Paper submission:

A Reification of a Strategy for Geometry Theorem Proving

Noboru Matsuda
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260 USA
Tel: 412-624-2662
mazda@isp.pitt.edu

Kurt VanLehn[1]
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA 15260 USA
vanlehn@cs.pitt.edu

Keywords: Interface design, Cognitive approaches. (5000 words)

[1]This research was supported by NSF grant number 9720359 to CIRCLE: Center for Interdisciplinary Research on Constructive Learning Environments. http://www.pitt.edu/~circle

Abstract

For many years, designers of ITS have tried to use the power of a well-designed graphical user interface (GUI) to help students learn complex problem solving strategies. The basic idea is to display the reasoning in a graphical, manipulable form — to "reify" (make real) the reasoning. In this paper, we point out flaws in some common techniques for reification and suggest a new one. Our basic idea is to reify the search process rather than the structure of the ultimate solution. We show how this reification technique can be applied to a specific complex problem solving task domain, geometry theorem proving with construction, for which we are building an ITS.

1 Introduction

Learning a complex problem solving task domain not only involves learning a great deal of knowledge, it often involves learning a complex strategy for applying that knowledge. For instance, a medical student must not only learn many medical facts, but also complex diagnostic reasoning strategies [12]. A geometry student must learn not only many axioms and theorems, but also how to weave them into a proof [3]. A physics student must not only learn many laws of physics, but also how to solve a problem with them [17].

Forward chaining and backward chaining are the two main strategies used by AI problem solvers for task domains of this type, where domain knowledge can be represented as a large number of small pieces of knowledge, which we will call operators. Forward chaining consists of repeatedly finding an operator whose preconditions are satisfied and executing it. Backward chaining consists of finding an operator whose effects match a current goal, and establishing subgoals with the unsatisfied preconditions. When all the subgoals are satisfied, the operator is executed. However, if one or more of the subgoals cannot be achieved, the solver must back up and try a different operator for achieving the goal.

AI studies indicate that neither strategy is uniformly better than the other. Which one is best depends on the task domain and even the particular problem being solved.

Many studies of expert and novice human problem solvers have focused on whether they use forward chaining, backward chaining or a combination of the two strategies (for a review, see [16]). The evidence is seldom decisive. It seems that most solvers use a combination of the two, although the mixture may change with levels of expertise.

Most ITS for such task domains allow students to use any mixture of forward chaining and backward chaining that they wish. Scheines et al. [13] showed that when students using their propositional logic ITS are allowed to use both strategies, they learn faster than when they are restricted to using only one strategy.

All these studies suggest that an ITS for such task domains should allow students to use a mixture of backward and forward chaining. That is, they should use the following non-deterministic procedure, which is applied repeatedly until the problem is solved:

Do one of the following:

A. Find an operator whose preconditions are satisfied in the current state and whose effects are not. Execute it.

B. Do both:

   1. Remove a goal ***g*** from the set of goals G.
   2. Non-deterministically choose either (a) a fact in the current state that unifies with ***g***, or (b) an operator whose effects unify with ***g***. If a fact is found that unifies with ***g***, then ***g*** is satisfied. If an operator is found whose preconditions are satisfied, then execute it. If an operator is found that has unsatisfied preconditions, then place them in G as new goals. Otherwise, no facts or operators that unify with ***g*** can be found, so back up.

Forward chaining consists of always taking the A option. Backward chaining consists of always taking the B option. The ITS should allow the student to take either.

Step B-2 mentions "backing up." Backing up consists of returning to an earlier state where step B-2 was done, and doing it differently. That is, one chooses a different fact or goal to unify with the goal ***g***. Thus, backing up requires remembering states where choice B-2 was done.

In some task domains, the preconditions of operators sometimes need to be totally or partially ordered. When such preconditions are turned into goals during step B-2, the set G must preserve their ordering and integrate it with the ordering of other goals in G. In particular, if all operators have totally ordered preconditions, then G is always totally ordered. In fact, G can be conveniently represented as a last-in-first-out stack, because that convention automatically integrates the orderings of the new goals and the existing goals.

Although this strategy may be simple by AI standards, is still not easy for students to master. Many ITS design their GUI in order to explicate the strategy on the assumption that such reification (making real) will facilitate learning. There are 3 traditional designs:

Operators:

1. $D \rightarrow C$
2. $A \wedge B \rightarrow C$
3. $a1 \rightarrow A$
4. $a2 \rightarrow A$
5. $b \rightarrow B$

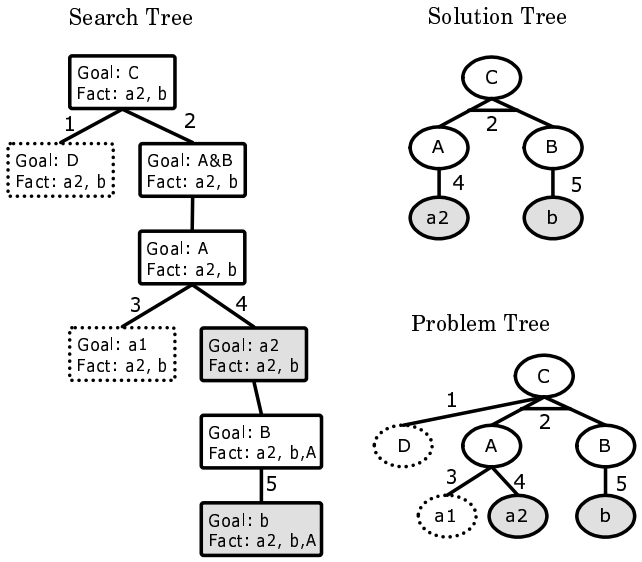


Figure 1: An example of the trees.

