On the Design of Novel Notations and Actions to Facilitate Thinking and Learning

Kenneth R. Koedinger
Department of Psychology
Carnegie Mellon University
Pittsburgh, Pa 15213
EMAIL: koedinger@psy.cmu.edu

ABSTRACT

There are certain domains, for example, word problems or high school geometry, which even well-motivated students find particularly difficult. I claim that such domains are *enigmatic* to students because the problem solving notation used in current instruction does not provide a good reflection of the underlying problem solving processes necessary to be successful in these domains. I argue that some domains are not enigmatic in this sense, like algebra equation solving, and thus, are not good candidates for model-based notation design. However, domains that are enigmatic, like geometry theorem proving, can be made less so by (1) developing a cognitive model of skilled problem solving in the domain and (2) designing novel notations and actions that reify the underlying structures and processes of the model. In previous research, we found that skilled geometry problem solvers plan proofs much differently than the problem solving approach implied by textbook solution examples (and by previous cognitive models). A new cognitive model of this perceptually-based planning approach has been used to design novel geometry notations and actions that facilitate a way of thinking about geometry proofs that is more like the thinking of skilled problem solvers. These notations and actions have been implemented in a computer environment called ANGLE (**A** **N**ew **G**eometry **L**earning **E**nvironment).

Appears in *Proceedings of the International Conference on the Learning Sciences*. Charlottesville, VA: Association for the Advancement of Computing in Education. 1991.

ENIGMATIC DOMAINS

There are certain subjects, for example, high school word problems and geometry proofs, which even well-motivated students find particularly difficult. I call such domains *enigmatic domains*. A key claim about them is that:

> Students find enigmatic domains particularly mysterious because the problem solving notation used in current instruction does not provide a good reflection of the underlying problem solving processes students must acquire to be successful.

While cognitive science tools can be used to identify these underlying problem solving processes, achieving effective instruction is not as simple as describing these processes or strategies to students. People are not very good at learning from explicit procedural instruction (e.g., consider trying to learn geometry problem solving by reading an AI program that solves geometry problems). Instead, people learn more easily from example solutions or cases (Anderson, Farrell, and Sauers, 1984; Riesbeck and Schank, 1989; Ward and Sweller, 1990; Zhu and Simon, 1987). However, not all examples are created equal: The ease of learning from example solutions depends crucially on the similarity between the example's notation and the problem solving process the student is supposed to learn. By developing a better understanding of these problem solving processes, particularly in enigmatic domains, we may be able to design notations that take some of the mystery out of acquiring enigmatic skills.

To illustrate the distinction between enigmatic and non-enigmatic domains, I contrast two domains: algebra equation solving and geometry theorem proving. This distinction is important because if the current notation is already a good reflection of the underlying problem solving process (i.e., in a non-enigmatic domain), designing better instruction may prove difficult. This appears to be the case for algebra equation solving.

A Non-Enigmatic Domain: Algebra Equation Solving

Field tests of two different intelligent tutoring systems for algebra have shown that while students clearly learn from these tutors, they do not appear to learn more effectively than students in a normal classroom (Lewis, 1989; Stasz, Ormseth, McArthur, and Robyn, 1989). In addition, mathematics teachers report that algebra equation solving is not a particularly difficult skill for well-motivated students to acquire. It would be useful if we could characterize what it is about current instruction and/or the skill itself that makes learning algebra equation solving much more straightforward for students than learning some other skills like geometry theorem proving.

One way to characterize the task of learning mathematics problem solving is in terms of two subtasks: 1) learning the basic problem solving operators, *operator application* and 2) learning how to choose among operators, *operator selection*. In algebra, both these subtasks can be accomplished using general weak methods – methods that most high school students have (at least) tacit knowledge of. Table 1 shows an example algebra solution.

Table 1. An example solution in the domain of algebra equation solving.

1)	3x - 13	= 2(x - 3)	
2)	3x - 13	= 2x - 6	Distribute
3)	3x - 13 - 2x	= - 6	x's to left side
4)	3x - 2x	= - 6 + 13	Num's to right side
5)		x = 7	Simplify

Learning problem solving operators in this domain can be done in a straightforward way by focussing on pairs of successive steps and inducing the operation that gets from one step to the next. Looking at the transition between steps 2 and 3 and similarly between steps 3 and 4, students can induce that terms, like `2x` in step 2, can be moved from one side of an equation to another by switching the sign of the term. For example, in going from step 2 to 3, `2x` goes from the right hand side (RHS) in step 2 to `-2x` on the LHS in step 3, while in going from step 3 to 4, `-13` goes from the LHS to `+13` on the RHS.

