Its Applications, pp. 581–605.

SHIN, S.-J. 1992. A semantic analysis of reasoning involving Venn diagrams. *In* Reasoning with Diagrammatic Representations. Technical Report SS-92-02. American Association for Artificial Intelligence, Menlo Park, CA, pp. 85–90.

SHIN, S.-J. 1995. The Logical Status of Diagrams. Cambridge University Press, Cambridge.

SHRAGER, J. 1990. Common sense perception and the psychology of theory formation. *In* Computational Models of Scientific Discovery and Theory Formation. *Edited by* J. Shrager and P. Langley. Morgan Kaufmann, San Mateo, CA, pp. 437–470.

SLOMAN, A. 1971. Interactions between philosophy and artificial intelligence: The role of intuition and non-logical reasoning in intelligence. Artificial Intelligence, **2**:209–225.

SLOMAN, A. 1993. Varieties of formalisms for knowledge representation. Computational Intelligence, **9**(4):413–423.

STAHOVICH, T. F., R. DAVIS, and H. SHROBE. 1996. Generating multiple new designs from sketches. *In* Thirteenth National Conference on Artificial Intelligence, Montreal, pp. 1022–1029.

STENNING, K. 1993. Depictive versus descriptive representations: A distinction, but what is the difference. Computational Intelligence, **9**(4):353–355.

STENNING, K., and J. OBERLANDER. 1991. Reasoning with words, pictures and calculi: Computation versus justification. *In* Situation Theory and its Applications. *Edited by* J. Barwise, J. M. Gawron, G. Plotkin, and S. Tutiya. University of Chicago Press, Chicago.

STENNING, K., and J. OBERLANDER. 1992. Implementing logics in diagrams. *In* Reasoning with Diagrammatic Representations. Technical Report SS-92-02. American Association for Artificial Intelligence, Menlo Park, CA, pp. 91–95.

STENNING, K., and J. OBERLANDER. 1995. A cognitive theory of graphical and linguistic reasoning: Logic and implementation. Cognitive Science, **19**:97–140.

TABACHNECK, H. J. M., A. M. LEONARDO, and H. A. SIMON. 1994. How does an expert use a graph? A model of visual and verbal inferencing in economics. *In* Sixteenth Annual Conference of the Cognitive Science Society, Atlanta, GA. *Edited by* A. Ram and K. Eiselt. Lawrence Erlbaum, Hillsdale, NJ, pp. 842–847.

THAGARD, P., and S. HARDY. 1992. Visual thinking in the development of Dalton's atomic theory. *In* Proceedings of the Ninth Canadian Conference on Artificial Intelligence, Vancouver, BC.

ULLMAN, S. 1985. Visual routines. *In* Visual Cognition. *Edited by* S. Pinker. MIT Press, Cambridge, MA, pp. 97–159. (*Reprinted from* Cognition: International Journal of Cognitive Psychology, **18**:1–63, ISSN 0010-0277.)

WANG, D. 1995. Studies on the formal semantics of pictures. Ph.D. dissertation, Institute for Logic, Language, and Computation, University of Amsterdam.

WU, W.-T. 1978. On the decision problem and the mechanization of theorem proving in elementary geometry. Scientia Sinica, **21**:157–179.

YATES, J., M. BESSMAN, M. DUNNE, D. JERTSON, K. SLY, and B. WENDELBOE. 1988. Are conceptions of motion based on a native theory or on prototypes? Cognition, **29**:251–275.