- A **solution tree** is an AND-tree that shows how the top level goals were derived (proved) via operator applications.
- A **search tree** is an OR-tree showing the states generated while searching for a solution.
- A **problem tree** is an AND/OR graph that combines the solution tree and the search tree. It shows not only the operator applications that led to a solution, but those that were tried in vain.

Figure 1 shows an example of those trees on a tiny problem with five operators. The nodes with dotted lines show dead-ends. The numbers on the links indicate the applied operators. An arc between two links shows an AND-node. As shown in the figure, only the search tree keeps track of problem-solving steps.

Most ITS utilize a problem tree or a solution tree. For example, Geometry Tutor [2], Angle (a successor of Geometry Tutor) [3], and an intelligent algebra tutor [6] offer students a partial problem tree to search a solution, and Algebra Tutor [10], Related

RATES TUTOR (a calculus tutor) [14], GRACE TUTOR (a COBOL tutor) [7], and GIL (a LISP tutor) [11] all offer students a solution tree to build up a solution.

Although these three tree displays have been used for years in AI textbooks, and thus are easy for AI researchers to understand, students without AI training often find them difficult to understand. More importantly, they may not aid students in learning the target strategy. Considering the steps of the strategy listed earlier, one sees that the student needs to attend to the following information:

- the operators
- the facts in the current state (e.g., to determine if an operator's preconditions are met).
- the goals in G that can be selected (i.e., all the goals if G is unordered; those goals that are first, if it is partially ordered; last goal pushed onto the stack if G is totally ordered).

In addition, when students are backing up, they need to attend to a list of states in which step B-2 was done.

Although this is the information that students should attend to, the three traditional displays do not emphasize it. None of the 3 trees show the operators. The current state facts can in principle be shown by the search tree, but often they are omitted because there are too many to display inside a node; the solution tree and problem tree do not display current state facts. Many goals are displayed by the trees, but none distinguish the selectable goals from goals that have already been satisfied or those that can't be selected now because they are not first in the partial order. The states available for backing up are shown in the search tree display, but they are not distinguished from states where step B-2 was not done and hence should not be backed up to. Hence, none of the traditional displays show all the information that students need, and what information they do provide is buried indistinguishably in irrelevant information of the same type. No wonder students find these displays confusing.

Moreover, the tree structure of the two most popular displays (solution tree and problem tree) is irrelevant to learning the strategy. It is intended to teach that the logical structure of the solution is hierarchical. While interesting and perhaps important, that is often not an instructional objective of the traditional versions of the courses that teach those cognitive skills, so one may legitimately wonder whether the ITS should teach it. That is, displaying the hierarchical structure of solutions may be both distracting and irrelevant.

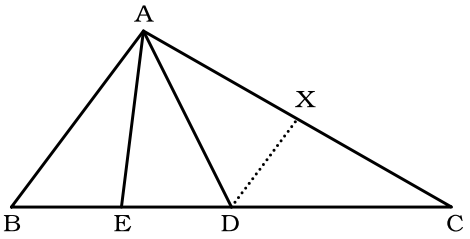
In this paper, we discuss a GUI design that follows directly from the requirement of learning the target strategy. We have incorporated that GUI into an ITS that we are building which will teach advanced geometry theorem proving. In the remaining sections, we first discuss the major issues involved in teaching advanced geometry theorem proving, followed by an overview of the previous work. In section 3, we introduce a GUI to reify a search tree generated by GRAMY, a theorem prover for auxiliary line problems, that uses the combined strategy mentioned earlier. Finally, in section 4, we discuss how the reification provides students with better scaffolding.

2 Geometry Theorem Proving

2.1 The domain

In geometry theorem proving, the operators to be applied are geometry axioms. A state is defined by a diagram, the known geometric relations in the diagram (the facts, for short), and the goal(s) to prove.

We especially focus on teaching construction of auxiliary lines and points, which is one of the most challenging and creative parts of geometry theorem proving. Figure 2 shows an example of a problem requiring construction. The original diagram includes neither a segment DX nor a point X. One must draw additional lines, such as DX, for example, to prove this problem. Theorem proving with construction is seldom taught at schools in some countries, e.g, the United States. However, it is commonly taught in many other countries such as Japan, Taiwan, France, etc.



Given: $BC = 2AB$

D midpoint of BC

E midpoint of BD

Goal: $\angle EAD = \angle DAC$

Figure 2: An example of a problem that needs a construction.

For theorem proving with construction, state transitions change not only the facts and the goals, but also the configuration; the configuration is changed by a construction.