Since many possible operators can apply to a particular step, students must learn to make good operator selections in addition to learning the operators themselves. Because of the nature of the algebra domain, students need to learn very little special-purpose operator selection knowledge. Instead, a the domain-general heuristic, *difference-reduction*, can be used to reliably select among operators. The difference-reduction heuristic is to choose operators that reduce (visible) differences between the current state and the goal state. (This heuristic is used, for example, as a key component of Newell and Simon's (1972) means-ends analysis method.) For example, from step 2 a student could notice that one of the differences between this state and the desired goal state (`x = ?`) is that there is an x-term on the RHS of step 2. This difference can be reduced by applying the "move-to-other-side" operator described above.

In algebra, the tasks of learning both operator application and operator selection are fairly straightforward because the notation of example solutions facilitates the use of general problem solving and learning methods. That these to general methods can work to learn algebra equation solving is evidenced by an early machine program that was successfully programmed to learn algebra in exactly this way (Neves, 1978).

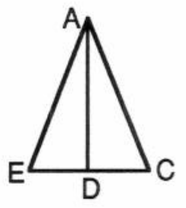
An Enigmatic Domain: Geometry Theorem Proving

In contrast, the situation in geometry is much different. First, learning operator application is not so straightforward as it is in algebra. See Table 2 which shows an example solution from a geometry textbook. It is not easy to induce operators from successive steps in solution partly because successive steps are not always related, for example, step 4 comes from step 1 not step 3. The numbers in the parentheses on the reasons side of the proof can help make this connection but, in fact, examples in traditional textbooks often do not use these numbers. Even if the student makes the proper connections between steps, the job of inducing an operator that gets from say step 1 to step 4 is not easy. In fact, if the student only considers the statement notation itself, $\angle AED \cong \angle ECA$ and $\overline{AE} \cong \overline{AC}$, this induction is impossible. One must also consider the objects referred to by these statements (the angles and segments in $\triangle EAC$) in order to have any chance of inducing the **Opposite-Sides** operator (in a triangle, the sides opposite congruent angles are congruent).

Table 2. An example solution in the domain of geometry theorem proving.

Given: **$\angle AED \cong \angle ECA$**
D midpoint $\overline{EC}$

Prove: **$\overline{DA}$ bisects $\angle CAE$**



PROOF:

Statements:	Reasons:
1. $\angle AED \cong \angle ECA$	**1. Given**
2. D midpoint $\overline{EC}$	**2. Given**
3. $\overline{DC} \cong \overline{ED}$	**3. Def. of midpoint (2)**
4. $\overline{AE} \cong \overline{AC}$	**4. Opposite sides (1)**
5. $\overline{AD} \cong \overline{AD}$	**5. Reflexive (diagram)**
6. $\triangle EAD \cong \triangle CAD$	**6. Side-Side-Side (3, 4, 5)**
7. $\angle CAD \cong \angle DAE$	**7. Corresponding-parts (6)**
8. $\overline{DA}$ bisects $\angle CAE$	**8. Def. of bisector (7)**

The task of learning operator selection is even more problematic. Domain-general heuristics like difference reduction are not effective in this notation. There is no natural way to characterize the differences, for example, between the givens **$\angle AED \cong \angle ECA$** and **D midpoint $\overline{EC}$** and the goal **$\overline{DA}$ bisects $\angle CAE$** in a way that could aid in the selection of operators to reduce these differences.

Previous models of geometry problem solving (e.g., Gelernter, 1963; Anderson, Boyle, and Yost, 1985) have operator application knowledge that corresponds with the steps in two-column proofs, that is, the definitions, postulates, and theorems of geometry. Koedinger and Anderson (1990) called this problem space the *execution space* since it is made up of operators that correspond with the steps that problem solvers conventionally write down (or "execute") in solving problems. (The problem space discussed above for algebra is the execution space for that domain.) As further evidence that domain general search heuristics are not effective in this problem space, these previous computer models were only successful through the use of many domain specific heuristics.

IDENTIFYING IMPLICIT PLANNING

A straightforward way to model problem solving in many domains is as a heuristic search in the execution space – the only 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, as it seems to in algebra, his or her learning job is made easier. 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.

Although heuristic search in the execution space is a natural candidate for modeling problem solving in a domain, it may not be the problem space representation that skilled problem solvers typically use in this domain. It may be possible to find a better problem space representation in which solutions can be planned. Plans developed in this alternative representation can be executed in the conventional one. This is exactly what we have found that skilled geometry problem solvers do (Koedinger & Anderson, 1990). They plan key proof steps using a perceptually-based problem space representation (called the *diagram configuration space*) and deal with the details of the execution space after a complete (abstract) proof plan has been discovered. This approach yields a dramatic reduction in the geometry search space from more than 100,000 states in the execution space on a particular proof problem to just 12 in the diagram configuration space. In addition, the diagram configuration model (DC) provides a more accurate account of the planning behavior we have observed in the verbal reports of skilled problem solvers. Further details on the DC model can be found in Koedinger and Anderson (1990).

