CHAPTER 6  

Examples versus Rules
6: Examples versus Rules

Acknowledgment
This chapter is written in collaboration with Dieter Wallach, who has developed the model in section 6.3.

An important topic in skill learning is the question of what type of knowledge is learned. Two explanations dominate the discussion. The rule-learning explanation assumes rules are learned by generalizing examples. The instance-based explanation assumes a set of examples is retained. This explanation assumes improved performance can be explained by the fact that a solution is retrieved from memory instead of being calculated again. Both types of explanation are compatible with ACT-R, and this chapter will explore the question of how to choose between the two. The central idea will be that the type of learning with the best expected gain will dominate performance. This will be demonstrated using two models. The first of these models the Sugar Factory task, a task in which performance can be explained by instance learning alone. The second models the Fincham task, in which the expected gain of both the use of instances and the use of rules can be examined in detail.

6.1 Introduction

The models in the previous chapter made an important assumption about learning new skills, the assumption that they are represented as production rules. An alternative account of skill learning is that people store examples, and later retrieve these examples if they encounter the same or a similar situation.

The question whether skills are realized as abstract rule-like entities or as sets of concrete instances is one of the central distinctions in cognitive science, spreading across fields as diverse as research on memory, problem solving, categorization or language learning (Logan, 1988; Hahn & Chater, 1998; Redington & Chater, 1996; Plunkett & Marchman, 1991; Lebiere, Wallach, & Taatgen, 1998). Hahn and Chater (1998) proposed that the distinction between instance- and rule-based learning mechanisms cannot be based on different types of representations, but must be seen within the framework of their use in problem solving. We extend their argument and emphasize the necessity of an integrated investigation of human skill acquisition using a comprehensive theory of cognition.

The view of skill acquisition as learning and following abstract rules has dominated theories of skill acquisition over the last decades, whether encoded in production systems (Newell & Simon, 1972; Anderson, 1993), stored as logical implications or represented in classifier systems (Holland, Holyoak, Nisbett & Thagard, 1986). While these approaches differ in many aspects, they share the assumption that cognitive skills are realized as abstract rules that are applied to specific facts when
solving problems. In ACT-R, it is assumed that people start out with concrete examples of previous problem solving episodes that are generalized to abstract rules. These rules can be applied in subsequent problem solving and can thus account for increased performance. Discontinuous improvements in cognitive performance (Blessing & Anderson, 1996) can be taken as further evidence for the acquisition of rules. While Anderson (1993) describes the view that cognitive skills are realized as (production) rules as "one of the most important discoveries" in cognitive psychology, Logan (1988) argues for domain-specific instances as the basis for cognitive skills. According to this instance theory, general-purpose procedures or algorithms are applied to solve novel problems. Each time such a procedure is used in problem solving, its solution is retained as a separate instance. For new problems, the solution can be calculated, or a previous one can be retrieved and applied to the current problem. The retrieved solution can be used as a whole, in part, or in an adapted version to obtain the solution of the new problem.

An important source of evidence for the instance-based approach is the fact that repeating a certain specific example of a problem increases performance on this example, but not on other ones. The fact that participants cannot verbalize abstract knowledge about the problems solved is frequently cited as further evidence against some form of generalization, as implied by rule-based skill theories. ACT-R, however, assumes that rules themselves cannot consciously be inspected, so this second source of evidence is not as convincing as the first.

Evidence for the fact that knowledge is represented as production rules comes from research on the directional asymmetry of rules. A production rule has two parts, a condition and an action, which we informally denote as ‘IF condition THEN action’. In a production system, control always flows from the condition to the action. In many practical cases, the condition and the action are both part of a pattern, for example the pattern AB. A rule like ‘IF A THEN B’ can be used to complete the pattern given A. In an instance approach, the pattern AB can be stored as an instance, and retrieved given either A or B. If participants are trained to complete some pattern AB on the basis of A, a rule approach predicts that they learn the rule ‘IF A THEN B’, and the instance approach predicts that they learn the instance AB. If participants are consequently asked to complete AB on the basis of B, the instance approach would not predict a decrease in performance. The rule-based approach, however, suggests that a new rule would have to be learned for the ‘IF B THEN A’ case, resulting in worse performance.