One of the difficult tasks for geometry theorem proving is identification of possible substitutions for labels in an axiom against a *subset* of the labels in diagram (i.e., unification). For example, at the initial state shown in Figure 2, an application of the midpoint axiom (i.e., if a point m is the midpoint of a segment xy, then $xm = my$) to segment BC and point D requires a substitution $\{m/D,\ x/B,\ y/C\}$.

Forward chaining is an application of an axiom with all the preconditions satisfied to deduce new facts. For example, in Figure 2, the fact that "E is midpoint of BD" makes the midpoint axiom applicable forwards over the segment BD and point E. This deduces a new fact that $BE = ED$.

Backward chaining is to apply an axiom that has a conclusion matching the current goal to spawn a subgoal as a conjunction of the preconditions. For example, in Figure 2, backward chaining with the axiom of isosceles triangle (i.e., if $xy = xz$ in $\triangle xyz$, then $\angle xyz = \angle xzy$) establishes a subgoal of $BA = BD$ to prove that $\angle BAD = \angle BDA$.

For non-auxiliary line problems, a chain of the forward and the backward chaining eventually reaches to a state where the goal matches a known fact, i.e., a proof completes. However, such situation never occurs for auxiliary line problem. One must extend the diagram so that the forward and the backward chaining meet. This procedure is fulfilled by a partial unification; that is, a substitution for a *subset* of labels of an axiom against a *subset* of the labels of diagram. For example, to prove $\angle EAD = \angle DAC$ in Figure 2, a backward application of the axiom of triangle congruent (i.e., if $\triangle xyz \equiv \triangle stu$, then

$\angle xyz = \angle stu$) with a substitution $\{x/E,\ y/A,\ z/D,\ s/D,\ t/A,\ u/\text{nil}\}$ implies an auxiliary point X (matched with u) as shown in the figure.[1] The same construction can be carried out by forward chaining, for example, with the axiom of corresponding angles (i.e., if xy traverses two parallel segments ab and cd at point p and q, then $\angle xpb = \angle xqd$) to conclude $\angle ABD = \angle XDC$.

Note, however, that the above procedure only requires either the forward or the backward chainer. Since the branching factor of backward chaining on this procedure is much smaller than forward chaining, our theorem prover, GRAMY, utilizes backward chaining for construction.

2.2 ITS for Geometry Theorem Proving

Many ITS and CAI have been developed for geometry theorem proving so far, yet no one has developed an ITS to teach auxiliary line construction.

Geometry Tutor, an early geometry ITS developed by Anderson *et al.* [2], provided students with a GUI where the students could build up a partial problem tree. That is, they could build a proof by making a connection between the givens placed at the bottom of a screen and the goal placed at the top of the screen. The students used graphical icons representing the geometric theorems as the connectors. They either hung an icon from a current goal to establish new subgoals, or placed an icon to hook the known facts to it in order to derive a new fact. The former corresponds to a backward reasoning, and the latter corresponds to a forward reasoning. Furthermore, Geometry Tutor provided students with hints on theorem application in terms of the structure of a problem tree. However, since it utilized a problem tree, it had the potential flaws discussed in the previous section.

A recent descendant, Angle [3], extended the tutoring capabilities of Geometry Tutor by providing the students with a diagrammatic representation of the theorems, called

[1] In this case, the construction is further complicated. One must draw XD to be parallel to AB. This construction is done by diagrammatic information, that is, by placing $\triangle DAX$ that is congruent with $\triangle DAE$, one can "see" that XD might be parallel to AB. Since diagrammatic reasoning is beyond the scope of this article, we do not further discuss it.

diagrammatic configuration schemas, instead of the labeled icons used in GEOMETRY TUTOR. It actually provided students with better help in the sense of reifying the theorem applications; students could "see" how to apply theorems. However, it still used a partial problem-tree GUI.

Several geometry systems have been developed without explicitly using AND/OR trees. For example, CABRI GEOMETRY [15] provides students with a GUI that allows them to change the shapes of problem configurations while maintaining certain constraints given in the problem. By moving the apex of an isosceles triangle, for example, the students can observe that the apex moves along the perpendicular bisector of the base of the triangle. Although that kind of scaffolding works fairly well, since the system does not directly deal with the problem-solving strategies, it is a different kind of instructional tool than the one addressed in this paper.

3 A Reification of Strategy for Geometry Theorem Proving

3.1 GRAMY: A Geometry Theorem Prover for Auxiliary Line Problem

GRAMY is a geometry theorem prover with a diagrammatic reasoning system that we will use as the expert model in our tutoring system. It is powerful enough to solve most theorems requiring auxiliary lines in a corpus gathered from the literature.

A geometric theorem is codified by first-order logic together with its topological configurations [5]. We refer to such an enriched knowledge piece as a diagrammatic schema (or, DS for short). The topological configuration of a theorem is represented by semantic network. It is the topological configuration that supports the practical strength of GRAMY in following manner:

- It works as a filter to avoid establishing an infeasible subgoal. For example, to prove $\angle EAD = \angle DAC$ in Figure 2, a "reckless" prover might try $\triangle AED \equiv \triangle ADC$, because they are two triangles that have the angles in the original goal. GRAMY does not make this vain effort, because the corresponding DS (the axiom of triangle congruent)

requires two triangles that appear congruent. The "congruentness" is estimated by the coordinates of vertices.

