Journal of Experimental Psychology:
Learning, Memory, and Cognition
1996, Vol. 22, No. 3, 576-598


0278-7393/96/$3.00

# How People Learn to Skip Steps

Stephen B. Blessing and John R. Anderson
Carnegie Mellon University

Novices often explicitly apply in a domain each necessary operator while solving a problem, whereas experts often skip steps, and as a result, the solution procedures they use are often organized differently from those of novices. Using an algebra analog, the authors examined this change in process. In 2 experiments, people learned the rules of the task and then solved many problems. Their solution procedures were monitored, and concurrent verbal protocols were taken. When participants started overtly skipping steps, they appeared to be performing them mentally but later started to use new transformations, thereby covertly skipping steps as well. An adaptive control of thought—rational model (J. R. Anderson, 1993) of problem-solver behavior within this task was developed and evaluated with respect to existing theories of skill acquisition.

As people solve the same type of problem again and again, not only do they get faster at doing that type of problem, but very often the process they use to solve the problems changes as well, often resulting in skipped steps (Koedinger & Anderson, 1990). This reorganization and skipping of steps allow people to solve problems more quickly, efficiently, and easily. One might expect this change in process to result in performance discontinuities in a person's acquisition of a skill because the person is undergoing what may be a radical reorganization of how that skill is performed. However, Newell and Rosenbloom (1981) showed that a variety of skills are acquired at a steady rate, one that follows a power function (an equation of the form $y = a + bx^c$, where $a$, $b$, and $c$ are constants). In this article we examine the step-skipping phenomenon and its apparent relationship to the power law of learning.

Step skipping is often thought of as a compositional process, in which a person who used to take two or more steps to do a task now takes only one. Intuitively then, if a person does a problem in fewer steps in completing a task, he or she should take less time in performing that task. Research by Charness and Campbell (1988) showed that compositional processes account for about 70% of the speedup associated with acquiring a new skill, with the rest of the speedup accounted for by becoming faster at the operations themselves. Work done by Frensch and Geary (1993; Frensch, 1991) also indicated the importance of compositional processes, as distinct from a general speedup in performance, in learning a task. It is evident from this research that composition is an important component to acquiring a skill.

Stephen B. Blessing and John R. Anderson, Department of Psychology, Carnegie Mellon University.

The work presented in this article was supported by National Science Foundation Grant 91-08529 and Office of Naval Research Augmentation Award for Science and Engineering Research Training Grant N000149311010. We thank Richard Carlson, Jill Larkin, Marsha Lovett, and Herbert Simon for their comments on drafts of this article.

Correspondence concerning this article should be addressed to Stephen B. Blessing, Department of Psychology, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890. Electronic mail may be sent via Internet to blessing+@cmu.edu.

Given the importance of these compositional processes, a number of skill acquisition theories attempt to account for the change in process, including step skipping, that occurs when a person becomes expert in a particular domain. The adaptive control of thought—rational theory (ACT-R; Anderson, 1993) predicts that any new procedural skills arise by analogy to examples. Newell and Rosenbloom (1981; Rosenbloom & Newell, 1986) proposed that people acquire skill by a chunking process, in which responses are acquired in larger and larger chunks (patterns). Logan's (1988, 1990) instance theory states that skilled performance is generally characterized as the retrieval of a specific memory, and not the use of a procedure. As can be seen even from just these short descriptions, the proposed mechanisms in these theories differ greatly. These theories are discussed in more detail shortly, but first we want to introduce the phenomenon that we address in this article.

When asked to solve the equation $-x - A = B$ for $x$, some people who are facile in algebra will immediately write that $x = -B - A$. However, people who have just learned how to solve simple linear equations may not be able to go immediately from the initial problem statement to the solution. Rather, they may have to perform three steps, which may correspond to adding $A$ to both sides, collecting $-A + A$ on the left, and multiplying through by $-1$ to solve successfully for $x$. At some point, people will probably learn to collapse into a single step the actions of removing the $A$ from the left-hand side of the equation, changing the sign on the $B$, and adding $-A$ to the right-hand side. What happens during this transition, and how does it take place? Under what circumstances do people start skipping steps? In the experiments described in this article we examine this process. First, we discuss some of the learning mechanisms that have been proposed to account for step skipping.

## Step-Skipping Explanations

There are a few theoretical approaches to step skipping that differ in their details but that involve the same problem-solving framework (Newell & Simon, 1972). As such, they make use of many of the ideas germane to that framework (e.g., operators and production rules, among others). An operator is a rule that can be applied to transform the current problem state into a

different one. Operators are often conceived of as production rules, which are condition–action (IF–THEN) statements that act on the current problem state. A number of these explanations amount to the claim that step skipping involves replacing smaller grain operators by larger grain operators. Three such explanations are rule composition, chunking, and analogy. Another view, Logan's (1988, 1990) instance theory, differs from the previous three by stating that truly expert performance is not characterized by the application of operators but rather by retrieval of a specific example. We discuss each of these four explanations in turn.

### *Rule Composition*

One view of how step skipping occurs is by collapsing operators that follow each other in a problem solution into a single operator (Anderson, 1983; Lewis, 1978). Such collapsing of adjacent operators is referred to as *rule composition.* These composed operators would then produce step-skipping behavior by not outputting the intermediate products that the single operators produced. For example, when first learning algebra and asked to solve the equation presented earlier, $-x - A = B$, the student would probably solve the problem in three steps (the application of three operators) by first adding a $+A$ to both sides, simplifying both sides to get $-x = B + A$, and finally multiplying through by $-1$ to get the final line, $x = -B - A$. Operator composition might compose the first two steps into a single operator that directly produced $-x = B + A$, and then compose this with the third step to create a single operator that produces the final result. Rule composition is an attractive idea because it provides a very concise account of how step skipping occurs.

Work by Lewis (1981), however, casts doubt on whether step skipping can merely be due to composition of adjacent steps. He investigated what he called *powerful operators.* These operators do the work of a series of others, as in the combination of the transposition and collection operators in algebra (shown previously). Their critical feature is that they can combine nonadjacent steps together, a condition that would not happen with standard rule composition. For example, a novice learning algebra would probably solve the equation $x + 2(x + 1) = 4$ by first multiplying both $x$ and 1 by 2, and then adding $x$ and $2x$ together for the next step. However, an expert would probably immediately write $3x = 2$, with the 2 and the $x$ being multiplied and then added to the $x$ before the 2 and the 1 are multiplied and subtracted from the 4. Thus, nonadjacent steps are being combined to produce the operator that writes out the $3x$. Lewis referred to this as the problem of going from a *two-pass* system (solving a problem in multiple steps) to a *one-pass* system (solving a problem in a single step).

### *Chunking*

Newell's (1990) theory of cognition, *Soar,* has a single mechanism that attempts to provide a psychologically plausible account for all of skill acquisition. This mechanism, called *chunking,* operates when an impasse in the problem-solving process occurs. An impasse occurs when the Soar system reaches a state in which no operator applies or in which the knowledge to apply an operator does not exist. Once an impasse happens, Soar will set a subgoal to solve the impasse and then creates a new problem space in which to solve the subgoal. Once the subgoal has been solved, Soar creates a new operator, a chunk, using the results obtained after the impasse is resolved. The chunk is built by starting at the results of the impasse and backtracing (i.e., working backward) through the elements before the impasse that gave rise to these results. The operator that represents the chunk is then constructed by using these original elements that occurred before the impasse as the condition side of a production rule and the results after the impasse as the action side. These chunks could be thought of as a composite operator because the operators used to solve the subgoal would be represented by the resulting chunk. Under such a scheme, the learner builds larger and larger chunks. For example, a person may eventually create a single chunk (in the form of a production) that would produce $x = -B - A$ when presented with $-x - A = B$. It is important to note that within Soar, not only is the process by which a system solves a problem important, but the product is as well. Because of this, Soar avoids the problem of nonadjacent operators inherent in the rule composition view. In building the chunk, Soar backtraces from the results of solving an impasse to the conditions before the impasse.

### *Compiling Rules From Analogy*

Within Anderson's (1993) ACT–R framework, cognitive skills are realized by production rules. Skilled performance is characterized by executing these production rules to do the task at hand. The only mechanism by which new production rules can enter the system is through analogy to an existing example in declarative memory. When given a task to do for which no production rules apply, ACT–R attempts to locate a previously acquired declarative example that exemplifies the application of the correct operator that is similar to the current goal. If a similar enough example is found, the analogy mechanism produces a production rule that will generate the correct action. In short, novices solve problems by referencing examples, whereas experts apply operators. Step skipping in which this analogy mechanism is used could be produced by forming analogies between examples that incorporated the application of multiple operators. A model of such a process, including a fuller account of the analogy mechanism, is presented after a discussion of the empirical results.

### *Retrieval of Examples*

In Logan's (1988, 1990) theory, performance on a task is governed by a race between executing the appropriate operator and retrieving a specific example. When first starting to do a task, people will apply the necessary operators. Eventually a strong enough association will be made between a specific stimulus and its appropriate response, and so direct retrieval of the response can take place. In comparison to Anderson's (1993) theory, Logan's makes the opposite prediction—that novice performance is marked by application of operators, whereas skilled performance is done by retrieval of past examples. Step skipping in such a framework could perhaps

occur once the task is done by direct memory retrieval. If one made the assumption that the retrieved memory trace of a problem solution could contain multiple steps of that problem solution, the problem solver could only produce the last step contained in the instance, thus eliminating intermediate steps. This would allow the problem solver to skip intermediate steps.

The two previous theories are not necessarily mutually exclusive, and evidence that supports both theories has been collected. In a study done by Anderson and Fincham (1994), some participants were given only examples of rule applications. Participants were able to extract the correct rules from the examples and were able to apply them to problems presented to them. Even in the absence of repeated examples, participants still exhibited power law learning, contrary to what the strong version of Logan's instance theory would predict. However, in unpublished follow-up studies, Anderson and Fincham found that learning was faster when examples were repeated. Carlson and Lundy (1992) also tested participants' ability to learn rules, varying the consistency of both the examples and the sequence of operation. They found separable benefits of both. That is, participants were able to learn faster either if the data were consistent (i.e., repeated examples) or if the order of operations was consistent.

## The Current Experiments

The process by which people start skipping steps has not adequately been explored. Since the work done by Lewis (1978), few researchers have examined how people learn to skip steps in solving problems. One issue that arises when people begin to skip steps is whether they are actually mentally skipping steps or are only physically skipping steps. That is, people may just be going through each step in the problem mentally, and only performing some of the steps, as opposed to mentally going straight from the first step to the last step. We refer to this as *overtly* skipping steps versus *covertly* skipping steps. We discussed the above-indicated theories in terms of how they account for covert step skipping, but no experiments have been done to test their claims.

In the work presented in this article we examined the development of step skipping within a particular task, one similar to algebra. By a close examination of people performing our task, a better understanding of the skill acquisition process could be attained. Because the task was relatively easy, participants could reach expert status within a single experimental session. By comparing their knowledge and use of procedures at the beginning to that of the end, we established the two endpoints for which a process has to account. Furthermore, by examining the participants' actions between these two endpoints and from concurrent verbal protocols (Ericsson & Simon, 1990), we were able to investigate participants' use of examples, their step-skipping behavior, and their change from making each step explicit to skipping certain steps. From this analysis we were able to develop a process model of the observed behavior, part of which we have implemented as an ACT–R (Anderson, 1993) model.

Table 1
*How the Symbols Used in This Task Map Onto Algebraic Symbols (Experiments 1 and 2)*

| Algebraic symbol | Symbol in task |
|---|---|
| + | ® |
| − | ♥ |
| * | # |
| / | © |
| Operands | Δ |
| | Γ |
| | Φ |
| | Ω |
| x | 𝒫 |
| = | ↔ |

## Experiment 1

When and how do people start skipping steps when solving problems? Experiment 1 was designed to explore this question. By instructing participants in the rules of our task and letting them solve problems in it, but not forcing them to make all steps explicit, a situation was created in which we could study how problem solvers naturally learn to skip steps. Also, participants were instructed to think aloud while solving the problems. By studying these protocols, the relation between physically skipping steps and mentally skipping steps can be more closely examined. It could be the case that problem solvers who are overtly skipping steps are still going through each step mentally, in which case no compositional processes are occurring.

As stated previously, the task we designed was an analog of algebra. The rules defining the task are given in Appendix A. In place of the standard four operators and Roman letters, we used Greek and various other symbols to mask the similarity to algebra. Table 1 lists the symbols we used and how they map onto the standard algebraic symbols. In most of our examples, we used the standard algebraic symbols so that the reader could use previous knowledge to decode parts of the task. Table 2 contains an example of one of the hardest problems, with all of the steps needed to solve the problem made explicit. The first step in solving this problem is to add ®Φ to both sides of the character string (the ↔ divides the string into left and right halves) in accordance with Rule 1. For the second step, the ... ♥Φ ... ®Φ is deleted from the left-hand side by application of Rule 5. For Steps 3 and 4, #Ω is added to both sides of the string (Rule 1 again), and then the ... ©Ω ... #Ω is deleted from the left-hand side (Rule 3). For the final step,

Table 2
*Sample Problem*

| Step | What participants saw | Algebraic mapping |
|---|---|---|
| Given step | ♥Φ©Ω♥𝒫↔♥Δ | $-A \div B - x = -C$ |
| Step 1 | ♥Φ©Ω♥𝒫®Φ↔♥Δ®Φ | $-A \div B - x + A = -C + A$ |
| Step 2 | ©Ω♥𝒫↔♥Δ®Φ | $\div B - x = -C + A$ |
| Step 3 | ©Ω♥𝒫#Ω↔♥Δ®Φ#Ω | $\div B - x * B = -C + A * B$ |
| Step 4 | ♥𝒫↔♥Δ®Φ#Ω | $-x = -C + A * B$ |
| Step 5 | 𝒫↔®Δ♥Φ#Ω | $x = +C - A * B$ |

Rule 8 is applied to eliminate the ♥ from in front of the $\mathscr{P}$. It should be noted that the rules are constructed such that this is the only order of steps possible in solving this problem.