Essentially, we found that skilled subjects were skipping steps in the process of developing a proof plan. The regularity of this step-skipping within and between subjects suggested that subjects had "thinking steps" (operators) 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 that are hidden in the traditional curriculum.

DESIGNING INTERFACE NOTATIONS AND ACTIONS TO REIFY IMPLICIT PLANNING

Building a cognitive model of implicit planning is the first step in this approach to model-based notation design proposed in this paper. Once an accurate model of implicit planning has been developed the challenge is to find a way to communicate this model to students. Since people seem to learn best by doing, directly communicating it to them doesn't usually help much. By design, this model contains problem solving process that are not reflected in the notation of the current curriculum. Thus, it may be possible to invent new notations which reify the previously hidden structures and processes. These notations can be the basis for interface design.

We have applied this approach in the design of ANGLE: **A N**ew **G**eometry **L**earning **E**nvironment. The cognitive model we developed, DC, played a key role in guiding the design process. Figure 1 shows the interface of ANGLE as it stands in the middle of the solution of the problem in Table 2.

An obvious design principle for a computer interface is that it should be easy to use and learn. A less obvious principle has to do with the role of interface as a subtle, yet ever-present, form of instruction. This implicit instruction comes both in the form of the *notations* used in the computer interface and also in the *actions* allowed by the interface. Notations should be created which mirror the important underlying representations of the problem solving model, while actions should be created which mirror the important underlying processes of the problem solving model. In this way, students can begin to internalize both the desired representations as they use the interface notations and the desired processes as they perform interface actions.