- The topological configuration in the DS also constrains auxiliary-line construction. GRAMY uses a single heuristic of construction; if no DS is applicable to achieve a goal, then choose a DS that has the goal in its conclusion, and partially overlap a configuration of the DS against the problem figure. The auxiliary lines thus appear as a difference in the overlap. That is, a construction is carried out in a backward chaining step.

GRAMY uses iterative-deepening search with five successor functions; forward, backward, construction, transitive, and transitive construction. The last two successors need explanation. The transitive successor establishes a subgoal $A = X$ to prove $A = B$ given that $B = X$ is a known fact. The transitive construction successor establishes a subgoal $A = X \wedge X = B$ to prove $A = B$ given there exists a DS that achieves $A = X$ by drawing a construction. In sum, GRAMY has one forward chaining successor, and four backward chaining successors including two backward chainers for construction (with and without transitivity).

3.2 Reification of a Search Tree

In order to facilitate learning the target strategy, which involves search, we decided to utilize a familiar user interface for search — searching a hypertext. When one searches a hypertext, one clicks on links to move forward. At each step, one sees only the current page, not the ones that have been visited already. When one reaches a dead end, one clicks on the Back button to return to a previously visited page. This kind of interface is used on popular web browsers, so most students should be familiar with it.

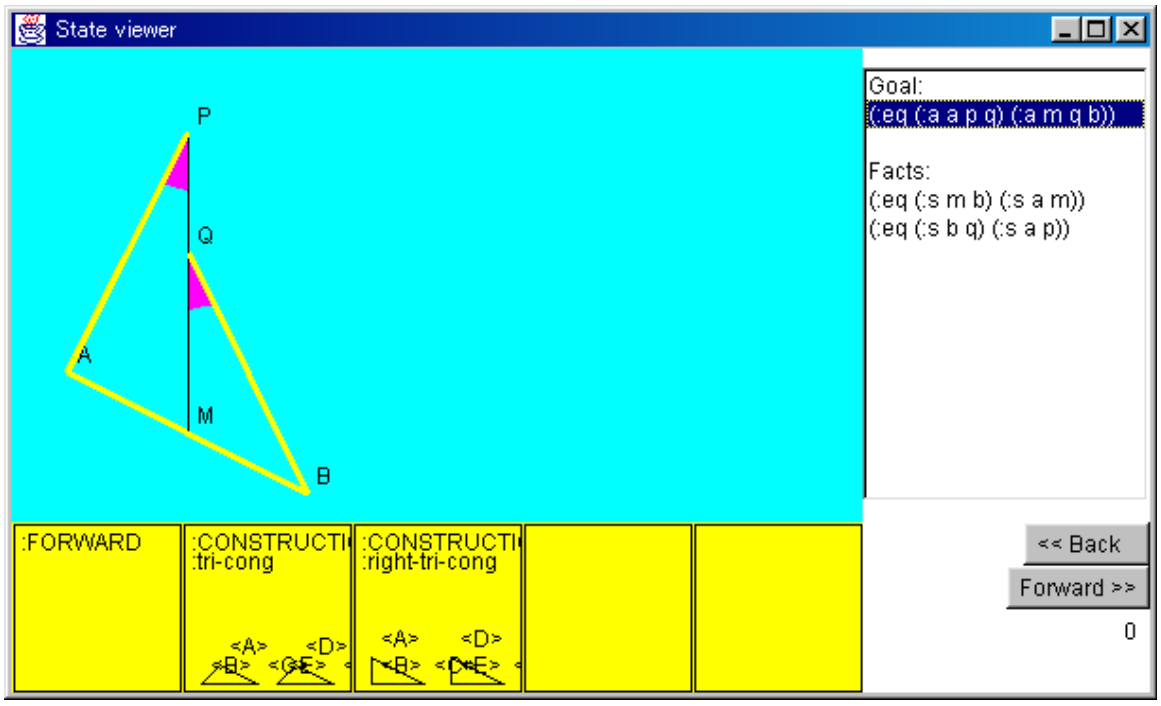
To adapt this for geometry theorem proving, we use "pages" that display just the information required by GRAMY's strategy — the configuration, the facts, the goals, and the relevant operators. Instead of clicking on a link to search forward, the student applies an

operator. This causes a new state to be displayed. The student can use a Back button to return to earlier states.

Figure 3 is an actual view of the GUI (called STATE VIEW) taken from a computer screen. It includes the diagram, the goal to prove, the facts deduced so far starting with the given premises, and the applicable geometry axioms. By "applicable," we mean either the axiom has a conclusion that matches against the goal (thus, can be used for a backward chaining), or it has all the premises satisfied (thus, it immediately derives a new factual statement). By displaying only applicable operators, we are providing scaffolding that should help students learn the strategy. The applicable DSs are lined up with the iconic representation illustrating configuration of corresponding axioms.