## Method

*Participants.* Twelve Carnegie Mellon University undergraduates were paid to participate in this experiment. We placed an ad on a local electronic bulletin board, which many undergraduates read, to solicit the participants.

*Materials.* We constructed an algebra analog for this experiment. Differences existed between this task and algebra, and so the mapping was not perfect. For example, the division–multiplication operator pair acted more like the addition–subtraction operator pair than in standard algebra. Also, this task had a more limited order of operations. There are rules of precedence (Rules 10 and 11 in Appendix A), but to keep the task simple, parentheses were not used as some of the allowable manipulations would look strange in algebra. Also, any operator was allowed in front of $x$, so it was possible to end up with an equation that looked like $*x = A + B$. The order of operations was constrained so that at each step in any problem, only one rule was applicable. That is, at any intermediate step in solving a problem, only one operator can be used to achieve the next step in the problem. There was never a choice between operators.

In all, the task consisted of 11 rules that the participants used to manipulate the character strings to achieve the goal state, which was to "isolate" (i.e., solve for) the $\mathscr{P}$ symbol on the left-hand side of the equation. One rule (Rule 1 in Appendix A) resembled the algebraic rule that states the same thing can be added to both sides of an equation (e.g., one can add a $+A$ to both the left- and right-hand sides), four rules (Rules 2–5) dictated how operators and operands could be removed from one side of an equation (e.g., if $+A - A$ appeared on the left-hand side, it could be deleted from the equation), four rules (Rules 6–9) described how to remove a sign in front of the $x$ symbol when one appeared, and two rules (Rules 10 and 11) provided information on precedence and the order of operations. The division removal rule (Rule 9, which applied when the equivalent of $/x$ appeared by itself on the left-hand side) depended on what the right-hand side of the equation contained. Because of this, the division removal rule had two incarnations: an easy one and a hard one.

These rules were presented to participants as screens of information, one rule per screen. The screens were similar to one another, with a schematic of how the rule applies, a short text description of the rule, and an example of an application of that rule. The task was implemented as a Hypercard 2.1 stack (Apple Computer, 1991) which was run on an accelerated Apple Macintosh IIci computer connected to a two-page monitor.[1]

*Procedure.* All participants were told to think aloud while solving the 200 problems they were given. (Because of a computer error, 1 participant received only 167 problems. This has been accounted for in the data, and the results do not suffer from this incident.) Before starting the task, each participant was instructed on what it meant to think aloud and was given two think-aloud warm-up tasks: one a mental addition problem and the other counting the windows in their childhood home. The rest of the experimental instructions was part of the Hypercard program. The participants had to go through all of the rule screens before starting to solve the problems. Once they started solving problems, participants could refer back to a particular rule screen by clicking on the appropriate button. Each participant received 50 problems of four different types, which are detailed in Table 3. The problems were randomly generated on-line within certain parameters to ensure that each rule was presented an equal number of times to each participant. The program kept track of the time between each symbol clicked and what symbol was clicked, so a complete record of the participants' responses to each problem was obtained.

Each problem was presented in a box near the top of the screen. The participant then used an on-screen keypad that contained all of the symbols used in the task to click out, with the mouse, a correct step that would follow from the problem or from one of the lines the participant had already clicked. A delete key was available to erase any character they had clicked. The participant's lines appeared in a box below the problem. Once the participant had clicked out a step, he or she clicked a special key to have the computer check the answer. If the step they had clicked out was a correct one, the computer would respond, "Good," and the participant could continue with the problem. If the line clicked out was the problem's solution, then the computer would respond, "Excellent," the box containing the participant's lines would clear, and a new problem would appear. If, however, the line was incorrect, the computer would respond, "Try again," the participant's line would be erased from the box below the problem and moved to a different location, and the participant would then have another chance to click out a correct line. If the second attempt was not correct, the computer would respond, "Here's the correct line," the next correct step (following from the last correct line) would appear, and a dialog box would appear listing which rule the participant should have applied. The participant could also click on a button with a ? to receive a hint. This feature was used very rarely, only 10 times in total by the 12 participants.

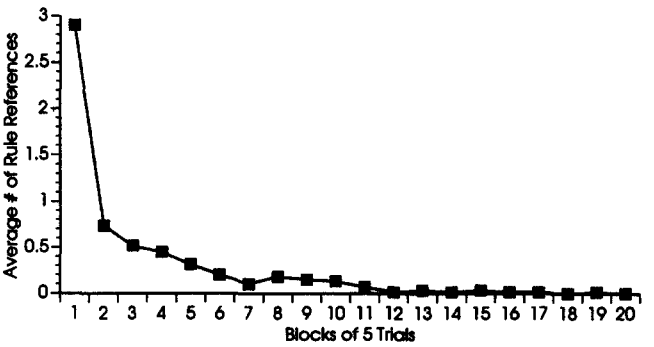


*Figure 1.* Mean number of rule references per problem by problem block (Experiment 1).

Table 3
*The Four Different Types of Problems (Experiment 1)*

| Step | Prototype equation[a] |
|---|---|
| Two step | $x \oplus A = \oplus B$ |
| Three step | $\oplus x \oplus A = \oplus B$ |
| Four step | $x \oplus A \oplus B = \oplus C$ |
| Five step | $\oplus x \oplus A \oplus B = \oplus C$ |

[a]The $\oplus$ represents any of the four operators. An operator can optionally appear as the first symbol on the right-hand side. $A$, $B$, and $C$ represent any of the four constants used. For Type 2 and Type 4 equations, the $x$ can appear in any of the positions.

[1] The maximum temporal resolution of Hypercard is 1/60 of a second. The program that we used to collect data was written to minimize any inaccuracy due to this fact. Furthermore, the timing data of interest were sufficiently long enough that any inaccuracies due to poor temporal resolution should not matter.

### *Results*

Participants spent an average of 13.5 min going through the introduction and the rules of the task (15 screens in all). The average amount of time working on the 200 problems was 91 min, with the extremes being 61 min and 120 min. Each participant made references back to the rules 29.7 times on average. Figure 1 shows the average number of times participants referred back to the rules per problem for the first 100 problems (the number of referrals stays essentially at zero for the last 100 problems). Note that we have blocked trials together in this graph, and in the graphs to follow, to aid readability. Participants appeared to rapidly master the rule set.

Before examining the step-skipping behavior of the participants, and all of them did learn to skip at least some steps, it is worth looking at the overall time it took participants to work on the problems. Figure 2 graphs the total time participants spent on a problem of a particular type by block. For each problem type, a power curve of the form $y = a + bN^c$ (where $a$ is the asymptote, $b$ is the performance time on the first block, $N$ is the block number, and $c$ is the learning rate) has been fit to the data (Newell & Rosenbloom, 1981). As can be seen, the fit to the data was quite good. However, behind this quite regular quantitative finding, which was found in many different tasks, lie large differences in the qualitative aspects of how participants were solving the problems. As we see, different participants began skipping different steps at different times.

Figure 3 depicts the overt step-skipping behavior of the participants. Each line represents a particular kind of step that a participant could skip and shows the number of participants that had consistently skipped that step as a function of the number of opportunities to skip the step. Referring to the

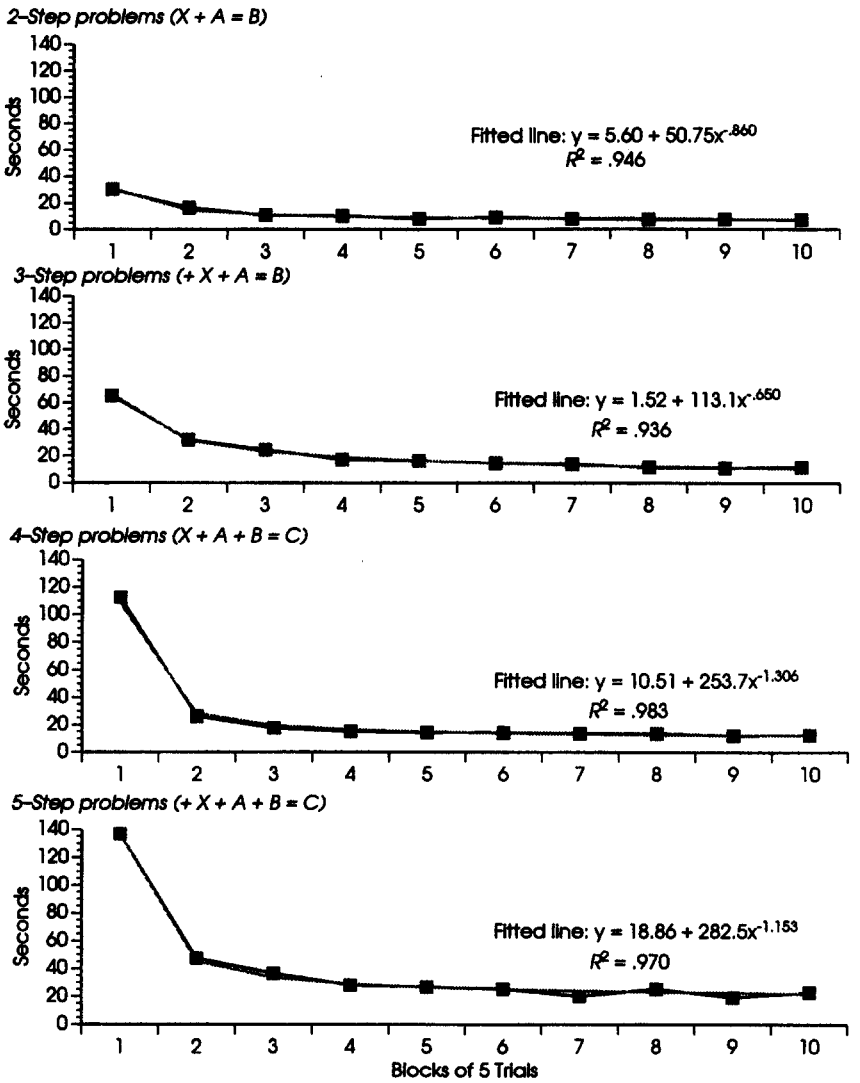


*Figure 2.* Overall time for solving problems by block for each problem type (Experiment 1).

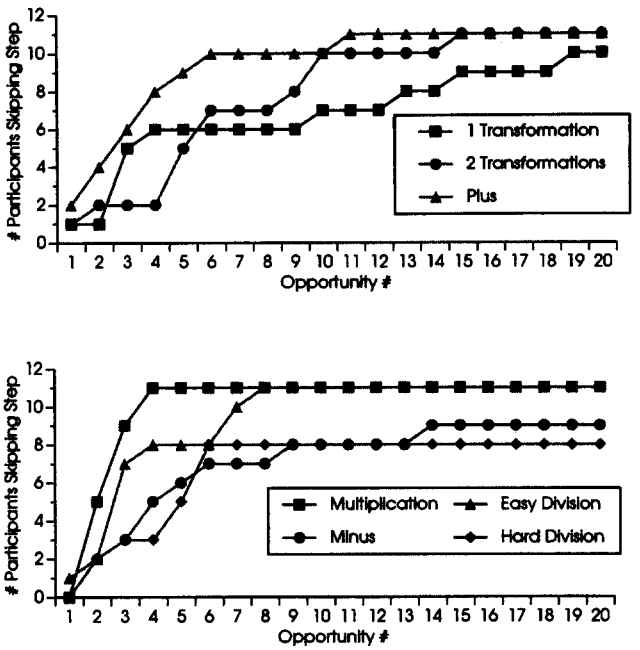


*Figure 3.* Step-skipping behavior of participants (Experiment 1).

example displayed in Table 2, skipping a one-transformation step would be skipping either Step 1 or Step 3. A participant would be credited with this if they clicked Step 2 or a later step on receiving the problem statement or Step 4 or 5 after clicking Step 2. Skipping two-transformation steps, possible only in four- and five-step problems, would occur when the participant received the problem statement and then clicked out Step 4 or Step 5. The other types of step skipping depicted in Figure 3 refer to the steps in which an operator was removed from in front of the $x$. From the example in Table 2, participants would have skipped the minus removal step if they did not click out Step 4. That is, participants were credited as having skipped the operator removal steps when they did not explicitly click out the step involving the operator and $x$ by themselves on the left-hand side of the character string. Throughout the problem set, there were 300 opportunities to skip a one-transformation step (the two- and three-step problems each had one, and the four- and five-step problems each had two), 100 opportunities to skip two transformations at once (once on each four- and five-step problem), and 25 opportunities to skip all four rules dealing with removing each operator in front of the $x$. (Given the random nature of the problem generator, it would be expected that 9 of the 25 problems containing division removal would involve the hard version of that rule. In fact, each participant received an average of 8.1 problems involving the hard division removal rule.) Because of these differences in rule application opportunities, the graph in Figure 3 does not represent the order in which the participants began to skip the rules.

