1 Reifying Implicit Planning in Geometry: Guidelines for Model-Based Intelligent Tutoring System Design

Kenneth R. Koedinger
John R. Anderson
Carnegie Mellon University

This chapter addresses the problem of how basic cognitive science research can be translated into effective "cognitive tools" for learning. We present a set of guidelines that have been used to design a second generation Intelligent Tutoring System (ITS) for geometry and are intended more generally to aid the design of other computer or noncomputer based learning environments. Similar efforts (Anderson, Boyle, Corbett, & Lewis, 1990; Anderson, Boyle, Farrell, & Reiser, 1987; Collins, Brown, & Newman, 1989) have focused on how cognitive theories or principles can be generally applied to instructional design in any domain. In contrast, this chapter illustrates how cognitive science *methodologies* can be used in the instructional design process and we focus on a particular class of domains, ones where instructional innovations may pay off most. Although instructional innovations can certainly benefit from the application of general instructional principles, a large share of the instructional benefit often derives from insights (of researchers or teachers) into the particular instructional domain itself. This chapter describes a set of methodologies intended to help instructional designers gain such insights.

This approach is intended to apply to, and in fact to help identify, a particular class of *enigmatic domains*. These are domains that students find particularly mysterious, for example, geometry proofs and algebra word problems, such that even well-motivated students have difficulty (Koedinger, 1991 provides an operational definition of enigmatic domain). We elaborate a methodology for designing instruction, particularly computer-based instruction, for such domains.

A key point is to emphasize how empirical studies and cognitive modeling of skilled problem solvers can uncover *implicit planning:* thought processes of successful students that are not represented in the notation of conventional in-

struction. In turn, this model of implicit planning can be used to help design novel instruction. We present a detailed example in the domain of geometry where we have built a second generation cognitive model and ITS for geometry proof problem solving. We illustrate how this model has driven the design of ANGLE (A New Geometry Learning Environment).

A second point is to provide some general guidelines for translating a cognitive model of implicit planning into design specifications for the components of an ITS. Given a cognitive model of skilled problem solving, designing the expert component (i.e., the subject-matter knowledge) of an ITS is straightforward as it can simply be the computer implementation of this model. In the case that a novice model is also available, the expert component can be supplemented with buggy rules to provide the tutor with knowledge of students' common mistakes. These are the essentials of the model-tracing approach in which the tutor is designed to follow the student's problem-solving efforts relative to a cognitive model of ideal and buggy behavior (Anderson, Boyle, & Reiser, 1985).

Translating a cognitive model into design specifications for the interface and tutoring components of an ITS is not as straightforward. We take the model-tracing approach a step further here by providing some guidelines for how a cognitive model of skill can be used in the design of the interface and tutoring components. It is the role of these components to reify and articulate aspects of the model so as to communicate them to students. The guidelines we propose are a generalization of the model-based approach taken in the development of ANGLE. There are 5 steps:

1. Identify the *execution space*.
2. Look for *implicit planning* in verbal reports.
3. Model this implicit planning.
4. Use the model to drive tutor design.
5. *Test* the tutor implementation.

We turn now to description of the development of ANGLE. This description comes in 5 subsections following the 5 steps in the guidelines for model-based tutor design we are proposing. In the following section we generalize these steps in 5 parallel subsections.

ANGLE: A 2ND GENERATION TUTOR FOR GEOMETRY

The Geometry Execution Space

Geometry proof problem solving is hard. For a typical geometry proof, the search space of possible geometry rule applications (i.e., definitions, postulates, and theorems) is quite large. To get an idea of the size, we performed a detailed

analysis of a typical problem (this problem appears in Figs. 1.4–1.7). Of the 27 definitions, postulates, and theorems that are introduced prior to such a problem in a traditional curriculum, 7 can be applied at the beginning of this problem. Some of these rules can be applied in more than one way yielding 45 possible inferences that can be made from this problem's givens. Using the results of these inferences, essentially as new givens, we did the same thing over again and found that 563 inferences can be made at this second layer. At the third layer the options really explode as there are more than 100,000 possible inferences. The number of options continues to increase at further layers—at minimum it takes 6 such layers of inferences to reach the problem goal.

For a student to be effective at solving geometry proof problems, he or she must have a method for dealing with this complexity. Many methods are possible as illustrated by the variety of artificial intelligence programs and cognitive models that have been developed for this domain (Koedinger & Anderson, 1990 provide a review). While these previous models differed in the general search strategies and domain specific heuristics they employed, most were built on top of the same underlying problem space representation—one that has the formal geometry rules as operators.[1] We call this problem space the *execution space* and, in general, we define the execution space for a domain as a problem space made up of states that correspond with the steps that problem solvers conventionally write down or *execute* in solving a problem in the domain. The preceding analysis illustrates an exhaustive search of the execution space in geometry.

To more effectively search this problem space, model builders have incorporated more sophisticated search strategies and included more domain specific heuristic knowledge. The search strategies have ranged from backward search in Gelernter's (1963) original "geometry theorem proving machine" to an opportunistic bidirectional search (works both forward and backward) in Anderson, Boyle, and Yost's (1985) cognitive model of skilled geometry students. The heuristic knowledge has ranged from using the problem diagram to reject goal statements that are clearly false (in Gelernter's model) to the addition of specific contextual features that predict the relevance of a rule (in Anderson et al.'s model). Despite these improvements, the key point is that the underlying problem representation has remained the same.

As the definition of execution space suggests, this problem representation is a reflection of the way geometry proofs are conventionally written, for example, in traditional geometry textbooks. Although such textbooks do not explicitly describe a problem-solving method for doing proof problems, the implied method is search in the space of formal rules. One indication of this method is the way traditional textbooks are organized around formal rules: introducing a rule, illustrating it, providing exercises with it, and then on to the next. A second

[1] See Newell (1990) or Newell and Simon (1972) for a review of the problem-solving terminology used here.

indication is that the steps in example problem solutions correspond with geometry rules. It seems quite natural then, for students reading such a text (and cognitive scientists, for that matter) to view their problem-solving task as one of searching for an appropriate rule to apply at each step in the problem.

In summary, early geometry problem-solving models assert heuristic search in the execution space as a psychologically plausible characterization of geometry problem solving. Similarly, traditional geometry instruction can be interpreted as implicitly making the same assertion. This assertion is a good first approximation as evidenced, for example, by the Anderson, Boyle, and Yost (1985) cognitive model and its use in the Geometry Proof Tutor which has been shown to be an effective instructional device (Anderson, Boyle, Corbett, & Lewis, 1990).

However, when we took a closer look at the behavior of skilled geometry problem solvers, we found that their mode of attack is distinctly different from the approach of these earlier models. Instead of making local heuristic decisions within the execution space, skilled problem solvers focus their search by initially developing a global solution plan using a representation that is more abstract (and more compact) than the execution space. They were bringing informal perceptual and conceptual knowledge to bear in their initial planning and were leaving the formal details for last. It seemed important to be able to characterize this knowledge both as a goal in and of itself and for pedagogical purposes.

Evidence of Implicit Planning: Step-Skipping

We have referred to the kind of planning we observed in skilled subjects as *implicit planning* to emphasize that it occurs in a problem space that is not explicitly represented in the notation of conventional instruction. In other words, implicit planning is (informal) planning in a problem space other than the (formal) execution space. The identification of implicit planning in a domain provides a first step in the design of novel instruction in that it indicates a kind of thinking which is not supported by or taught in current instruction.

A key piece of evidence for implicit planning in geometry was the observation that in their initial attempt at a problem solution skilled subjects were not thinking about every step in the execution space solution. Instead, they developed a solution sketch in which numerous execution steps were skipped. Consider the protocol in Table 1.1 of a skilled subject solving the problem shown in Fig. 1.1.

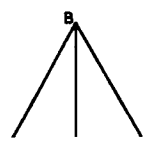


FIG. 1.1. A geometry proof problem solved by a skilled problem solver whose verbal

TABLE 1.1
A Verbal Protocol for a Skilled Subject Solving the Problem in Fig. 1.1

		Planning Phase
B1:	We're given a right angle–this is a right angle,	Reading given: rt <ADB
B2:	perpendicular on both sides [makes perpenedicular markings on diagram];	Inference step 1: AC ⊥ BD
B3:	BD bisects angle ABC [marks angles ABD and CBD]	Reading given: BD bisects <ABC
B4:	and *we're done*	Inference step 2: ΔABD = ΔCBD
		Execution Phase
B5:	We know that this is a reflexive [marks line BD].	In this phase, the subject refines and explains his solution to the experimenter.
B6:	we know that we have congruent triangles; we can determine anything from there in terms of corresponding parts	
B7:	and that's what this [looking at the goal statement for the first time] is going to mean...that these are congruent [marks segments AD and DC as equal on the diagram].	

The left side of the table contains the protocol and the right side indicates our coding of the subject's actions.

