International Journal of Computational Intelligence Systems, Vol.3, No. 1 (April, 2010), 21-27

Geometry Tutoring Supported by an Intelligent Drawing Interface and Automatic Problem Solving

Hyung Joon Kook
Department of Computer Engineering, Sejong University
Seoul, Korea
E-mail: kook@sejong.ac.kr

Received: 15-05-2009
Accepted: 01-12-2009

Abstract

In a scientific domain, learning comprises studying a finite set of principles of the domain and applying them to solve a wide variety of problems. Therefore an intelligent tutoring system in a scientific domain is required to possess an adequate methodology to deal with this principle. We suggest a tutoring architecture for geometry where the domain principles are automatically converted to *inference operators* for use by the domain-independent, inferential portion of the tutoring system. An important part of this architecture is an intelligent drawing interface that facilitates automatic conversion of the figures in geometry into an internal form that is suitable for problem-solving tutoring. During student problem solving, the system monitors the student's steps, tracks a step that has multiple inferences, and gives hints and explanations. We discuss the advantages of our approach in enhancing the performance and interactivity of the tutoring system.

Keywords: geometry tutoring, problem solving, inference operators, knowledge compilation, multiple inference, intelligent drawing interface.

1. Introduction

The task of building an intelligent tutoring system has been considered hard, since it requires years of human efforts to codify diverse knowledge involved in an educational domain, e.g., basic concepts, principles, and problem-solving strategies for a wide range of selected training problems. This is especially true in scientific domains. Learning in science is a process made up of studying a finite set of general *principles* and then developing the skills to apply them in an infinite variety of situations. A computer tutor in science should, therefore, possess an adequate strategy to represent and to use the *principles* in tutoring.[1, 2] Although such a strategy is expected to be commonly applicable to various scientific domains, a technique developed for tutoring in a domain cannot always be applied directly to other domains because each scientific domain is taught and learned in a context radically different from any other. For example, learning in mechanics involves reasoning about physical objects, motions, forces, etc., while the main contexts of plane geometry are figures, angles, etc.

We have set as our long-term research goal the development of a modular science tutoring architecture, in which the common, sharable strategies of science tutoring are built in and modifiable independently of the domain-specific knowledge supplied by the domain authors. We believe that authoring of domain-specific knowledge must be facilitated in a human-friendly interface that is designed to minimize the burden on the human author who supplies the knowledge (e.g.,

concepts and principles) of the domain. More importantly, the requirement for the tutoring architecture suggests that the system should provide a mechanism for compiling the knowledge of the domain into a format suitable for tutoring during the reasoning part of the teaching process.[3, 4]

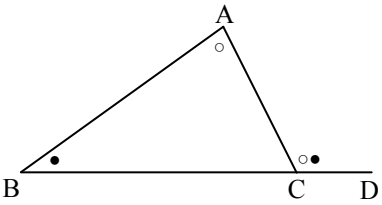
We have designed one such compilation mechanism and built on it a tutoring system for geometry problem solving called CyberTutor. Although geometry has been a popular domain for numerous intelligent tutoring systems research,[5, 6, 7] few of the system have yet reached the level of building a modular architecture like the one we are developing. A major obstacle to such modularization seems to be that, in geometry the domain knowledge is taught and learned in a unique context, i.e., the *figure*, an application that is uncommon in other domains. This paper reports on the design and the implementation of the tutoring system with a focus on the knowledge compilation scheme with regard to figures for obtaining the inference operators and the benefits of using them in tutoring problem solving.

In the following sections, we will show how the geometry principles exemplified by figures can be represented in the computational framework, and compiled into a set of inference operators to use in the inferential part of the tutoring system. Next, we will present the working mechanism of the inferential components of the tutoring system. That aspect will be followed by a brief demonstration of a sample tutoring session by CyberTutor. Finally, conclusions will be presented with discussions on potential areas of extensions.