As noted, Figure 3 graphs consistent step skipping. Consistent skipping is defined not only as skipping the step on that opportunity but also on the next three opportunities in which that step could be skipped. In 72 of the 84 cases, once a participant started to skip a step they continued to skip that step. Thus, it was seldom the case that a participant would skip a step once, then on the next opportunity not skip that step. Of the 12 cases in which a participant would relapse into making a past-skipped step explicit, all were either the minus or the hard division operator elimination step. It should be stressed that this was a measure of overt step skipping, not covert step skipping. That is, this graph depicts only steps that participants were physically skipping. It could be the case that participants were still mentally going through each transformation, and so were not covertly skipping steps. However, by examining protocol data, to be discussed later, more light can be shed on this particular issue.

One of the curious features of the data is that we never saw a participant presented with a four- or five-step problem skip to Step 3 (see Table 2). That is, participants would first start skipping steps like Step 3. The additional complexity that occurred when an operator and operand were added to both sides, one of which would be taken away again on the next step, made this an unnatural step to skip. Participants only skipped to equations as simple as the original. It appears, for this task at least, there was somewhat of a natural progression in how steps were skipped, that of skipping the cancellation steps first (Steps 1 and 3), then the extra addition step (Step 2), and then the step that removed the operator in front of the $x$ (Step 4).

The subjectively easier steps (in both our view and the view of the participants in postexperiment questioning) were generally skipped first. It is also the case that the more practiced rules were skipped first as well. The one-transformation step, for all participants, was always the first one skipped, usually followed by doing two transformations as one step. In all cases, once participants started skipping the one-transformation step for one operator–operand pair, they skipped it for all pairs. The typical order that the operator removal steps were skipped went (a) plus and multiplication (which were essentially identical rules), (b) easy division, (c) minus, and then (d) hard division. Not all participants skipped all steps, as is evident from Figure 3. One participant only skipped the single-transformation steps. When asked why he made all the other steps explicit, he responded, "[I]f I tried to do two steps at once, it'll be just like four times the chance for error."

An interesting feature of the graph is that in almost all cases, the participants had to click out at least one instance of a step before it could be skipped. Only 1 participant never did a single transformation in two steps, only 1 participant never did two transformations on separate lines, 1 participant immediately skipped easy division, and 2 participants immediately skipped the plus-operator elimination step. For the rest, however, at least one occurrence had to be completed and, usually more, before that step was skipped.

The graph in Figure 4 shows the mean number of lines clicked out by the participants for each problem throughout the experiment. When this number is one, the participants are clicking out the answer as their first response after the problem is displayed, skipping all intermediate steps. At the start of the experiment, however, the participants were clicking out multiple lines per problem, and so this number is higher. Figure 5

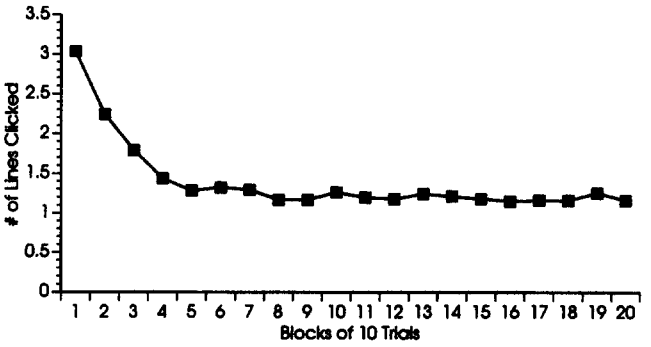


*Figure 4.* Mean number of lines clicked by problem block (Experiment 1).

graphs the mean seconds per character (i.e., total solution time divided by number of characters clicked), averaged over all participants, for each problem separated by the four different problem types. As can been seen, this measure steadily decreased throughout the experiment. Furthermore, as is evident from Figure 4, the number of characters that each participant clicked out per problem also decreased. This means that not only were participants getting faster at clicking out their answer, but they were also clicking out fewer lines (and therefore fewer symbols) per problem. All participants by the end of the experiment, except for the one, clicked out only the final line for most problems. Thus, hiding behind the regularity in Figure 2 are two more regularities depicted in Figures 4 and 5—a reduction in the number of physical actions and a reduction in the time per action.

As can be seen from Figure 5, the major difference in time per character occurs between problems with an even number of steps and problems with an odd number of steps. The difference between even and odd steps (two step = four step < three step = five step) was significant by a Duncan multiple range test ($p < .01$). The problems with an odd number of steps are the ones in which the participant had to remove a sign in front of the $x$. The importance of that fact becomes more apparent in Experiment 2.

All participants, except for the one, eventually got to the point where they would click out the final line (i.e., the solution) of a problem as their first line of input. The graph in Figure 6 examines the individual latencies between clicking the symbols for these solutions, which we termed *perfect solutions,* for the three-step problems. Because each participant started skipping steps at a different time, participants had a differing number of these perfect solutions. Each point on the graph represents at least 8 participants. The largest latency was before the first keypress (the $x$), in which the best fitting function was a power curve (see Table 4). The latencies between the other keypresses remained relatively constant throughout the problem set. An analysis of the other lines, the nonperfect solution lines, revealed a similar pattern. These results are suggestive that participants plan the whole line they are going to type next and then simply click out the sequence of symbols that constitute the line. The time to click out each line could therefore be divided into *planning* time and *execution* time. Planning time would be the time spent before the first keypress, and execution time would be the amount of time spent per character clicking out the rest of the line.

We fitted power functions, as in Figure 2, to the planning and execution time. Table 4 shows the best fitting power functions for each type of problem in the data set and with various constraints placed upon the parameters. As can be seen in Table 4, the fit was still good even when there were only six free parameters (where $b$ varies for each type of problem and $a$ and $c$ are constrained to be the same across problem types). Performing $F$ tests on the basis of the deviations and the number of free parameters did reveal that something is lost in planning time when going from 12 free parameters to 6 free parameters, $F(6, 28) = 8.44$, $MSE = 0.305$, $p < .05$. In contrast, nothing was lost by imposing the constraints on the execution time, $F(6, 28) = 1.07$, $MSE = 0.006$. Despite the

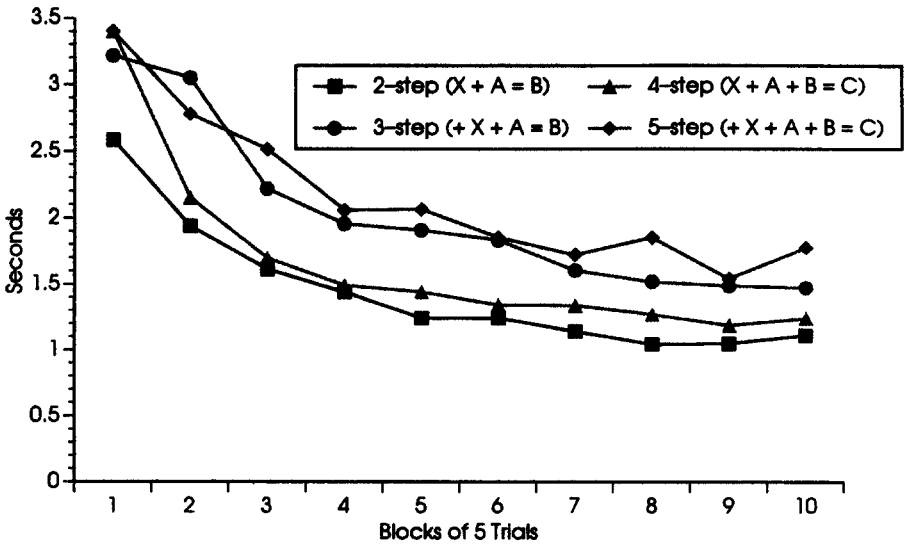


*Figure 5.* Mean seconds per character by block for each problem type (Experiment 1).

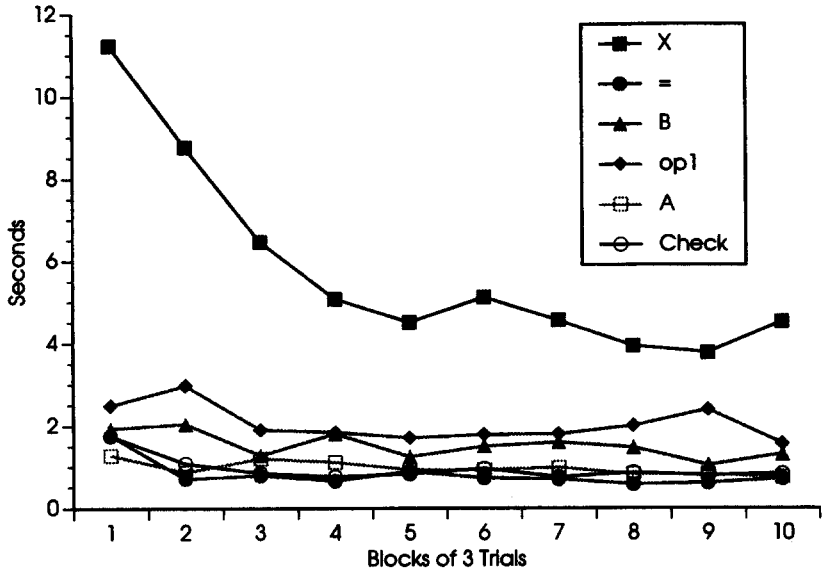


*Figure 6.* Latency by block between each character in "perfect" solutions of three-step problems (Experiment 1); op = operator.

slight loss of fit, it is impressive that the differences among conditions could be reduced to a 6-parameter model with little loss in prediction. In the 6-parameter model, the values of $a$ and $c$ are constant across number of steps skipped (and $a$ estimates to be zero). Constraining $a$, the intercept term, implies that participants are converging to the same physical limit in producing these lines. Constraining $c$, the exponent, implies that participants have the same learning rate for each type of problem. The 1 parameter that separates the conditions is the multiplicative factor $b$. The variations in this parameter were quite large, particularly for the planning times. This parameter was large when more steps were being skipped, suggesting that participants may still have been going covertly through the steps and only overtly skipping them.

### *Discussion*

An analysis of people learning this task sheds light on how people learn to skip steps, both overtly and, as to be discussed shortly, covertly. At the beginning, participants performed the task according to the given rules. When given a problem, they would either think over or refer back to the rules and pick the

Table 4
*Best Fitting Power Functions ($y = a + bN^c$) for Symbol in Final Line by Problem Type*

| Step | Planning | | Execution | |
|---|---|---|---|---|
| | Latency | $R^2$ | Latency | $R^2$ |
| | 12 free parameters[a] | | | |
| Two step | $6.59N^{-0.545}$ | .95 | $1.14N^{-0.202}$ | .89 |
| Three step | $1.50 + 9.95N^{-0.628}$ | .96 | $0.81 + 0.74N^{-0.889}$ | .94 |
| Four step | $1.73 + 9.87N^{-1.151}$ | .99 | $0.33 + 0.99N^{-0.243}$ | .76 |
| Five step | $13.52N^{-0.375}$ | .94 | $1.58N^{-0.171}$ | .74 |
| | 9 free parameters[b] | | | |
| Two step | $0.77 + 5.95N^{-0.705}$ | .94 | $0.47 + 0.68N^{-0.425}$ | .88 |
| Three step | $2.04 + 9.49N^{-0.705}$ | .96 | $0.49 + 1.00N^{-0.425}$ | .92 |
| Four step | $10.92N^{-0.705}$ | .98 | $0.65 + 0.68N^{-0.425}$ | .76 |
| Five step | $4.09 + 9.78N^{-0.705}$ | .89 | $0.77 + 0.83N^{-0.425}$ | .73 |
| | 6 free parameters[c] | | | |
| Two step | $6.19N^{-0.477}$ | .95 | $1.13N^{-0.192}$ | .89 |
| Three step | $11.23N^{-0.477}$ | .96 | $1.40N^{-0.192}$ | .89 |
| Four step | $8.90N^{-0.477}$ | .96 | $1.36N^{-0.192}$ | .76 |
| Five step | $14.96N^{-0.477}$ | .93 | $1.63N^{-0.192}$ | .74 |

*Note.* Parameters were constrained to be positive. If zero, it was not included in the equation.
[a]No constraints. [b]$c$ constrained to be the same fit value across problem type. [c]$a$ and $c$ constrained to be the same fit value (which for $a$ was zero) across problem type.

one rule that should be applied at that point. Some participants flipped back and forth between the screen with the problem and the relevant rule screen, making sure the rule was applicable in that situation. Soon, however, participants did not refer back to the rules. When they did refer to the rules, it appeared to the experimenter that they paid almost sole attention to the example at the bottom of the screen and not to the text description or the schematic.

The protocol data, as the following discussion shows, are useful at shedding further light on the processes the participants were going through as they learned the task. For example, as people went through the experiment, they also began to reorder the parts of a problem they considered first. This was evident from their protocols (see the first sequence in Appendix B). However, many of the protocols were simply verbalizations of what the participants were clicking. As such, a rigorous analysis of them would prove not useful. We present the protocols in the following text and in Appendix B as representative of the more substantive comments that the participants made. The participants at the beginning of the experiment, as per the given rules, would move the operator and operand pairs from the left-hand side to the right-hand side and would then consider the operator in front of the $x$, if one existed. By the end of the problems, participants were first considering the operator in front of the $x$, if one existed, and then performing the transposing operations. This represents a major strategy shift, but one that makes solving the problems in one step easier, as can be seen in Appendix B. When not skipping steps, it is obviously of no use to worry about any operator in front of the $x$ until the very last step (Appendix B, Sequence 1, first problem). Even when participants began to skip all intermediate steps, they initially continued to consider the operator in front of $x$ last (Appendix B, Sequence 2, first problem). However, later on, all 11 of the participants who consistently skipped all intermediate steps looked to see if an operator existed in front of the $x$ (e.g., Sequence 1, second and third problems; Sequence 2, third problem—"I have to swap the ®s and ♥s," realizing there is a ♥ in front of the ℛ) before the end of the problem set.

