Providing Natural Representations To Facilitate Novices' Understanding in a New Domain: Forward and Backward Reasoning in Programming *

J. Gregory Trafton and **Brian J. Reiser**
trafton@clarity.princeton.edu
reiser@princeton.edu

Cognitive Science Laboratory
Princeton University

Abstract

In many domains, novices exhibit a bias in the direction in which they reason about problems. Earlier studies of LISP programmers using a graphical representation suggested that novice LISP programmers tend to reason forward, working from initial input data toward the goal. We examined novice programmers learning LISP using the GIL programming tutor and manipulated the direction subjects were allowed to reason (forward, backward, or free). Subjects who were required to work backwards (from goal toward givens) exhibited more difficulty solving the problems than subjects working forward or subjects left free to chose their direction. Backward subjects required more time to solve problems, made more errors, and required more time to plan each solution. We suggest that these effects and preferences occur because forward reasoning is more congruent with the way novices reason about computer programs, resulting in an increased working memory load for subjects required to work backward.

Introduction

Novices reason in a different fashion from those who have acquired more expertise in a domain. Experts are better able to recognize the important structural features of a problem and initiate an appropriate solution procedure, while novices require more search to construct a solution (e.g., Chi, Feltovich, & Glaser, 1981). In some domains, this results in a noticeable contrast in the *direction* of the reasoning steps taken. Larkin and her colleagues have demonstrated that physics experts reason *forward* from given information, applying appropriate equations to infer new quantities until the goal is reached (Larkin, McDermott, Simon, & Simon, 1980). In contrast, novices use means ends analysis, starting with a desired unknown and searching for equations that include that variable, then generating new subgoals to fill in terms unknown in that equation, and so on, thus reasoning *backward* from the goal toward the given information.

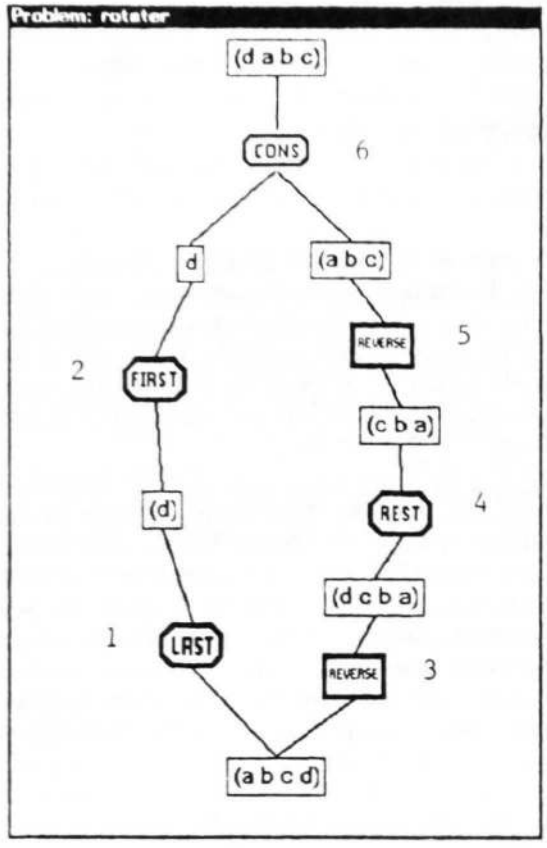
In evaluating the source of this intriguing difference, it is important to consider how novices reason in different domains. Rist (1989) found that novice Pascal programmers tended to write their solutions in essentially forward fashion, writing each section of the program from the beginning of the block of code to the end, rather than working backward from the goal of the block to the initial statements. Anderson, Conrad, and Corbett (1989) found that novice LISP programmers tend to construct their solutions "top-down, left-to-right." This corresponds essentially to backward reasoning, constructing the final actions taken on the input data before the initial actions.

What determines the direction in which novices reason in a domain? It may be merely a preference due to students' naive judgments about how to construct solutions. Alternatively, novices may reason in a particular direction because it simplifies the problem solving. Thus, novices may reason backward in physics because this is the simplest way they can select among possible inferences, since they do not have the problem schemas that enable forward inference (Chi et al., 1981).