At the point where he said "we're done," this subject had developed a correct solution sketch for this problem in only 13 seconds. He plans this solution sketch without looking at the goal statement (see Koedinger & Anderson, 1990 for an explanation of this curious behavior) and in the remainder of the protocol he elaborates the solution sketch, reads the goal statement, and explains how it is proven. His words "we're done" indicate his realization that the two triangles ABD and CBD are congruent and that therefore he knows everything about the whole problem—as he explains later: "we can determine anything from there in terms of corresponding parts."

Figure 1.2 shows the 7-step execution space solution to the problem. Apart from the givens and goal, the statements which the expert mentioned while solving this problem are numbered in Fig. 1.2, while the skipped steps are circled.

Consistent with the evidence that verbalizations are accurate reflections of working memory states (Ericsson & Simon, 1984), we conclude that the expert only makes certain key inferences in his search for a solution while skipping other, apparently minor inferences.[2] A person following the implicit message of the traditional curriculum would work in the execution space (as do previous

[2]Ericsson and Simon (1984) stipulate that this claim is true only for verbal reports of problem solving that are given concurrently with problem solving. These are the kind of protocols we collected.

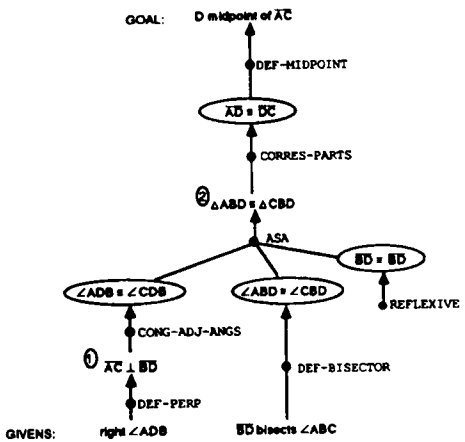


FIG. 1.2. The final solution for the problem shown in Fig. 1.1. The givens of the problem are at the bottom and the goal is at the top. The lines represent inferences with the conclusion at the arrow head, the premises at the tails, and the justifying geometry rule at the dot in between. The statements Subject R mentioned during planning (see Table 1.1) are numbered while the ones he skipped are circled.

problem solving models) and require at least seven thinking steps to come up with a solution plan.[3] In contrast, this subject came up with a solution plan in two thinking steps. It became clear to us that this subject (and our other skilled subjects; see Koedinger & Anderson, 1990) were bringing to bear a different kind of knowledge than that involved in heuristic search of the execution space.

DC: A Model of Implicit Planning in Geometry

Based on these observations, we tried to design a model of expert geometry theorem proving that would be both more powerful and more like skilled humans than previous systems. The model we came up with, the Diagram Configuration model (DC), is briefly summarized here. DC has one major knowledge structure, diagram configuration schemas, and three major processes: diagram parsing, statement encoding, and schema search.

The core idea of the DC model is that experts have their knowledge organized according to diagrammatic schemas which we call *diagram configuration schemas*. These are clusters of geometry facts that are associated with a single prototypic geometric image. Figure 1.3 shows two diagram configuration schemas.

The *whole-statement* and *part-statements* attributes of a schema store the facts that are associated with the geometric image stored in the *configuration* attribute. The configuration is a prototypic configuration of points and lines which is commonly a part of geometry diagrams. In Fig. 1.3, the configuration on the left is a prototype for any set of lines that form two congruent triangles with a side in

[3]More than seven thinking steps would result from pursuing blind alleys or arriving at a longer

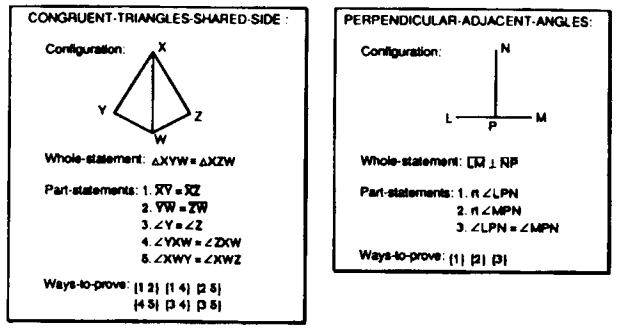


FIG. 1.3. Two examples of diagram configuration schemas. The numbers in the ways-to-prove indicate part-statements. Thus, in the CONGRUENT-TRIANGLES-SHARED-SIDE schema {1 2} means that if the part-statements $\overline{XY} = \overline{XZ}$ and $\overline{YW} = \overline{ZW}$ are proven, all the statements of the schema can be proven.

common. The whole-statement is the geometry statement that refers to the configuration as a whole. The part-statements refer to properties of the parts of the configuration. The whole-statement of the CONGRUENT-TRIANGLES-SHARED-SIDE schema refers to the two triangles involved while the part-statements refer to the corresponding sides and angles of these triangles. The *ways-to-prove* are used to determine whether inferences can be made about a configuration. They indicate subsets of the part-statements which are sufficient to prove the whole-statement and all of the part-statements. For example, the first way-to-prove of the CONGRUENT-TRIANGLES-SHARED-SIDE schema, {1 2}, indicates that if the part-statements $\overline{XY} = \overline{XZ}$ and $\overline{YW} = \overline{ZW}$ have been proven, the schema can be proven—that is, all the other statements of the schema can be proven.

Our basic proposal is that skilled problem solvers plan in terms of these schemas rather than the statements of geometry. The problem solver tries to establish that various schemas are true of the diagram associated with a problem. Establishing one schema may enable establishing another. Because there are a small number of schemas possible for any particular problem diagram, the search space of schemas is much smaller than the execution space.

DC does a type of instance- or model-based reasoning by treating the problem diagram as a specific instance or model (Johnson-Laird, 1983) of the abstract problem statement indicated by the givens and goals of the problem. This use of the diagram as a model allows DC to do some inductive reasoning to aid in the solution of what is nominally a deductive problem. DC induces relationships that look true in the problem diagram (this involves measurements of relative sizes of segments and angles) and uses these relationships to constrain the search space, which would otherwise include many syntactically well-formed statements that are clearly not true in the model

The logic of this trick relies on the fact that since the diagram is an instance of the proof being constructed, any relationship that can be proven will appear true in an accurately drawn diagram. As a concrete model, the diagram has the advantage of making important relationships clearly apparent (visible in this case) whereas they are only implicit in abstract statements. These perceptual cues have the effect of allowing the problem solver to avoid lots of semantically impossible or irrelevant inferences that may appear relevant when trying to syntactically apply the formal execution space rules.

We can explain the subject protocol in Table 1.1 in terms of the DC model as follows. In B1, he reads the first given and as he marks it in the diagram, he recognizes an instance of the PERPENDICULAR-ADJACENT-ANGLES schema. Since this given satisfies one of the ways-to-prove of this schema, the schema's properties (i.e., the whole-statement and part-statements) can be proven. Correspondingly in B2 the subject says "perpendicular on both sides" which, though not technically well-formed, essentially expresses the three part-statements of the PERPENDICULAR-ADJACENT-ANGLES schema in one neat phrase. In B3, he reads the second given, encodes it as $\angle ABD \cong \angle CBD$, and marks this in the diagram. At this point (or possibly before) he makes a perceptual recognition that two of the triangles in the diagram look congruent and share a side, that is, he identifies an instance of the CONGRUENT-TRI-ANGLES-SHARED-SIDE configuration. Thus, this schema is cued and considering the ways-to-prove, he sees that a part-statement of the PERPENDICULAR-ADJACENT-ANGLES schema ($\angle ADB \cong \angle CDB$) and the encoding of the second given ($\angle ABD \cong \angle CBD$) satisfy one of them ({4 5}). Because he knows he can prove all the properties of this schema and since this schema constitutes the whole diagram, he is able to say "we're done" at this point without reading the problem goal.

Evaluation of DC. In Koedinger and Anderson (1990), we established the empirical accuracy of DC by showing that it does a good job of accounting for the steps that subjects mention or skip in their verbal reports of proof planning. We also demonstrated its computational efficiency by illustrating how much smaller the schema space is than the execution space. For example, in contrast to the greater than 100,000 states in the complete execution space for the problem discussed on p. 17, the complete DC space for this problem contains only 8 states.

A Point About Learning. It is important to consider whether the DC model and the method of skilled subjects is distinctly different from that of search in the execution space. One competing hypothesis is that what we've observed in skilled subjects is the result of a skill acquisition process (e.g., ACT*'s composition, Anderson, 1983; or Soar's chunking, Newell, 1990) which has chunked

together execution space operators into *macro-operators*. If this were the case, that is, if it were necessary to learn the execution space operators first before learning schemas, then training students in the execution space (as the traditional curriculum implicitly does) would be appropriate. However, we not believe this to be the case.

Although it is true that DC's schemas can be *represented* as a number of execution space macro-operators, there is good reason to believe that schemas are not actually *derived* (learned) from execution operators. In Koedinger and Anderson (1989, 1990) we argued that, among other things, there is too much regularity in the kinds of steps experts skip to be accounted for by a macro-operator type mechanism. The number of possible ways of syntactically combining execution space rules into macro-operators is far greater than the relatively small number of schemas that correspond with expert step-skipping. Macro-operator learning mechanisms cannot explain this limited, but sufficient, set of schemas in a parsimonious way.