Lastly, another difference between participants at the beginning of the experiment and those same participants at the end was the use of shortcuts. These shortcuts were also involved in making the problems easier to solve in one step. At the beginning of the experiment, of course, participants had no opportunity to make use of shortcuts because they were following the given rules. However, participants picked up a few shortcuts that aided in manipulating the character strings. Perhaps the most useful shortcut, and one that most participants acquired, helped in solving problems in which a minus sign was in front of the $x$. As an example, here is a problem with its worked-out solution:

$$-x + A = +B$$

$$-x + A - A = +B - A$$

$$-x = +B - A$$

$$x = -B + A.$$

If this was among the first of the problems presented to participants, they would transpose the $+A$ to the other side by using the inverse operator of plus, and when it came time to eliminate the minus sign from the $x$, the $-A$ would go back to what it was. Even when participants started doing these types of problems in one step, many would think of "double reversing" (4 participants said essentially that) the plus sign in front of the $A$. However, if this was near the end of the problem set, participants would see that there was a minus sign in front of the $x$, would know that the plus sign in front of the $A$ would stay the same on the other side, and so would just carry it over (9 participants explicitly mentioned this property). The example in Appendix B, Sequence 2 is typical of how participants progressed through this task.

Once participants started doing problems, they soon began to skip steps. Many participants began skipping the single-transformation step after only two or three problems. Most of the time when participants began to skip steps, they would just click out a line, skipping a step in the process (Appendix B, Sequence 2, Problems 1 and 2 contain an example of this). Occasionally (three times), participants would explicitly say before clicking out the line "I think I'll skip a step here," and in the next line they would skip a step. Also, sometimes participants would have a line partially clicked out and would then delete the symbols they had, saying, "Oh, I can skip this step" (this was explicitly said twice), and would then click out a new line, skipping the step they had previously started to click. This directly raises the issue of overt versus covert step skipping. On the basis of the protocols, it appears that when participants started to skip steps, they were merely doing all the same operations in their head and were only overtly skipping steps. Later on it appears that participants were actually changing the rule base and therefore mentally skipping steps as well, as evidenced by participants planning to copy directly a + over rather than going through a double reversal in their plan.

It would be interesting to graph the problems in which no step skipping occurred versus problems in which some of the intermediate steps were skipped versus problems in which all the intermediate steps were skipped, and then examine the best fitting power curve for each graph. In such a way, we could get a better handle on how these individual power curves conspired to produce the good fitting power curves seen in Figure 2. That is, it would be interesting to see if discontinuities exist between when people go from clicking out all of the steps to clicking out only some of them. Unfortunately, many of these graphs would have too few data points to properly fit a curve because participants began skipping some steps after only three or four exposures to the rule. The next experiment was designed such that people were explicitly told when to start step skipping so that such a comparison could be made. Also, in Experiment 2 we attempted to show that people can choose to initiate step skipping at any point in the experiment, even on the first trial. In contrast to some theoretical analyses, step skipping does not require first overtly performing the steps. When people start step skipping right away in this task it is highly plausible they are only overtly step skipping, having had no chance to rehearse skipping steps before beginning to solve problems. As these participants have never experienced

the results of the step combinations, it is hard to imagine that they have rules yet to produce the results directly.

## Experiment 2

In this experiment participants were told when they were allowed to start skipping steps. By exercising more control over participants' step-skipping behavior, a better analysis could be done of what transfers between making steps explicit and skipping steps. One group of participants was instructed to skip all intermediate steps immediately; whereas two other groups began the experiment clicking out all of the required steps. At different specified points in the experiment, these two groups were told when they had to begin skipping all intermediate steps. It was expected, on the basis of the participants in the first experiment, that participants in the first group, those who had to start overtly skipping steps immediately, would have a difficult time initially performing the task. By comparing across these three groups, inferences could be made as to whether the rule base that the participants used changed as the experiment progressed.

The task used for this experiment was almost identical to the one used in Experiment 1. However, this experiment contained no four- or five-step problems. A set of problems that took only one step, of the form $\oplus x = \oplus A \oplus B$, was added. These changes were made to ensure that all rules were used an equal number of times. One important feature of these materials is that, unlike Experiment 1, the final step contained the same number of symbols independent of number of steps skipped. This meant that when participants skipped steps, they were clicking out the same kind of line no matter how many steps were skipped.

### Method

*Participants.* Thirty-six Carnegie Mellon University undergraduates participated in this experiment for course credit and pay. Data from 6 other participants were collected but not used in the analyses. Four of these participants used the hint facility on almost every problem, and 2 participants made more than three times the number of errors made by the other participants. These participants were evenly distributed through the three groups used in this experiment.

*Materials.* As stated in the introduction, the task used for this experiment was similar to the task used in Experiment 1. One-step problems were added, however, and the four- and five-step problems removed. Because there were no four- or five-step problems, the two rules that dealt with precedence (Rules 10 and 11 in Appendix A) were no longer needed, so this task made use of only nine rules. Finally, Rule 9 describing how to remove the division sign in front of $x$ was also changed so that no matter what appeared on the right-hand side of the equation, the same procedure (switch the position of the two right-hand side operands) was applied.

*Procedure.* The procedure was similar to that of Experiment 1. These 36 participants, however, did not think aloud while solving the problems. The participants initially went through the nine rule screens and the screen of examples. Depending on which group the participant was placed in, the instructions differed. One group of participants was instructed that they must immediately skip all intermediate steps. That is, when given a problem, the first line they clicked out had to be the final line in the problem's solution. The other two groups were informed that to start with they must make each of the problem's steps explicit, and at some point during the experiment, they would be told when they had to start skipping steps. They were not told when that point would be within the problem set, but when that point arrived, a dialog box would appear on the screen saying that the participant must now skip all intermediate steps. For one of the two groups of participants, this dialog box appeared after they had completed 24 problems, and for the other group after 96 problems (halfway through the problem set). Twelve participants were in each of the three groups. Participants were allowed to take a break whenever they needed. Only 2 participants took a break, and no participant complained about the length of the experiment.

### Results

Table 5 summarizes the amount of time participants spent on the experiment in each of the three groups. Time spent reading the instructions did not differ significantly among the three groups, $F(2, 33) = 0.87$, $MSE = 9.14$, $p > .1$. As to be expected, though, the time spent solving the problems did differ between the three groups, $F(2, 33) = 21.76$, $MSE = 66.48$, $p < .01$. Participants who could not start skipping steps until after 96 problems had to click out at least 288 lines; whereas participants in the other groups had to click out at least 192 (skip immediately) or 216 lines (skip-after-24 problems). Even though participants who skipped steps immediately took a little longer per problem initially (see Figures 9 and 10), the number of lines required of the skip-after-96 group caused them to take longer in completing the experiment.

We also did an analysis of variance of how often participants entered incorrect lines. An error was when the participant clicked the checkmark button, but the current line was not a valid line. Participants in the three groups made statistically the same number of total errors, $F(2, 33) = 1.39$, $MSE = 2.65$, $p > .1$. Even though participants in the skip-after-96 group had to click out more lines, we still expected participants who had to start skipping steps immediately to do worse, on the basis of our observations in Experiment 1. Figure 7 graphs the probability of a participant making an error on a line for each type of problem as a function of problem block, which was split into each of the three groups. Participants improved over the experiment, $F(7, 231) = 25.61$, $MSE = 0.62$, $p < .001$. Not surprisingly, participants made more errors on the problems with more steps, $F(2, 66) = 21.15$, $MSE = 1.07$, $p < .01$. The data were a bit erratic, but there was a significant interaction between problem block and group, $F(14, 231) = 2.11$, $MSE = 0.62$, $p < .01$, which might be interpreted as participants making more errors when they first had to skip steps in the skip-immediate and skip-after-24 group. There was also a significant two-way interaction between problem block and

Table 5
*Mean Time Spent Reading Instructions and Solving Problems (Experiment 2)*

| Step | Reading instructions[a] (min) | Solving problems (min) |
|---|---|---|
| Skip immediately | 10.1 | 38.2 |
| Skip after 24 problems | 9.5 | 44.1 |
| Skip after 96 problems | 11.1 | 59.5 |

[a]Thirteen screens.

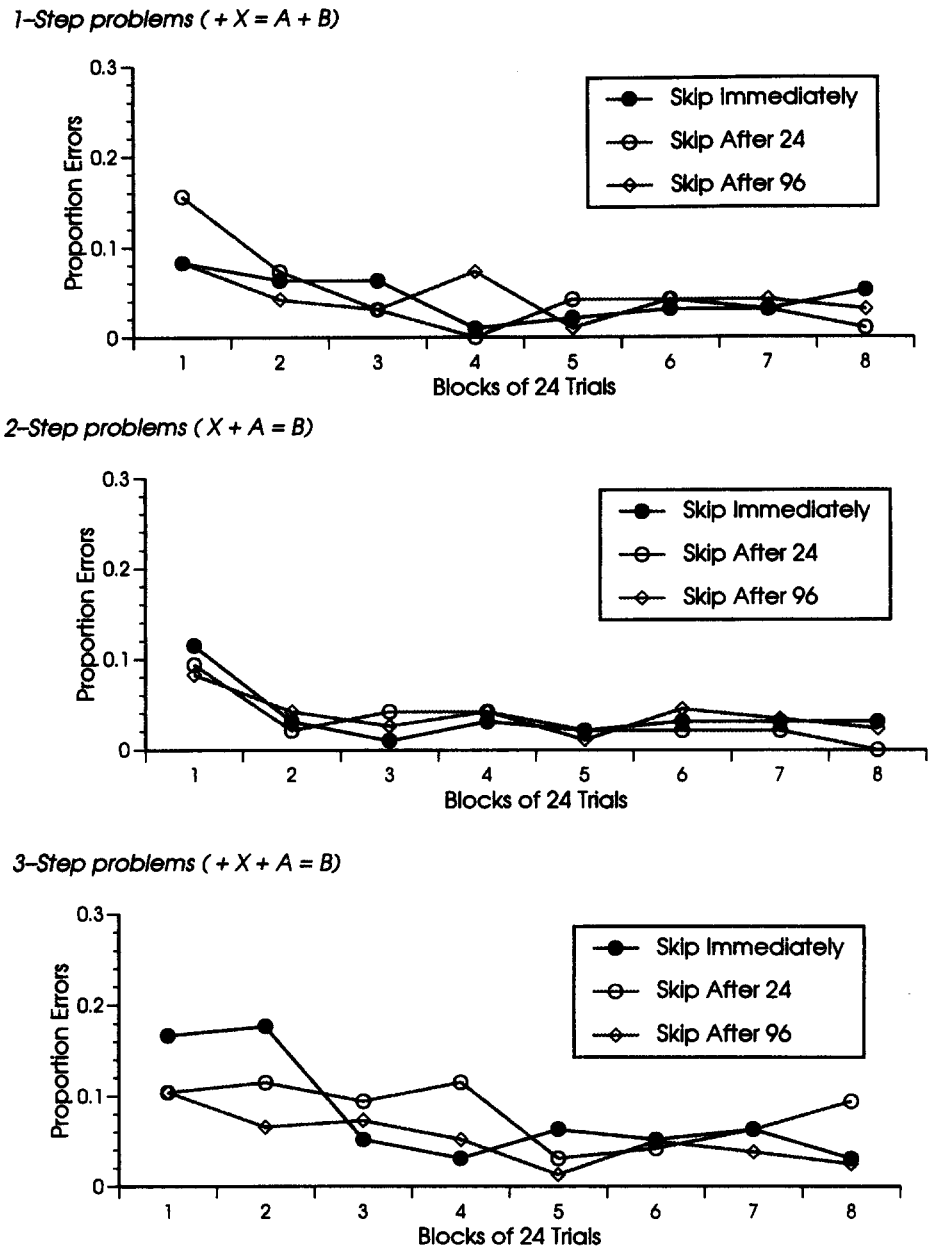


*Figure 7.* Probability of making an error by block for each problem type (Experiment 2).

number of steps, $F(14, 462) = 3.16$, $MSE = 0.56$, $p < .001$, which could be interpreted as the participants improving less rapidly in the three-step condition. There were no other significant effects.

