Journal of Experimental Psychology: General
2008, Vol. 137, No. 3, 548–565

0096-3445/08/$12.00 DOI: 10.1037/0096-3445.137.3.548

The Acquisition of Robust and Flexible Cognitive Skills

Niels A. Taatgen
Carnegie Mellon University and University of Groningen

David Huss, Daniel Dickison, and John R. Anderson
Carnegie Mellon University

The authors introduce a model of skill acquisition that incorporates elements of both traditional models and models based on embedded cognition by striking a balance between top-down and bottom-up control. A knowledge representation is used in which pre- and postconditions are attached to actions. This model captures improved performance due to learning not only in terms of shorter solution times and lower error rates during the task but also in an increased flexibility to solve similar problems and robustness against unexpected events. In 3 experiments using a complex aviation task, the authors contrasted instructions that explicitly stated pre- and postconditions with conventional instructions that did not. The instructions with pre- and postconditions led to better and more robust performance than other instructions, especially on problems that required transfer. The parameters of the model were estimated to obtain a quantitative fit of the results of Experiment 1, which was then successfully used to predict the results of Experiments 2 and 3.

Keywords: control, cognitive modeling, skill acquisition, learning from instructions, embedded cognition

Humans have the remarkable ability to acquire almost any skill given suitable instructions and practice. Models of skill acquisition (Anderson, 1982, 1987; Logan, 1988; Newell & Rosenbloom, 1981) have mainly focused on two aspects of skill acquisition: improvement in speed and reduction of errors. Despite the focus on speed and reduction of error, *expertise*, which can be considered the end product of skill acquisition, is associated not only with speed and accuracy but also with flexibility and robustness (e.g., Chi, 2006). *Flexibility* refers to the ability to apply a skill to new problems that are different from the problems that served as the basis for training. *Robustness* is associated with the ability to protect skilled performance from various disturbances, including unexpected events, interruptions, or changing demands. Disturbances can also have internal causes, like forgetting what to do next or making an error, and robustness, in the context of the present article, also includes recovery from errors and handling situations if part of the task knowledge has been forgotten. Robustness finally includes the ability to perform a skill in parallel with other tasks, without any obvious need for complex strategies to schedule resources between multiple tasks. The goal of this article was to discuss a model that can explain the improvement in speed and the reduction in errors, but also explain why flexibility and robustness improve with practice. The key to understanding this improvement is to assume that cognitive task control has both an internal or top-down component, and an external or bottom-up component. The bottom-up component is crucial to enable the model to adapt quickly to changes in the task environment without needing extra knowledge or inference capabilities. The learning process involves discovering the right knowledge structures to support optimal control and then optimizing these structures for fast performance.

Niels A. Taatgen, Department of Psychology, Carnegie Mellon University and Artificial Intelligence, University of Groningen, Groningen, the Netherlands; David Huss, Daniel Dickison, and John R. Anderson, Department of Psychology, Carnegie Mellon University.

This research was supported by NASA Grant NRA2-38169 and Air Force Office of Scientific Research Grant FA-95500710359. We thank Stefani Nellen and Ion Juvina for their helpful comments on the manuscript, Ion Juvina for help in collecting the data, and Peter Polson and Karl Fennell for their advice.

Correspondence concerning this article should be addressed to Niels Taatgen, Department of Psychology, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. E-mail: taatgen@cmu.edu

Overview of Models of Skill Acquisition and Skilled Performance

Traditional Models of Skill Acquisition

Although many models of skill acquisition have been proposed, most of them share a mechanism in which the model starts with general knowledge and, through experience, gains more specialized knowledge (Anderson, 1987; Crossman, 1959; Fitts, 1964; Fitts & Posner, 1967). This specialized knowledge has the advantage of faster memory access and, assuming the specialized knowledge is correct, a reduction in errors.

Logan's (1988) instance theory assumes people start out with a general-purpose algorithm to solve the problem. Each correct solution is stored as an instance in memory. For each new problem, there is a potential race between instances in memory and the general-purpose algorithm: Whichever produces the answer first wins and determines the action. Instance retrieval is generally faster than the algorithm, which explains the speedup in performance as instances accumulate.

Newell and Rosenbloom's (1981) chunking mechanism explains skill acquisition by specializing general problem-solving strategies. The model starts out with a general strategy to solve the problem, which may entail setting subgoals to solve partial problems. Whenever a partial problem is solved, a new production, or *chunk*, is created, which can solve a partial problem for that specific case on future occasions. An even more fine-grained version of this idea is used in the ACT-R architecture (Anderson,

2007; Anderson et al., 2004). ACT-R uses *productions* to represent skilled knowledge, which are knowledge structures that map goals, results of memory retrieval, and perceptual input onto actions.[1] ACT-R has a learning mechanism called *production compilation* (Taatgen & Anderson, 2002), which combines any two productions that have been used in sequence into one new production. Any memory access is substituted into the new production, essentially making it a specialization of the original productions.

Despite many differences in details, the three models have a similar account of the general characteristics of skill acquisition: They all assume an initial general strategy that can solve the problem, and they all explain the speedup in performance and the reduction in errors by specialization and more efficient use of that strategy. The main difference is the grain size of specialization: In instance theory, it is the problem as a whole; in chunking, it is at the level of individual subtasks (which can include several steps), and in production compilation, it is the level of individual steps in problem solving.

Characteristic of all three models is a stress on the internal representation of knowledge to perform the task. This knowledge determines the next action, or it can be used to plan ahead multiple actions. None of the three models addresses robustness and flexibility directly: The nature of the learning process implies specialization, which generally means that the learned knowledge cannot be used for cases that have not been seen before. All three models do have some means to infuse the specialization process with some generalization, which can potentially help increasing flexibility. In instance theory, a similar but not identical instance can be retrieved and applied to the present situation (Lebiere, Wallach, & Taatgen, 1998); in chunking, a constant can be replaced by a variable (Newell, 1990), and in production compilation, the general strategy of analogy can be specialized to achieve generalization (Taatgen & Anderson, 2002). Although generalization can improve flexibility because it helps cover more situations, it can also lead to overgeneralization, which leads to the choice of inappropriate actions that reduce robustness instead of improving it.

Embedded Models of Skill Acquisition

A different view of skilled performance can be found in theories of embedded cognition (e.g., Clark, 1997). These theories stress that intelligent behavior, including skilled performance, cannot be studied without acknowledging the impact of interaction with the world. Brooks's (1991) work in robotics exemplifies this notion: Whereas early robotics work focused on the robot planning future actions on the basis of a task representation (e.g., Fikes & Nilsson, 1971), Brooks's robots had no plan or representation of a plan. Instead the robot's behavior was directly controlled through its perceptual system. For example, if the robot's sensors would perceive an object in the robot's path, then these percepts would lead directly to motor actions that would avoid the object without any elaborate intervening planning steps. This new architecture made the robot's behavior much more robust because it automatically corrected errors in the robots trajectory, was able to handle unknown obstacles in the robot's path, and made it possible to put the robot in an unknown environment.

Evidence that a tight coupling between perception and action is also evident in human behavior can be found in Prinz's (1997) work on the relation between perception and action planning. He found that when the coding of a percept can stimulate the code for an action directly (e.g., responding by moving the hand to the left button when a target on the screen moves left), response times are much faster than when this is not the case. More direct evidence has been found in monkey research, in which single-cell recordings of the visual cortex and the part of the motor cortex that involves eye movements show that regions corresponding to candidate eye movements are activated almost directly by visual stimuli in the corresponding field of view (Roelfsema, Lamme, & Spekreijse, 1998).

A property that these approaches share is that the world itself, or a relatively straightforward encoding of the world, is the main drive for determining actions. Glenberg and Robertson (1999) have shown that making connections between instructions and perceptual input is important for a proper understanding: In an experiment in which people were instructed to use a compass, one group received just instructions, and the other received instructions in which key terms in the instructions were linked to a picture of the compass. The opportunity to link instructions to perceptual input proved to lead to better performance. Kirsh (1995) argued that people actually tend to organize the world around them to facilitate their performance. For example, in the game of Tetris, expert players rotate pieces physically (by pressing keys) rather than mentally because physical rotation is faster than mental rotation (Kirsh & Maglio, 1994). Similar observations, but in the domain of aviation and other contexts, have been made by Hutchins (1995).

The Role of Cognitive Control and Task Representation in Skill Acquisition

The embedded cognition theories of skilled performance provide what is lacking in the traditional approaches: explanations of why skills are robust to unexpected changes in the environment or changes in the task. However, the direct connection between perception and action in the embedded approach makes it hard to give it instructions for a new task and see how these instructions are developed into a robust skill (although some systems permit training the system from the bottom up, e.g., Botvinick & Plaut, 2004). The key to a theory that explains both the taskability that traditional theories provide and the robustness that embedded theories offer is to analyze why the former leads to brittle skills and the latter to robust skills.

Traditional models of complex task performance structure the knowledge for the task in a sequence of steps, each of which leads to an action. To properly sequence the steps, the model maintains some sort of internal control state that is used to determine the next step. This leads to an organization of knowledge in lists, or, in the case of complex tasks, a hierarchy of lists (e.g., Card, Moran, & Newell, 1983), and to models that base their actions primarily on an internal representation of the world. Aligning the internal representation with the real world requires additional actions and knowledge. The alternative used in embedded models is only to map what is perceived onto actions, which is a much more simple

[1] Although productions are also often referred to as *rules*, we avoid the term here because ACT-R's productions lack many properties usually associated with rules.

mapping. This simplicity means that it is much more likely that the available knowledge will cover all possible situations.

An illustration of this is Larkin's (1989) example of making coffee. When analyzed in detail, making coffee turns out to be a complicated task that consists of many mutually dependent steps. A traditional model would call for an internal representation that keeps track of the state of the coffee machine (i.e., Is there water in the reservoir, coffee in the grinder? Is the lid on the pot? etc.) and uses planning to determine the appropriate next action. This planning process can break down if certain things in the world are not consistent with the internal state. For example, if the coffee reservoir is already full when the model wants to fill it, then it does not know what to do. Larkin observed that people are able to navigate through these steps almost without effort, and are also able to adapt their plan on the fly. The key observation she made is that individual steps are not triggered by a planning process but rather by conditions that can be perceived in the world, for example, an empty filter holder cues placing a filter in it. Informal investigation of errors people make in preparing coffee revealed that errors are often related to aspects of the task that cannot be observed directly, for example, forgetting to fill an opaque water reservoir. We suggest that people do not mentally organize the steps required to make coffee as a list. Instead, the conditions under which a step can be carried out play an important role in organizing knowledge and selecting the next steps. For example, the step to put ground coffee in the filter is not triggered by the previous step of grinding coffee but by the fact that there is an empty filter in the holder, and ground coffee in the grinder. Many of these conditions can be perceived, whereas others have to be remembered (e.g., whether the opaque container has been filled). This view of planning is consistent with the embedded theory of skilled behavior. It also is consistent with the more applied work of Norman (1988), whose numerous examples illustrate that perceived *affordances*, actions that the current environment allows, are the main determiner of action selection and errors if the environment is poorly designed. Studies of eye movements when making tea (Land, Mennie, & Rusted, 1999) and sandwiches (Hayhoe, Shrivastava, Mruczek, & Pelz, 2003) show that people repeatedly sample the environment and exhibit a tight interaction between perceptual and motor actions, evidence against internal representations and explicit planning. Agre and Shrager (1990) have studied how people get faster when using a photocopier. They showed that this can be attributed to the fact that people learn to use perceptual cues to key their next action. For example, when the light of the photocopier has flashed, the original can be turned to the next page: It is not necessary to wait for the copy.