The observation that DC schemas correspond with the perceptual structure of geometric objects has led us to propose that, in contrast to macro-operator learning. DC schemas are the result of an inductive learning mechanism that acquires expert operators directly from interactions with the learning environment and not so much from modifying and combining previously acquired operators. This inductive learning mechanism must be capable of identifying categories of percepts (configurations) and attaching the corresponding formal properties (whole-statement and part-statements) to these percepts.

Further support for this proposal comes from data collected in our first evaluation of ANGLE (summarized on p. 33). The hypothesis that DC schemas are learned from an earlier execution space representation predicts that step skipping will not be apparent in novices, but will only emerge as enough expertise is acquired. In contrast to this hypothesis, we found evidence that novices are also occasionally skipping steps quite early in the skill acquisition process.

DC Summary. In the next section we describe how DC has been used as the expert component for a second generation ITS for geometry called ANGLE. We end this section by comparing DC with the expert component of the first generation Geometry Proof Tutor (GPT). See Table 1.2. In GPT's expert component, the knowledge was organized around the *formal rules* of geometry. These formal geometry rules appeared as parts of production rules that also contained context cues to indicate likely situations in which the formal rule would be usefully applied. The productions implement a bidirectional heuristic search strategy that has a local view in that decisions are only made about the next formal step in the proof (either forward or backward). In contrast, ANGLE's expert component is organized around informal knowledge of geometric percepts (configurations) and associated concepts (part-statements and ways-to-prove). It searches a different

TABLE 1.2
Differences Between GPT's and ANGLE's Expert Components

	Knowledge	*Strategy*
GPT Expert	Formal rules	Local heuristic search
ANGLE Expert	Percepts & concepts	Global planning

problem space that takes a more global planning view where each decision typically involves many proof steps. We describe how these different cognitive models lead to different tutoring systems in the following section.

Tutor Design: From DC to ANGLE

In this section, we discuss the design of ANGLE and in particular, how its major components, the expert, interface, and tutoring component have been influenced by the cognitive model of problem solving described earlier. The expert component is the subject-matter knowledge of the tutor, the interface component determines how the student interacts with the system, and the tutoring component provides the verbal advice the system gives the student.

Table 1.3 summarizes a methodology for model-based tutor design that indicates a way to translate from a cognitive model of problem solving into design specifications for the interface and tutoring components of an ITS. Given a model of successful students' implicit planning, one needs to find ways to communicate the knowledge representations and cognitive processes that make up this model.

The interface component provides an implicit form of instruction in that students can learn through their perception and interaction with it. Thus, one avenue to teaching the problem-solving model is to invent interface *notations* and *actions* that reify the underlying representations and processes of the model. Interface notations should mirror the model representations such that students can begin to internalize these representations through repeated perception and use of the notations. Similarly, interface actions should mirror the model processes such that students can learn these processes through repeated performance of these actions.

TABLE 1.3
Ways to Communicate a Problem-Solving Model in an ITS

	Representations	*Processes*
Interface	Notations	Actions
Tutor Advice	Vocabulary	Hints and Explanations

Whereas the interface component provides an implicit form of instruction, the tutor component provides a direct way to instruct students on the representations and processes of the problem solving model. Both ANGLE's tutoring messages and the off-line text materials use specific vocabulary to directly articulate the schema representations used in DC. Similarly, the content and order of ANGLE's *hints* and *explanations* have been designed to articulate the problem-solving processes (e.g., diagram parsing, abstract planning) incorporated by DC.

Reifying the Model Through Interface Notations and Actions. The ANGLE interface includes a number of examples of notations that reify representations in the DC model. The most prominent and important are icons for representing the generic schema categories and for representing student-selected instances of these schemas (see Fig. 1.4). These icons provide a concrete image to which students can attach the related conceptual knowledge about the part-statements

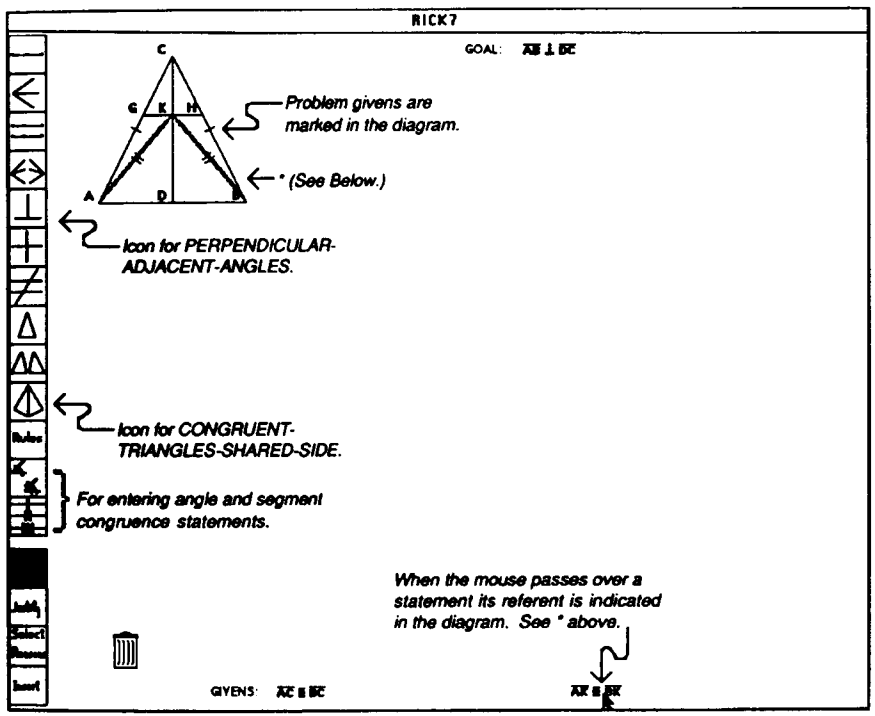


FIG. 1.4. The ANGLE interface at the start of a problem. The italicized annotations are not part of the interface.

and ways-to-prove. In particular, they reinforce the perceptual character of schemas.

Following GPT, ANGLE incorporates a graph representation of proofs in contrast to the two-column format of traditional geometry instruction. Figure 1.5 illustrates a complete proof plan represented in the proof graph notation, while Fig. 1.7 illustrates the complete proof after execution. This proof graph notation is important because it reifies the search process:

1. by explicitly indicating how a correct solution must be a chain of steps linking the givens to the goal,

2. by allowing the entering of subgoals as possible future links in the solution chain, and

3. by explicitly indicating dead-end solution attempts which are a common part of problem solving even for experts.

Collins and Brown (1990) discuss how this type of "spatial reification" provides a vehicle for learning through reflection.

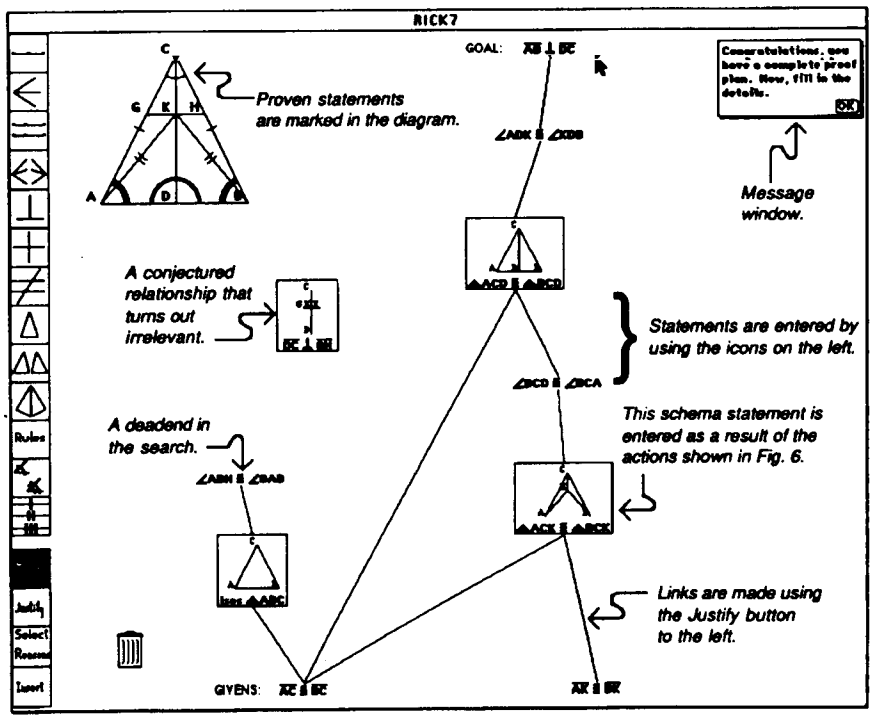


FIG. 1.5. A completed proof plan. The student performed some extra steps that aren't relevant to the solution. The italicized annotations are not part of the interface.