As in Figure 2 for Experiment 1, Figure 8 graphs the total time participants spent on a problem of a particular type by block. Only the curves for the two groups who had to do the intermediate steps before beginning to skip them are shown. Unlike Figure 2 for Experiment 1, there was a discontinuity for these participants who started skipping steps after problem 96 at the point where they were required to skip steps (marked by a vertical line in the graphs). The participants who skipped after 24 problems were in such a steep part of their learning curve that the discontinuity was less apparent, particularly with the trials blocked. In Experiment 1, the different times when participants started skipping steps masked such discontinuities. In fitting the curves shown in Figure 8, we made the assumption that once participants were required to skip steps, only the asymptote (the *a* parameter) would change, but not the multiplicative factor (the *b* parameter) or the learning rate (the *c* parameter). We also made the assumptions that the learning rate did not differ at all between the curves and that the different two-step and three-step problems had the same *a* and *b* parameters (these assumptions are similar to the most

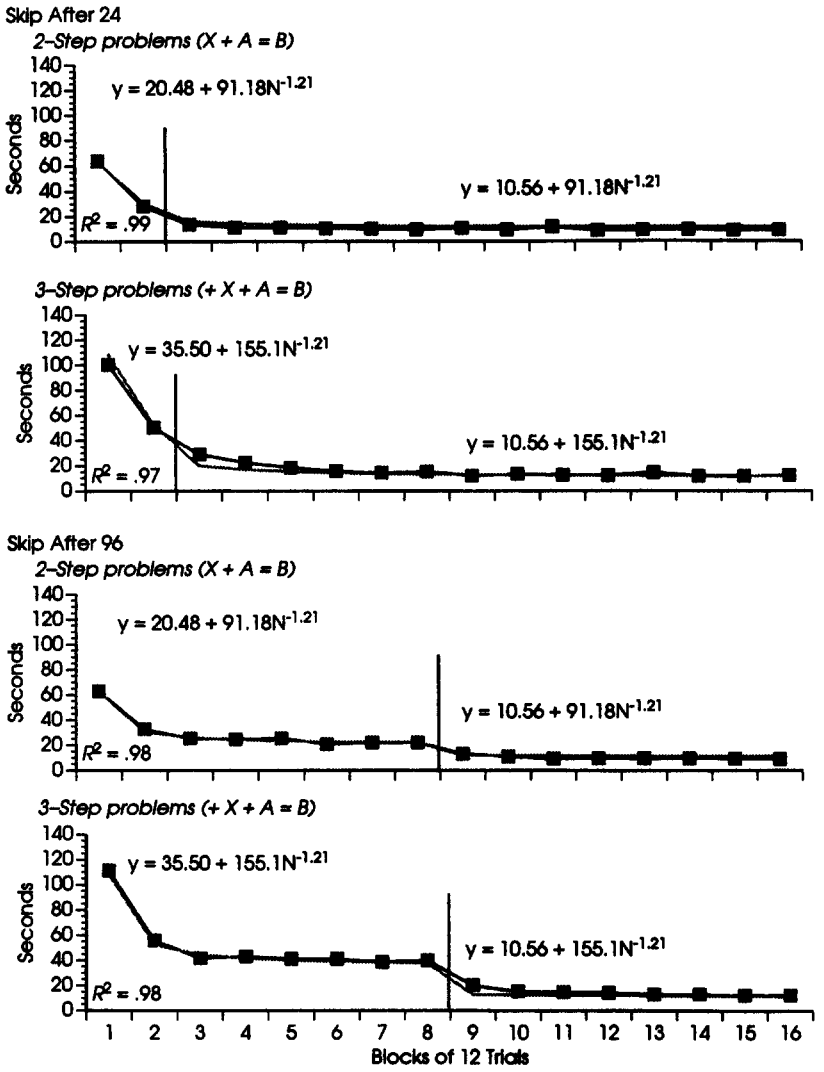


*Figure 8.* Overall time by block for each problem type (Experiment 2).

constrained model in Table 4). This resulted in a model with a total of six free parameters. When participants started to skip steps, then, they were essentially on the same learning curve, but the point to which they reached asymptote had been shifted down. This shift down corresponds to the decrease in the number of symbols that needed to be clicked out. The best fitting curves and their equations are also shown in Figure 8. This produced a better fit than not varying the $a$ parameter at all over blocks, $F(1, 58) = 42.05$, $MSE = 235.80$, $p < .01$. Although this gave a fairly satisfactory global fit, as in Experiment 1, we looked at the timing of each symbol to get at the microstructure of the skill being learned.

On the basis of the observation from Experiment 1 that most of a participant's time was spent planning the first symbol, we split participants' time solving the problems of this experiment into two categories: planning time and execution time. A participant's planning time was the time spent before clicking out the first symbol of a line, and execution time was the rest of the time spent on that line. Figures 9 and 10 graph the execution and planning time, respectively, that participants spent on a per character basis on each type of problem, which was broken into the three groups.

Concerning participants' execution time, there was no difference between the three groups, $F(2, 33) = 0.09$, $MSE = 1.34$, $p > .1$. Participants did spend differing amounts of time executing the different problem types, $F(2, 66) = 40.18$, $MSE = 0.12$, $p < .01$, and got faster as they went through the experiment, $F(7, 231) = 75.97$, $MSE = 0.14$, $p < .01$. A multiple-comparison test showed that one- and two-step problems did not differ, but both differed from three-step problems. This could be interpreted as some planning time seeping into the execution time (or alternatively, it could amount to some concurrency of the two processes). There were also significant interactions between problem block and group, $F(14, 231) = 2.00$, $MSE = 0.14$, $p < .05$, and problem block, group, and number of steps $F(28, 462) = 2.36$, $MSE = 0.06$, $p < .001$. These interactions could be interpreted as participants taking longer when they have to skip steps, particularly for three-step

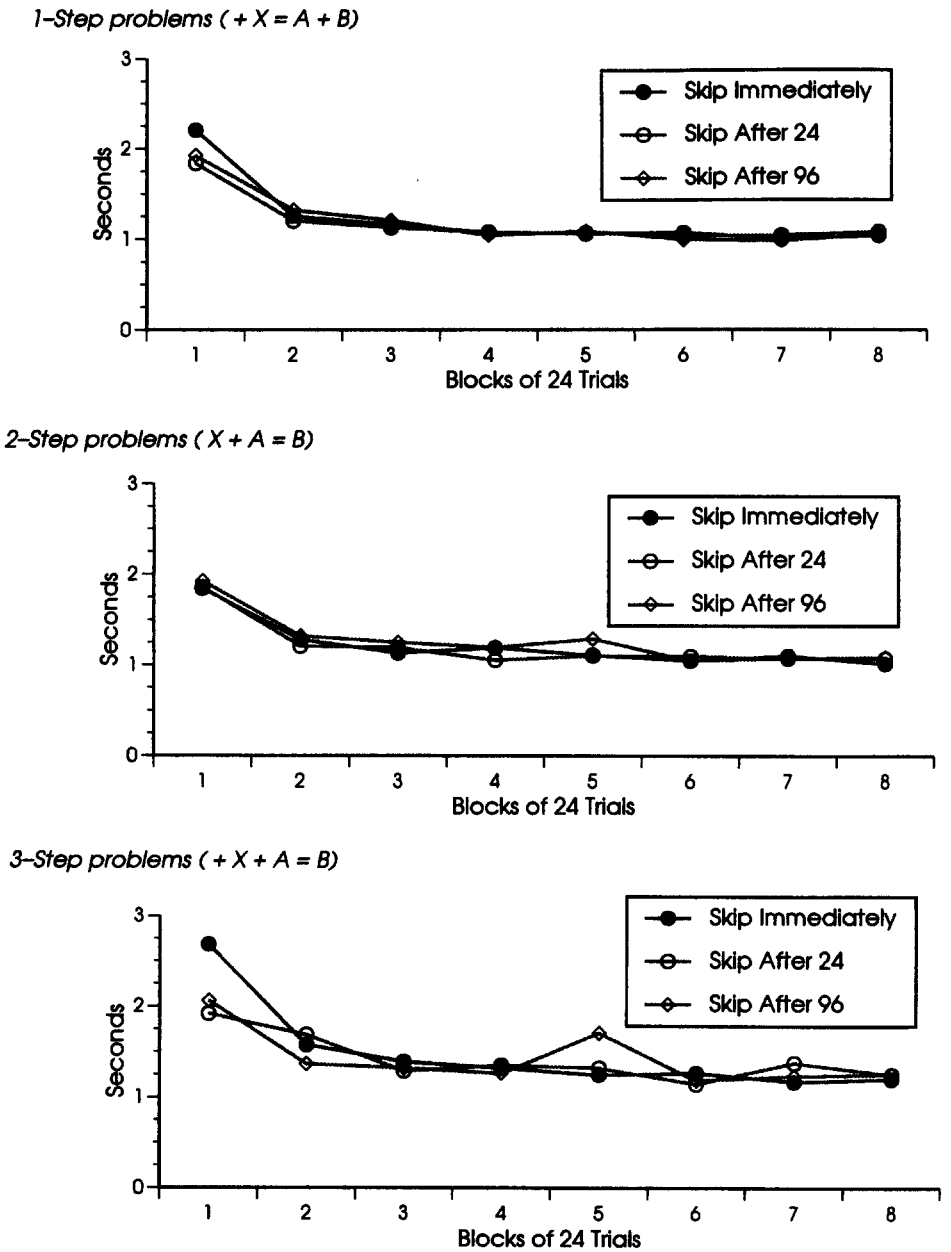


*Figure 9.* Execution time by block for each problem type (Experiment 2).

problems at the beginning of the experiment. No other interactions were significant.

Looking at planning time, a significant difference was detected between the groups, $F(2, 33) = 4.22$, $MSE = 32.77$, $p < .05$. As in execution time, participants also differed in planning the different problem types, $F(2, 66) = 41.85$, $MSE = 4.88$, $p < .01$, and got faster at planning as they went through the experiment, $F(7, 231) = 41.52$, $MSE = 2.70$, $p < .01$. All interactions in planning time were highly significant—group by number of steps, $F(4, 66) = 14.42$, $MSE = 4.88$, $p < .001$, group by problem block, $F(14, 231) = 34.42$, $MSE = 2.70$, $p < .001$—number of steps by problem block, $F(14, 462) = 13.43$, $MSE = 1.67$, $p < .001$, and group by problem block by number of steps, $F(28, 462) = 18.46$, $MSE = 1.67$, $p < .001$. All of these can be interpreted as participants being slowed when they first have to skip steps, particularly in the three-step condition. This increase in planning time is evidence for covert execution of the steps that participants were now being asked to skip overtly. The participants who started skipping steps after 24 problems had longer latencies, in comparison to the group who skipped steps immediately, during the first block of trials in which they skipped steps, but in the next block of trials they were performing the same. Looking closer at the planning time for the three-step problems solved in the last 96 problems, the

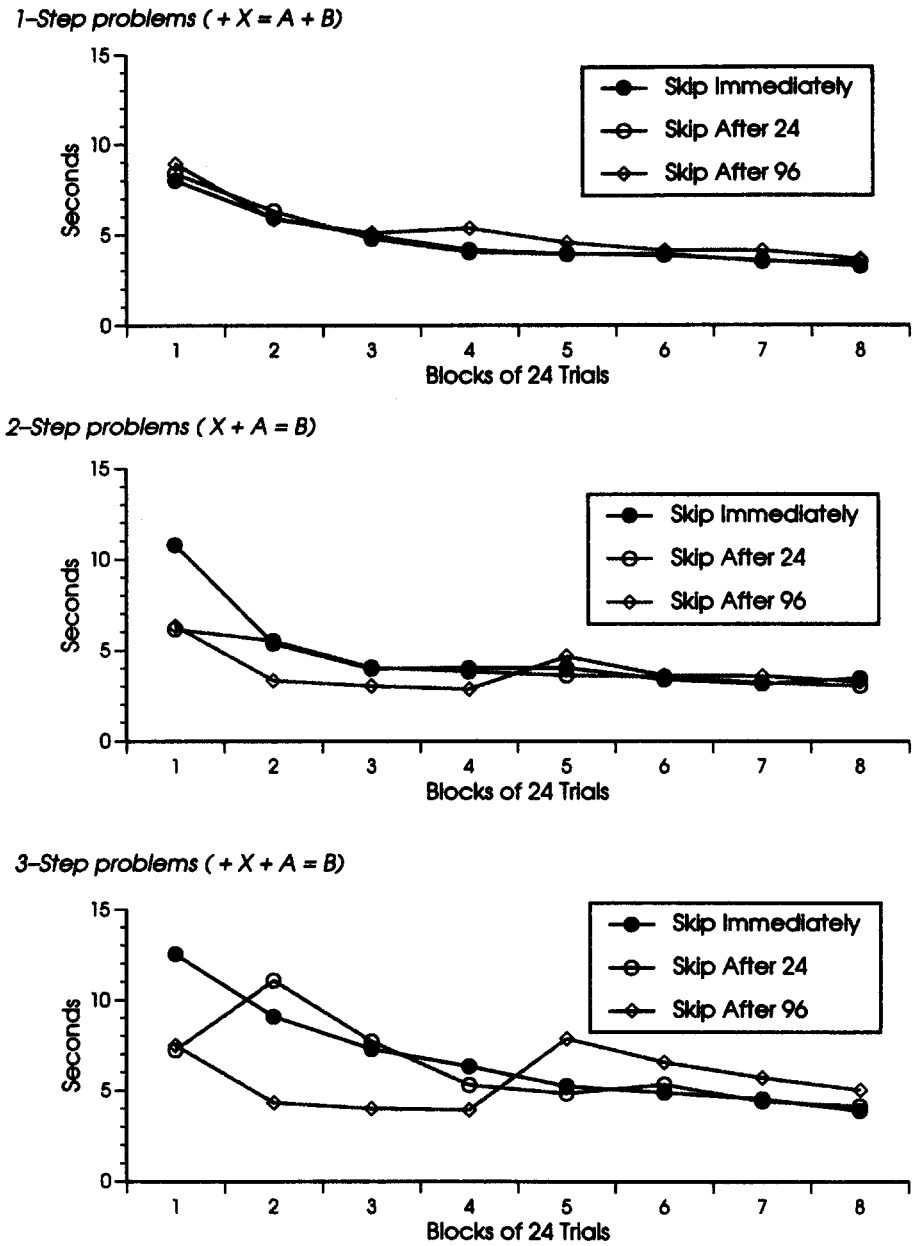


*Figure 10.* Planning time by block for each problem type (Experiment 2).

times were significantly different from one another, $F(2, 33) = 3.52, MSE = 11.96, p < .05$. The line in Figure 10 representing the group of participants who did not skip steps until after Problem 96 is significantly different from the lines representing the other two groups by a Duncan multiple range test ($p < .05$), meaning that these participants never reached the performance level of the other two groups.