2. The Geometry Models

Unlike other domains, geometry principles are usually presented with figures to graphically describe the contexts of the principles. A geometry principle is usually stated as a logical proposition about a figure representing the context in which the principle is applicable. Such a figure contains a set of geometrically-meaningful information, which may be divided into its components, quantities and relations. The *components* are the objects, such as points, lines, and figures like triangles and circles. The *quantities* are the quantitative attributes associated with the components, such as lengths, angles, perimeters, and areas. The *relations* are the configurations among the components, such as '*Point A is on Circle C*' and '*Direction of Line AB is shifted to the right from that of line BC*'. The collection of the components, the quantities and the relations may be referred to as *contexts*, since they serve as the contextual backgrounds for the higher-level *propositions* of the domain.

We have employed a computational framework called *model*, as a natural way to represent a geometry principle. Fig. 1a shows an example principle, followed by the corresponding model in Fig. 1b. Our notion of models corresponds to the notion of the models in the ACT-R theory[5], in that a model is a declarative data structure with slots to hold the contexts as well as the proposition.



"Exterior angle of a triangle is equal to the sum of opposite inner angles"

Fig. 1a. A geometry principle.

```
(MODEL M27
   (PRINCIPLE "Exterior angle of a triangle is equal to the
       sum of opposite inner angles")
   (COMPONENTS
       (POINTS (A) (B) (C) (D))
       (LINES (AB (ENDS A B)) (BC (ENDS B C))
         (AC (ENDS A C)) (CD (ENDS C D))
         (BD (ENDS B D))))
   (QUANTITIES
       (ANGLES (ABC (SIDES AB BC))
           (BAC (SIDES AB AC)) (ACB (SIDES AC BC))
           (ACD (SIDES AC CD))))
   (RELATIONS
        (ON (POINT C) BD)
       (SHIFT (DIRECTION AB AC)
           (DIRECTION BC AB) (DIRECTION AC BC)
           (DIRECTION CD AC)))
   (PROPOSITION (ACD = BAC + ABC))
)
```

Fig. 1b. A model corresponding to the principle in Fig. 1a.

3. The Drawing Interface

It is important that an authoring system facilitate an automated environment for generating *models*, since encoding a large number of these models manually would be too tedious and time-consuming. The drawing interface we are building enables an author to supply geometry principles by drawing figures and entering propositions in a user-friendly way; i.e., the contexts of the models are specified automatically, as the author draws the figures using a series of mouse actions.

The drawing interface provides the usual facility for entering figures; i.e., a drawing palette and mouse-selectable icons of basic *components* (e.g., line, circle, triangle, etc.). When the user of the drawing interface enters the components, implicit information about the figure is also derived, and all of these are transformed into an internal data structure, namely the *specifications* of the figure. The derivation of implicit information is performed by a search procedure that attempts to specify underlying *components*, *quantities* and *relations* formed by the components drawn explicitly. Among them, implicit components are such objects as line segments, arcs, chords, sectors or polygons that are inferred either by decomposing a composite component (e.g., a line intersecting a circle produces a chord, arcs, and segments of the line and the circle), or by grouping simple objects into a larger object (e.g., three intersecting line segments produces a triangle). Implicit *relations* are information about *point inclusions* and *line shifts*, while implicit *quantities* are those quantities associated with the components (e.g., length, angle, perimeter, circumference, area).

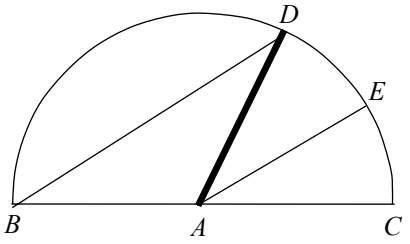


Fig. 2. An example figure.

An example is given in Fig. 2. Suppose line segment *AD* (shown thick) is entered into the existing figure. Then, from this particular configuration, the search mechanism derives a set of new components, i.e., triangle *ABD*, and sectors *ABD*, *ACD* and *ADE*. Also derived are new relations formed between line segment *AD* and existing line segments *AB*, *BD*, *AE* and *AC*; they are '*AB shifts from AD*' and '*AD shifts from BD*' among others. A number of associated quantities are also specified; they are the angles between the line segment just entered and the existing line segments, and the areas and the perimeters of the derived components. All these derived information are added to the internal specifications of the figure. Main benefit from this design of the drawing interface is the automatic specification of all implicit, but contextually legitimate information, without intervening an author's drawing activity. As a result of such automatic specification, this information is made accessible (or addressable) in tutoring.