Following a conventional notation used on paper, ANGLE uses hash marks in the diagram to indicate part-statements that have been proven (see, for example, segments AK and BK in Fig. 1.5). These markings reify an efficient equivalence class representation of segment and angle congruence in contrast to the more cumbersome binary relationships of the formal notation. For example, to indicate that 3 angles are all congruent to each other using the hash mark notation, one can mark all three with the same marking—the marking serves as a token of the equivalence class containing all three angles. In contrast, the formal notation requires three binary statements to represent this situation, for example, $\angle 1 \cong \angle 2$, $\angle 2 \cong \angle 3$, and $\angle 1 \cong \angle 3$.

Finally, as a further aid to the acquisition of schemas, ANGLE highlights a schema within the problem diagram whenever the mouse passes over the corresponding schema instance icon in the proof graph. This is intended to reinforce the relationship between the schema and the rest of the diagram. This is a kind of dynamic notation that cannot be feasibly employed with paper and pencil.

Whereas interface notations were created to reify representations in the DC model, interface actions were designed to reify processes. Following the major processes in DC, ANGLE interface actions are broken down into (1) diagram parsing actions, (2) planning actions, and (3) execution actions. The grain size of parsing and planning actions is designed to emphasize schemas, while the focus on statements and rules is left for execution actions.

The *diagram* parsing actions are those done in order to enter schemas. First the student selects a schema type and then indicates the lines within the diagram that make up an instance of this schema. Figure 1.6 shows the entry of an instance of the CONGRUENT-TRIANGLES-SHARED-SIDE schema. This particular way of constructing a statement, as opposed to the way a student constructs one in GPT or on paper, is meant to reinforce the relationship between the schema instance and the problem diagram in which it is embedded.

The *planning actions* are those done in order to justify schemas and part-statements. To some extent these actions are more elaborate than those made in a two-column proof on paper: In ANGLE, the student must explicitly indicate the premises that lead to a conclusion (that is, by drawing the lines between them), while in the typical two-column proof these links are only implicit. On the other hand, the planning actions are for the most part less elaborate than those required for a two-column proof. Certain details required in a two-column proof can be left out while constructing a plan. Students can omit (a) certain statements usually required in a complete proof, for example, the reflexive statement $\overline{CK} \cong \overline{CK}$ in the first inference in Fig. 1.5, and (b) the rules or "reasons" that usually appear in the right column of a two-column proof.

After a proof plan has been discovered (as shown in Fig. 1.5), the student can fill in the details that were left out in the proof plan by performing *execution actions* which include adding new statements and inserting rules. Figure 1.7 shows the final result of adding all the details to the proof plan in Fig. 1.5. The

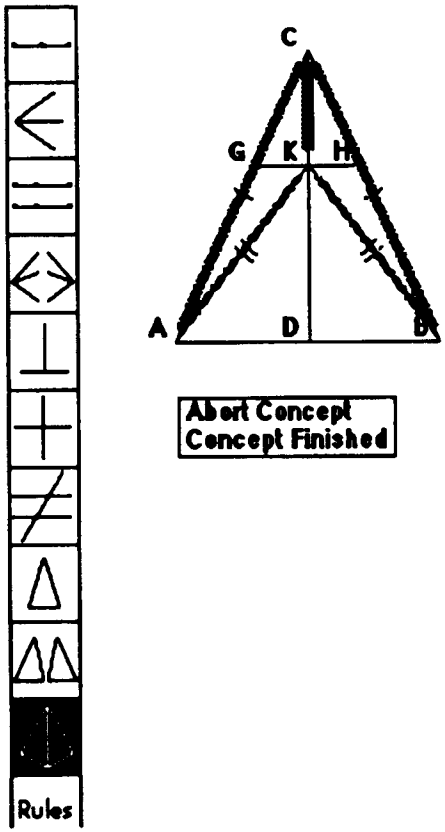


FIG. 1.6. Selection of an instance of the CONGRUENT-TRIANGLES-SHARED-SIDE schema: $\triangle ACK \cong \triangle BCK$. This method of schema selection is intended to reify the diagram parsing process.

idea of the separation of planning and execution is that, just as in DC, the student need not worry about the details while searching for a plan. When a plan is found, he need only fill in the details along the path that leads to the goal.

Any tutor interface (or notational scheme for that matter) is implicitly taking an instructional stance about what things are difficult and/or important to learn and what things are not. Three important aspects of the instructional stance taken by ANGLE's interface are worth making explicit: (1) learning about the logical linkages between proof steps is difficult and important, (2) learning the details of proof execution is less important and perhaps less difficult than learning proof planning, and (3) learning how to parse geometry diagrams into particular chunks (DC-schemas) is difficult but it is important for successful search in a vast problem space. The first point is shared by GPT's interface, but not by the two-column notation used on paper. The last two points are special to ANGLE's interface.

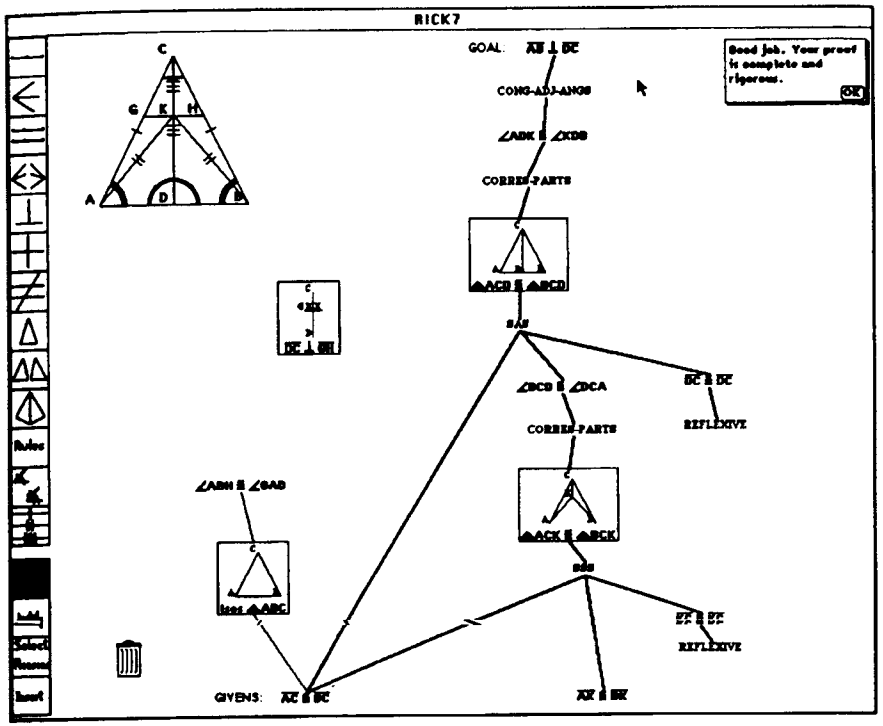


FIG. 1.7. The final, detailed solution in the execution space. The dark links indicate execution inferences while the lighter links indicate planning inferences. Note that the planning inferences that are not relevant to the solution are not executed.

Articulating the Model Through Tutor Vocabulary and Hints. While the interface component was designed to reify model representations and processes with interface notations and actions, the tutor component was designed to articulate model representations and processes with the vocabulary and explanations employed by the tutor.

In supplemental text materials, schemas are explicitly presented to students in a format similar to that shown in Fig. 1.3. We decided to call schemas "concepts" as this word is much more familiar to most students. In addition to talking about "concepts," the vocabulary of the schema slot-names, "configuration," "part-statement," and "ways-to-prove" are an explicit part of both the tutor advice and the supplementary text materials.

The content of the feedback ANGLE gives when students make logical errors (i.e., an illegal justification attempt) was also designed to help students learn the schema representation. This logical error feedback first enforces the constraint

(1) that only part-statements of a schema can be used to prove it. When this constraint has been satisfied the tutor then checks (2) that the part-statements the student selected match one of the ways-to-prove. In this way, students can learn that when trying to prove a schema, they need not consider any statements besides the part-statements of that schema.

One straightforward way of communicating model processes is through the content of the hints the tutor provides. ANGLE's hints for entering a particular schema explicitly direct the student toward diagram parsing (a key process in the cognitive model). The hint to enter a CONGRUENT-TRIANGLES schema, for example, encourages careful diagram parsing by asking students to count the triangles in the diagram.

Besides the content of hints, a less obvious way of communicating model processes is through the order in which hints are given. In other words, the order in which hints about appropriate problem-solving steps are given should correspond with the order of these steps in the cognitive model. Following through on this design guideline results in one of the biggest differences between ANGLE and the first generation GPT.

Consider the situation in Fig. 1.8. If a student were to ask for help from GPT at this point, it would focus on using the corresponding-parts rule to make the inference from $\triangle BCD \cong \triangle YZW$. The message would be something like "Notice two triangles which are congruent. Think about what this tells you." This is not necessarily a bad hint. But, it lacks a strategic context that might help the student decide which of the six corresponding parts it might be useful to prove. We've observed students blindly following this advice, proving nonessential parts congruent (e.g., $\angle CBD \cong \angle ZYW$), getting the advice again, proving another set of nonessential parts congruent (e.g., $\angle CDB \cong \angle ZWY$), and so on. Eventually, they stumble upon the right corresponding parts, in this problem both $\overline{DC} \cong \overline{WZ}$ and $\angle DCA \cong \angle WZX$ are useful, and only then will GPT start giving advice towards proving $\triangle ACD \cong \triangle XZW$.