Although the embedded theories seem to offer a better explanation for skilled behavior, they have two shortcomings. A first shortcoming is that sometimes not all the information is available in the world, making it necessary to maintain some mental representation after all. The solution we propose to this problem is to maintain an internal representation, but only change it when strictly necessary, a principle called the *minimal control principle* (Taatgen, 2005, 2007). This means that control is derived from the environment, or bottom-up, as much as possible, and that top-down control, derived from an internal state or representation, is used only when necessary. The second shortcoming of embedded theories is that it is not evident how the relatively direct coupling between perception and action can be learned in cases in which the learning is partially based on instruction and not just on direct experience. The solution we propose for this is slightly more elaborate. The idea is that the direct coupling between perception and action is the result of a learning process that starts by retrieving task knowledge from memory. This knowledge is not structured in a hierarchy but is cued by the environment. For example, a novice at making coffee may have read in the instructions of the coffee maker to put the lid on the carafe before switching on the machine. The lid can therefore serve as a cue to retrieve the appropriate action from memory. If the lid would be out of sight for some reason, then the novice might forget to put it onto the carafe. The transformation of the memory retrieval process to a process that links perception (and internal state) to action directly is carried out by the production compilation mechanism (Taatgen & Anderson, 2002). This mechanism, which we discuss in detail in the *Learning* section, replaces the memory retrieval process by a direct perception/action mapping.

In order for perception to cue relevant actions, actions need to be augmented by a representation of what is perceived when the action is applicable. We take this idea a step further and also add to each action a representation of its expected outcome. Encoding task knowledge in terms of precondition-action-outcome triples results in what we call an *operator* representation because of its similarity to representations used in artificial intelligence (AI; Stanford Research Institute Problem Solver (STRIPS) operators, Fikes & Nilsson, 1971) and in the early work of Newell and Simon (1963). Although STRIPS has also been used to control a robot, most of the earlier AI systems matched preconditions only to an internal representation of the problem, giving rise to the criticism that they only modeled cognition "in the head." In our approach, the preconditions of an operator can be matched to both internal states and to what is perceived in the world. However, matching the precondition to conditions in the world produces behavior with maximal flexibility.

To summarize, our theory of skills acquisition consists of a set of components. The first component is the operator representation of task knowledge, in which recall of the relevant knowledge is mainly driven by perceptual input. The second component is the principle of minimal control that specifies that the task representation should have a control structure that is as small as possible. The third component is the production compilation mechanism that gradually transforms memory retrieval of task knowledge into direct perception–action mappings. In the present article, we make these three components explicit in a model of skill acquisition that we apply to a complex task. This model aims to explain how missing knowledge is filled in, how problems that do not match the instructions exactly can nevertheless be solved, and how partially completed problems and error states can be handled.

To test the model, we conducted three experiments using a complex task. To test the assumptions of the operator representation, we constructed two instruction sets (or, in the case of Experiment 3, four instruction sets). One of the instruction sets, which we call the list instructions, only lists the actions that have to be taken, whereas the other set, the *context instructions*, also makes the pre- and postconditions of actions explicit. If people indeed use an operator representation, then adding pre- and postconditions should have a substantial benefit, more than other additions to the action list (which we explore in Experiment 3).

The Task Domain: Flight Management Systems

The task used in all experiments was derived from automation in modern airplanes. Many passenger airplanes use flight management systems (FMS) to help pilots control the airplane. On a routine flight, the FMS can perform almost the whole flight with the exception of take off and landing. The task of the pilot is to supply the FMS with the right information and parameters (e.g., the load of the plane, and, most importantly, the route it has to fly) to do its job. This route consists of a list of waypoints that the plane has to follow from the source to the destination airport. Waypoints are sometimes specific radio beacons, but they are often just points on the map with particular coordinates. Although the route in the FMS has both a vertical and a lateral component, the task focuses on the lateral part of the task, which is not unlike the kind of routes that are produced by route planning services for cars.

Interacting with the FMS is typically taught as part of the pilots' supplementary training when they start flying a plane that has an FMS. Training consists of a phase in which procedures on the FMS are learned in the classroom, followed by a phase in which they are applied in a simulator. Procedures are specified as lists of steps to carry out. Although 102 different procedures have been identified for the Boeing 777 FMS (the system we used for our experiments), knowledge of only around 25 procedures is needed for FAA certification, and therefore the training focuses on those procedures. The idea behind this is that the pilots can study and/or discover the remaining procedures on their own. Experience from training itself shows, however, that it is very hard for pilots to learn the required procedures, let alone discover any new procedures (Sherry, Polson, Fennell, & Feary, 2002). Memorizing the procedures during the classroom phase of training turns out to be so hard that it is virtually useless for the second phase of training in the simulator. Pilots' troubles include problems with forgetting particular steps in a procedure, not knowing how to pick up a partially completed procedure, and poor generalization. For example, the procedure to fly toward a waypoint at a certain heading is identical to the procedure needed to land the plane (which involves approaching the end of the runway at a certain heading), but pilots have great trouble executing the former while having no problems with the latter. Fennell, Sherry, Roberts, and Feary (2006) categorized steps in procedures as *recall* or *recognition*. Recognition steps were steps that had cues in the environment, for example, a label on a key or a line of text on the screen, whereas recall steps had no perceptual cues. In an experiment with pilots new to the FMS, researchers found that if a procedure had no recall steps, but only recognition steps, then the error rate was 6%; if it had one recall step, then it was 13%; and if it had two recall steps, then it was 74%. This indicates that steps that cannot be inferred from the interface are the main source of errors, which is another indication that people's internal representation of instructions tends to be triggered by external events.

The FMS task is a very suitable task for exploring flexibility and robustness because the starting point is a training system that, for many pilots, leads to overspecialized, inflexible and brittle skills. From the perspective of our minimal control principle, this is perfectly understandable: If instructions are learned as lists of steps, then the individual steps cannot be related to the current environment, and therefore cognitive control has to be completely internalized.

General Task Description for All Experiments

For the purposes of the experiments, we chose two FMS procedures participants had to learn and carry out. Both procedures are part of *lateral navigation*. Lateral navigation involves planning and modifying routes. A route is a sequence of waypoints (points on the map) that the plane will follow and is typically programmed into the FMS before the flight starts. However, the route is often changed during the flight. A possible reason for such a change is that when traffic is light, air traffic control allows the plane to skip a few waypoints and fly directly to a waypoint further in the flight plan. The procedure to make this modification is called *direct-to*. Another reason for changes in the flight plan is bad weather. This sometimes requires the pilot to change his heading to a new waypoint that was not previously on the flight plan and proceed with the flight plan after that new waypoint. This change requires two procedures: the *direct-to* procedure to change the waypoint that the plane is currently heading for and the *remove-discontinuity* procedure, which tells the FMS how to connect the newly entered waypoint to the rest of the flight plan.

Participants had to carry out route modifications on a simulated FMS, consisting of a keyboard to enter information into the scratchpad, a display listing part of the current route, a navigational display showing a map with waypoints and the current route, and a pane that shows the current task (see Figure 1). The main difference between the simulation and a real FMS was that participants had to operate the keyboard by clicking on the keys with the mouse. In the real airplane, the navigational display and the FMS are not next to each other but require a larger eye and possibly head movement. For the purposes of the experiments, this only changes some of the perceptual motor characteristics of the task, which are not the object of the study.

In Experiments 1 and 2, there were two conditions for the instruction of the procedures: "list procedures" and "context procedures." List procedures, consisting of numbered lists of steps, were adapted from the United Airlines training program. The context procedures instructed participants on not only the steps that they had to take but also the conditions for carrying out a step, and what its result was. Table 1 lists both procedures in both styles. Each of the experiments consists of series of problems that participants have to solve, that is, directives of air traffic control that have to be entered in the FMS. Figure 2 illustrates how the direct-to procedure is carried out. In some of the problems, the learned procedures could be carried out literally, but in some more complicated problems, participants had to make modifications to make it work. Given that the context instructions are closer to what we hypothesize as the internal representation of the task, we expected participants to be faster and more accurate after receiving the context instructions than after the list instructions. We also expected that, in particular, accuracy would be higher in the context condition for problems that were more complex and that require participants to go beyond what was given in the instructions. Finally, we expected participants in the list instruction to carry out every step in the instructions, even if they were not necessary, but that the participants in the context condition would more readily skip such steps.

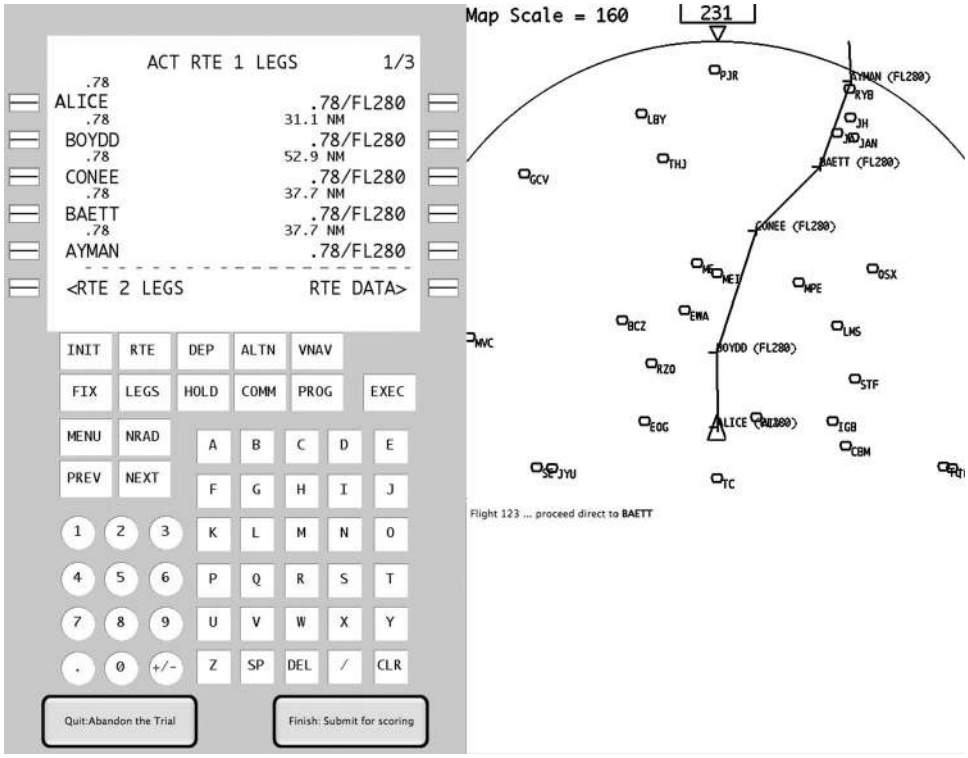


Figure 1. The flight management system (FMS) experiment. The left of the display shows the keyboard and displays contents of the actual FMS unit, together with a button to either quit or indicate that the task is completed. The top right of the display shows the navigational display that can be used to verify the route. The bottom right of the display shows the current problem and will display feedback after the participant has pressed *finish*. The rectangles left and right of the FMS display are also keys, called *line keys*. The keys to the left of the display are named 1L–6L, and the keys to the right are 1R–6R. The bottom line on the FMS display is called the *scratchpad* (which is empty in the figure). Text typed on the keyboard will appear in the scratchpad and can be transferred to a line in the display by pushing a line key. For example, typing "BOYDD," which is one of the waypoints on the map, puts BOYDD in the scratchpad. Pushing the *1L* key would then put BOYDD next to the *1L* key on the display, replacing ALICE. An alternative method to put text in the scratchpad is by pushing one of the line keys. For example, pushing the *1L* key copies ALICE into the scratchpad. The LEGS page, which is currently displayed, lists the current route the plane is taking, starting with a list of waypoints consisting of ALICE, BOYDD, CONEE, BAETT, and AYMAN and continues on two more pages. The Navigation Display on the top right shows a graphical version of the route. The triangle represents the current position of the plane with a line connecting the upcoming waypoints.