Research on novice programmers provides an interesting case of forward and backward reasoning. Reiser, Kimberg, Lovett, and Ranney (1991) argued that the graphical representation employed in GIL (Graphical Instruction in LISP), an intelligent tutoring system for LISP programming, provides a more natural external representation for novices to record their reasoning than the traditional text form of programs. Rather than simply retrieving sequences of functions to achieve particular programming goals, novices tend to reason about the behavior of the relevant functions on an example to construct a chain of operations that achieves the desired result. Thus, GIL is designed so that students build a program by constructing a graph

*This research was supported in part by contract MDA903-87-K-0652 from the Army Research Institute, by research grants from the James S. McDonnell Foundation, and the Xerox Corporation University Grant Program. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of these institutions.

of program constructs and the data each manipulates. Students specify the data input to each function in their program and the data that results from it. A complete program consists of a graph specifying how a chain of functions transforms the input data to achieve a particular type of output (see Figure 1).




```
(defun rotater (lis)
  (cons (first (last lis))
    (reverse (rest (reverse lis)))))
```

Figure 1: A completed GIL graph and the corresponding text form of the solution. The original GIL input is the list *(a b c d)* at the bottom of the screen; the goal data is the list *(d a b c)* at the top. The modal sequence of solution steps is indicated by the number shown next to each function.

Students are free in GIL to reason at any step either from the goal data toward the original inputs (a backward step) or from the inputs toward the goal data (a forward step). Reiser, Ranney, Lovett, and Kimberg (1989) found that novices exhibited a strong bias for forward reasoning (95% forward steps, 5% backward steps). Reiser et al. argued that novices can reason more easily about the behavior of programs by working forward from the input data rather than backward from the goal. Interestingly, the order of reasoning exhibited by GIL subjects is opposite that of the surface form of the solution. For example, the first function encountered in the body of the text form (the "append") is in fact the last component of the solution assembled by GIL subjects (see figure 1). If students' reliance on forward steps in GIL accurately represents their reasoning, then novices do not appear to plan their solution in the order in which the functions appear in the text form of the solution. If students are led to enter their code in an outside-in left-to-right sequence, as students working with the text representation may be, they may be led to construct a chain of functions for each subgoal before entering it into their solution. Reiser et al. argue that GIL's scaffolding of forward reasoning is one reason why students solve problems more easily with GIL than standard text LISP.

The preference of the GIL students for working forward in LISP appears to conflict with Anderson et al.'s finding that their subjects tended to construct programs using backward reasoning. One possibility is that subjects were not comfortable enough with the editing freedom Anderson et al.'s tutor provided to construct their code from the inside out, whereas GIL's graphical representation is more neutral in biasing between forward and backward steps. Indeed, it seems reasonable to expect that novices will feel compelled to enter the components of their solutions in the "surface order," the order in which these components appear in the final form of the solution. Such a concern with a mismatch between natural ways of reasoning about a problem and the format of a completed solution have been the motivation for the graphical formats used in a number of interactive learning environments (Anderson, Boyle, & Reiser, 1985; Collins & Brown, 1988).

If novices indeed reason more naturally in one direction than another, then it should be possible to facilitate their reasoning by providing a learning situation that supports that direction of reasoning. The initial evidence in programming relies only on subjects' preferences. The present study attempts to explicitly evaluate whether the direction of reasoning affects the difficulty of learning to solve programming problems. We used the GIL tutoring system because it makes it possible to restrict the direction in which subjects' reason, and it allows fine-grained analyses of the time course and errors in subjects' reasoning.

Table 1 summarizes the three versions of our GIL tutor used in this experiment.

Table 1: Versions of GIL

Forward	Only forward steps permitted. From input data toward goal data.
Backward	Only forward steps permitted. From goal data toward input data.
Free	Forward or backward steps allowed throughout.

The Free condition provides an opportunity to observe how novices choose to reason. The Forward and

Backward conditions enable us to examine whether there is an additional load placed on problem solving by requiring students to reason in the backward direction. We expect the Free and Forward conditions to be similar, and problem solving to suffer in the Backward condition.

Method

Subjects worked through two chapters of an introductory LISP programming textbook and solved problems assisted by GIL.