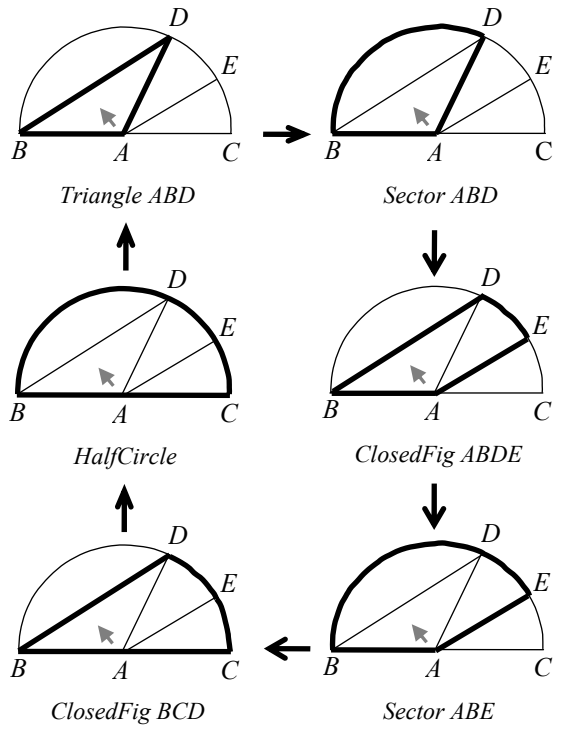


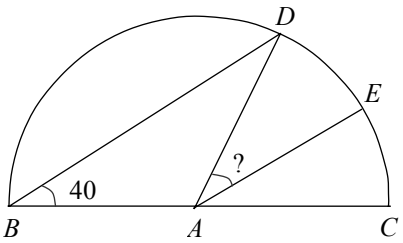
Fig. 3. Mouse toggles around the closed figures surrounding the mouse point.

To exploit the benefit further, we have implemented a graphic facility to enable cyclic selection of individual components among an overlapped cluster of components with a series of mouse toggles. Fig. 3

shows an example of such a cyclic selection in which, with the selection mode set to *closedfigure* (other possible modes are *line, arc, angle* and *area*), each mouse toggle selects and highlights each different closed figure surrounding the mouse point. The capability to access a specific part of a figure individually is very useful in tutoring. When the tutor wants to designate a component or a quantity, the tutor may *highlight* it to draw the student's attention. Such a feature is especially useful when a component having an irregular shape (thus, hard to name) has to be designated; e.g. the third-toggled closed figure in Fig. 3 which is surrounded by line segments *AB*, *AE* and *BD*, and arc *DE*.

4. The Inference Operators

For the finite set of general models to deal effectively with an infinite variety of problems, we employ the notion of *knowledge compilation*[8]. The models generated using the drawing interface are *compiled upon* the specific contexts of a problem to produce a set of *inference operators*. These operators are context-specific since they are obtained by converting the general contexts used in the model into the particular contexts of a problem (e.g., labels used in the figure). The compilation process applies only to those models whose contexts match the problem contexts. As a consequence, the range of the principles covered by the resultant operators is confined to only those principles that are *contextually* relevant to that specific problem.



Given BD//AE and ∠ABD = 40, find ∠DAE.

Fig. 4a. A sample problem.