One of the striking results in Figure 10 is that, particularly for participants who skipped immediately, the differences in planning time among the one-, two-, and three-step problems disappeared over the course of the experiment. In the first block of 24 trials these participants were averaging 8.01, 10.78, and 12.52 s for the one-, two-, and three-step problems, respectively, whereas at the end of the experiment they averaged 3.23, 3.45, and 3.88 s, respectively.[2] It seems that, while participants were going through multiple transformations in planning these steps initially, or overtly skipping steps,

[2] The comparable means for the first block of data from the other two groups were 8.38, 6.18, and 7.49 for the participants who started skipping after Problem 24 and 8.93, 6.34, and 7.49 for the participants who started skipping after Problem 96. The comparable means for the last block of data from the other two groups were 3.46, 3.02, and 4.13 for the participants who started skipping after Problem 24 and 3.62, 3.25, and 5.02 for the participants who started skipping after Problem 96.

they wound up planning all steps as single transformations, or covertly skipping steps. Protocols collected from 6 additional participants; 2 in each of the three groups, confirmed such a hypothesis. Like participants in Experiment 1, once they had to start overtly skipping steps, they went through each individual step mentally before clicking out the final line. After they had experience doing this, their protocols suggested, such as through the use of the shortcut mentioned previously, that they began to mentally skip steps as well.

We fit a simple mathematical model to the planning and execution times to more fully explain these findings. It assumes that planning of single steps and execution of steps speed up from the beginning of the experiment according to a power function. As in the final model fit in Table 4, we assumed separate parameters for $a$, $b$, and $c$ for execution and planning, but these were constant across problem types (and as in Table 4, these constraints have the same implications). The model for execution and one-step planning is simply

$$a + bN^c, \tag{1}$$

where $N$ is the number of blocks. However, the model is more complex in the case of planning multiple transformations. The model assumes that when participants are first asked to do a multistep problem in a single step they mentally go through all of the steps. That is, at first they are only overtly skipping but are covertly still planning each line. However, each time that they do so there is a probability of forming a collapsed operator that will do it in a single step (and therefore covertly skipping as well). Thus, if a participant has been required to solve a two- or three-step problem in a single step $M$ times, then the probability of still planning it in three steps is $P^M$ and the probability of mentally planning it in one step is $1 - P^M$, where $P$ is the probability of not collapsing a step. The latency for both planning and execution time is described by

$$a + (\text{no. of steps})^* P^M bN^c + (1 - P^M)bM^c, \tag{2}$$

where $N$ is the total number of blocks, and $M$ is the number of blocks in which the participant had to skip steps. The best fitting parameters, found by minimizing the sum-squared error, are displayed in Table 6. The overall $R^2$ for this model was .941. Figure 11 presents the planning predictions, which is to be compared with Figure 10. Note that the model predicts a longer planning time when required to start skipping steps, just as the participants actually did. We estimated two values of $P$ for this model: .963 for two-step problems and .992 for three-step problems. This difference between the $P$ values corresponds to the relatively less difficulty participants had in adapting to the skipping instructions for two-step problems. There are at least two possible factors to point to in understanding this difference. First, participants only have to compose two steps rather than three. Second, the two-step composition seems more natural in that it is very like algebraic movement rules and skips over an intermediate step.

Table 6
*Best Fitting Parameters for Planning and Execution Time (Experiment 2)*

| Time | Parameter | | |
|---|---|---|---|
| | *a* | *b* | *c* |
| Execution | 0.90 | 2.60 | −0.337 |
| Planning | 2.35 | 15.13 | −0.495 |

*Note.* $P_2 = .963$ represents value for the two-step problem; $P_3 = .992$ represents value for the three-step problem.

## General Discussion

Taken together, these two experiments provided a close examination of the step-skipping process, both overt step skipping and covert step skipping. The results of Experiment 1 supply a qualitative overview of how people begin to skip steps when left essentially on their own to learn how to solve problems. At first, the participants made all of the intermediate steps explicit. That is, they would apply the one necessary rule to the current line to arrive at the next line of the problem's solution. Only after participants applied each rule at least a couple of times did they begin overtly skipping that rule, and only later covertly skipping the rule. It was almost never the case that participants skipped a rule on the first opportunity they had to apply it. However, Experiment 2 indicated that overt skipping was a somewhat arbitrary choice on the participant's part, and they were capable of skipping right away. The experiment also indicated that their decision to skip brought a substantial reduction in total time due to the reduced execution time. Participants were still covertly planning each step, and it took longer for these covert planning steps to disappear. However, there was little evidence for multistep planning after the 192 trials.

The results of Experiment 1 demonstrate that even though participants showed an overall continuous improvement in performance (Figure 2), and the performance was consistent with the power law of learning, this masked large qualitative differences, such as step skipping (Figure 3). However, in Experiment 2, in which we controlled when participants were allowed to skip steps, discontinuities were observed (Figure 8). The separate parts of the discontinuities, though, were well fit by a power curve. Moreover, the clicking of individual symbols still followed a power law across the course of the experiments. Thus, it seems that the power law of learning is preserved at a more microscopic level.

It could be argued that discontinuities would not be observed when participants are not forced to skip steps, as in Experiment 1, because the two strategies (to skip or not to skip steps) are in competition with one another, as in a race model, and the faster of the two controls performance. When skipping steps becomes faster than not skipping steps, then that will be what the person does, without a discontinuity in performance.

Although other formalizations of how participants represent transformations are possible, we consider here and in the model to follow a production system framework. An important issue raised in these experiments was the grain size at which the participants represent these rules as productions. Participants could break up a rule into multiple productions for transforming individual characters, or they could have a single production rule that transformed a whole line mentally fol-

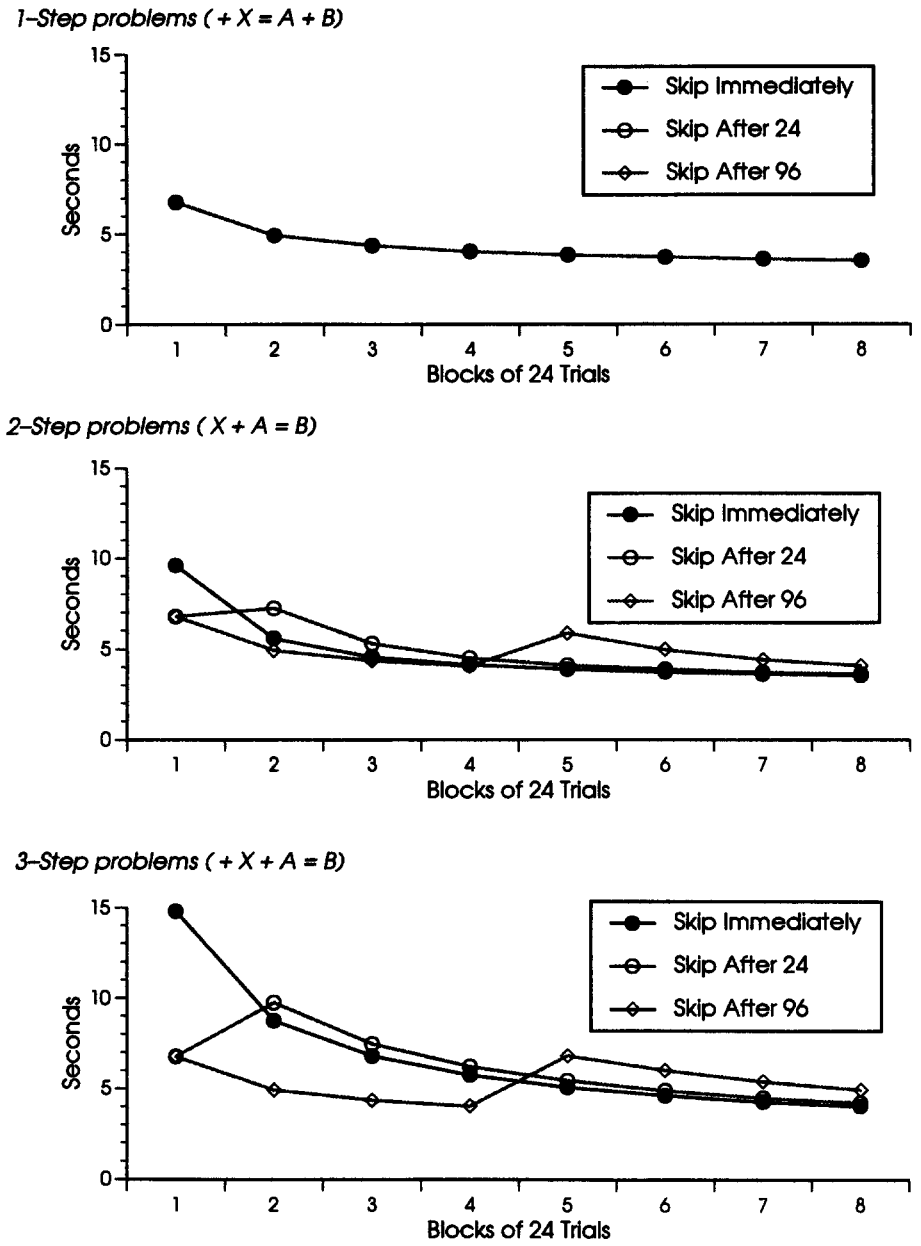


*Figure 11.* Predicted planning time by block for each problem type (Experiment 2).

lowed by a series of execution productions that clicked out each character. Evidence from these two experiments favors the latter possibility—that is, planning productions that apply at the line level. The longest latency for a line is before the first keystroke. This would be the time for the productions to fire that planned the line to be clicked. The time between all of the other keystrokes varied only a little as a function of condition. Once the production matches and fires, the next line is then ready to be clicked out, which would be handled by a set of execution productions that click out individual characters. While participants' planning time was greatly increased when they were instructed to skip steps, their execution times were almost unaffected.

### *An ACT–R Model*

We developed an ACT–R model (Anderson, 1993) that captures the important qualitative aspects of people performing this task (in particular, the processes discussed in Experiment 1's *Discussion* section). Appendix C contains a more detailed account of the model, particularly of the analogy mechanism. An important distinction within the ACT–R architecture is between declarative knowledge, one's knowledge of facts (e.g., Washington DC is the capital of the United States), and procedural knowledge, one's knowledge of how to perform actions (e.g., adding numbers together). One of the claims of the ACT–R theory is that all knowledge has declara-

tive origins. That is, the only way new procedural knowledge, in the form of production rules, enters the system is by the process of analogizing to some previous declarative knowledge. This mechanism operates by forming an analogy from examples stored in declarative memory to the current goal.

The model of our task within ACT–R (Anderson, 1993), therefore, makes use of analogy to examples and creates production rules as a result of the analogy process (either the ones that do the original one-step transformations or the later ones that skip steps). This fits in nicely with the participants' behaviors because in the early stages of performing the task, they would refer to the rule screens and look specifically at the example at the bottom of the screen. Also, participants would try to recall what they did last time when faced with a similar configuration of symbols. Thus, participants were relying on examples to start out with, just as ACT–R predicts.

In formulating this model, and in keeping with the analyses previously presented, we made the assumption that each step in a problem's solution could be conceived of as the application of essentially two kinds of operators: those that planned the step and those that executed the step. Whereas there would be different operators for planning different lines, which would roughly correspond to the rules on the rule cards, the system would use the same set of execution operators for each step. That is, an operator would decide what string of symbols to output for the next step (i.e., what rule to apply), and a set of execution operators would produce that string. Our model concerned itself with the planning operators.

To begin the simulation, the model has some examples available to it, such as those found at the bottom of the rule screens as well as some other declarative knowledge about the operators and operands involved. To illustrate how ACT–R (Anderson, 1993) proceeds, suppose that for its current goal it has been encoded to solve the problem $x + A = B$. It has no productions that apply. However, one of the examples that it has, that the next line after $x * C = D$ is $x * C/C = D/C$, is similar to that of the current problem. It also has the knowledge that / is the inverse operator of * and that − is the inverse operator of +. On the basis of that knowledge and the example given, the ACT–R analogy mechanism produces the following production (P):

IF the current goal is an equation of the form: (P1)
*x op1 CON1 = CON2*
and *op2* is the inverse of *op1*,
THEN the next step should be
*x op1 CON1 op2 CON1 = CON2 op2 CON1.*

Some of the mapping that the analogy mechanism does is relatively easy (e.g., the $x$ in the first line goes with the $x$ in the second line), but it does infer (by tracing through the working memory elements that contain *) that the / appears because of the knowledge that / is the inverse of *. After this production is constructed, it applies to the current goal to produce the result $x + A - A = B - A$. This result becomes the new goal.