An irritating impediment to learning geometry is constantly having to figure out the referents in the diagram of symbols such as $\angle ABC$ and AB. To scaffold students, the GUI takes care of some of this reference-finding for the students. When the student clicks on a fact or a goal in the right window pane, the relevant parts of the diagram are highlighted. For instance, in Figure 3, the goal $\angle APQ = \angle MQB$ has been selected. (Our current implementations use a LISP-style notation instead of the standard notation. This will be changed soon.) Although it does not show in the monochrome Figure 3, multiple goals and facts can be selected at the same time. Each is displayed with a different color, and the color of the marks in the diagram matches the color of the highlighting on the text. The most recently selected item blinks.

One difficulty we faced in designing this interface is that a state can have a large number of operator applications. If auxiliary construction is not used, then the number of applicable operators averages about 4 (i.e., the branching factor in experiments with non-auxiliary line problems from the literature). However, when auxiliary line construction is allowed, there can be as many as 40 distinct operator applications, which is too many to show at once in STATE VIEW. Moreover, many of them involving applying the same DS to different parts of the diagram (i.e., the same axiom with different unification).



`(:eq` X Y`)` means $X = Y$,
`(:s` A B`)` means AB,
`(:a` A B C`)` means $\angle ABC$.

Figure 3: A representation of a proof step.

Our solution is to show only the icons for the distinct DS at the bottom of STATE VIEW. When an icon is clicked, all the constructions determined by the corresponding DS appear in a pop-up window. Figure 4 shows an example of the pop-up window showing possible constructions by the axiom of congruent triangles.

A pop-up window is also used for DSs applied by forward chaining. If there are forward chaining available in a state, an icon saying "`FORWARD`" appears. When this icon is clicked, a pop-up window is launched with all the forwardly applicable DSs. (See Figure 3.)

The third kind of icons are for backward chaining without construction. They are lined one by one at the bottom of STATE VIEW. Consequently, if the same axiom is applicable for backward chaining with and without construction at the same time, the axiom appears

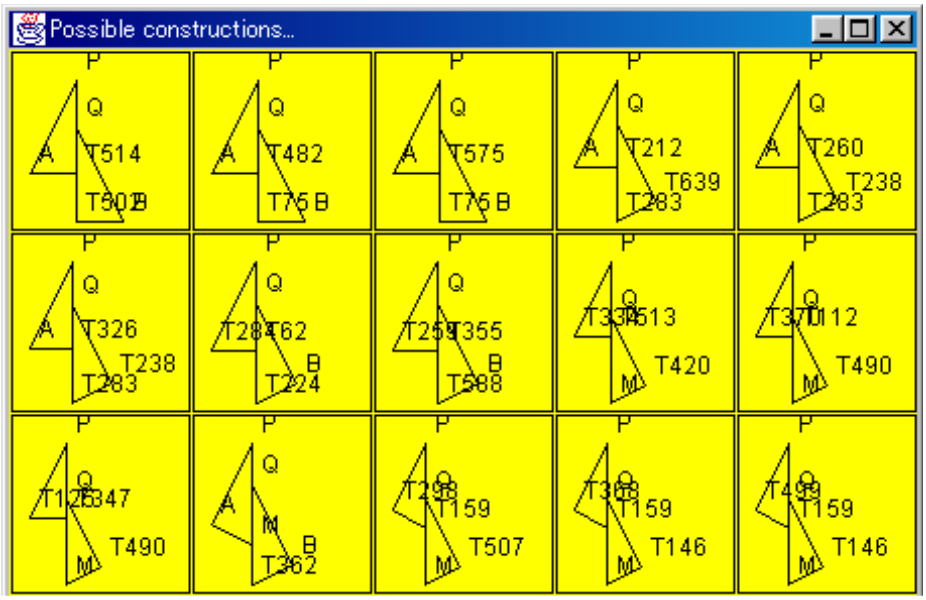


Figure 4: An example of DSs in a pop-up window.

in two different icons at the bottom of the window; one for construction and the other for backward chaining.

When an icon in a pop-up window or at the bottom of STATE VIEW is clicked, a configuration of the corresponding axiom is overlapped against the problem figure. In other words, a unification is reified. If the selected icon is for construction, then auxiliary lines also appear. In Figure 5, an application of the axiom of triangle congruent illustrates the auxiliary lines overlapped the diagram in STATE VIEW.

At the same time, the corresponding axiom appears automatically in a separate window, called the diagrammatic schema browser (or DS BROWSER for short), with which students can browse all the geometry axioms. (See Figure 6.) DS BROWSER shows the sentential expression of an axiom together with its configuration.