Subjects: The subjects were 30 undergraduate paid volunteers from Princeton University and other nearby colleges. All had taken no more than one semester of computer programming and had no knowledge of LISP. Ten subjects were assigned to each of the Free, Forward, and Backward conditions, balancing programming ability across conditions, using Math SAT as the predictor of programming ability (Mayer et al., 1986). Math SAT scores ranged from 420 to 760, with a median score of 625 and means of 624, 627, and 627 for Free, Forward, and Backward respectively.

Instructional Materials: The textbook explaining data structures and LISP functions was used for all conditions, with several slight wording changes to describe the process of chaining functions together. The Free condition text was neutral, explaining how one might chose to construct a chain either from the givens toward the goal or the reverse, and the Backward and Forward texts described the method for sequencing functions appropriate to that condition. All subjects solved the same fourteen problems, including one problem worked in conjunction with the Experimenter used to demonstrate how to construct programs in GIL.

Learning Session: Each subject was given the first of three sections of the text. After reading the first section, subjects were led through a brief demonstration to familiarize them with the learning environment. The Forward condition subjects saw only forward steps; the Backward condition subjects saw only backward steps, and subjects in the Free condition saw steps in both directions. After the demonstration, subjects worked through the remaining text and problems at their own pace.

Posttest: Subjects were given a posttest following the learning session, consisting of three near transfer coding problems. Subjects solved the problems on the computer using the same graphical interface with some additional editing features, but received no feedback. The computer did not constrain the subjects' direction of work on the posttest.

Results

We expected subjects in the Free condition to work primarily forward, so we used a planned comparison to contrast performance in the Forward and Free conditions with the Backward condition. In addition, we performed a second set of analyses to examine whether the performance in the learning conditions differed depending on ability. Subjects below the median SAT score (420-620) were classified as the "Low SAT" group and those above the median (630-760) were classified as the "High SAT" group. We performed an analysis of variance with SAT level and learning condition as independent variables.

We examined the number of forward and backward steps in the Free subjects. As expected, these subjects strongly preferred to work forward ($t(9) = 14.7, p < .001$). Subjects worked 95% of their steps forward and 5% of their steps backward. This finding supports the use of a planned comparison to contrast Forward and Free with Backward subjects.

To examine whether the direction of reasoning affected the difficulty of solving the assigned problems, we first considered solution time. This was defined as the total time required to solve the thirteen assigned problems, not including the demonstration problem or time spent reading the instructional text. Backward subjects took longer to solve the assigned problems than subjects working Forward or Free (47.6 min vs 31.5 and 33.4 min; $F(1, 26) = 9.08, p < .01$). Figure 2 shows solution time partitioned into the time spent on correct steps, error steps, initial planning, and cancelled steps. Not surprisingly, Low SAT subjects took longer than High SAT subjects (41.7 and 33.2 min, respectively; $F(1, 24) = 3.61, p < .10$). There is a suggestion that the effect of direction is stronger for the lower ability subjects, although this is not statistically reliable ($F(2, 24) = 2.23, p = .13$ for the interaction).

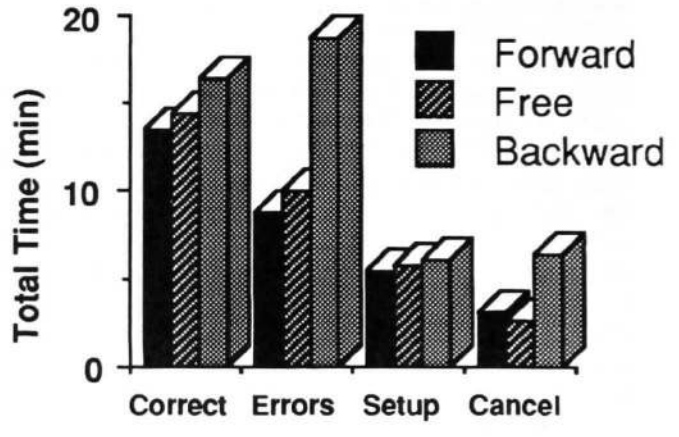


Figure 2: Distribution of learning session time. Time shown is the total amount of time spent on each event type across all 13 problems. Categories are: correct steps, recovery from errors, setting up a plan, and cancelled steps.