Experiment 1

Method

Participants. Thirty-one students from Carnegie Mellon University were paid for participation in the experiment (15 in the list condition, and 16 in the context condition).

Procedure. Participants first read through the general background information of the FMS task. They then started with a series of warm-up trials that taught them how to operate the FMS interface. These warm-up trials, taking about 10 min, consisted of typing in text on the keyboard, copying text from a line on the FMS to the scratchpad, and copying text from the scratchpad to a line. All of the text on the display during the warm-up was unrelated to the real FMS task. Participants then studied the direct-to and remove-discontinuity procedures for 5 min for the condition that they were in.

The experiment proper consisted of three main blocks of trials, each consisting of 12 problems. The first three problems in each block were problems for which the direct-to procedure could literally be applied (easy problems). The second set of three problems consisted of problems in which a new waypoint had to be entered, making it necessary to apply both the direct-to and the resolve-discontinuity procedure (medium problems). Each of the final six problems in a block contained some complication, making it impossible to apply the procedures literally (hard problems). These complications were one, or a combination of the following:

> One of the waypoints referred to in the problem would not be on the page visible in the FMS. Participants had to use the page-up/down keys to find them. Although the function of these keys was explained in the general background, they were not part of the procedures.

Table 1
Instructions in the List and in the Context Condition

List condition	Context condition
Direct-to: 1. Press the LEGS key. 2. Enter the desired waypoint in the scratchpad. 3. Push the 1L key. 4. If the word "discontinuity" appears on the screen, follow the procedure to remove discontinuities. 5. Verify the route on the Navigational Display. 6. Press EXEC.	Getting to the LEGS page You can see what page you are on by looking at the top line of the window. If the word "LEGS" is on that line, then you are on a LEGS page. If you want to change the route and you are not yet on the LEGS page, then press the LEGS key in order to go to the LEGS page. How to modify a waypoint The item in line 1 on the first LEGS page, displayed in magenta, is the waypoint you are currently flying to. You can change this item, by pressing the line key next to it. If you want to modify a waypoint, you enter the waypoint to replace it with into the scratchpad, and then press the line key corresponding to the waypoint you want to modify. How to confirm your results Use the NAV display to view the results of your modification. When you are satisfied with a modification, you can press the EXEC key to make it permanent.
Remove-discontinuity: 1. Press the LEGS key. 2. Press the line select key after the discontinuity. 3. Press the line key with the THEN prompt. 4. Press EXEC.	Discontinuities in the route When the text "Route discontinuity" appears on the LEGS page, the route should be made continuous. In order to make the route continuous, two points on the route have to be reconnected. The LEGS page will show the last connected waypoint followed by "THEN" and a line with boxes. Enter the waypoint that you want to fly to after the last connected waypoint on that line.

The waypoint to be modified was not the waypoint that the airplane was currently flying toward, but one later in the flight plan. This was not covered by the procedures and required some generalization.

Half of the hard problems only required using the direct-to procedure; the other half required both procedures. Although the problems were presented in a fixed order of difficulty within a block (3 easy, 3 medium, 6 hard), the problems within a category of difficulty were assigned randomly (e.g., there were 9 easy problems overall, and they were randomly assigned to a block and a position within the easy problem slots of a block). The whole experiment lasted approximately 1 hr. Once the main phase of the experiment had started, participants were not allowed to refer back to the instructions. If participants were

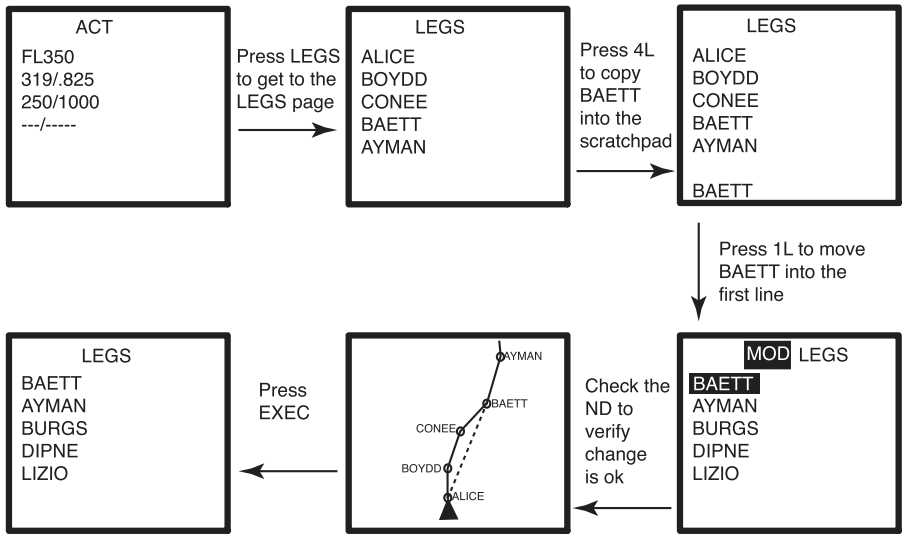


Figure 2. Simplified display of the direct-to procedure based on Figure 1. This example assumes air traffic control has given the directive to proceed directly to BAETT and that the flight management system (FMS) is initially not on the LEGS page. Pressing the *LEGS* key brings it to the LEGS page, showing BAETT as the fourth waypoint in the flight plan. Pressing the *4L* key, which is next to BAETT, brings it to the scratchpad (the bottom line on the display). Pressing *1L* will put it in the top line of the flight plan as a modification (indicated by white-on-black shading). The next step is to verify that the modification is what was intended. The navigational display (ND) indeed shows a dotted line from the airplane (the triangle) directly to BAETT. Finally, the *EXEC* key is pushed, making the modification final.

unable to solve the first problem, then the experimenter would reiterate the relevant part of the procedure to help them. No help was offered at any later time in the experiment.

Results

Figure 3 shows the proportion correct for each problem in the two conditions. The average correctness is 77.0% in the list condition and 91.1% in the context condition. An analysis of variance (ANOVA), with block and problem difficulty as within-subject factors, condition as the between-subject factor, and subject as a random factor, showed a main effect of condition, $F(1, 29) = 7.09$, $MSE = 1.85$, $p = .014$; a main effect of block number, $F(1, 29) = 56.1$, $MSE = 2.01$, $p < .001$; and a main effect of problem difficulty, $F(2, 58) = 7.12$, $MSE = 0.42$, $p = .002$. Furthermore, there was an interaction between condition and block, $F(1, 29) = 6.7$, $MSE = 0.24$, $p = .015$; an interaction between block and problem difficulty, $F(2, 58) = 6.6$, $MSE = 0.28$, $p = .003$; an interaction between condition and problem difficulty, $F(2, 58) = 3.43$, $MSE = 0.20$, $p = .039$; and no three-way interaction. This indicates that correctness was significantly lower in the list condition and that there was substantial learning between blocks. The interaction between condition and problem difficulty indicates the instruction type had different effects on accuracy for different levels of difficulty. Welch t tests comparing the two conditions for each type of problem revealed that the difference can mainly be explained by the hard problems: For the easy problems, there was no significant difference in difficulty (list condition: 90.4% accuracy; context condition: 93.8% accuracy), $t(28.8) = 1.15$, $p = .26$; for the medium problems, the difference approached significance (list condition: 77.0% accuracy; context condition: 91.0% accuracy), $t(18.9) = 1.99$, $p = .062$, whereas for the hard problems, the difference was very clear (list condition: 70.4% accuracy; context condition: 89.9% accuracy), $t(17.9) = 2.6$, $p = .018$. These results confirm our prediction that the benefit of context instructions would be stronger in those problems that required participants to go beyond the actual procedures instructed. Nevertheless, the context instructions also provide a benefit for the medium problems. The model that we discuss in the *A Model of Learning From Instructions* section explains this by assuming that participants sometimes forget some of the instructions and that context instructions are better at enabling recovery from forgetting.

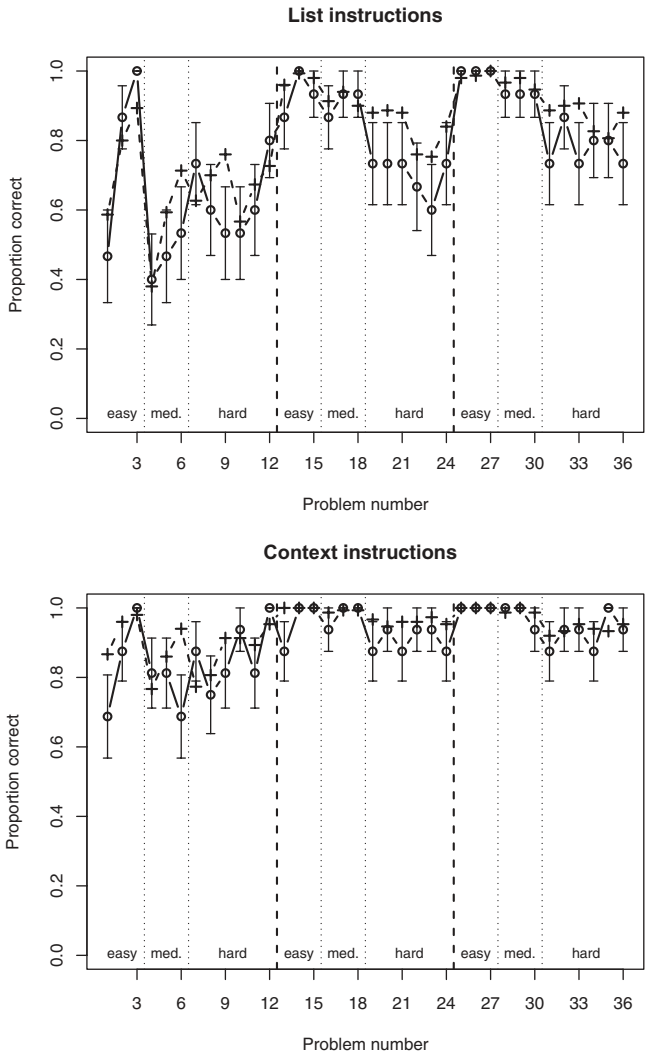


Figure 3. Proportion correct for all the 36 problems in the two conditions, which are divided in three blocks (marked by vertical dashed lines) with three levels of difficulty each (marked by vertical dotted lines). Circles are the empirical data with standard error bars, and the dotted line with crosses is the model fit, which is discussed in the *Model Fits* section. med. = medium.