ANGLE approaches this situation more strategically, that is, starting with $\triangle ACD \cong \triangle XZW$ and moving back to the enabling part-statements $\overline{DC} \cong \overline{WZ}$ and $\angle DCA \cong \angle WZX$. If the student needs help, ANGLE would first provide *schema selection hints* to encourage her to look in the diagram for triangles that she might prove congruent. The student is advised toward entering $\triangle ACD \cong \triangle XZW$ as an island subgoal. See Fig. 1.9.

Next, she should work on finding part-statements which she can prove from $\triangle BCD \cong \triangle YZW$ and which are relevant to this goal. If she has trouble, ANGLE provides *part-statement justification hints*. In this situation, the most general such hint would suggest: "Look for OVERLAPPING concepts. That is, look for a part-statement which appears both in $\triangle ACD \cong \triangle XZW$ and in a concept you've already proven."

Figure 1.9 shows the addition of the necessary part-statements. If the student has trouble here, ANGLE would provide a *schema justification hint* that would

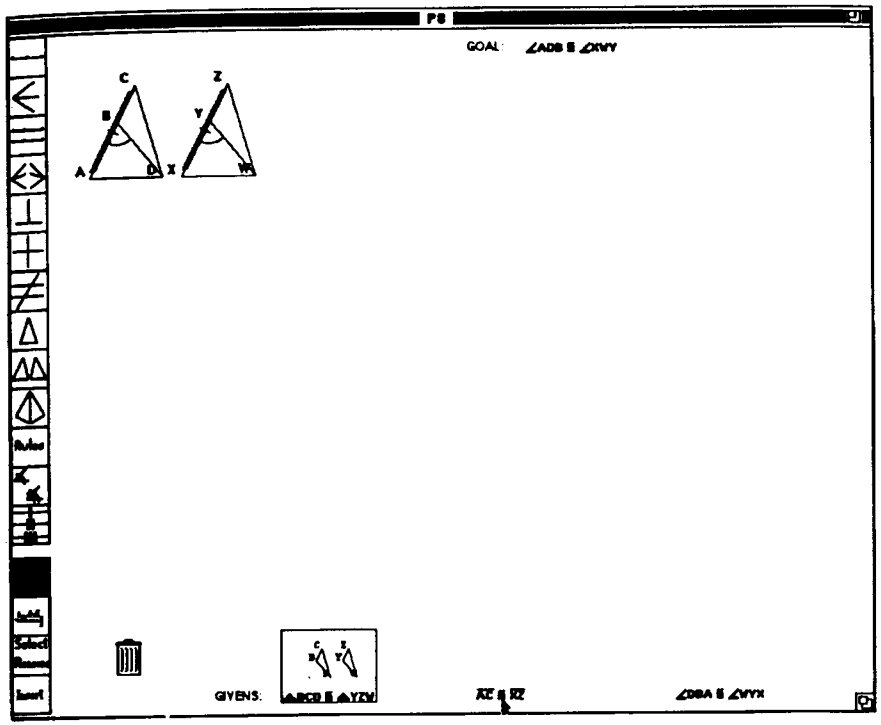


FIG. 1.8. The context for a strategic hint. The next hint will focus on the selection of the $\triangle$**ACD** $\cong$ $\triangle$**XZW** schema and only then, will the hints focus on part-statements that follow from $\triangle$**BCD** $\cong$ $\triangle$**YZW.** These hints suggest to select those part-statements which will help to prove $\triangle$**ACD** $\cong$ $\triangle$**XZW.**

work her toward finding the appropriate premises ($\overline{DC} \cong \overline{WZ}$, $\angle DCA \cong \angle WZX$, and the given $\overline{AC} \cong \overline{XZ}$) with which to justify $\triangle ACD \cong \triangle XZW$.

ANGLE Summary. Table 1.4 presents contrasts between the first generation GPT and the second generation ANGLE along the dimensions of tutor design discussed before: interface notations and actions to reify model representations and processes and tutor vocabulary and hints to articulate model representations and processes. While the interface notations of GPT focus on the formal rules of geometry, ANGLE's notations focus on diagram configuration schemas. The interface actions of GPT reify rule application by breaking it down into three steps: selecting premises (the if-part), typing the rule name, and entering the conclusion (the then-part). In contrast, ANGLE's interface actions are at a larger grain size reifying the processes of diagram parsing and schema justification and search. In the tutor component, GPT's messages exclusively use the vocabulary

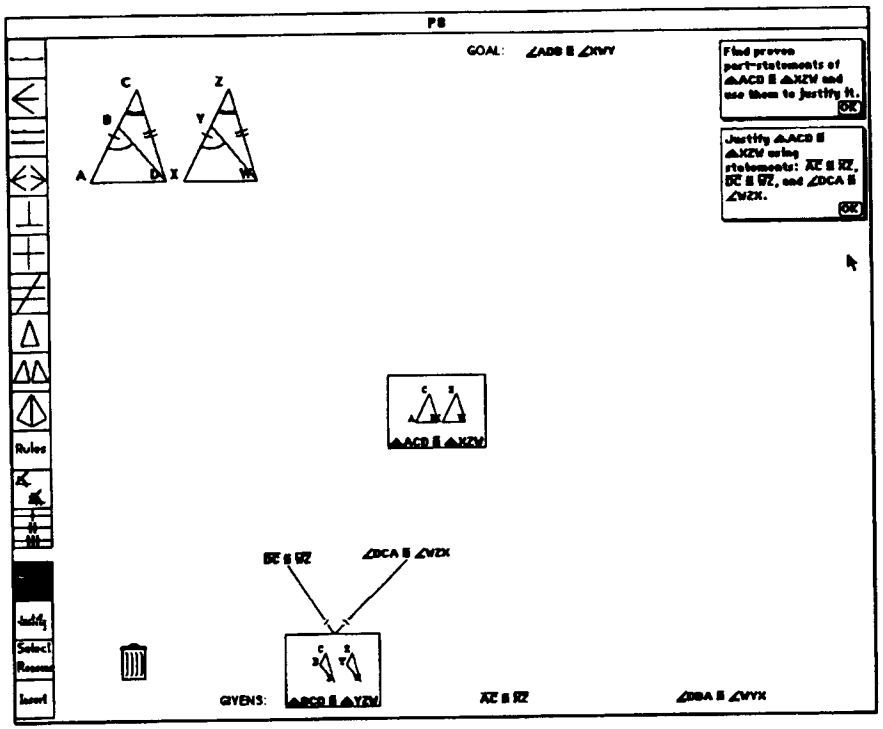


FIG. 1.9. The needed part-statements have been added. Both a general hint and the bottom-out hint for the next step are shown in the message windows in the top right. The focus is on justifying **△ACD ≅ △XZW** using the proven part-statements.

TABLE 1.4
Contrasts Between the Interface and Tutor Design of GPT and ANGLE

	Interface Component		*Tutor Component*	
	Notations	*Actions*	*Vocabulary*	*Hints*
GPT	Rules	Premise Rule Conclusion	Traditional	Local only: Execution steps
ANGLE	Schemas	Diagram parsing Schema search	"concepts" "part-stmts" "ways-to-prove"	Global: Schemas Local: Rules

of the traditional curriculum, while ANGLE's messages (and supplementary text materials) make choice use of novel vocabulary to articulate crucial aspects of the schema representation. Lastly, GPT's hints articulate its underlying model of heuristic search in the execution space and thus, they are at the grain size of formal geometry rules (execution steps). In contrast, ANGLE's hints articulate the processes in DC and thus, start at the larger grain size of schemas and then move to the more detailed grain size of rules.

Testing the Tutor Implementation: A Formative Evaluation of ANGLE

This section describes some of the results of our first empirical study of ANGLE. The central goal of our empirical evaluations of ANGLE is to test the hypothesis that the development of more accurate and powerful cognitive models of problem solving can lead to major improvements in the instruction of problem solving, particularly within the context of intelligent tutoring systems. In this study, we compared GPT and ANGLE—two tutors for the same domain whose primary difference is the cognitive model that underlies them. While the cognitive model underlying GPT is a reasonably good one, it has some flaws. It does not do a good job of capturing the more strategic abilities of effective problem solvers. The DC model improves on GPT's model by providing a sound explanation of these abilities. The question is: can this improvement lead to more effective instruction?

ANGLE is still in an early development stage. In particular, the current implementation of the feedback scheme and the wording of the feedback messages are only a reasonable first pass attempt. It takes iterations of usage and redesign to get a good sense for how to make them most effective. ANGLE represents about 1.5 person-years of effort. This study was its first extensive trial. In contrast, GPT represents about 10 person-years of effort and has been through at least 2 major iterations of study and redesign.

Method Summary. 30 high school students were paid to participate. To reduce the amount of declarative instruction in geometry and focus on proof problem solving, we required these students to have just completed a high school geometry course. They came in the lab for six 2-hour sessions and started with pretesting, then worked with either GPT or ANGLE for 8 hours of tutoring, and finished with posttesting. Half the students worked with GPT, the other half with ANGLE. Given the relatively short training time, a subset of the geometry curriculum was used involving the topics of perpendicularity and triangle congruence. It covered 8 rules in the execution space and 4 schemas in the diagram configuration space.