There are several possible reasons why subjects in the Backward condition may have taken longer than the other subjects. First, we examined their errors, which can be costly to repair. When an error is made,

GIL prints an explanatory error message, and subjects must read the message and try to fix the error. Indeed, as Table 2 suggests, it appears that the Low SAT subjects in the Backward condition made more errors than either the Forward or Free subjects, while errors produced by the High SAT subjects were not affected by the learning condition ($F(2,24) = 2.89, p < .10$ for the interaction). Differences in solution time may also result from the time subjects spent repairing errors. Subjects who have more difficultly constructing and implementing a plan may spend more time on errors that they do make. Indeed, Backward subjects took more time to repair each error than did Forward and Free subjects (115 vs. 68 and 75 sec; $F(1,26) = 4.36, p < .05$). Furthermore, Backward subjects also spent more time aborting steps ($F(1,21) = 4.8, p < .05$), supporting the hypothesis that these subjects found reasoning more difficult.

Table 2: Mean number of errors in the learning session

	Low SAT	High SAT
Forward	8.2	8.2
Free	8.0	9.2
Backward	17.6	6.0

Interestingly, subjects in the Backward condition spent more time planning or setting up each problem before they started working on it ($F(1,26) = 7.01, p < .05$). Although the differences are small (mean planning time 37 sec vs. 26 and 28 sec), this suggests Backward subjects found constructing an overall plan more difficult than other subjects.

Finally, we examined subjects' performance on the posttest. Backward subjects spent more time on the posttests than the Forward or Free subjects (19.7 vs. 14.9 and 16.3 min; $F(1,25) = 4.78, p < .05$). However, their scores on the posttest problems did not differ ($F(2,22) = 1.06, n.s.$). The posttest allowed all subjects to work freely, so it was possible to examine the direction subjects chose to work. Not surprisingly, the Forward and Free conditions worked almost exclusively forward as they did during the learning session. However, the Backward subjects performed almost three times more forward than backward steps. Although they were unaccustomed to working forward, they chose to do so, and this may have resulted in their longer solution times on the posttest.

Discussion

In the domain of LISP programming, subjects given the choice of which direction to work prefer to work forward. This preference is apparently motivated by the difficulty in reasoning backward in this domain. Subjects in the Backward condition took longer to solve the problems than did subjects who worked forward. This difference is primarily due to the number of errors these subjects made and the difficulty they had in repairing them. The effect appears to arise predominantly in the lower ability subjects; the higher ability subjects are less hampered by requiring them to reason backward. The Backward subjects also had more difficulty in constructing and implementing a plan, as evidenced by their increased initial planning times and greater tendency to abort steps.

The preference for forward reasoning appears to be a strong one. Even after working all the learning session problems using backward steps, the Backward condition subjects still chose to perform a majority of their steps on the posttest using forward reasoning. If this strong preference holds in the more advanced parts of the curriculum in this domain, then these subjects' problem solving during the learning session (in which they were forced to use backward reasoning) may not have prepared them for future problem solving as well as the instruction the Forward and Free groups received. Indeed, the Backward condition subjects took longer to solve the posttest problems, presumably because these subjects were relying on less practiced techniques of forward reasoning.

Taken together, these results support the conclusion that, unlike physics, in which novices tend to use backward reasoning, in LISP programming, backward reasoning is more difficult than forward reasoning. Subjects' reliance on forward reasoning is not merely a preference — subjects allowed to use this type of reasoning perform better than subjects who are required to use backward reasoning, despite the fact that backward reasoning better corresponds to the order of solution components in the traditional form of LISP programs.

We consider three types of explanations for these results.

Naive Causal Models: Novices tend to reason about how programs behave as they try to construct a program to achieve a particular goal. Functions have an underlying directionality — they take in input data and return output data. Novices can understand how functions embed by viewing an embedded function call as a causal chain, in which the output of one function becomes input to another. This type of causal reasoning naturally starts with the first inputs, the initial "cause," and progresses toward the final result, rather than reasoning backward from the desired output to select a function and necessary input that yields that result. The temporal directionality of operators may have some effect on the direction people choose to reason in other domains. For example, chess operators are typically applied in a forward direction (e.g., castling will develop the rook) rather than backward (e.g., to develop the rook, castle), although both of these processes may play some part in the reasoning process.