The average solution time for correctly solved problems in the list condition is 28.7 s; in the context condition, it is 24.8 s. An ANOVA of the log solution times of correctly solved problems, with block as the within subject-factor, condition and problem difficulty as between-subject factors, and subjects as the random factor, showed a main effect of condition, $F(1, 29) = 5.85$, $MSE = 3.91$, $p = .022$; a main effect of block, $F(1, 29) = 231.9$, $MSE = 35.8$, $p < .001$; and a main effect of problem difficulty, $F(2, 58) = 130.9$, $MSE = 10.5$, $p < .001$. In addition, there was an interaction between block and problem difficulty, $F(2, 58) = 31.7$, $MSE = 4.48$, $p < .001$, but no other interactions. Figure 4 shows the average solution times for the correct trials in both conditions. Unlike accuracy, there is no interaction between instruction and problem difficulty for latency, and, as we will see, this is a prediction of our model.

A Model of Learning From Instructions

In the introduction, we made a case that people internally use an operator representation for task knowledge and instructions. The first experiment gives qualitative support for this representation. We want to take this one step further and supply quantitative support by providing a simulation model that fits accuracies and solution times in detail. The advantage of this model over a purely verbal account is that it can be used to make quantitative predictions for new experiments in the same domain and can serve as a basis for models that make predictions in other domains. We indeed use the model to make predictions for Experiments 2 and 3,[2] in which the same instructions are given, but additional problems have to be solved.

The general approach is to have a single model that can take two representations of instructions, list and context. The model has been built in ACT-R (Anderson, 2007; Anderson et al., 2004) and

[2] The model of the experiment is available for downloading from the ACT-R models page at http://act-r.psy.cmu.edu/models

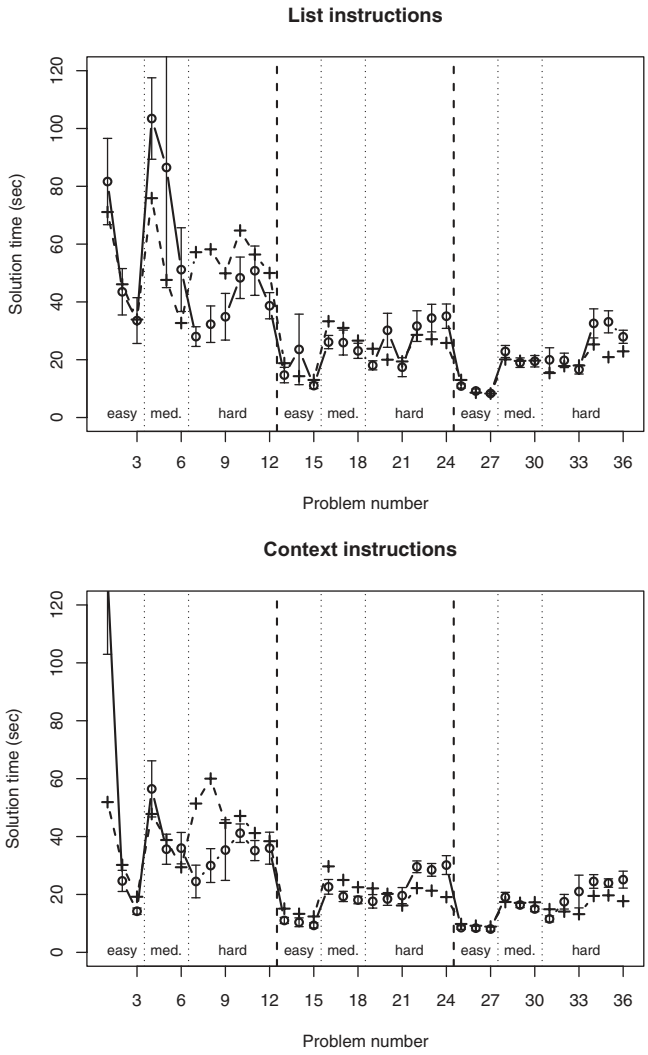


Figure 4. Average solution times for correctly solved problems for all 36 problems in the two conditions, which are divided in three blocks (marked by vertical dashed lines) with three levels of difficulty each (marked by vertical dotted lines). Circles are the empirical data with standard error bars, and the dotted line with crosses is the model fit, which is discussed in the *Model Fits* section. med. = medium.

is a further development of our work on learning from instructions (Anderson, Taatgen, & Byrne, 2005; Taatgen, 2005; Taatgen & Lee, 2003).

The ACT-R Architecture

One of the basic assumptions of the ACT-R architecture is that there are two long-term memories: *declarative memory* and *procedural memory*. Declarative memory is used to store facts and experiences and is basically passive: It retrieves memories upon request. Procedural memory contains condition–action patterns and productions, which map internal and external states onto actions. These productions play an active role because as soon as the production's conditions are fulfilled, it triggers an action (assuming it wins the competition with other productions whose conditions are also fulfilled). Figure 5 shows an overview of the architecture, with procedural memory as the central component, surrounded by modules that are either part of internal cognitive processing or communicate with the outside world. Although procedural memory is the central component, its scope is limited: It can only access the end product of processing in the modules as made available in *buffers*. For example, productions cannot match any arbitrary fact in declarative memory but, instead, only the fact that is present in declarative memory's buffer. Buffers therefore serve the role of communication ports between the different components of the cognitive system. Procedural memory is a fast system because it can select actions on the basis of input in 50 ms. It therefore supports an almost direct coupling between perception and action, assuming the right production is in memory to make the mapping.

We do not discuss all the details of the ACT-R architecture here but focus on the mechanisms in the architecture that are important for the model and discuss them along with the main features of the model.

Representation of Instructions

Although many models based on productions (in ACT-R, but also other architectures) start with a task representation in the form of a set of productions, this is not a plausible account of how new skills are acquired. As was already pointed out in early work with ACT-R's predecessor, the ACT* (Anderson, 1982), it is much more likely that instructions for a new task are first stored in declarative memory. These instructions are retrieved step by step by productions and are then interpreted and carried out by other productions.

A main problem in the interpretation process is finding the appropriate next step to be carried out in memory. As pointed out in the introduction, the easiest way to decide this is to make the individual steps of an instruction into a list and retrieve and perform these steps in order. The alternative, which we used in this article, is to use preconditions to determine what step to do next, what was the basis for context instructions. This representation is augmented with a postcondition that specifies the expected outcome of a step.

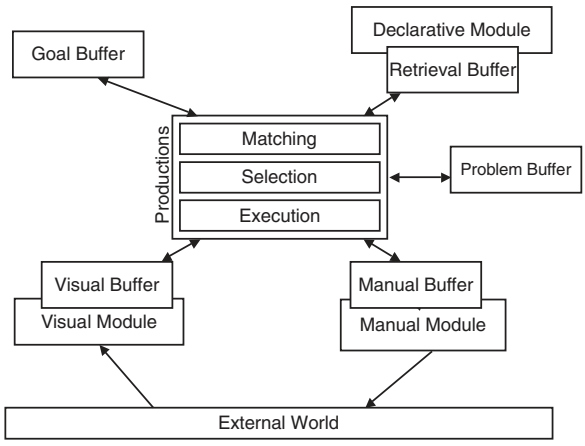


Figure 5. Overview of the ACT-R architecture.

In order to use the same model for both conditions of the experiment, we used a representation of instructions that contains a precondition, an action, and a postcondition. In case of the list instructions, the pre- and postconditions are symbols whose only purpose is to link the instructions together in a list. In the context condition, the pre- and postconditions are representations that can be matched to the state of the world. Table 2 lists the representations for the two conditions. As an example, the list-condition representation specifies "Direct 1" as a precondition for the action to type the destination (the second line in Table 2). This means that the internal state has to be "Direct 1" in order to carry out this action. The "Press LEGS" action, on the first line in the table, is an action that sets the state to "Direct 1." In other words, the cue to type the destination is the internal state that is reached after pressing the *LEGS* key. The context-condition representation specifies "On LEGS page" as a precondition for typing the destination and refers to an external instead of an internal state. This means that typing the destination in this representation is cued by the perceptual cue of being on the LEGS page.

The Basic Decision Cycle

Figure 6 illustrates the basic decision cycle of the model. In the first step, the model perceives the current state of the environment, in this case, the state of the FMS, producing the *perceived state*. Besides a perceived state, there is also an *expected state*, which is set to the postcondition of the previous operator (the value is initially set to START). On the basis of both the perceived and expected state, an operator is retrieved from declarative memory. This retrieval is based on ACT-R's memory activation: The operator with the highest activation will be retrieved (see Anderson, 2007, for the details). For the purpose of this model, the important factor in this process is that operators that share attributes with the current expected and perceived state receive extra activation and are therefore considered first. Because the retrieval process is noisy, the retrieved operator is checked for applicability. If it cannot currently be applied, then a new operator is retrieved until an applicable operator is found, or until the retrieval process fails. To continue the example from the previous paragraph, suppose the model has just pressed the *LEGS* key in the list condition. Its expected state is set to "Direct 1," because that is the postcondition of that action, and it has a perceived state of being on the LEGS page. The expected state now spreads activation to both the "Type destination" and the "Press line key with destination" actions. The perceived state (being on the LEGS page) has no associations with any of the instructions, so it does not contribute to the process. As a consequence, the most likely outcome is that either "Type destination" or "Press line key with destination" is selected. In the case of the context instructions, both the expected state and the perceived state are "on LEGS page" after pressing the *LEGS* key. Because both concur, this gives a stronger activation boost to both applicable actions, leading to a faster response and a lower probability that a wrong operator is retrieved (due to noise on activation).

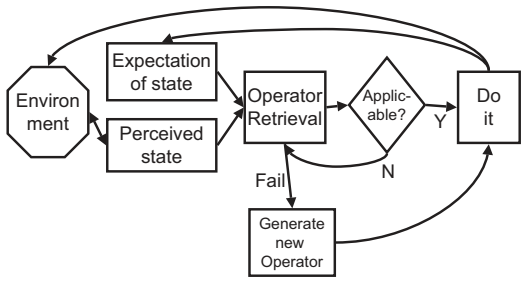


Figure 6. Outline of the model. N = no; Y = yes.

When the operator is applicable, it is executed, resulting in an action that leads to a change in the environment, in the perceived

Table 2
Representation of the Instructions in the Model for the Two Conditions of the Experiment

List condition			Context condition		
Pre	Action	Post	Pre	Action	Post
Start	Press LEGS.	Direct 1	On INIT page	Press LEGS.	On LEGS page
Direct 1	Type destination.[a]	Direct 2	On LEGS page	Type destination.[a]	Destination in scratchpad
Direct 1	Press line key with destination.[a]	Direct 2	On LEGS page	Press line key with destination.[a]	Destination in scratchpad
Direct 2	Press 1L.	Direct 3	Destination in scratchpad	Press 1L.	Modification done
Direct 3	Check for discontinuity.	Direct 4 or Direct 6[b]	Modification done	Check navigational display.	Check ok
Direct 4	Check navigational display.	Direct 5	Check ok	Press EXEC.	Goal achieved
Direct 5	Press EXEC.	Goal Achieved			
Discon 6	Press LEGS.	Discon 7	Discontinuity	Press line key after discontinuity.	Discontinuity in scratchpad
Discon 7	Press line key after discontinuity.	Discon 8	Discontinuity in scratchpad	Press line key with discontinuity.	Modification done
Discon 8	Press line key with discontinuity.	Discon 9			
Discon 9	Press EXEC.	Direct 4			

Note. Discon = Discontinuity.
[a] The general instructions give two methods to enter a waypoint into the scratchpad; therefore, each is represented by a separate operator. [b] This step sets the expected state to Direct 4 when there is no discontinuity, or to Direct 6 when there is one.

state and in the expected state. The cycle is repeated until the goal is reached or time runs out (the model gives up after 120 simulated s).