Again, no productions apply to this new goal. However, the model has another example available to it, that the line that follows $x * C/C = D/C$ is $x = D/C$. Similar to last time, with this example and its other knowledge, the analogy mechanism creates this production

IF the current goal is an equation of the form: (P2)
*x op1 CON1 op2 CON1 = CON2 op2 CON1*
and *op2* is the inverse of *op1*,
THEN the next step should be
*x CON2 op2 CON1.*

After firing, this production creates the final result, $x = B - A$. The model now has two productions that transform lines and a complete, worked-out solution of the form

$$x + A = B$$

$$x + A - A = B - A$$

$$x = B - A.$$

Once a production has been created to do a step, the person now has two bases for performing the task: either by reference to the production or by analogy to an example. This competition between production and example will be a function of the strength of the production and the activation of the example. The production will acquire strength with practice and gradually come to dominate the competition. This is consistent with the fact that in Experiment 1 participants did not stop referring to examples after their first opportunity. Presumably, this is because the resulting production was not yet strong enough and needed to be recreated by further analogy.

Let us now consider what may happen when the simulation is presented with a new, but similar, problem to solve (e.g., $x/B = D$). The simulation has one production that will apply, Production 1, but its strength will be low. It also has two declarative structures to which it can analogize: the original example and the declarative trace left from solving the first problem. The production's strength is low, and the activation from the example of the previous problem will be high, so the analogy mechanism will create a production that is identical to Production 1. This identity will be noted, and instead of creating a new production, the original production will gain strength. A similar process will occur to solve the next, final line of the problem.

The question now arises as to how the model will begin to overtly skip steps (as the participants did in the first experiment). Because it is quicker to solve problems by clicking out fewer lines, it is advantageous to skip steps, but the cost is an increased memory load because the person has to maintain a declarative structure representing the current line while retrieving and applying the next rule. Once participants, and the model, have gained a familiarity with the declarative structures that represent the task's problem space (or the experimenter dictates that intermediate steps must now be skipped), instead of clicking out intermediate steps, the intermediate steps can be maintained in working memory and the appropriate rules applied mentally (to decrease memory load) and some steps would be skipped. This corresponds to only overt step skipping. That is, the model, like the participants when they started to skip steps, still goes through the intermediate steps but does not click them out.

Once steps are overtly skipped, the solver will see on the screen a pair of lines with the step skipped and can store that

pair in declarative memory to be used to form an analogy. In other words, once a solver or the model solves $x/B = D$ by mentally applying analogized productions similar to Production 1 and Production 2 and clicking out only $x = D * B$, a declarative structure will be formed linking $x/B = D$ with its solution, $x = D * B$. In that way, the system can analogize from a current problem to that new fact and produce the following production that skips a step:

IF the current goal is an equation of the form: (P3)
  *x op1 CON1 = CON2*
  and *op2* is the inverse of *op1*,
THEN the next step should be:
  *x = CON2 op2 CON1*.

Eventually, this production will accrue enough strength (by being generated repeatedly by the analogy mechanism) that it will be more powerful than the declarative examples. Once this happens, the model, by executing this production, will be covertly skipping steps.

In summary, all of the production rules start out by analogy to examples. The examples are either the line transformations on the rule screens or a line transformation it creates while solving the problem. Once ACT–R starts overtly skipping steps, it creates new examples of line transformations that contain skipped steps from which it can learn new rules that correspond to macrooperators. The new rules it learns gradually accrue strength and come to replace analogy to examples, which corresponds to covertly skipping steps. It is important to note that the rules created operate on whole lines and return whole lines as their answer. This corresponds to the result in both experiments that participants spent most of their time before clicking out the first symbol and then a relatively similar time clicking out the rest of the symbols of a line. That is, what the participants apparently did, and what the ACT–R model (Anderson, 1993) does, is to figure out what to do with the entire line and then click out the result.

## *Conclusions*

One question of interest is if any person noticed the relationship between this task and algebra. This can be addressed best with respect to the 12 participants in Experiment 1 from whom protocols were obtained. Only 1 participant in Experiment 1 reported being aware of the similarity between this task and algebra. Another participant after working on 60 problems in Experiment 1 began to make a mapping between the symbols in this task and algebraic symbols but did not complete the mapping. When questioned after the experiment, the participant reported thinking there was similarity but did not believe the mapping was going to provide any help in performing the task, so he abandoned it. Almost all participants in both experiments, when questioned after their participation, reported a similarity between this task and a logic type of task. From this finding, it appears that participants were not directly using their knowledge of algebra to perform the task.

This indirectly leads to the question of generality of these findings. What kinds of tasks lead to the sort of step skipping exhibited in the task used in these experiments? Step skipping is not possible, or simply is not done, in all tasks. Counting and simple addition represent two such tasks. A person counting cannot skip objects to be counted.[3] Children learning addition go directly from using some counting strategy to direct retrieval of the example (Siegler, 1986); they do not go from counting by ones to counting by twos to counting by fours. Retrieval of the answer can be conceived as a special kind of step skipping, but it is one that eliminates the algorithmic nature of the process. In our case, we argue that participants were not retrieving the final line but were still computing it (see below for more detail). Certainly, the kinds of tasks that support the type of step skipping discussed in this article are typically formal, procedurally oriented domains, such as algebra or physics. In such domains, steps are characterized by the application of a rule to the current state of the problem to reach the next meaningful state on the path to solution. Steps can be skipped by simply skipping the intermediate steps between the problem statement and its solution. This can be done by recognizing any sort of pattern that may be contained in the problem statements directly to reach the answer or some other intermediate step. However, it is the case that other less formal procedures support step skipping. For instance, students who are learning to draw human figures do so by drawing an oval for the head and then seven more for the other parts of the body. They then put hash marks on the ovals to serve as guides for such things as shoulders and hips. As the person becomes a more expert drawer, they begin to skip the steps of drawing certain ovals and the hash marks. This skipping of steps coincides with a more intuitive understanding of how to draw the human figure.

The ACT–R theory (Anderson, 1993) claims that any new procedural knowledge comes from declarative knowledge in the form of examples. The theory posits only a single mechanism to account for both the learning of the original rules and of the composed rules. Participants' behavior while performing this task is consistent with such a view (in how they referred to the examples and by their protocols), and an ACT–R simulation was created that modeled many of the important aspects of participant's performance. That is, both the participants and the model initially did each step in the solution procedure (when left on their own, as in Experiment 1) before skipping steps. In doing each individual step, the participants and model formed an analogy from an existing example to the current problem. When beginning to skip steps, they were most likely applying each separate operator mentally in order to skip a step. After noticing the relationship between these new pairs of lines, though, they formed by analogy a new, single operator that skips steps.

The other theories of skill acquisition also can be examined in light of these results. The rule composition view might seem to have the same problem with these data that Lewis (1981) raised. That is, people are composing together the planning steps, but these are not adjacent—for instance, the double reversals of the operators are separated at least by execution

[3] Subitizing (Chi & Klahr, 1975) could perhaps be construed as a case of skipping steps while counting, and indeed, people can be trained to enumerate recognized, repeated patterns (Lassaline & Logan, 1993). However, subitizing does not have the flavor of collapsing a sequence of different procedures into a single operation.

steps. However, the model we described proposed that people consciously reorganized their problem solving so that they went through a series of planning steps mentally before executing the results. Then the planning steps would be adjacent and composition could produce macrooperators. The remaining difficulty for the composition theory is that it does not deal with the original learning of the operators and the evidence for participant's reliance on examples.

Logan's (1988, 1990) instance theory as it is currently realized cannot adequately account for these data because at most only one or two examples were ever repeated for a participant, yet his or her performance still improved over the course of the experiment. Furthermore, once participants started to produce $x = B - A$ from $x + A = B$, they also would produce $x = C * D$ from $x/D = C$. That is, once they started to do single transformations in one step, they would do so for all operators. This generalization phenomenon poses a major problem to Logan's (1988, 1990) theory.

Modifications to instance theory could perhaps be made to explain the data in these experiments. For example, the generalization problem mentioned above could perhaps be solved by making the instances used by the problem solver smaller grained. That is, instead of using whole problems, or even whole lines, an instance could be only part of a line (an operator–operand pair perhaps). In that way, these instances could be pieced together by multiple retrievals to form a whole line. However, if that were the case, one would not expect to see the relatively large up-front planning time that was seen in these experiments. Rather, the expectation would be more consistent intercharacter times as the smaller grained instances were recalled. Another potential way to mend instance theory would be to allow instances to accept variables and thus allow nonexact retrieval matching to occur. This is perhaps a better solution, but it would seem to be basically introducing production rules into the theory, which would take away its distinct character. Moreover, in addressing the question of how such instances are variabilized, one comes against the same issues that ACT–R's (Anderson, 1993) analogy mechanism was designed to solve.

In these experiments we have examined closely the power law of learning. At a gross level, as in Figure 2 from Experiment 1, our results were fit quite nicely by a power law. However, as we looked more deeply at people's performance at learning our task, we found discrete qualitative changes. Because these changes occurred at different points for different participants, the average result looked like a continuous power function. When we controlled the point of these qualitative changes (Experiment 2), we noticed discontinuities in the power law in terms of overall time. However, when we decomposed this overall time into its components, we found that these components still obeyed a power law. In summary, time to perform a complex task approximates a power law because its components do, and qualitative changes in these components are typically masked by averaging procedures.

## References


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

Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, NJ: Erlbaum.

Anderson, J. R., & Fincham, J. M. (1994). Acquisition of procedural skills from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 20,* 1322–1340.

Carlson, R. A., & Lundy, D. H. (1992). Consistency and restructuring in learning cognitive procedural sequences. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 18,* 127–141.

Charness, N., & Campbell, J. I. D. (1988). Acquiring skill at mental calculation in adulthood: A task decomposition. *Journal of Experimental Psychology: General, 117,* 115–129.

Chi, M. T. H., & Klahr, D. (1975). Span and rate of apprehension in children and adults. *Journal of Experimental Child Psychology, 19,* 434–439.

Ericsson, K. A., & Simon, H. A. (1990). *Protocol analysis: Verbal reports as data.* Cambridge, MA: MIT Press.

Frensch, P. A. (1991). Transfer of composed knowledge in a multistep serial task. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 17,* 997–1016.

Frensch, P. A., & Geary, D. C. (1993). Effects of practice on component processes in complex mental addition. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 19,* 433–456.

Hypercard 2.1 [Computer software]. (1991). Cupertino, CA: Apple Computer.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14(4),* 511–550.

Lassaline, M. E., & Logan, G. D. (1993). Memory-based automaticity in the discrimination of visual numerosity. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 19,* 561–581.

Lewis, C. H. (1978). *Production system models of practice effects. Dissertation Abstracts International, 39,* 5105B. (University Microfilms No. 79-07, 120).

Lewis, C. H. (1981). Skill in algebra. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 85–110). Hillsdale, NJ: Erlbaum.

Logan, G. D. (1988). Toward an instance theory of automatization. *Psychological Review, 95,* 492–527.

Logan, G. D. (1990). Repetition priming and automaticity: Common underlying mechanisms? *Cognitive Psychology, 22,* 1–35.

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

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 1–55). Hillsdale, NJ: Erlbaum.

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

Rosenbloom, P. S., & Newell, A. (1986). The chunking of goal hierarchies: A generalized model of practice. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 123–140). Los Altos, CA: Morgan Kaufmann.

Siegler, R. S. (1986). Unities in strategy choices across domains. In M. Perlmutter (Ed.), *Minnesota Symposium on Child Development.* (Vol. 19, pp. 1–48). Hillsdale, NJ: Erlbaum.

# Appendix A

## Rules Used in Experiment 1

For Rules 2–5, participants were told that *X* represents any of the operand symbols, Δ, Γ, Ω, Φ (in the experiment they were called *object symbols*), and that the goal of each problem was to "isolate" the $\mathscr{P}$ on the left-hand side of the ↔.

### Rule 1

This rule allows you to take any connector symbol (©, ®, #, ♥) plus any object symbol (Δ, Γ, Ω, Φ) and add them to both the left-hand side and the right-hand side of any character string. For example, if the current line was:

$$\mathscr{P}\#\Phi\leftrightarrow{}^{\circledR}\Delta$$

You could add ©Φ to both sides to get:

$$\mathscr{P}\#\Phi^{\copyright}\Phi\leftrightarrow{}^{\circledR}\Delta^{\copyright}\Phi$$

Suppose the current line was $\heartsuit\mathscr{P}{}^{\circledR}\Delta\leftrightarrow\#\Omega$ and you wanted to use Rule 1 with ♥Δ for the next line. That line would look like:

$$\heartsuit\mathscr{P}{}^{\circledR}\Delta\heartsuit\Delta\leftrightarrow\#\Omega\sigma\Delta$$

### Rule 2

Schematically, Rule 2 states:

The current line (on one side) contains: [. . .]#X[. . .]©X
You may write: [. . .][. . .]

So, for Rule 2 to apply, the character string must contain the connector # and some object symbol together, and then the connector © and that same object symbol later on in the character string, but still on the same side.

To illustrate:

The current string is: $\mathscr{P}\#\Delta^{\copyright}\Delta\leftrightarrow{}^{\circledR}\Omega^{\copyright}\Delta$
You may rewrite it as: $\mathscr{P}\leftrightarrow{}^{\circledR}\Omega^{\copyright}\Delta$

### Rule 3

Schematically, Rule 3 states:

The current line (on one side) contains: [. . .]©X[. . .]#X
You may write: [. . .][. . .]