Finally, when an icon is "selected" by double clicking on it, then the selected icon is highlighted, and a new state is expanded by applying the selected DS to the current state (i.e., where the icon is clicked). Then, the contents of STATE VIEW are updated accordingly. The diagram is modified if the applied axiom is for a construction. The goal is replaced if

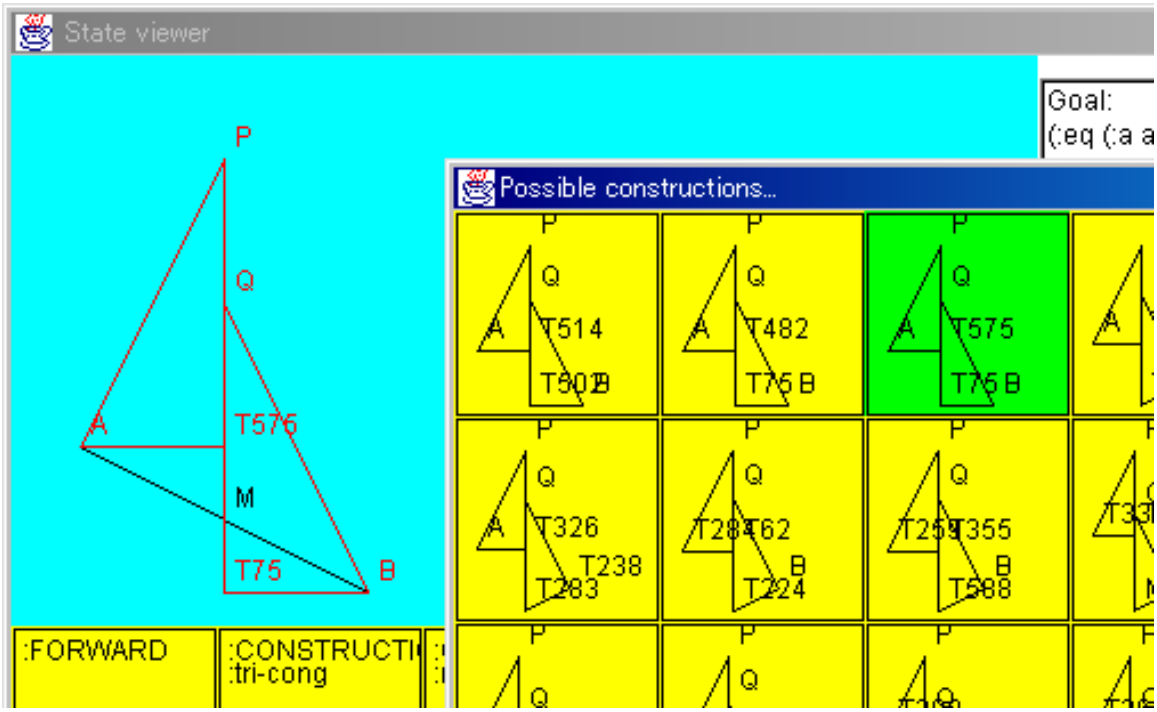
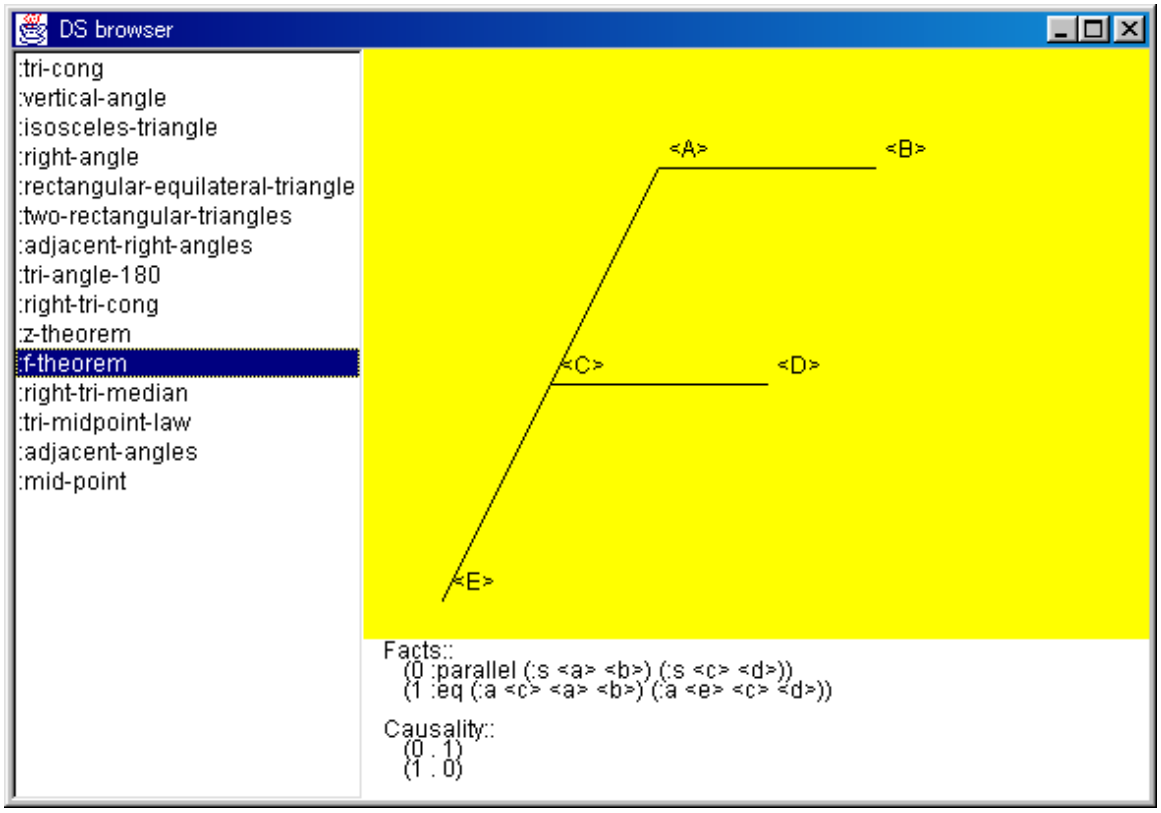