Forgetting and Discovery of Operators

One of the properties of learning from instructions is that instructions are forgotten easily. People have some ability in handling situations in which they have forgotten some of the instructions. Moreover, for the harder problems in the FMS task, participants need to discover some operators themselves. To simulate the aspect of forgetting operators, each operator in the model had a 25% probability of being forgotten, that is, not being present in declarative memory at the beginning of the simulation. Forgetting operators leads to situations in which the model does not know what to do, that is, the retrieval attempt of an operator fails. The model uses a relatively simple strategy to handle the situation: It takes a random action and tries to perceive what the result of that action is. It then judges whether this action has brought it closer to the goal. If that is the case, then the model will create a new operator with the old state as precondition, the random action as action, and the new state as postcondition. Random actions were drawn from the following set: pressing the *LEGS* or *RTE* key, typing any of the waypoints mentioned in the problem, pressing any line key with any of the waypoints mentioned in the instructions, or the line key below it, pressing a line key with the term *discontinuity* or the line key below it, pressing the *Erase* key, pressing the *CLR* key, or checking the navigational display. The model's judgment of whether an action would bring it closer to the goal was based on the number of steps from the goal. Although it is likely participants have a sense of whether an action brings them closer to the goal, the model's perfect knowledge is an oversimplification.

As an example, suppose the model has entered the destination in the scratchpad but has forgotten that the next action is to push the *1L* key. It would fail to retrieve an applicable operator and resort to randomly picking an action. Many of the possible actions do not change anything (e.g., pressing the *LEGS* key), produce a state that is earlier in the sequence (e.g., pressing *CLR* or *Erase* put the model in the "On LEGS page" state), or may even put the model in an error state (e.g., typing the destination while there is already something in the scratchpad). However, once the correct action is picked, the model will notice progress and learn a new operator with that action, the original perceptual state ("Destination in Scratchpad") as precondition and the resulting perceptual state ("Modification done") as postcondition. Although the learned operator is identical in both instructional conditions, the context condition is better able to use it. After the learned "1L" operator has been carried out, both the expected and perceived state will be "Modification done." The context instructions can pick up from that state straight away because the "Check Navigational display" operator has "Modification done" as a precondition, but the list-instruction model has no knowledge on how to proceed because it can only match operators with respect to the order in the list. By guessing an action, it has lost the thread of the instructions.

Some of the participants did in fact use a guessing strategy to try out new actions. It is, however, likely that there are other strategies to find a next action, for example, based on means–ends analysis. The simple guessing strategy turned out to be enough for the model to learn the missing steps.

Learning

Although ACT-R has a number of learning mechanisms, the mechanism that explains most of the learning in the model (in addition to the discovery of new operators) is *production compilation* (Taatgen & Anderson, 2002). Production compilation encodes operators directly into productions, making it unnecessary to retrieve and test operators from declarative memory. The basic process is very simple: If two productions fire in sequence, then they are combined into a single new production that is added to procedural memory. If the first of these productions makes a request to declarative memory, and the second production uses the retrieved fact to perform an action, then the retrieved fact is substituted into the new production. Table 3 shows an example of this process, with two productions from the model translated into pseudo-English. In the example, the two original productions retrieve and interpret instructions: The first production initiates the retrieval of an operator for the current task, and the second production presses a key after checking whether the preconditions have been fulfilled. The newly learned production, which combines the two original productions and the retrieved operator, immediately recognizes the current state, presses the key, and sets the appropriate expected state. The advantage of the new production is that it can bypass the retrieve-operator and check-applicability cycle and maps perception directly onto action. In behavioral terms, it therefore produces a considerable speedup and a reduction in errors.

The first time a production is created, its utility value is set to zero. Utility values of productions are used in ACT-R to decide which one to use if there are multiple candidates. In the case of a

Table 3
Example of Production Compilation

Original Production 1	Original Production 2	Learned production
IF the goal is to do a task THEN send a request to declarative memory for an operator for that task Operator example: Precondition Not on LEGS page Action Press LEGS key Postcondition On LEGS page	IF the goal is to do a task AND an operator that specifies a press action has been retrieved from declarative memory AND the precondition of the operator matches the current state THEN Press the specified key AND set the expected state to the postcondition of the operator	IF the goal is to do the FMS task AND the state is Not on the LEGS page THEN Press the LEGS key AND set the expected state to "on LEGS page"

Note. FMS = flight management system.

new production, its utility value has to compete with the utility value of the production that it originated from (the "old" production), so a new production initially has no chance to win the competition with the old production. However, each time the production is recreated, its utility value will be increased to approximate the value of the old production. If the utility of the old and the new production are close enough, then the new production has some probability of being used and begins gaining its own experiences. The consequence of this is that new productions are only introduced slowly, which is consistent with the notion that skill acquisition is slow. The learning speed is controlled by a learning parameter that determines how fast the utility of the new production converges with the utility of the old production.

The productions that are learned on the basis of the list instructions are less general than productions learned on the basis of the context instructions. The production in Table 3 can fire each time the FMS is not on the LEGS page, no matter what the reason is. The production that would be learned on the basis of the list conditions would require that the current expected state equals "Start," which is only true at the beginning of a trial.

Model Fits

The model was run 150 times for each of the two instruction sets, after which the accuracies and solution times for correct trials were averaged. Three parameters were estimated to fit the data: the retrieval threshold, which determines the minimum activation needed to retrieve facts from declarative memory, the learning rate, which affects how quickly new productions can compete with old productions, and visual attention latency, which determines how long it takes to perceive the current state of the environment. The model fits are shown alongside the data in Figures 3 and 4. The model fits the data quite well ($R^2 = 0.88$ and root-mean-square error of approximation [RMSE] = 0.078 for list-condition accuracy; $R^2 = 0.82$ and RMSE = 0.043 for context-condition accuracy; $R^2 = 0.70$ and RMSE = 10.2 for list-condition solution time; and $R^2 = 0.52$ and RMSE = 9.1 for context-condition solution time).

Consistent with the idea that we described in the introduction, the model shows that instructions that are formulated as operators with pre- and postconditions lead to better performance than instructions that are just a list of actions. More in detail, the model explains superior performance in the context condition as due to basically two factors. First, attempts to retrieve operators are more successful in this condition. In the list condition, the perceived state (e.g., being on the LEGS page) is different than the expected state (e.g., being in state *Direct 1* after pressing the *LEGS* key, see Table 2). Only the expected state can be used to determine the next step because the instructions make no reference to a real-world state. In the context condition, the expected and perceived states are generally the same, and, therefore, both help activate the appropriate operator through spreading activation. Therefore, operator retrieval in the list condition will fail more often than in the context condition.

A second factor is that the model is better able to cope with gaps in its knowledge in the context condition. As we discussed earlier, these gaps can occur because of failures in retrieval or, in the difficult problems, because the model finds itself in situations for which it does not have operators. The model with a context representation is much more likely to be able to pick up successful operation application after such a guess because it can recognize operators relevant to the state that appears after the guess. The ability to continue on after a knowledge gap is much more important in the difficult problems because there are more knowledge gaps. This is what accounts for the interaction between instruction and difficulty.

Interestingly, at knowledge gaps in the list condition, the model discovers operators that are "context-style" and not "list-style." Eventually the list-condition model ends up with a mixture of operators that refer to internal states and operators that have pre- and postconditions referring to the perceived state. Thus, with practice, the knowledge representation in the list condition becomes more and more like the knowledge representation in the context condition.

It is also worth noting that context instructions produce better performance because they skip unnecessary steps. For example, the first step is to push the *LEGS* key. If the FMS is already on the LEGS page, then this step is unnecessary. In 32 of the 36 problems, the FMS already displayed the LEGS page at the start of the problem. This means that optimal performance entails pressing the *LEGS* key 0.11 times per problem on average. Figure 7 shows the number of presses of the *LEGS* key in the model and the data for both conditions. Participants in the list condition press the *LEGS* key much more often than needed, whereas participants in the context condition are close to the optimal level. The model fits these proportions correctly.

A second example is the use of the *EXEC* key. This key is normally used as the last step in a route-change procedure because it commits the FMS to the route change. The list instructions (as taken from United Airlines) specify that the *EXEC* key has to be pressed not only at the end of the direct-to procedure but also at the end of the procedure to resolve a discontinuity. As Figure 8 shows, this leads to extraneous presses of the *EXEC* key in problems with a discontinuity. In the list condition, neither the participants nor the model follow the instructions to the letter because the average number of key presses on the EXEC key condition for disconti-

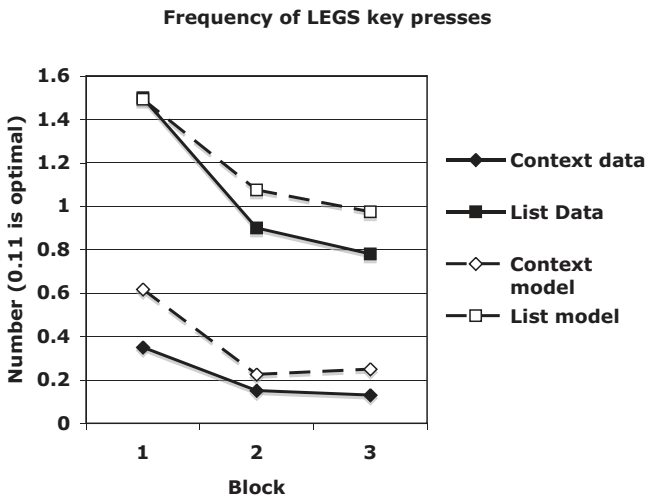


Figure 7. Frequency of presses on the LEGS key for the three blocks and two conditions, empirical data and model.

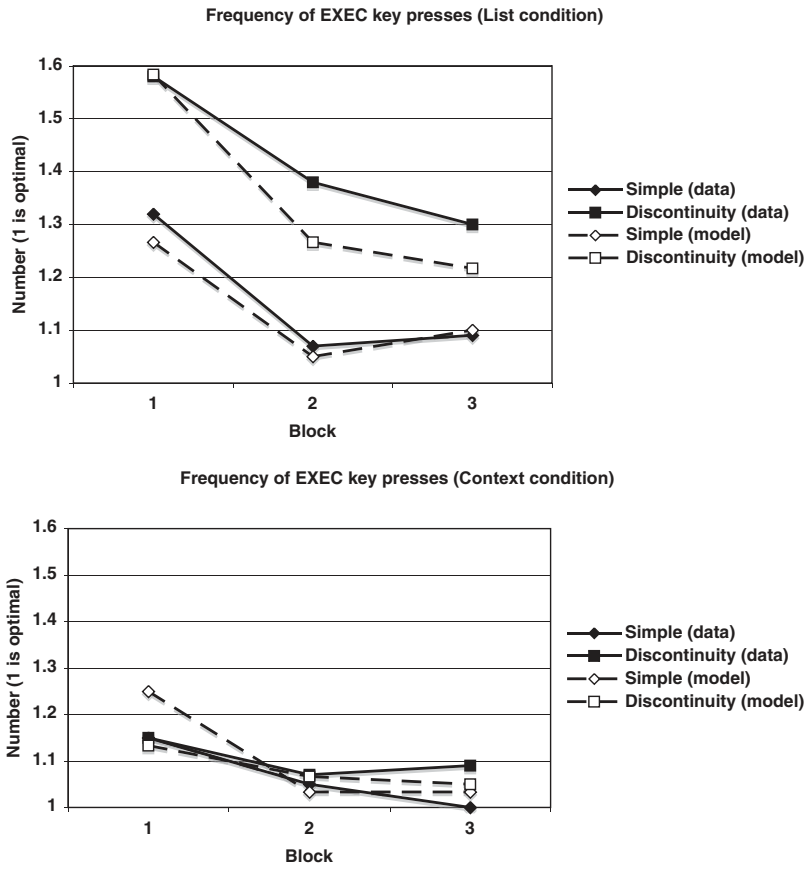


Figure 8. Frequency of presses on the EXEC key for the three blocks and two conditions, empirical data and model, and whether or not the problem involved a discontinuity (which, according to the literal list instructions, would require a second press on EXEC). Top panel represents the list condition; bottom panel represents context condition.

nuities is around 1.3 eventually, instead of the 2 that the instructions prescribe. The model learns to skip the extra *EXEC* key when it forms its own operators during exploration at knowledge gaps. As expected, these extra presses on the *EXEC* key are virtually absent in the context condition. The model also fits these data correctly.