The contexts having been matched already, these operators need not carry contextual information and thus can consist only of the propositions in the forms instantiated for the problem at hand. Fig. 4b shows some of the inference operators produced for the sample problem shown in Fig. 4a. We can see that the operators shown here solely consist of compilations from relevant models. They correspond to such principles as *"(P1) Alternate interior angles between parallel lines are equal", "(P2) If two sides of a triangle are equal, then the base angles are equal", "(P3) The sum of the angles in a triangle is 180", "(P4) Lengths of two radii of a circle are equal", "(P5) The sum of two supplementary angles is 180", "(P6) Exterior angle of a triangle is equal to the sum of opposite inner angles"* etc. Note that, some operators included in this operator set are apparently unusable ones (e.g., OPR2,4,5), since their left-hand side propositions do not hold. Still these operators are legitimate as far as the contexts are concerned and in fact are useful for capturing misconceptions in the inference steps that students might take.

(OPR1) BD//AE → ∠ADB = ∠DAE	*(P1)*
(OPR2) BD//AC → ∠ADB = ∠CAD	*(P1)*
(OPR3) AB = AD → ∠ABD = ∠ADB	*(P2)*
(OPR4) BD = AD → ∠ABD = ∠BAD	*(P2)*
(OPR5) AB = BD → ∠BAD = ∠ADB	*(P2)*
(OPR6) ∠ABD + ∠ADB + ∠BAD = 180	*(P3)*
(OPR7) AB = AD	*(P4)*
(OPR8) AD = AE	*(P4)*
(OPR9) AC = AD	*(P4)*
(OPR10) ∠BAD + ∠CAD = 180	*(P5)*
(OPR11) ∠CAD = ∠ADB + ∠ABD	*(P6)*

Fig. 4b. Some of the inference operators produced for the sample problem in Fig. 4a.

5. The Inference Engine

For tutoring of a problem, stored principles in geometry are combined with the specific problem to automatically produce a homogeneous set of logical consequences, the *inference operators*. These operators are subsequently used by the domain-independent, inferential part of the system for tutoring on that specific problem.

CyberTutor employs three types of inference strategies: *propositional*, *algebraic*, and *quantitative*. The *propositional inference* is the common backward inference about the propositions in the working memory: i.e., the inference operators and the set of propositions given in the problem. When a student enters a problem-solving step in the form of an assertion,

that student's step is taken as a hypothesis and set as the goal to be proved by the system. In the goal-driven search, the propositions in the working memory are then searched for matches. For those inference operators whose RHS's match the goal, the system tries to match their LHS propositions, possibly by sub-goaling them until the initial goal is proved or unproved. If proved, the student's step is taken as valid; otherwise, it is considered to be invalid.

The *algebraic inference* module handles the inference involving an algebraic manipulation. Real problem solving in geometry often involves algebra. Students may enter a step in the form of an algebraic expression which is not derivable from a direct application of the propositional inference described above. For instance, a student's step, "$x = 50$" is taken as a goal hypothesis by the algebraic inference module, and proved if the propositions in the working memory include, e.g., "$x + y = 180$" and "$y = 130$". The algebraic inference takes more time than the propositional inference does, since it involves solving simultaneous equations. In order to minimize performance degrade, this inference is used only as a complement to the propositional inference.

Finally, the *quantitative inference* module handles the situation when the goal hypothesis consists of an algebraic expression involving quantities and if both the propositional and the algebraic inferences fail to validate the goal hypothesis. In this case, the module extracts the quantities from the expression and sets each quantity as the sub-goal of the subsequent inference, and keeps working backwards until desired quantities are solved for. Previous research on geometry tutoring has concentrated mainly on purely-proving types of problems, while ignoring the algebraic aspects involved in problem solving. In fact, many geometry problems often demand algebraic manipulations in the course of the problem solving, and some problems ask directly to solve for the value of an unknown quantity. Using a separate built-in algebraic manipulation module would not suffice, since such a "black box" module would not allow the tutor to access the information about the algebraic inference performed within it. Kerber et al.[9] have advocated an integration of computer algebra into mechanized reasoning systems, in which the uniform framework allows to solve a large class of problems that are not automatically solvable by separate systems. Such a notion of integration is shared by us and even extended further in the design of the inference engine. One of the achievements of our research is the generalization of the tutoring scope to cover problems involving the algebraic and the quantitative inferences in a way that is transparent, and therefore, *accessible* for tutoring.