Constraint of the Problem Space: Because functions are many-to-one mappings, the search of the problem space is more constrained when reasoning forward than

when reasoning backward. That is, a function operating on an input has a unique output, but a function could take many possible inputs that would produce a desired output. Furthermore, there are a greater number of applicable functions when reasoning backward than forward. Thus, students can select between fewer potential choices when reasoning forward. This factor may also explain why novices rely upon backward reasoning in physics. In physics, means-ends analysis is profitable because there is only one goal quantity (e.g., velocity), and there are fewer equations that contain that quantity than the equations that could be used to chain forward from all the given information. For other domains, the direction of work with the fewer applicable operators should be preferred.

Complexity of Planning: There are several reasons to expect the planning to be simpler when reasoning forward. First, the subgoals are more independent when reasoning forward. Many problems involve combining separately obtained objects. When reasoning backward, it is necessary to know what form the inputs will be to know how to combine the subgoals. This often requires further look-ahead, because the form of the inputs to the step will determine which function to use to combine them. In contrast, when reasoning forward, it is necessary only to determine that a particular subgoal is needed, and then one can reason to obtain that data independently of whatever else will be needed to combine with that subgoal. Students can then make greater use of opportunistic reasoning. The student may realize constraints on a subgoal of the program when building a chain to achieve its sibling subgoal. In contrast, when backward reasoning begins with combining subgoals together, it is necessary to commit to a particular combination of subgoals. Many domains have operators that combine objects in a similar fashion (e.g., SAS or ASA for geometry). This hypothesis predicts that reasoning forward should be easier for those domains that have these types of operators.

A related point concerns the ease of determining whether a particular inference is profitable. GIL students reason using a concrete example. Candidate forward inferences can be evaluated by comparing the results and determining whether each would be useful for obtaining the goal data, which typically entails determining whether the candidate result is a component of the goal. Backward reasoning requires determining whether a particular input to a step can easily be obtained from the original inputs. It is typically easier to see whether a particular object is a component of the goal than to determine whether an object is of the form that can easily be obtained from the inputs.

We suggest that these factors together explain why forward reasoning is simpler in this domain. The greater look-ahead required, the larger number of alternatives to be considered, and the difficulty in mentally simulating the execution of a program when reasoning backward all contribute to an increased working memory load, which results in greater difficulty in constructing and carrying out a plan. It is also possible that the Backward subjects are indeed reasoning forward, but just entering their solutions in the backward sequence as required by the tutor. This too would result in an increased memory load, because subjects would have to construct an entire chain of reasoning before entering the outermost (first backward) step.

In future work, we plan to examine how features of the domain can be used to explain the direction that novices reason, and how problem solvers change their direction of reasoning as they gain expertise in a domain.

Acknowledgements: We are grateful to Michael Ranney for many contributions to this research, and to Adnan Hamid for programming assistance.

References

Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*, 228, 456–462.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, 13, 467–505.

Chi, M. T. H., Feltovich, P., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5, 121–152.

Collins, A. & Brown, J. S. (1988). The computer as a tool for learning through reflection. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems.* New York: Springer-Verlag.

Larkin, J. H., McDermott, J., Simon, D., & Simon, H. A. (1980). Models of competence in solving physics problems. *Cognitive Science*, 4, 317–345.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM*, 29, 605–610.

Reiser, B. J., Kimberg, D. Y., Lovett, M. C., & Ranney, M. (1991). Knowledge representation and explanation in GIL, an intelligent tutor for programming. In J. H. Larkin & R. W. Chabay (Eds.), *Computer assisted instruction and intelligent tutoring systems: Shared issues and complementary approaches.* Hillsdale, NJ: Erlbaum.

Reiser, B. J., Ranney, M., Lovett, M. C., & Kimberg, D. Y. (1989). Facilitating students' reasoning with causal explanations and visual representations. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence and Education* (pp. 228–235). Springfield, VA: IOS.

Rist, R. S. (1989). Schema creation in programming. *Cognitive Science*, 13, 389–414.