Experiment 2

An advantage of a cognitive model is that it can be used to make predictions. In order to test the predictive power and robustness of the model, we designed a second experiment in which the instructions were identical to the first experiment, but with a new level of complexity in the actual tasks. The model predictions were generated before the actual data were collected.

A property of robustness is that a skill can be applied to situations that deviate from the original instructions. This implies the ability to assess the current situation, even if it is a situation that has not been encountered before. It also implies the ability to come up with new actions if a situation cannot be resolved with the current knowledge. In Experiment 1, we tested this by introducing hard problems that could not be solved literally with the given instructions. In Experiment 2, we introduce a second category of hard problems: problems that are halfway completed, or that are in an error state. These problems reflect an aspect of real-life interaction between pilots and the FMS that is not apparent from the way instructions are formulated, which is that pilots often share tasks on the FMS. It is possible that one of the pilots initiates a procedure and then asks the co-pilot to take over. The co-pilot then has to assess the state of the system and has to decide what steps are still needed to reach the goal. This situation is similar to the situation in which a procedure is interrupted and resumed later on because at the moment of resumption, the state of the system has to be reassessed (assuming the pilot has not remembered this state). To simulate this in the experiment, we included blocks of trials in which the task was already partially completed. To make it even harder for the participants, some of the half-completed trials contained an error. For example, in some of the problems, the destination would already be in the scratchpad, sometimes a wrong destination.

Although the model was not designed to be able to handle these cases, it should in principle be able to solve them, especially the context-condition model, because its precondition-matching strategy should be able to pick up on the current state of the system. Also, the model should be able to recover from problems containing an error at least some of the time because it has to recover from

the errors it makes during exploring unknown actions. Experiment 2 therefore was a strong test of the model because the data that it had to predict extrapolated from the data on which it had been fit. On this basis, we were able to make a prediction of the outcome of the experiment before actually conducting it.

Method

Participants. Forty students from Carnegie Mellon University were paid for their participation in the experiment (20 in the list condition, and 20 in the context condition).

Procedure. The procedure was identical to the procedure of Experiment 1, except that each of the three blocks now consisted of 24 problems. The first 12 problems were the same as in Experiment 1: 3 easy problems, followed by 3 intermediate problems, and then six hard problems. Problems 13–24 in each block were partially completed. Half of the partially completed problems contained an error. The partially completed problems, either with or without error, consisted of equal proportions of easy, intermediate, and hard problems. All problems within the partially completed half of the block were presented in random order. Participants were not informed that procedures could be partially completed, nor of the possibility that an error was already present; in fact, the instructions they received were identical to those of Experiment 1. The experiment lasted, on average, 1.5 hr.

Results

Empirical results. Figure 9 shows the proportion correct for each problem in the two conditions. The average correctness in the list condition is 78.8%; in the context condition, it is 89.0%. An ANOVA, with block and problem difficulty as within-subject factors, condition as a between-subject factor, and subject as a random factor, showed main effects of condition, $F(1, 38) = 4.5$, $MSE = 2.54$, $p = .040$; of block, $F(1, 38) = 35.3$, $MSE = 2.67$, $p < .001$; and of problem difficulty, $F(4, 152) = 26.0$, $MSE = 1.32$, $p < .001$. In addition, there were significant interactions between condition and problem difficulty, $F(4, 152) = 2.82$, $MSE = 0.14$, $p = .027$, and between block and problem difficulty, $F(4, 152) = 5.45$, $MSE = 0.24$, $p < .001$. Welch t tests, comparing the two conditions for each problem type, are listed in Table 4. Overall, Experiment 2 replicated the results of Experiment 1, and extended it by showing some effect of instruction on partially completed problems, although this effect only approached significance in the case of the partially completed problems with an error. The main effect of instructional manipulation was not as strong as in Experiment 1. This was probably due to the fact that the experiment was twice as long, and the difference between the conditions tends to get smaller with practice.

An ANOVA of the log solution times was conducted, with block and problem difficulty as within-subject factors, condition as a between-subject factor, and subject as a random factor. The average solution time for correctly solved problems in the list condition was 27.2 s, and in the context condition, it was 23.0 s, $F(1, 38) = 3.60$, $MSE = 4.53$, $p = .066$. There was a main effect of block, $F(1, 38) = 120.9$, $MSE = 52.1$, $p < .001$, indicating a learning effect; a main effect of problem difficulty, $F(4, 152) = 115.2$, $MSE = 12.9$, $p < .001$; and an interaction between problem difficulty and block, $F(4, 152) = 27.4$, $MSE = 4.12$, $p < .001$.

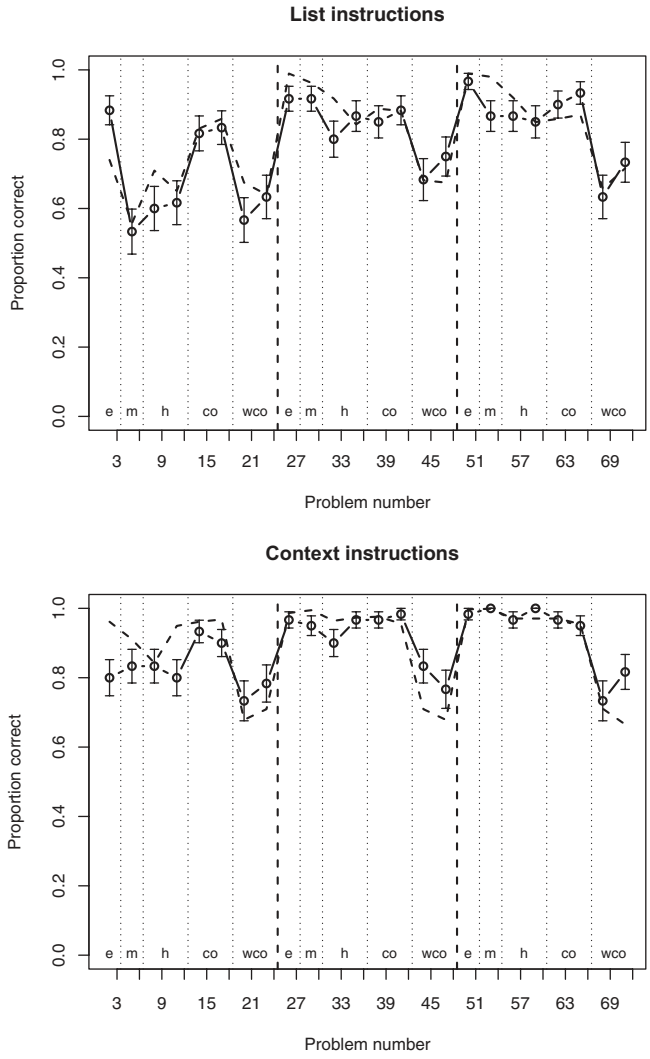


Figure 9. Proportion correct for all the 72 problems in the two conditions averaged in groups of three. The experiment is divided in three blocks with five types of problems each (e = easy, m = medium, h = hard, co = copilot, wco = copilot with error). Circles are the empirical data with standard error bars, and the dashed line represents the model prediction. Although in the real experiment co and wco problems were offered in random order, they are sorted into separate categories within a block for clarity.

Figure 10 shows the average solution times for the correct trials for each trial for both conditions combined.

Model prediction. The prediction of the model is shown alongside the data in Figures 9 and 10 and in Table 4 ($R^2 = 0.78$ and RMSE = 0.064 for list-condition accuracy; $R^2 = 0.66$ and RMSE = 0.072 for context-condition accuracy; $R^2 = 0.75$ and RMSE = 5.9 for solution times). A first noteworthy result was that the unmodified model was able to do the new problems at all because it was not designed to solve them. This shows that the model indeed exhibited the flexibility and robustness that we were seeking. For the partially completed problems without error, the predictions were very accurate. For the partially completed problems with an error, the model predicted the list condition accurately but slightly underestimated performance in the context con-

Table 4
Proportions Correct in Experiment 2 by Problem Type

Problem type	List correct	Context correct	*t*	Model list correct	Model context correct
Easy	92.2%	91.7%	$t < 1$	90.6%	98.2%
Medium	77.2%	92.8%	$t(32.0) = 2.28$, $p = .030$	83.3%	96.8%
Hard	76.7%	91.1%	$t(26.6) = 2.47$, $p = .020$	84.9%	92.6%
Copilot without error	86.9%	95%	$t(22.8) = 1.47$, $p = .15$	86.5%	96.4%
Copilot with error	66.7%	77.8%	$t(24.9) = 1.86$, $p = .074$	67.4%	69.2%

dition. This was probably due to the fact that the model has no real error recovery strategies: It will just start trying things out when it is in an unrecognized state. Participants in the context condition might be better able to recognize an error state and therefore also better able to cope with one. The latency predictions in Figure 10 are also quite accurate, including the predictions for the partially completed problems for which we had no previous data. The only exception was an underestimation of the solution times for the first few problems (similar to the underestimation in Experiment 1).

Experiment 3

The instructions for Experiments 1 and 2 were designed to achieve a maximum difference between the two conditions. The disadvantage is that the version of the context instructions that we used may have had advantages in addition to the advantage provided by a pre- and postcondition because they may generally be more readable. This might mean that participants extract additional information from the instructions that may lead to better performance. In order to separate out the different effects, we constructed four different sets of instructions. The first set of instructions consisted of the list instructions as we have used them in Experiments 1 and 2, which we refer to as the *simple list instructions*. To contrast our specific manipulation of instructions with alternatives, we constructed extended list instructions, which provided, in addition to the list instructions, an explanation of how the information on the screen corresponds to the route of the airplane, and how the line keys can manipulate this information. The idea is that additional explanation provides an understanding of the system from which participants can derive the actions they have to take in case the literal instructions do not suffice, which we refer to as the *extended list condition*. The third set consisted of the context instructions, but they are now reduced to the list instructions with a short description of the pre- and postcondition, which we refer to as the *simple context instructions*. Because it might not always be easy to recognize a particular precondition, we constructed a fourth set of instructions, the *graphics context instructions*, which consisted of the context conditions with a second addition: a screenshot of the pre- and postcondition with the relevant feature highlighted. The Appendix shows the new extended list and simple context instructions.

The model predicts, consistent with the earlier experiments, that there should be an advantage of both context conditions over the simple-list condition, and not much of a difference between the two context conditions. The model cannot predict the exact impact of the extended-list condition but has no specific reason to suppose it would offer a distinct advantage over the simple-list condition.