6. CyberTutor

A problem-solving tutoring session in CyberTutor (shown in Fig. 5) comprises a series of stepwise interactions between the student and the tutor. Student problem solving is monitored so that each step entered by the student is checked for validity. This validity check is performed by using the backward inference mechanism on the propositions in the working memory, and if necessary, the algebraic and quantitative inference mechanisms as well.

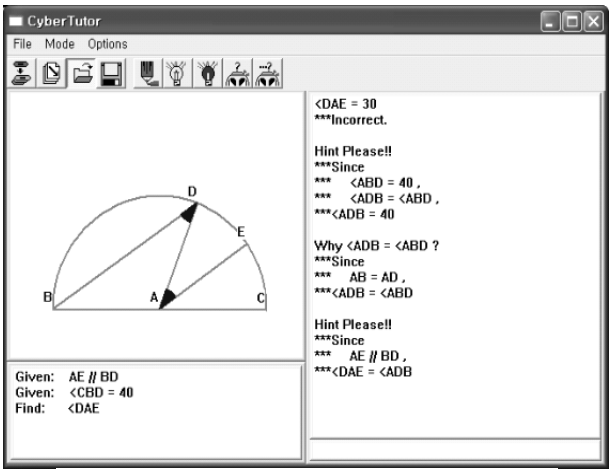


Fig. 5. The CyberTutor interface.

Initially, a problem selected by the student is loaded into the system. At load-time, the problem is compiled *upon* the system's library of geometry models to produce a set of inference operators. It is worth mentioning here that we are in the process of upgrading our drawing interface to facilitate the student entering a problem outside the library of problems, i.e., one supplied by the student himself. The strength of the proposed tutoring framework is that it is applicable not only to the selection of stored problem, but also to a wider range of problems, since the solution procedures for the problems are not stored along with the problems

in the library, but generated dynamically during the problem solving. This is important since students sometimes want to work with *their own* problems that are outside the system's library. Accommodation of this demands on the tutoring system's supports for an intelligent interface for students' authoring problems and automatic problem-solving, which we believe are naturally supported in our proposed tutoring framework.

Besides tracking student steps comprising single or multiple inferences, CyberTutor is also capable of providing hints and explanations at points where students cannot proceed further, as illustrated in Fig. 5. Highlighted hints are generated by using the system's current ability to solve the problem, i.e., the system is made to generate partial steps of problem solving. If the student asks for an explanation of the tutor's step, the system responds by back-chaining on the inference made for generating the step.

7. Conclusions

We have presented the design methodology of a tutoring system for problem solving in geometry. The main design issues involved were the knowledge compilation schemes through an intelligent drawing interface, the inferential mechanisms based on inference operators, and the pedagogical strategies on differing skill levels of students. To tutor a problem, the problem is first compiled with the domain principles stored in models to produce a set of inference operators. Then, these operators are passed to the inference part of the system, which consists of a set of domain-independent problem-solving strategies, namely propositional, algebraic, and quantitative inference strategies. They work together to assist the system's tutoring activities in a way accessible for pedagogy.

As for the problems-solutions database, a stored solution approach would be both unwise and useless. It is unwise because the human author has the burden of specifying all problem-solving steps for each problem, and remains useless because the system is still not prepared to deal adequately with the off-the-track steps students can take. There are also a line of automated geometry proof generating and problem solving systems[10, 11] that are capable of generating proofs and solutions to a large class of geometry theorems and problems. Although such a system may well be used as a legitimate supplementary to geometry learning or geometry-based systems, e.g., CAD,[12] it falls short of an *interactive tutor* for the same reason mentioned above. In the proposed design, the solution steps are not pre-stored in the library, but rather generated automatically at tutoring time in the form of the inference operators. The inference operators resemble the *production rules*,[5] as both are derived from declarative knowledge of the domain. But the operators are tactically more useful in several respects. First of all, the search space is greatly reduced since the models (i.e., the principles) that are irrelevant to the sample problem have been pruned out during compilation. In fact, a typical problem in a scientific domain, including geometry, can usually be solved using only a few principles. Additionally, the time-taking process of context matching is completed with compilation; hence the interactive performance of the system is enhanced still further.