Two versions of the tests (A & B) were used to counterbalance test order. Here we report on the results of one of the four tests given—the proof construction test

which provides the most direct measure of students' proof performance. Both the A & B versions of the proof test had four proof problems on it, the first item was of medium difficulty (about the difficulty of the problem in Fig. 1.1: 5 execution steps or 2 schemas in the shortest solution), the second two were difficult (8 execution steps, 3 schemas) and the last was very difficult (10 or 11 execution steps, 4 schemas). A detailed overall score was taken by giving one point for each correct execution step (32 points for version A, 31 for B). In addition to the test results, both tutors collected detailed on-line data of student-tutor interactions.

Overall Results: ANGLE About Equal to GPT. On the proof test, ANGLE subjects went from a mean pretest performance of 11.0 to a posttest performance of 19.6, while GPT subjects went from 11.9 on the pretest to 23.6 on the posttest. Consistent with the previous results of GPT, both groups showed significant learning ($p < .001$), however, there is no significant difference on posttest performance ($p > .13$). At this early stage of development, ANGLE appears to be matching up with a proven tutor. However, there is room for improvement and this study helped identify some weaknesses in the implementation of ANGLE that we can remedy.

Better Instruction of Planning Than Execution. In addition to the overall score reported earlier, the proof tests were also scored using a planning measure where subjects got one point for every correct "schema step" in this solution. Schema steps are those that uniquely indicate a schema (e.g., whole-statements). Using this measure we find that ANGLE subjects went from a mean score of 5.5 to 9.7 (on a scale of 12), while GPT subjects went from 5.9 to 10.0. The trend in favor of GPT in the overall score disappears in this planning score. To better understand what was going on here, we counted the number of skipped steps in the proof solutions—that is, cases where the subject had a correct schema step but some of the corresponding execution steps were missing. We found that while skipping about the same number of steps on the pretest, ANGLE subjects skipped significantly more steps on the posttest, 3.5 to GPT subjects' 1.0 ($p < .001$). In other words, although they were planning equally as well, ANGLE subjects are making more execution errors on the posttest.

This result is not surprising given that the development of ANGLE to this point has primarily focused on tutoring facilities for planning. ANGLE's tutoring facilities for execution actions were rather minimal and these results would seem to indicate that such minimal execution tutoring is not sufficient for efficient instruction of proof execution skills.[4]

[4]It should be pointed out that proof execution skill (i.e., skill at the formal details of proofs) is not necessarily a primary instructional goal. Some educators are more concerned with more informal, but accurate, reasoning along the lines of DC's conceptual planning.

On-Line Tutor Data: Too Much Flexibility? The on-line tutor data provided additional pointers to areas where ANGLE can be improved. From observations during the study, it appeared that ANGLE subjects were floundering more often and for longer periods than GPT subjects. This observation was corroborated by the fact that in the fixed 8 hours of training GPT subjects finished significantly more problems than ANGLE subjects, 48.5 to 39.1 respectively ($p < .05$). The number of problems completed was highly correlated with posttest performance ($r = .73$) suggesting that this factor is an important one in learning. Part of the problem with ANGLE seemed to stem from the greater flexibility and complexity of its interface relative to GPT's.

The ANGLE interface was more complex both because it had more types of actions to learn (necessary to implement the distinction between planning and execution), and because it allowed more alternative sequences of actions. Although the ideal sequence of actions is to start by alternating between diagram parsing and planning actions and then finish with a series of execution actions (i.e., following the DC model's approach), ANGLE allowed students to perform diagram parsing, planning, and execution actions in any order. While providing such flexibility seemed like a laudable implementation goal, the interaction of this flexibility with certain other implementation decisions had some negative consequences.

Bottom-out Hint Problem. One of the more serious problems involves a decision all ITS designers must face: What to do if the student is really stuck and the hints the tutor is giving are not helping. Many ITSs, including ANGLE and GPT, follow a general-to-specific hinting strategy where the first hint in a particular context is very vague, not giving too much away, and if the student still has trouble, the hints get progressively more specific. Eventually they *bottom-out* at the most specific hint. In both ANGLE and GPT, bottom-out hints are given only after the student has made numerous logical errors or requests for help (4 or 5 on average) without making any progress, that is, performing a legal step. One type of bottom-out hint that guarantees continual progress toward the solution is to have the tutor do the next step for the student.[5] This is the approach taken in GPT. In ANGLE, we took the slightly less extreme approach of telling the student what to do next without actually doing it.

Unfortunately, students were not always able to translate ANGLE's bottom-out hints into the appropriate actions and because of the flexibility designed into ANGLE, students were not forced to perform them. In turns out that this situation arose quite often—bottom-out hints were given on 23% of the 561 problems solved.[6] The fact that students often did not immediately perform the suggested

[5]At the other extreme, an ITS designer might decide that in a classroom situation it is sufficient to direct the student to the teacher, for example, "You seem to be having a lot of trouble. Please ask your teacher for help."

[6]Some of the early protocol data was lost. Students actually solved 586 problems.

next step is measured by two statistics. Since ANGLE repeats the bottom-out hint if the student continues to have trouble, one relevant statistic is the number of times a particular bottom-out hint was repeated on average. Bottom-out hints were repeated about once (0.97) on average. The second statistic is more telling and leads to a more dramatic result. It is the amount of time that is spent between the time the bottom-out hint is first given and the time the student performs the suggested next step. Because there is more than one solution to these problems, it is possible for the student to finish the problem without ever performing the suggested next step (this happened for only 8% of the bottom-out hints given). In the cases where students do end up eventually performing the suggested next step, about 75 seconds elapse on average. Multiplying this by the number of bottom-out hints given, we find that the students in this study spent almost one half hour on average (26.6 minutes) between being told what to do next and actually doing it. (26.6 minutes is about 6% of the total 8 hours of tutor instruction.) Recall, that bottom-out hints are given only after the student has already given evidence of floundering (e.g., 4 or 5 logical errors or requests for help) and thus, this period of time might be characterized as extreme floundering.

It is possible, on the other hand, that some useful work is going on during this time. For example, the student may be pursuing alternative paths, working out details, or entering future subgoals. Nevertheless, the student ends up needing to perform the suggested action and this statistic clearly indicates a big discrepancy between the way we expected the student-tutor interaction to go and the way it actually did.

One straightforward remedy to this problem is to do what is done in GPT and have the bottom-out hint provide the next step for the student. Another alternative is a simple "interface tutor" that will kick-in after bottom-out hints are given and guide the student in performing the suggested action.

The Whole-statement Encoding Problem. One of the difficulties encountered in thinking about how the ANGLE interface should reify the problem solving model involved the process of *whole-statement encoding* in which the DC model converts whole-statements that appear in a problem given or goal into the corresponding schema. The question was whether and how this whole-statement encoding process might be reified in ANGLE interface actions.

Here again, we opted for flexibility in the interface. On the one hand, whole-statements that appeared in the givens or goal could be treated, for the purposes of making justifications, just as if they were schemas. This is a short cut in which the whole-statement encoding process is left implicit. Alternatively, the student could enter the corresponding schema and link it to the whole-statement, and thus explicitly perform the whole-statement encoding process. The interface responded by putting the schema (whole-statement plus configuration instance) in the given or goal position that the whole-statement had occupied. An example

of the result is shown in Fig. 1.8 where the given on the left, which originally appeared simply as $\triangle BCD \cong \triangle YZW$, has been encoded as the corresponding schema.

Although the student was free to do it either way, in the case that the student needed help at this point, the tutor suggested that they explicitly perform the whole-statement encoding step. Two problems arose. While the interface actions to perform this step were the same as those necessary to select and justify any schema, the effect was different (i.e., the schema replaces the whole-statement to which it was linked). Understanding this interface operation was another thing for students to learn that may have distracted them from learning geometry.

A second problem was the unanticipated awkwardness that sometimes resulted from the way the schema hint templates got applied to this situation. Whereas the majority of these hints read just fine, one of them, the bottom-out hint for justifying a planning action, turned out to be particularly awkward. Since schemas were referred to in the hint messages using the same label as the label for the whole-statement, the justification bottom-out hint message, which has the generic form "Justify ⟨ the desired schema ⟩ using statements: ⟨ the necessary statements ⟩," ended up as "Justify ⟨ whole-statement ⟩ using statements: ⟨ whole-statements ⟩." For example, if the whole-statement $\triangle ABD \cong \triangle CBD$ appeared as a given and the corresponding TRIANGLE-CONGRUENCE-SHARED-SIDE schema (labeled $\triangle ABD \cong \triangle CBD$) had been entered, then if the student happened to need a bottom-out hint, it would read "Justify $\triangle ABD \cong \triangle CBD$ using statements: $\triangle ABD \cong \triangle CBD$."