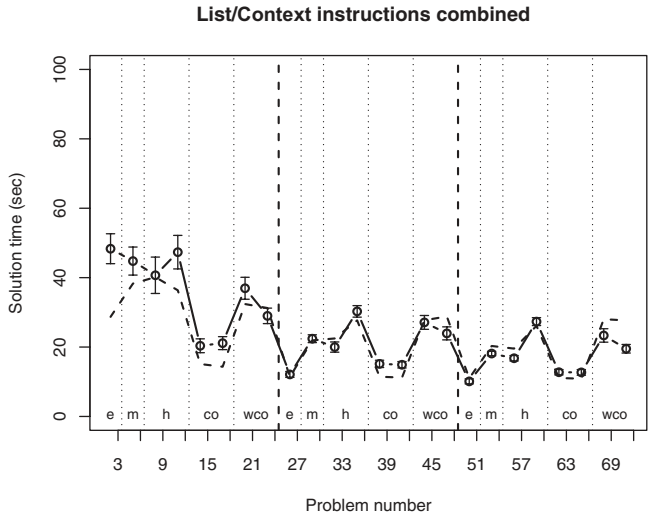


Figure 10. Average solution times for correctly solved problems for all 72 problems averaged in groups of three. The experiment is divided in three blocks with five types of problems each (e = easy, m = medium, h = hard, co = copilot, wco = copilot with error). Circles are the empirical data with standard error bars, and the dashed line represents the model prediction.

Method

Participants. One hundred students from Carnegie Mellon University were paid for their participation, with 25 participants in each of the four conditions.

Procedure. The procedure was identical to Experiments 1 and 2, except that the experiment consisted of 3 easy problems, followed by 3 medium problems, and two blocks of 18 problems. Each block of 18 problems consisted of 3 easy, 3 medium, 6 hard, 3 partially completed, and 3 partially completed with error problems. Within the two 18-problem blocks, the order of problems (including problem type) was fully randomized.

Results

Consistent with the earlier experiments, there was a main effect of block on log solution time, $F(1, 96) = 286.7$, $p < .001$, and on correctness, $F(1, 96) = 51.1$, $p < .001$, a main effect of problem difficulty on log solution time, $F(4, 384) = 158.8$, $p < .001$, and on correctness, $F(4, 384) = 67.6$, $p < .001$, as well as an interaction between block and problem difficulty on solution time, $F(4, 384) = 24.7$, $p < .001$, and correctness, $F(4, 384) = 6.23$, $p < .001$. However, contrary to the two earlier experiments, there was no interaction between condition and problem type.

Given the predictions of the model, there should be an advantage of both context conditions over both list conditions, and there should be no differences between either the two context conditions or the two list conditions. In fact, the difference between the two list conditions and the two context conditions is quite significant in the case of time and marginally so for accuracy (21.3 s vs. 26.5 s), $t(92.2) = 2.97$, $p = .004$ (83.9% vs. 79.9%); $t(93.7) = 1.66$, $p = .10$. The predictions of the model are that both context conditions should take 21.8 s and have 90.9% accuracy, whereas both list conditions should take 28.6 s and have 80.8% accuracy. Participants are performing somewhat less accurately than a priori predictions of our model. As for the two context conditions, there were no significant differences between simple context and graphics context (22.3 s vs. 20.3 s), $t(40.3) = 1.43$, $p > .10$ (84.9% vs. 82.9%); $t(46.6) = 0.66$, $p > .10$. The differences between the simple-list and extended-list conditions were also not significant (26.0 s vs. 26.9 s), $t(44.1) = 0.80$, $p > .10$ (77.6% vs. 82.1%); $t(43.3) = 1.19$, $p > .10$. Figure 11 depicts detailed outcomes of the experiment with the model predictions ($R^2 = 0.79$ and RMSE = 0.067 for list-condition accuracy, $R^2 = 0.44$ and RMSE = 0.119 for context-condition accuracy, $R^2 = 0.56$ and RMSE = 9.2 for list-condition solution time, and $R^2 = 0.56$ and RMSE = 6.9 for context-condition solution time).

Discussion

Although the difference is smaller in Experiment 3 than in Experiments 1 and 2, context instructions still provide a significant improvement over list instructions both in terms of correctness and solution time. In contrast, an extended-list instruction provides no significant improvement over simple-list instructions. The graphics context instruction also leads to no significant advantage over

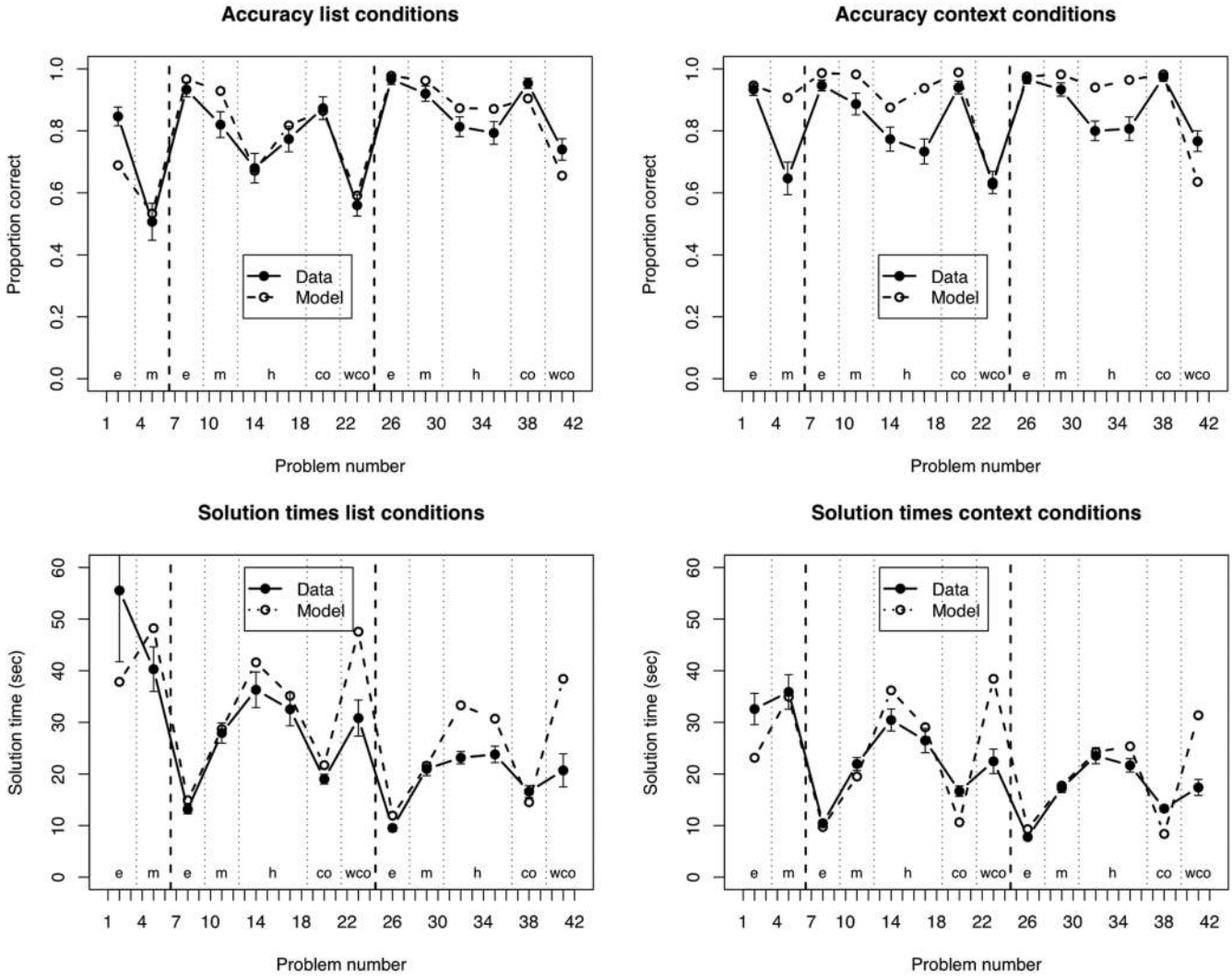


Figure 11. Accuracies and solution times for all 42 problems, averaged in groups of three (e = easy, m = medium, h = hard, co = copilot, wco = copilot with error). Both list conditions and both context conditions are averaged. Except for the first six problems, the problems were presented in random order within a block but are sorted by problem type in this graph for clarity.

the simple-context instructions. The interaction between problem difficulty and condition disappeared in Experiment 3, but this is probably due to the smaller difference in accuracy between the context and the list condition as a result of the reduced context instructions. Alternatively, the fact that the order in which the problem types were offered was now fully randomized may have reduced the power to find the interaction because each participant now receives the problem types in a different order. The model still captures the main effects in the data, although it slightly overestimates the correctness in the context condition.

General Discussion

In this article, we presented a model of skill acquisition that takes into account that cognitive control is both internal, in terms of an internal control state that keeps track of progress, and external, in the sense that the environment can prompt the next action. It thereby shows how two traditions in explaining skilled behavior, the traditional problem-solving approaches and the embedded cognition approaches, can be combined, alleviating the weaknesses of both. From the traditional approaches, we take the concept of a task structure that can guide the right sequence of actions and that is open to change and discovery. We go beyond those approaches by explaining how people can respond to unexpected situations and how they can fill in the gaps in their knowledge. The key to this is the shift of the focus of control from an internal representation to perception, which is more consistent with embedded models of skilled performance, and thereby inherits the robustness of such models. Use of a minimal task structure allows the model to go beyond simple embedded models in the sense that it can base its performance on both instruction and experience.

The model gives a detailed account of the various stages of learning. It starts out with an incomplete set of knowledge and uses discovery learning to fill in the gaps. Although knowledge is still declarative, behavior is characterized by a memory retrieval process, in which perception and possibly an internal state cue memory for the right action. This retrieval process is slow, with a high probability that a wrong operator is retrieved. The process of production compilation gradually transforms these representations into productions that directly map perception onto action, producing the tight coupling that embedded cognition theorists propose. This offers a solution to what many have indicated as an ignored aspect of problem solving: Information is in the world as well as in the head (e.g., Brooks, 1991; Clark, 1997; Hutchins, 1995). A context-style task representation allows people to offload control to the world: Given a proper understanding of the task, people can select their actions on the basis of not only an internal state but also the state of the world.

The model assumes that the representation of individual steps that make up a skill consists of an action with pre- and postconditions. This implies that instructions that are formulated in these terms should lead to better performance than instructions that just list the actions. Our experiments showed that context instructions lead to better performance than list instructions in terms of accuracy, solution time, and flexibility, supporting the idea that that representation is close to our internal representation of task knowledge. In addition, a model constructed on the basis of these representations provides for an accurate fit and prediction of both accuracy and latency data. Experiment 3 showed that other additional information, like additional explanations of how the system works or screenshots of the display, has little or no impact on performance. This is consistent with Kieras and Bovair's (1984) conclusion that a mental model is only useful insofar as it supports exact and specific control actions and that general principles, metaphors, and analogies are generally not very useful. Recent research by Catrambone and Yuasa (2006) also supports the usefulness of connecting preconditions to actions in instructions. In their experiment, participants had to formulate database queries and received instructions that either emphasized the connection between conditions and action or mainly elaborated on just the conditions. They found that the instructions connecting condition and action produced significantly lower error rates.