Figure 5: An example of reified DS application.

the axiom is applied backwards. The new facts are added if the icon for forward chaining is selected.

The students can observe how a proof develops in State View by selecting an applicable DS (i.e., an icon) at each state. They can return to a previous state by clicking the Back button on State View. When they return to a state, the operator application they made when exiting it earlier is still highlighted. This facilitates a methodical (e.g., left to right) exploration of the operator applications. Also, the number at the right-bottom corner of State View shows a depth of the search.

As added scaffolding, the students can be constrained to conduct a depth-limited search. That is, when the search depth reaches a maximum (found by running GRAMY), the student is told that they should proceed no further, and no applicable DS appears. Thus, the student must back up to a previous state and take another path.



$(i.j)$ in causality means that fact j is a logical consequence of fact i. i might be a list for conjunctive premises.

Figure 6: Diagrammatic Schema Browser

In sum, this state exploration facility provides students with powerful scaffolding on learning the problem-solving strategy as follows:

(S1) Each state in a search tree is reified by visualizing a diagram, goals to be proven, known facts, and applicable axioms (i.e., DSs).

(S2) The successors expanded at each node in the search tree are reified as the clickable icons.

(S3) An application of an axiom is reified by overlapping a configuration against the diagram, and also showing sentential expressions in Ds Browser.

(S4) For the auxiliary line problems, construction is reified as an overlapped configuration against the diagram.

(S5) State expansion is reified as a "selection" of an applicable DS.

(S6) Backing-up is reified as turning to states visited earlier, and selecting an alternative icon.

These properties of the GUI are numbered for easy reference in the next section.

4 Discussion

Although this research project is at the very beginning stage, and no experimental evaluation has been done yet, several expectations of instructional benefit emerge by considering empirically established learning theory. Merrill and Reiser [8] have argued that a learning environment that holds the pedagogical principles listed in Table 1, called a reasoning-congruent learning environment, supports students in learning complex problem-solving strategies. Furthermore, they have also observed [9], by comparing three different learning environments in the same domain, that the more the reasoning-congruent principles held, the less difficulties (in terms of learning time, number of errors, and a number of trials to hit a solution) students experienced to attain a masterly level (in terms of a number of errors in post-test).

Their principle II is implemented by the scaffolding policies S1–S6 (listed at the end of preceding section). In particular, by listing selectable goals, the State View displays critical information that is normally invisible.

Next, students can see how an axiom would work against the diagram (S3), what auxiliary lines would be available (S4), and what state would appear by applying an axiom (S5). That is, a local problem-solving plan in their mind is reified. Thus, the principle III is held.

Students can return to a previous state (S6) where a selected axiom is highlighted (S5), so they can always keep track of their search. These scaffolding policies implement the principle IV.

Table 1: The principle of reasoning-congruent learning environment

Render invisible behavior visible.

I. Make students' own reasoning explicit by having them make predictions of behavior.

II. Render behavior visible by allowing student to access normally invisible states.

Support incremental planning and use of environment as note pad.

III. Minimize the translation process from the students' internal plans to the external representation of the solution.

IV. Have the structure of a partial solution remind the students of where they are in their solution plan and the search space of the domain.

V. Allow students to focus on subproblems on the way to solving the entire problem, thereby avoiding premature commitment and exploiting independence of subgoals.

Lead students to engage in more effective strategies.

VI. Proactively guide problem solving by encouraging students to use a more profitable set of tools for solving problems.

From Merrill and Reiser (1993) with the Arabic numbers retouched.

Each node in a search tree (i.e., a state) shown in STATE VIEW explicitly displays goals that can be selected and worked on in that state (S1). So, students never lost the focus on a goal hierarchy. Thus, the principle V is also held.

The desired problem-solving strategies (i.e., forward chaining, backward chaining, and backing up) are reified by all the policies S1–S6. Specifically, students can carry out forward and backward chaining by S1, S2 and S5. Meanwhile, they can carry out backing up by S6. So, the principle VI is implemented.

In sum, five principles out of six are held by our scaffolding policies. This implies that our reification technique should facilitate students' learning in geometry theorem proving with construction.

In addition to the above expectations, there exist other findings from literature that support our GUI.

First, since our GUI provides a justification of theorem application visually (S4) and literally (S3), it is also expected that students are unlikely to induce shallow, unjustified operators — that is, knowing what is true in the diagram without knowing why. A study on forcing students to state a reason for axiom application showed that always exposing the students to the rationale of a theorem application prevented them from learning shallow knowledge [1].