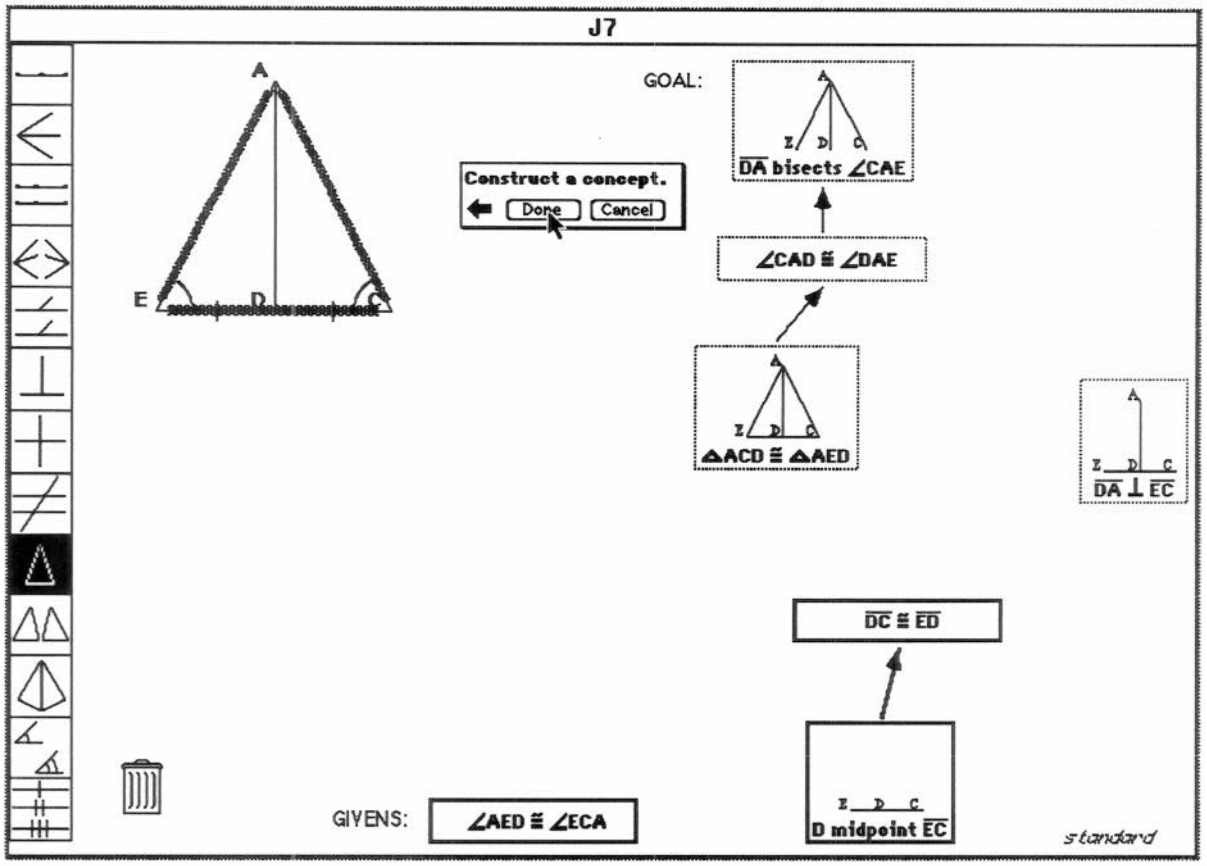


Figure 1. The ANGLE interface as it stands in the middle of the solution of the problem in Table 2. Students can work forward, backwards, or post "island" subgoals. An instance of the ISOSCELES-TRIANGLE schema is currently being instantiated in the problem diagram.

ANGLE Interface Notations

The ANGLE interface includes a number of examples of notations which reify representations in the cognitive model we developed. The key representations in the DC model are diagram configuration schemas that tie together perceptual knowledge of prototypical geometric categories with conceptual knowledge of properties and sufficiency conditions for these categories. The most prominent and important notational devices in ANGLE are the icons used for representing generic diagram configuration schemas and specific instances of these schemas. The generic schema categories are represented by icons in menu on the left in Figure 1. The icon for the ISOSCELES-TRIANGLE schema is currently highlighted (as are the segments in the diagram the make up an instance of this schema – discussed below). Student-selected schema instances are represented in the proof graph as a miniature picture of the relevant part of the diagram. For example, an instance the PERPENDICULAR-ADJACENT-ANGLES schema appears on the far right side of the screen and the miniature picture contains only the segments $\overline{DA}$ and $\overline{EC}$ that make up the perpendicular lines.

Following the Geometry Proof Tutor (Anderson, Boyle, and Yost, 1985), ANGLE incorporates a graph representation of proofs in contrast to the two-column format of traditional geometry instruction. This proof graph notation 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 posting 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).

In the proof graph proven statements appear in the bold boxes while unproven statements appear in the dotted boxes.

Following a conventional notation used on paper, ANGLE uses hash marks in the diagram to indicate segment and angle congruence statements that have been given or proven. For example, see the markings in diagram that indicate the angles in the given statement $\angle AED \cong \angle EDA$ are congruent. (The other proven statement $\overline{DC} \cong \overline{ED}$ is also marked in the diagram, but the markings are somewhat occluded by the highlighting.) These markings reify the equivalence class nature of segment and angle congruence in contrast to the binary relationships of the formal notation. In other words, 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 which cannot be feasibly employed with paper and pencil.

ANGLE Interface Actions

Following the major processes in the cognitive model, ANGLE interface actions are broken down into (1) diagram parsing actions and (2) planning actions. The diagram parsing actions are those done in order to post schema statements. 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 shows a student instantiating the ISOSCELES-TRIANGLE schema – note that only the segments that make up this instance are selected, $\overline{AD}$ is not. This particular way of constructing a statement, as opposed to the way a student constructs one in the Geometry Proof Tutor or on paper, is meant to reinforce the relationship between the schema instance and the problem diagram in which it is embedded.

Planning actions are those done in order to justify schemas and their properties. 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 $\overline{AD} \cong \overline{AD}$ can be omitted when the student goes to prove $\triangle ACD \cong \triangle AED$ in Figure 1, and b) the rules or "reasons" that usually appear in the right column of a two-column proof.

Any tutor interface (or notational scheme for that matter) is implicitly taking an instructional stance about what things are hard 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 hard and important, (2) learning the details of proof execution is less important and perhaps less hard than learning proof planning, and (3) learning how to parse geometry diagrams into particular chunks (DC-schemas) is hard but it is important for successful search in a vast problem space. The first point is shared by the Geometry Proof Tutor's interface, but not by the two-column notation used on paper. The last two points are special to ANGLE's interface.

REFERENCES

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

Anderson, J. R., Farrell, R. and Sauers, R. (1984). Learning to program in LISP, *Cognitive Science*, 8: 87-120.

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

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

Lewis, M. W. (1989). Developing and evaluating the CMU algebra tutor: Tension between theoretically and pragmatically driven design. Paper presented at the annual meeting of the American Educational Research Association, San Francisco.

Neves, D. M. (1978). A computer program that learns algebraic procedures. *Proceedings of the 2nd Conference on Computational Studies of Intelligence*, Toronto.

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

Riesbeck, C. and Schank, R. C. (1989). *Inside case-based reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Ward, M. and Sweller, J. (1990). Structuring effective worked examples. *Cognition and Instruction* 7(1): 1-39.

Zhu, X. and Simon, H. A. (1987). Learning mathematics from examples and by doing. *Cognition and Instruction* 4: 137-166.