Flexibility is evidenced in problems that required generalization (the hard problems) and in problems in which the task had to be picked up in the middle, possibly with an error (the copilot problems). Nevertheless, even when given list instructions, both the model and the participants developed flexibility, only slower and less complete. The model explains this by an exploration process in which operators are discovered, which have pre- and postconditions that can be related to the environment. One direct recommendation from this research is that the standard list procedures used in pilot training should be replaced by procedures for which the conditions and results of each action are explained explicitly. Are context instructions always better than list instructions? Probably so in situations in which useful state information can be extracted from the environment. Only in cases in which the environment cues the wrong actions is it possible that list instructions lead to superior performance (see Norman, 1988, for examples of how poor design leads to problems).

Besides empirical evidence, we have presented a model fitted to the data of Experiment 1. This model, in which the only difference between the two conditions was the representation of the instructions, could reproduce the accuracy and solution time data very well, and it was also able to explain some of the details on how often particular (the *LEGS* and the *EXEC*) keys were pressed. The model generalized to novel situations: It predicted the outcome of Experiments 2 and 3, which contained partially completed trials for which the model was not designed. The model not only solved these trials, it did so with accuracy and solution time comparable to those of participants. We believe that being able to make predictions is an important property of cognitive models because it counters some of the criticisms that cognitive models can fit any data set given enough free parameters (Roberts & Pashler, 2000). Generalizing the present model to other tasks is relatively straightforward because a large portion of it is task independent. A model for a different task would require the appropriate set of operators for that task and additional productions to implement actions that are not part of the FMS task. Otherwise, the mechanism for matching preconditions and carrying out operator actions carried over to any other task.[3]

The approach to skill acquisition taken in the present article can also be extended to the domain of multitasking. One proposal to understand how people are able to interleave multiple tasks efficiently is to assume a *general executive* (Kieras, Meyer, Ballas, & Lauber, 2000). The general executive is a top-down process that schedules processing between the different tasks. The problem of this approach is that such a scheduler runs into problems of computational intractability and that the scheduler needed to interleave two tasks is

[3] We are presently working on such a general system (ACT-R/Lisa), see http://www.ai.rug.nl/~niels/lisa.html for more information.

sometimes more complex than the task representation itself. If the steps in each of the tasks are represented as operators with preconditions, then the scheduling can be taken care of by a bottom-up process: Each time an operator of either task matches the current perceptual state, that operator will be executed, assuming the appropriate cognitive resources are available. This approach has been successful in modeling experiments of perfect time-sharing (Schumacher et al., 2001), in which participants, after sufficient training, are able to do two choice reaction tasks in parallel without any dual-task interference. By representing each of the steps in both choice-reaction tasks as operators with preconditions, the model was able to achieve perfect time-sharing in the same manner as participants did (Salvucci & Taatgen, 2008).

References

Agre, P. E., & Shrager, J. (1990). Routine evolution as the microgenetic basis of skill acquisition. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 694–701). Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–406.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem situations. *Psychological Review, 94,* 192–210.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of cognition. *Psychological Review, 111,* 1036–1060.

Anderson, J. R., Taatgen, N. A., & Byrne, M. D. (2005). Learning to achieve perfect timesharing: Architectural implications of Hazeltine, Teague, and Ivry (2002). *Journal of Experimental Psychology: Human Perception and Performance, 31,* 749–761.

Botvninick, M., & Plaut, D. C. (2004). Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review, 111,* 395–429.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, 47,* 139–159.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.

Catrambone, R., & Yuasa, M. (2006). Acquisition of procedures: The effects of example elaborations and active learning exercises. *Learning and Instruction, 6,* 139–153.

Chi, M. T. H. (2006). Two approaches to the study of experts' characteristics. In K. A. Ericsson, N. Charness, P. J. Feltovich, & R. R. Hoffman (Eds.), *The Cambridge handbook of expertise and expert performance* (pp. 21–30). New York: Cambridge University Press.

Clark, A. (1997). *Being there: Putting brain, body and world together again*. Cambridge, MA: MIT Press.

Crossman, E. R. F. W. (1959). A theory of the acquisition of speed-skill. *Ergonomics, 2*(2), 153–166.

Fennell, K., Sherry, L., Roberts, R. J., & Feary, M. (2006). Difficult access: The impact of recall steps on flight management system errors. *International Journal of Aviation Psychology, 16*(2), 175–196.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2,* 189–208.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of human learning* (pp. 243–285). New York: Academic Press.

Fitts, P., & Posner, M. I. (1967). *Human performance*. Monterey, CA: Brooks/Cole.

Glenberg, A. M., & Robertson, D. A. (1999). Indexical understanding of instructions. *Discourse Processes, 28*(1), 1–26.

Hayhoe, M. M., Shrivastava, A., Mruczek, R., & Pelz, J. B. (2003). Visual and motor planning in a natural task. *Journal of Vision, 3,* 49–63.

Hutchins, E. (1995). How a cockpit remembers its speeds. *Cognitive Science, 19,* 265–288.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science, 8,* 191–219.

Kieras, D. E., Meyer, D. E., Ballas, J. A., & Lauber, E. J. (2000). Modern computational perspectives on executive mental processes and cognitive control: Where to from here? In S. Mondell & J. Driver (Eds.), *Control of cognitive processes: Attention and performance XVIII* (pp. 681–712). Cambridge, MA: MIT Press.

Kirsh, D. (1995). The intelligent use of space. *Artificial Intelligence, 73,* 31–68.

Kirsh, D., & Maglio, P. (1994). On distinguishing epistemic from pragmatic action. *Cognitive Science, 18,* 513–549.

Land, M., Mennie, N., & Rusted, J. (1999). The roles of vision and eye movements in the control of activities of daily living. *Perception, 28,* 1311–1328.

Larkin, J. H. (1989). Display-based problem solving. In D. Klahr & K. Kotovsky (Eds.), *Complex information processing: The impact of Herbert A. Simon* (pp. 319–341). Hillsdale, NJ: Erlbaum.

Lebiere, C., Wallach, D., & Taatgen, N. A. (1998). Implicit and explicit learning in ACT-R. In F. E. Ritter & R. M. Young (Eds.), *Second European conference on cognitive modelling* (pp. 183–189). Nottingham, England: Nottingham University Press.

Logan, G. D. (1988). Toward an instance theory of automization. *Psychological Review, 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. 1–56). Hillsdale, NJ: Erlbaum.

Newell, A., & Simon, H. (1963). GPS, a program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought* (pp. 279–293). New York: McGraw-Hill.

Norman, D. A. (1988). *The psychology of everyday things*. New York: Basic Books.

Prinz, W. (1997). Perception and action planning. *European Journal of Cognitive Psychology, 9*(2), 129–154.

Roberts, S., & Pashler, H. (2000). How persuasive is a good fit? A comment on theory testing. *Psychological Review, 107,* 358–367.

Roelfsema, P. R., Lamme, V. A. F., & Spekreijse, H. (1998, September). Object-based attention in the primary visual cortex of the macaque monkey. *Nature, 395,* 376–381.

Salvucci, D. D., & Taatgen, N. A. (2008). Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review, 115,* 101–130.

Schumacher, E. H., Seymour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E., et al. (2001). Virtually perfect time sharing in dual-task performance: Uncorking the central cognitive bottleneck. *Psychological Science, 12*(2), 101–108.

Sherry, L., Polson, P., Fennell, L., & Feary, M. (2002). *Drinking from the fire hose: Why the FMC/MCDU can be hard to learn and difficult to use* (Honeywell Publication No. C69–5370–022). Morristown, NJ: Honeywell.

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: From dual tasks to complex dynamic skills. *Cognitive Science, 29*(3), 421–455.

Taatgen, N. A. (2007). The minimal control principle. In W. Gray (Ed.), *Integrated models of cognitive systems* (pp. 368–379). Oxford, England: Oxford University Press.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say "broke"? A model of learning the past tense without feedback. *Cognition, 86*(2), 123–155.

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45*(1), 61–76.

Appendix

Simple-Context and Extended-List Instructions in Experiment 3

Context condition:

When Air Traffic Control gives you a directive to proceed directly to some waypoint, use the following list of steps to accomplish this:

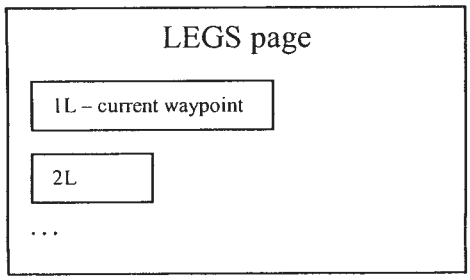
1. When you are not yet on the LEGS page, **Press the LEGS key** to get to the LEGS page.
2. When you are on the LEGS page, **Enter the desired waypoint in the scratchpad** so you can put it into the flight plan.
3. When you have your new destination in the scratchpad, **Push the 1L key** to move the waypoint from the scratchpad into the first line of the flight plan.
4. If "route discontinuity" appears on the screen, **follow the procedure to remove discontinuities** to remove the discontinuity.
5. When you have modified your route and have removed all route discontinuities, **Verify the route on the Navigational Display** to make sure that the FMS has made the changes as you intended them.
6. When you have verified that the route modification looks ok on the Navigational Display, **Press EXEC** to finalize the route modification.

There is a separate procedure for removing discontinuities:

1. When you are not yet on the LEGS page, **Press the LEGS key** to get to the LEGS page.
2. If you are on the LEGS page and there is a discontinuity in one of the lines, **Press the line select key after the discontinuity** to copy the waypoint after the discontinuity in the route into the scratchpad.
3. If you have the waypoint after the discontinuity in your scratchpad, **Press the line key with the THEN prompt** to copy the waypoint after the discontinuity into the line with the discontinuity, and reconnect the route.

Extended List condition:

The first left line (1L) of the LEGS page is used to show the waypoint that the airplane is currently flying to (see diagram). The waypoint at 2L is the next one on the route, and so on.



Whenever a waypoint is added to 1L (replacing what was there), the plane will fly directly to that waypoint.

When the added waypoint was not on the route, a discontinuity in the route occurs. This is signaled by the word "Discontinuity" and an empty space next to one of the line keys (2L, 3L, etc.). If one of the next waypoints on the route is copied in this empty space, the route becomes continuous.

The scratchpad (the empty space below 6L) can be used to copy and paste waypoints from and to line keys (1L, 2L, etc.). For example, when the scratchpad is empty, a mouse click on 1L would copy the waypoint from 1L to the scratchpad. When the scratchpad holds a waypoint, a click on 1L would paste the waypoint from the scratchpad to 1L. The keyboard can also be used to type waypoints in to the scratchpad.

The LEGS key on the keyboard is used to access the LEGS page. Next-page and Prev-page keys are used to browse through the waypoints on the route. The NAV display is used to view the results of a route modification. The EXEC key makes a modification permanent.

When Air Traffic Control gives you a directive to proceed directly to some waypoint, go through the following steps:

1. Press the LEGS key.
2. Enter the desired waypoint in the scratchpad.
3. Push the 1L key.
4. If the word "discontinuity" appears on the screen, follow the procedure to remove discontinuities.
5. Verify the route on the Navigational Display.
6. Press EXEC.

There is a separate procedure for removing discontinuities:

1. Press the LEGS key.
2. Press the line select key after the discontinuity.
3. Press the line key with the THEN prompt.

Received January 11, 2007
Revision received December 10, 2007
Accepted December 13, 2007 ■