In the present implementation of CyberTutor, a less-skilled student's step is usually tracked by a single or two inference steps of the system. On the other hand, a step involving a longer chain of inferences made by a more skilled student can also be tracked by applying multiple inference steps. Such a capability to track student steps, including a step made of multiple inferences, is important for building individual models of students. Most of the previous systems in this area of skills tutoring have confined the steps to a primitive length of inference, and by doing so, they failed to model the variety of individual skill levels. In this respect, we believe the proposed design has a solid potential for providing a personalized learning environment, if combined with an appropriate feedback facility.[13, 14, 15]

At the present, the drawing interface is specialized only for geometry. Learning in many other scientific domains also involves reasoning about figures. Mechanics, electric circuitry, and chemical reactions are several examples of such domains. We plan to investigate on the generalization of our approach to these domains. This generalization can be achieved by isolating the domain-independent features of the drawing interface from the geometry-specific features, then applying them to the specific features of other domains.

Overall, the proposed design principle is expected to be applicable to various scientific domains other than

just geometry, where the nature of problem solving is a stepwise inference that also uses domain principles as the search operators.

Acknowledgements

This work was supported by the faculty research fund of Sejong University in 2008.

References

1. A. Kohlhase and M. Kohlhase, Semantic Knowledge Management for Education, *Proc. IEEE,* **96**(6) (2008) pp. 970-989.
2. M. T. Mitchell, An Architecture of an Intelligent Tutoring System to Support Distance Learning, *Computing and Informatics* **26** (2007) 564-576.
3. I. Hatzilygeroudis and J. Prentzas, Knowledge Representation Requirements for Intelligent Tutoring Systems, in *Proc. Intelligent Tutoring Systems* (*Lecture Notes in Computer Science* **3220**), (2004), pp. 87-97.
4. M. P. Jarvis, G. Nuzzo-Jones and N. T. Heffernan, Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems, in *Proc. Intelligent Tutoring Systems* (*Lecture Notes in Computer Science* **3220**), (2004), pp. 541-543.
5. J. R. Anderson, F. Boyle, A. Corbett and M. Lewis, Cognitive Modeling and Intelligent Tutoring, *Artificial Intelligence* **42** (1990) 7-49.
6. T. McDougal and K. Hammond, Representing and Using Procedural Knowledge to Build Geometry Proofs, in *Proc. AAAI*, (1993), pp. 60-65.
7. A. Mitrovic, K. R. Koedinger and B. Martin, A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling, in *Proc. User Modeling*, (2003), pp. 313-322.
8. M. Cadoli and F. M. Donini, A Survey on Knowledge Compilation, *AI Communications* **10**(3-4) (1997) 137-150.
9. M. Kerber, M. Kohlhase and V. Sorge, Integrating Computer Algebra into Proof Planning, *J. Automated Reasoning* **21**(3) (1998) 327-355.
10. S. C. Chou, X. S. Gao and J. Z. Zhang, *Machine Proofs in Geometry: Automated Production of Readable Proofs for Geometry Theorems* (World Scientific, 1994).
11. F. Botana and T. Recio (eds.), *Automated Deduction in Geometry* (Springer, 2008).
12. S. Bhansali and T. J. Hoar, Automated Software Synthesis: An Application in Mechanical CAD, *IEEE Trans. on Software Engineering* **24**(10) (1998) 848-862.
13. A. Anolona, Advances in Intelligent Tutoring Systems: Problem-Solving Modes and Model of Hints, *Int. J. of Computers, Communications & Control* **2** (2007) 48-55.
14. K. R. Koedinger and J. R. Anderson, Reifying Implicit Planning in Geometry: Guidelines for Model-Based Intelligent Tutoring System Design, in *Computers as Cognitive Tools* (Erlbaum, 1993), pp. 15-45.
15. N. Matsuda and K. VanLehn, Modeling Hinting Strategies for Geometry Theorem Proving, in *Proc. User Modeling* (2003), pp. 373-377.