For Rule 3 to apply, the character string must contain the connector © and some object symbol together, and then the connector # and that same object symbol later on in the character string, but still on the same side (either the right-hand side or the left-hand side).

As an example:

The current string is: $^{\circledR}\mathscr{P}\,{}^{\copyright}\Delta\#\Delta\leftrightarrow\heartsuit\Phi\#\Omega$
You may rewrite it as: $^{\circledR}\mathscr{P}\leftrightarrow\heartsuit\Phi\#\Omega$

### Rule 4

Rule 4 states

The current line (on one side) contains: [. . .]®X[. . .]♥X
You may write: [. . .][. . .]

To apply Rule 4, the character string must contain the connector ® and some object symbol together, and then the connector ♥ and that same object symbol later on in the character string, but still on the same side (either the right-hand side or the left-hand side).

For example:

The current string is: $\mathscr{P}\,{}^{\circledR}\Delta\heartsuit\Delta\leftrightarrow\heartsuit\Phi\heartsuit\Delta$
You may rewrite it as: $\mathscr{P}\leftrightarrow\heartsuit\Phi\heartsuit\Delta$

### Rule 5

Rule 5 states:

The current line (on one side) contains: [. . .]♥X[. . .]®X
You may write: [. . .][. . .]

To apply Rule 5, the character string must contain the connector ♥ and some object symbol together, and then the connector ® and that same object symbol later on in the character string, but still on the same side (either the right-hand side or the left-hand side).

For example:

The current string is: $\#\mathscr{P}\,\heartsuit\Phi{}^{\circledR}\leftrightarrow\#\Omega^{\circledR}\Phi$
You may rewrite it as: $\#\mathscr{P}\leftrightarrow\#\Omega^{\circledR}\Phi$

### Rule 6

Once you have used the preceding four rules to eliminate all but the $\mathscr{P}$ symbol and its connector from the left-hand side, these next four rules will allow you to get rid of that connector and thus correctly transform the character string.

When the left-hand side contains only $^{\circledR}\mathscr{P}$, in the next line you write you may eliminate the ® from the left-hand side, and leave the right-hand side exactly the same.

As an example:

Current line: $^{\circledR}\mathscr{P}\leftrightarrow{}^{\copyright}\Omega^{\circledR}\Delta$
Your next line: $\mathscr{P}\leftrightarrow{}^{\copyright}\Omega^{\circledR}\Delta$

### Rule 7

Rule 7 is somewhat similar to Rule 6, in that when the left-hand side contains only $\#\mathscr{P}$, in the next line you write you may eliminate the # from the left-hand side and leave the right-hand side exactly the same.

As an example:

Current line: $\#\mathscr{P}\leftrightarrow{}^{\circledR}\Phi\heartsuit\Gamma^{\copyright}\Omega$
Your next line: $\mathscr{P}\leftrightarrow{}^{\circledR}\Phi\heartsuit\Gamma^{\copyright}\Omega$

### Rule 8

When the left-hand side contains only $\heartsuit\mathscr{P}$, in the next line you write you may eliminate the ♥ from the left-hand side, but then change any occurrence of ® in the right-hand side to ♥, and any occurrence of ♥ to ®, leaving the other symbols alone.

To illustrate:

Current line: $\heartsuit\mathscr{P}\leftrightarrow{}^{\circledR}\Omega\#\Gamma$
Your next line: $\mathscr{P}\leftrightarrow\heartsuit\Omega\#\Gamma$

(*Appendixes continue on next page*)

### Rule 9

Rule 9 is somewhat more complicated. When the left-hand side contains only ©$\mathscr{P}$, in the next line you write you may eliminate the © from the left-hand side. If a © appears on the right-hand side, then you may rewrite the right-hand side, swapping the positions of the two symbols on either side of the ©. When the right-hand side does not contain a © connector symbol, you may simply rewrite the right-hand side, but putting a © first.

As examples of this rule:

Current line: ©$\mathscr{P}$↔♥Δ♥Φ©Ω
Your next line: $\mathscr{P}$↔♥Δ♥Ω©Φ

But:

Current line: ©$\mathscr{P}$↔♥Δ♥Φ#Ω
Your next line: $\mathscr{P}$↔©♥Δ♥Φ#Ω

These last two rules are very important and provide a standard method of attack for all problems.

### Rule 10

Rule 10 states that you must eliminate all ♥ and ® connector symbols from the left-hand side before eliminating any # or © symbols.

Therefore, if the problem was #Δ®Φ♥$\mathscr{P}$↔♥Ω, you would have to eliminate the ®Φ before eliminating the #Δ.

### Rule 11

Rule 11 states that you must eliminate the connector and object symbols from the left-hand side in a left-to-right fashion. However, Rule 10 has precedence over Rule 11. So if the current line was $\mathscr{P}$#Γ®Φ©Ω↔♥Φ, you would need to eliminate the ®Φ (because of Rule 10), then the #Γ before you eliminate the ©Ω.

## Appendix B

### Sample Protocols From Experiment 1

| Problem | Protocol | Commentary |
|---|---|---|
| | Sequence 1 (Participant 1) | |
| Problem 1<br>®$\mathscr{P}$ #Γ®Φ↔®Ω<br>↓<br>®$\mathscr{P}$ ↔®Ω♥Φ©Γ<br>↓<br>$\mathscr{P}$ ↔®Ω♥Φ©Γ | Let's see, ®, ®$\mathscr{P}$ is equivalent to ®Ω, ®Ω then let's see, ® has precedence so it's ♥Φ©Γ.<br>(Computer: Good)<br>Good. And, ummm, uh, let's see, so which gives me $\mathscr{P}$ equivalent to, uhh, let's see, ®Ω, ♥Φ©Γ. | The participant first skips two transformations, not worrying about what is in front of the $\mathscr{P}$. After that line is checked, he then applies the rule that removes the ® in front of the $\mathscr{P}$. |
| Problem 2<br>®Φ♥Ω®$\mathscr{P}$ ↔♥Γ<br>↓<br>$\mathscr{P}$ ↔♥Γ♥Φ®Ω | Okay, ®$\mathscr{P}$, $\mathscr{P}$ equivalent to ♥Γ, uh, uh oh, ♥Γ, just, I can worry about that later, ♥Γ, ♥Γ, ♥Φ®Ω. | The participant apparently looks to see what symbol is in front of the $\mathscr{P}$. |
| Problem 3<br>®$\mathscr{P}$ ©Γ♥Δ↔Φ<br>↓<br>$\mathscr{P}$ ↔Φ®Δ#Γ | Um, $\mathscr{P}$, nothing interesting in front of it, goes to Φ, let's see, the ♥ gets precedence, so it's ®Δ#Γ. | The participant is considering the symbol in front of the $\mathscr{P}$ first. |
| | Sequence 2 (Participant 5) | |
| Problem 1<br>♥$\mathscr{P}$ ®Ω↔Φ<br>↓<br>$\mathscr{P}$ ↔Φ♥Ω | Okay. Okay. I have to eliminate the ®s with a ♥, and that ♥ will turn into a ®Ω, and then, that was supposed to have been, that was supposed to have been a ♥, okay. I'm not sure. . . . | First three-step problem done all in one line. The participant does all the steps in his head, first recalling the elimination rule, and then the rule concerning how to remove the ♥ in front of the $\mathscr{P}$. |
| Problem 2<br>♥$\mathscr{P}$ ♥Φ↔Ω<br>↓<br>$\mathscr{P}$ ↔Ω♥Φ | Okay. Get rid of the ♥s first with a ®, and then to get rid of the ♥s, you would change the ♥s to ®s. So I think I can do this on one line. This is that, and then I'll have to put, Φ, but it will just be ♥Φ. | Again, a similar three-step problem has been done on one line. He is faster and more sure of himself this time. Notice he mentioned the ®Φ changing to a ♥Φ. |
| Problem 3<br>♥$\mathscr{P}$ ®Γ↔♥Δ<br>↓<br>$\mathscr{P}$ ↔®Δ®Γ | Okay. Umm, I have to swap the ®s and ♥s. So the original was that, goes to that, and this would stay as it is. | However, this time, he says, "this [the ®Γ] would stay as it is." |
| Problem 4<br>♥$\mathscr{P}$ ®Δ↔®Φ<br>↓<br>$\mathscr{P}$ ↔♥Φ®Δ | Okay. Change that, to a ♥. | At the later problems, participants were very abbreviated in their protocols. |

*Note.* In the first column the problem (as it was presented to the participant) is shown. Below that (as directed by the arrows) is what the participants clicked. The second column is the protocol that the participants produced while solving the problem. Participants developed their own names for the symbols (e.g., "squiggle" or "alpha" for the $\mathscr{P}$), but in the protocol we substituted the actual symbol. The commentary in the third column explains the importance of the protocol.

# Appendix C

## The Adaptive Control of Thought—Rational (ACT-R) Model

This appendix elaborates on the ACT-R (Anderson, 1993) model, which was presented in the article. Because of space constraints, we focus on how the model composes, via the analogy mechanism, the production that covertly skips steps. A more detailed model can be found at the ACT-R World Wide Web site (http://sands.psy.cmu.edu/ACT/act/act-code.html).

We assumed a flat representation for the character strings. That is, the structure that contains the representation of a line is essentially just a list of the symbols placed in their correct slots. This assumption made it easier for the analogy mechanism, plus we had no real intuition as to what sort of hierarchical representation the participants actually used. In ACT-R (Anderson, 1993), our representation can be characterized by the following statement (*lhs* and *rhs* refer to the left-hand side and the right-hand side of the equation, respectively, as separated by the double arrow; *var* refers to variable; *op* refers to operator; and *con* refers to constant):

(WMEType equation lhs-var lhs-op1 lhs-con1 lhs-op2
lhs-con2 rhs-con1 rhs-op1 rhs-con2 achieved-by).

This creates a new working memory element (WME) type called *equation*, which has nine slots, one for each potential symbol, plus the achieved-by slot, which is a special slot used by the analogy mechanism telling the system the next step in a problem's solution. The only other important WME type is the one that defines the operators, which is defined by (WMEType operator reversed-by). Thus, these WMEs not only have the name of the operator but also the operator's inverse associated with them.

A problem can be expressed by the following WME:

(PROBLEM
ISA equation
LHS–VAR *X*
LHS–OP1 *
LHS–CON1 *C*
RHS–CON1 *D*),

which is the problem $X * C = D$. Because the ACHIEVED-BY slot is not listed, it is assumed to be nil. For the purpose of this appendix, it is assumed that this problem is the current goal. Furthermore, the system has already solved the problem $X + A = B$ and has noted that the answer is $X = B - A$. These can be expressed by

$X + A = B$
(SOLVEDPROBLEM1
ISA equation
LHS–VAR *X*
LHS–OP1 +
LHS–CON1 *A*
RHS–CON1 *B*
ACHIEVED–BY SolvedProblem2)

and

$X = B - A$
(SOLVEDPROBLEM2
ISA equation
LHS–VAR *X*
RHS–CON1 *B*
RHS–OP1 –
RHS–CON2 *A*).

The other two WMEs that are important for this illustration are the two that correspond to the operators used in the goal problem and SolvedProblem1:

(+
ISA operator
REVERSED–BY –)

and

(*
ISA operator
REVERSED–BY /).

When the system solved the first problem ($X + A = B$), it presumably either already had the productions needed to solve the problem or it had another example with which it analogized, which created two productions: one for going from $X + A = B$ to $X + A - A = B - A$, and another for going from $X + A - A = B - A$ to $X = B - A$. These productions will be in competition with the analogy mechanism. If the productions are weak, the system can analogize from the current problem ($X * C = D$) to SolvedProblem1. This will result in the following production, one that covertly skips steps:

```
(p ANALOGIZED–PRODUCTION127
  =Problem–Variable>
     ISA              equation
     LHS–VAR          =X–variable
     LHS–OP1          =*–variable
     LHS–CON1         =C–variable
     RHS–CON1         =D–variable
  =*–variable>
     ISA              operator
     REVERSED–BY      =/–variable
 ==>
  =Equation–Operator–Subgoal>
     ISA              equation
     LHS–VAR          =X–variable
     RHS–CON1         =D–variable
     RHS–OP2          =/–variable
     RHS–CON2         =C–variable
 !focus–on! =Equation–Operator–Subgoal)
```

(*Appendix continues on next page*)

The important thing to note about this production is that it found the mapping between the multiplication sign and the division sign to induce correctly how one goes from $X * C = D$ to $X = D/C$ (the three lines above the $\Rightarrow$ in the production). The analogy mechanism does this by finding the path from the + in the previously solved problem to the WME that declares that the reverse operator of + is −, where − is also used in the previously found problem. The other symbols can be trivially mapped from their location in the problem statement to their location in the problem's solution. The !focus-on! command sets the newly created WME as the system's new goal.

Although the current simulation does not attempt to model the latencies recorded in either of the experiments, the ACT–R (Anderson, 1993) theory does specify how such predictions would be made. The latency of the analogy process is defined as the sum of the latencies of the WMEs retrieved in the process, which include the examples and the intermediate retrievals. The analogy process and the instantiation of the production occur sequentially. Therefore, the latency of the resulting instantiation is defined as the latency of the analogy process plus the matching latency. More information regarding how these latencies are computed, including the underlying equations, can be found in Anderson (1993).

Received February 23, 1995
Revision received June 15, 1995
Accepted June 26, 1995 ■