This proved a bit confusing to students. Many of the cases where a student never performed the step suggested by a bottom-out hint were cases where the student got this hint, didn't know how to implement it, and was eventually able to perform a different step (i.e., one that involved skipping the whole-statement encoding step) with no help from the tutor.

Clearly one way to remedy the problems experienced with the whole-statement encoding step would be to improve the associated hint message. Another possibility would be to remove the flexibility, the option to skip the whole-statement encoding step, and always require the student to perform it. The idea of the whole-statement encoding step was to encourage students to begin problem solving by recognizing any statements in the givens or goal that refer to schemas (i.e., any whole-statements). However, from watching students, this recognition of given or goal statements in the diagram does not appear to be a great difficulty, especially in comparison to the difficulties that arose from allowing and sometimes encouraging the explicit whole-statement encoding step.

Thus, perhaps the best remedy might be to simply eliminate the need to explicitly perform whole-statement encoding. In addition, whenever a problem given or goal is a whole-statement, the tutor should display this statement as a schema complete with the diagram configuration.

Planning Before Execution Not Enforced. Following the cognitive model, ANGLE's feedback scheme always suggests planning moves first and only suggests execution moves once a complete plan has been found. However, consistent with the effort to make the system flexible, ANGLE allows students to integrate planning and execution. In fact, students rarely completed planning before beginning execution. One measure of this is the percentage of inferences that occurred after execution began, but before planning was finished. Thus, for example, if all the planning is done first this percentage should be 0. Instead, 47% of students' inferences were in this mixed stage. Students' tendency to mix planning and execution was not quite significantly correlated with posttest performance ($p = .06$). However, the pattern is that the students who began execution early scored better on the post-test. This is probably a reflection of students' prior familiarity with the execution space and the good students' better facility with it.

These data indicate that to accurately test the efficacy of teaching the DC model, it seems important to sacrifice some flexibility in the interface and enforce the planning first approach of the cognitive model. Only after completing a plan would students to allowed to begin proof execution.

Study Conclusions: Lessons Learned. Getting the student-tutor interaction right requires careful and lengthy user study that goes beyond informally watching a few students work with the system for a couple of hours. While we knew prior to the study that ANGLE's tutoring schemes were not perfect, we did not anticipate certain problems like the trouble students had understanding and implementing the bottom-out hints. The lesson is that such problems are necessarily a part of the development process and that tuning the interface and tutoring components is equally as important as following through on the implications of a cognitive model.

A second lesson from this initial evaluation is that the curriculum we used was too small (only 8 geometry rules and/or 4 schemas). This is important because a key difference between the problem solving method ANGLE teaches and the one GPT teaches is that ANGLE's method is more effective in the large search space of geometry rules (the execution space). By limiting the curriculum, the size of the search space is effectively reduced and if it is small enough, the search benefits of ANGLE's method become negligible. With a larger curriculum, then, we are more likely to see a difference between ANGLE and GPT.

GUIDELINES FOR MODEL-BASED TUTOR DESIGN

In this section, we recast the research agenda carried out in the development of ANGLE to this point, generalizing the key steps and presenting them in the form of guidelines for tutor design. Certainly there are other theoretically motivated

routes to successful tutor design, not to mention getting there by good intuitions or serendipity. These guidelines are provided as one possible route that may (1) be directly applied in some domains or (2) be used as a departure point for developing related approaches in other domains.

The approach can characterized in five steps:

1. Identify the *execution space.*
2. Look for *implicit planning* in verbal reports.
3. Model this implicit planning.
4. Use the model to drive tutor design.
5. *Test* the tutor implementation.

We discuss the significance of each step, illustrate it in the context of this project, and suggest how it might be generalized to other domains.

Identify the Execution Space

This step sets the stage for step 2 where one looks for underlying problem solving processes that are effectively hidden in current instruction. First, we need to know what aspects of the problem solving process are revealed, at least implicitly, by current instruction. This is the task of identifying the execution space. The execution space for a domain is the problem space most directly induced from the way problem solution steps are conventionally written down.[7] In other words, the operators of this space correspond one-to-one with the written problem steps.

As discussed earlier (see p. 17), the execution space operators for geometry are the various definitions, postulates, and theorems that appear as the "reasons" in the steps of the conventional two-column proof format. The execution space operators for algebra equation solving are the various rules (e.g., You can add the same number to both sides) for manipulating equations. In physics problem solving (e.g., the kind analyzed by Larkin, McDermott, Simon, and Simon (1980), the execution space operators might be the relevant physics formulas.

Another potential guide to the operators of the execution space is to look at the units of knowledge that are provided to students in their textbooks or lectures. Quite often, these units of knowledge correspond with the written problem steps. For example, the traditional geometry curriculum is organized around presenting and illustrating the very same rules that appear as reasons in two-column proof solutions. A similar situation is apparent in algebra and physics.

[7]Newell and Simon (1972, p. 144) referred to a "basic problem space" and gave examples of one in a number of domains. It is evident from their examples that what they meant by a basic problem space has some similarity to what we mean by an execution space. However, they did not provide the kind of operational definition we provide here for the execution space.

TABLE 1.5
An Example Solution in the Domain of Algebra Equation Solving

3x - 13	=	2(x - 3)	
3x - 13	=	2x - 6	Distribute
3x - 13 - 2x	=	- 6	x's to left side
3x - 2x	=	- 6 + 13	Num's to right side
	x = 7		

A straightforward way to model problem solving in these domains is as a heuristic search in the execution space—the trick is to find appropriate operator selection heuristics. From the perspective of a student, to the extent that the execution space provides a good characterization of skilled problem solving, his or her learning job is made easier. By definition, execution operators can be induced fairly directly from the steps of worked out examples and may be supported by verbal descriptions in textbooks and lectures. For example, consider algebra equation solving. An example worked out solution is shown in Table 1.5.

In this domain, the execution operators can be fairly directly induced from the steps in worked out examples like that in Table 1.5. This claim is supported by the fact that an early machine learning program did exactly that (Neves, 1981). In addition, the general difference-reduction heuristic turns out to be an effective means of selecting operators. Because this domain independent weak-method works in this domain, extra domain specific heuristics do not need to be learned.

While heuristic search in the execution space is a straightforward candidate for modeling problem solving in a domain, it may not be the problem space that skilled problem solvers typically use. The next step is to see if it is or not.

Look for Implicit Planning

The purpose of this step is to identify the nature of skilled problem solving in the domain and in particular, to see whether or not it deviates from heuristic search in the execution space. To do so, one can collect concurrent verbal reports (Ericsson & Simon, 1984) of skilled subjects solving problems in the domain. As Ericsson and Simon point out, subjects should not explain what they are doing, but merely report what they are thinking. To the extent that heuristic search in the execution space provides a good model, subjects' successive verbalizations should correspond with successive states in the execution space.

However, subjects' verbalizations could deviate from the execution space in a number of ways:

1. Multiple execution steps might be *aggregated* into a single verbalization.
2. Successive verbalizations may *skip* steps in the execution space.
3. Verbalizations may not specify an execution state in full detail, but rather

Regularities in such deviations suggest "thinking steps" that are not represented in the execution space. From the perspective of the student, these thinking steps are an implicit part of the planning process which, in contrast to the execution operators, cannot be directly induced from worked out examples. In other words, when there is implicit planning in the thinking of skilled problem solvers, there may be aspects of a successful problem solving method which are hidden in the traditional curriculum.

In the second Section we showed that skilled geometry problem solvers skip steps in execution space (item (2) above) while developing an initial proof plan. The knowledge that allows them to do so is hidden in the geometry curriculum. While it is possible to induce *some* execution operators from successive steps of worked out geometry proofs in a similar fashion to Neves' (1981) program for Algebra, this induction is not possible in general (see Koedinger, 1991 for an example). Furthermore, it is not possible to apply a general weak-method like difference-reduction or means-ends analysis to effectively perform operator selection in the execution space of geometry. There is just not much information in the syntactic differences between a goal state in geometry, like $\overline{AB} \perp \overline{DC}$, and a given state, like $\overline{AC} \cong \overline{BC}$, that could help in operator selection. As the step-skipping in the verbal reports reveals, successful search in geometry involves operators other than those in the execution space.

One may find no evidence for implicit planning in a domain, in other words, skilled subjects may work in the execution space. This seems likely, for example, in the domain of algebra equation solving. In such a case, building a tutor for this domain may be inappropriate. Conventional instruction may be adequate. In other words, finding no implicit planning would suggest that the conventional written display of problem solving steps corresponds fairly well with the thought steps necessary for successful problem solving in that domain. In this case, well-motivated students may not have much trouble in inducing the necessary operators. An ITS based on a model of skilled problem solving will not be much different from conventional instruction and thus, will be unlikely to help much. In fact, an ITS for algebra equation solving developed in our lab has been compared with a conventional classroom and although students learn with this tutor, they do not learn significantly better or faster than students in a normal classroom (Anderson, et al., 1990). A field test of the RAND Algebra Tutor had similar results: Students using the tutor learn, but no better than students in the control classes (Stasz, Ormseth, McArthur, & Robyn, 1989).