Second, the visualization of axiom application against the diagram (S4) might strengthen students' memory of the experience of theorem application. Lovett and Anderson [4] observed that the diagram, not the logical structure of the proof, appeared to serve as the basis for subjects' memory for geometry proof, and the quality of their memory for a specific problem-solving episode seemed to impact later success in solving analogous problems.

5 Conclucion

A reification of the problem-solving strategies in a complex domain is addressed. A GUI that reifies ingredients required by a target search strategy was implemented and its potential for facilitating learning was discussed. The learning environment is powerful enough so

that students can see how the problem-solving strategy is utilized to develop a proof in a single simple window. A central device in the learning environment (i.e., STATE VIEW) is implemented with the common GUI design that should be familiar to the students who have used web browsers.

Although we focus on a geometry theorem proving, the reification technique used in this paper is general so that it could be applied to other domains that need backward chaining as well (e.g., PROOF TUTOR [13] and ALGEBRA TUTOR [10], etc.).

References

[1] Vincent Aleven, Kenneth R. Koedinger, H. Colleen Sinclair, and Jaclyn Snyder. Combatting shallow learning in a tutor for geometry problem solving. In Barry P. Goettl, Henry M. Halff, Carol L. Redfield, and Valerie J. Shute, editors, *Proc. of ITS '98*, pages 364–373. Springer, 1998. Lecture Notes in Computer Science, No.1452.

[2] J. R. Anderson, C. F. Boyle, and G. Yost. The geometry tutor. In *Proceedings of the 9th IJCAI*, pages 1–7, 1985.

[3] Kenneth R. Koedinger and John R. Anderson. Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design. In S. Lajoie and S. Derry, editors, *Computers as Cognitive Tools*. Erlbaum, 1993.

[4] Marsha C. Lovett and John R. Anderson. Effects of solving related proofs on memory and transfer in geometry problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20(2):366–378, 1994.

[5] Noboru Matsuda and Toshio Okamoto. Diagrammatic reasoning for geometry ITS to teach auxiliary line construction problems. In *Proc. of ITS98*, Lecture Notes in Computer Science No.1452, pages 244–253. Springer-Verlag, 1998.

[6] David McArthur, Cathleen Stasz, and John Y. Hotta. Learning problem-solving skills in algebra. *Journal of Educational Technology Systems*, 15(3):303–324, 1986.

[7] Jean McKendree, Bob Radlinski, and Michael E. Atwood. The grace tutor: A qualified success. In C. Frasson, G. Gauthier, and G. I. McCalla, editors, *Proc. of Intelligent Tutoring Systems*, pages 677–684, Berlin, 1992. Springer-Verlag.

[8] Douglas C. Merrill and Brian J. Reiser. Scaffolding learning by doing with reasoning-congruent learning environments. In *Proc. of the Workshop in Graphical Reprentations, Reasoning and Communication from the World Conference on Artificial Intelligence in Education*, pages 9–16, Edinburgh, Scotland, 1993. The University of Edinburgh.

[9] Douglas C. Merrill and Brian J. Reiser. Scaffolding effective problem solving strategies in interactive learning environments. In *Proc. of the Annual Conference on the Cognitive Science Society*, pages 629–634. Erlbaum, 1994.

[10] Robert Milson, Matthew W. Lewis, and John R. Anderson. The teacher's apprentice project: Building an algebra tutor. In Roy Freedle, editor, *Artificial Intelligence and the Future of Testing*, chapter 3, pages 53–71. Erlbaum, Hillsdale, NJ, 1990.

[11] Brian J. Reiser, Daniel Y. Kimberg, Marsha C. Lovett, and Michael Ranney. Knowledge representation and explanation in GIL: An intelligent tutor for programming. In H. Larkin and R. W. Chabay, editors, *Computer-assisted Instruction and Intelligent Tutoring Systems*, pages 111–149. Erlbaum, 1992.

[12] Mark H. Richer and William J. Clancey. GUIDON-WATCH: a graphical interface for viewing a knowledge-based system. *IEEE Computer Graphics and Applications*, 5(11):51–64, 1985.

[13] Richard Scheines and Wilfried Sieg. Computer environments for proof construction. *Interactive Learning Environments*, 4(2):159–169, 1994.

[14] Mark K. Singley. The reification of goal structures in a calculus tutor: Effects on problem-solving performance. *Interactive Learning Environments*, 1(2):102–123, 1990.

[15] University Joseph Fourier of Grenoble and the CNRS. *CABRI Geometry*. `http://www-cabri.imag.fr`.

[16] Kurt VanLehn. Cognitive skill acquisition. In J. Spence, J. Darly, and D. J. Foss, editors, *Annual Review of Psychology*, volume 47, pages 513–539. Annual Reviews, Palo Alto, CA, 1996.

[17] Kurt VanLehn. Conceptual and meta learning during coached problem solving. In C. Frasson, G. Gauthier, and A. Lesgold, editors, *Proc. of the International Conference on Intelligent Tutoring Systmes*, pages 29–47, New York, 1996. Springer-Verlag.