Model this Implicit Planning

If evidence for implicit planning is found, the question becomes what is the knowledge that is responsible for the non-execution space inferences? Generalizations for how to come up with a model of implicit planning are difficult to come by. The diagram configuration schemas described on p. 21 evolved from an

(Newell & Simon, 1972; Sacerdoti, 1974). In particular, the first attempt at a model was based on applying the idea of equivalence classes as a way to collapse nearby problem states into one. Diagram configuration schemas evolved through a generalization of the equivalence class idea and the incorporation of perceptual categories as the organizational element.

In the case of aggregation or step-skipping verbalizations (types (1) and (2) above), it is possible that the implicit planning knowledge is the result of composing execution operators. On p. 23, we summarized the arguments against a macro-operator interpretation of the genesis of diagram configuration schemas. These arguments (see Koedinger & Anderson, 1990) suggest some general criteria for distinguishing macro-operators derived from execution operators from operators that merely bear a macro-operator relationship with the execution operators.

In the case of verbalization type (3), the abstract problem space ideas are likely to be relevant. For example, Newell and Simon's (1972, p. 152) augmented problem space for cryptarithmetic provides a model of such verbalizations (e.g., abstract features of states like "the number must be even").

Use the Model to Drive Tutor Design

Once an accurate model of implicit planning has been developed the challenge is to find a way to communicate this model to students. This issue was discussed on p. 24. As people seem to learn best by doing, directly communicating a method to them doesn't usually help much. By design a model of implicit planning contains problem-solving processes that are not reflected in the notation of the current curriculum. Thus, it is necessary to invent new notations which reify the previously hidden representations and processes. These notations can be the basis for interface design. For example, diagram configuration schemas became the basis for the icons in ANGLE's concept menu and for the representation of concept instances in the proof graph. In addition, the computer medium affords the possibility of inventing *novel actions*, which can more directly reify the processes of the model. For example, the interface actions for selecting a schema instance from the problem diagram reify the diagram parsing process (see Fig. 1.6).

The goal of the design of tutoring strategies and messages should be to support and articulate the problem solving method. In ANGLE, the tutoring strategy of focussing on concepts as the grain size for next step advice is intended to support the approach of the DC method of searching in the space of schemas (see p. 30 and Figs. 1.8 and 1.9). The use of terms like "concept," "part-statement," and "ways-to-prove" is intended to articulate crucial aspects of this novel notation.

Test the Tutor Implementation

Although cognitive models can guide the design of the interface and tutoring

menting these guidelines. There is little theory available to guide these implementation decisions, though some guidelines have been proposed (e.g., see Chabay & Sherwood, 1992). Thus, it is crucial to test and tune an ITS implementation. What seems right intuitively may not be effective. In addition, because of the great complexity of such systems, all possible interactions cannot be anticipated. Long periods of use by multiple students is the only way to "shakedown" possible unanticipated interactions.

Two examples from the formative evaluation of ANGLE are most notable. First, there was the problem with bottom-out hints and more generally, with the conflict between interface flexibility and user confusion. Intuitively it had seemed that learning would be facilitated by both (1) always requiring students to enter proof steps even in the case that the tutor has told them what to do, and (2) always allowing students the flexibility to ignore hints, for example, if they had something else they wanted to do. However, it turned out that students did not use this flexibility effectively and often engaged in long periods of floundering as a result of ignoring or not fully understanding a bottom-out hint.

Second, there was the problem with whole-statement encoding. Here again, there were well-motivated goals to (1) maximize the student's involvement, that is, by not doing the whole-statement encoding step for them, and (2) allow flexibility, that is, allowing students to either explicitly perform or skip this step. However, the result was, on the one hand, added complexity to the interface with little or no added instructional impact and, on the other hand, an interaction with the generic hinting scheme that resulted in a confusing tutor message. It is unanticipated problems such as these that make experimental testing a very necessary part of ITS design and instructional design in general.

CONCLUSION

We have presented a 5-step guide to employing cognitive science methodologies in the design of novel instruction. The guidelines are aimed at domains where learning from examples/cases and from problem solving is apt to be more effective than an emphasis on reading texts and listening to lectures. Thus the steps focus on the nature of examples and problem solving:

- Step (1) "Identify the execution space" provides a method for analyzing worked-out solution examples in current instruction to see what kind of problem solving approach (including representations and strategies) is implied by these examples.
- Step (2) "Look for implicit planning" provides a method for deciding whether this implied problem solving approach is a reasonable characterization of the thought process of skilled problem solvers.
- Step (3) "Model this implicit planning" provides a method for developing a

the characterization implied by current instruction is wanting. As such, these guidelines are focused on domains that students find particularly difficult (for good reason) and for which instructional innovations can show the greatest payoff.

- Step (4) "Use the model to drive tutor design" provides a method for translating the cognitive model developed in step 3 into novel interface representations and actions as well as tutoring messages and strategies that can help make previously hidden thought processes more visible to students.
- Step (5) "Test the tutor implementation" is a recognition that no design guidelines are 100 percent prescriptive—there is no guarantee, even if the guidelines are followed exactly, that a system will be effective.

While much can be learned by testing instructional systems in the lab (not only about the effectiveness of the system but also about the nature of cognition), there is also much to learn from testing these systems in the classroom. We are currently preparing for a field test of ANGLE in a Pittsburgh high school in the spring of 1992. Although this chapter focuses specifically on the design of computer-based learning tools, we would be amiss if we did not mention an equally important issue, that is, the instructional context in which computer-based learning tools are placed and, in particular, the role of the teacher. The impact of the classroom use of LOGO, for example, has been shown to depend critically on teachers providing an effective instructional context to support learning with LOGO (Klahr & Carver, 1988; Lehrer, Littlefield, & Wottreng, 1991; Lehrer, Randle, & Sancilio, 1989). A number of researchers and teachers have offered different perspectives on the role of the teacher, and the social context more generally, in the classroom success of the Geometry Proof Tutor discussed earlier (Epstein & Hillegeist, 1990; Shofield, Evans-Rhodes, & Huber, 1990; Wertheimer, 1990). We are paying to close attention to such issues in our preparations for the field test of ANGLE.

ACKNOWLEDGMENTS

This research was supported by Grant 86-17083 from the National Science Foundation and by a James S. McDonnell Foundation Postdoctoral Fellowship to the first author.

REFERENCES

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R., Boyle, C. F., Corbett, A., & Lewis, M. (1990). Cognitive modelling and intelligent tutoring. *Artificial Intelligence, 42*, 7–49.

Anderson, J. R., Boyle, C. F., Farrell, R., & Reiser, B. J. (1987). Cognitive principles in the design of computer tutors. In P. Morris (Ed.), *Modelling cognition*. New York: Wiley.

Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science, 228*, 456–468.

Anderson, J. R., Boyle, C. F., & Yost, G. (1985). The geometry tutor. In *Proceedings of the International Joint Conference on Artificial Intelligence-85*. Los Angeles: IJCAI.

Chabay, R. W., & Sherwood, B. A. (1992). A practical guide for the creation of educational software. In J. H. Larkin & R. W. Chabay (Eds.), *Computer-assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Collins, A., & Brown, J. S. (1990). The computer as a tool for learning through reflection. In H. Mandl & Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. New York: Springer.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Epstein, K., & Hilligeist, E. (1990). Intelligent instructional systems: Teachers and computer-based intelligent tutoring systems. *Educational Technology*, November, 13–19.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: The MIT Press.

Gelernter, H. (1963). Realization of a geometry theorem proving machine. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought*. New York: McGraw-Hill.

Johnson-Laird, P. N. (1983). *Mental models*. Cambridge, MA: Harvard University Press.

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology, 20*, 362–404.

Koedinger, K. R. (1991, August). On the design of novel notations and actions for thinking and learning. In *Proceedings of the International Conference on the Learning Sciences*. Charlottesville, VA: Association for the Advancement of Computing in Education.

Koedinger, K. R., & Anderson, J. R. (1989). Perceptual chunks in geometry problem solving: A challenge to theories of skill acquisition. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14*, 511–550.

Larkin, J., McDermott, J., Simon, D., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science, 208*, 1335–1342.

Lehrer, R., Littlefield, J., & Wottreng, B. (1991). *Seeding Mindstorms with LogoWriter: Using LOGO in the elementary classroom*. Fontana, WI: Interactive Education Technologies.

Lehrer, R., Randle, L., & Sancilio, L. (1989). Learning pre-proof geometry with LOGO. *Cognition and Instruction, 6*, 159–184.

Neves, D. M. (1981). Learning procedures from examples. Doctoral dissertation, Department of Psychology, Carnegie Mellon University.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5*, 115–136.

Schofield, J. W., Evans-Rhodes, D., & Huber, B. R. (1990). Artificial intelligence in the classroom: The impact of a computer-based tutor on teachers and students. *Social Science Computer Review, 8*(1), 24–41.

Stasz, C., Ormseth, T., McArthur, D., & Robyn, A. (1989, March). *An intelligent tutor for basic algebra: Perspectives on evaluation*. Paper presented at the annual American Educational Research Association meeting, San Francisco.

Wertheimer, R. (1990). The geometry proof tutor: An "intelligent" computer-based tutor in the