Another apparent source of evidence stems from the fact that rules are more general than instances, which are assumed to be represented in a relatively unprocessed form (Redington & Chater, 1996). If participants show increased performance on examples they have not encountered before, some generalized knowledge can be postulated as the basis of the observed performance. This second source of evidence is, however, unreliable. It assumes that stored examples can only be used when the
new example is literally identical to one of the stored examples. If one or more old examples (or fragments of them) can be used to improve performance on a new example in a less direct fashion, generalization is also possible in an instance-based setting. Consequently, if generalization in transfer experiments is used as evidence against instance theory, it must be made clear that the answer to a certain problem cannot easily be derived from answers to previous problems. As Redington and Chater (1996) have pointed out, surprisingly simple models, relying on represented fragments of observed stimuli, can perform exceedingly well in transfer tasks without acquiring any abstract knowledge. An example of such a model will be discussed in section 6.3 when we demonstrate the scope of a purely instance-based approach in accounting for data that Broadbent and his colleagues (Broadbent, 1989) have interpreted as evidence against ACT-R’s claim that production rules are learned on the basis of examples. Their results on dissociations between knowledge and performance seem to imply that participants can acquire rules to successfully operate complex systems without showing an increased performance in answering questions about the system’s behavior. Our instance model will provide a very simple explanation for this dissociation result.

6.2 Learning strategies

The learning mechanisms in ACT-R are all quite basic, and can be used in several different ways to achieve different results. In chapter 4, it is argued that the learning mechanisms of ACT-R correspond to the psychological notion of implicit learning, since they are always at work, do not change due to development and show few individual differences. Explicit learning, on the other hand, is tied to intentions — to goals in ACT-R terms — and can better be explained by a set of learned strategies.

In this chapter we will discuss a paradigm for skill learning that involves both implicit learning and an explicit strategy. Figure 6.1 shows an overview of this paradigm. First we assume people have some initial method or algorithm to solve the problem. Generally this method will be time-consuming or inaccurate. Each time an example of the problem is solved by this method, an instance is learned. In ACT-R terms, an instance is just a goal that is popped from the goal stack and is stored in declarative memory. Since this by-product of performance is unintentional, it can be considered as implicit learning.

Other types of learning require a more active attitude. If the initial method is too time consuming, one may try to derive an abstraction to increase efficiency. If the initial method leads to a large number of errors, new relationships in the task may be deduced or guessed in order to increase performance. The next step, from abstraction to production rule, can only be made if the abstraction is simple enough to convert to a production rule. Since proceduralization is usually not considered
something that is under conscious control, it is a form of implicit learning as well. This idea is not entirely consistent with ACT-R’s production compilation mechanism. We will return to this issue in the discussion at the end of the chapter. Both the application of abstractions and the firing of new production rules will create new instances. Regardless of what is going on due to explicit learning, implicit learning keeps accumulating knowledge.

If we have that many ways of learning, what type of learning will we witness in a particular experiment? To be able to answer this question we go back to the principle of rational analysis. According to this principle, the type of learning that will be principally witnessed is the type that will lead to the largest increase in performance. If we have a task in which it is very hard to discover relationships or abstractions, learning will be characterized primarily by implicit instance learning. In tasks where each instance is different from the others, but where generalization is relatively easy, the best explanation of performance will probably involve the learning of rules.

Before discussing specific models, both learning instances and production rules will be examined in more detail. The abstractions used in this chapter are still very simple structures, and will be elaborated in the next chapter.

**Instance-based learning**

The last thirty years have seen a number of different experimental paradigms investigating the concept of implicit learning in domains as diverse as learning artificial grammars (Reber, 1967), sequence learning (Willingham, Nissen, & Bullemer, 1989) or learning to control complex systems (Berry & Broadbent, 1984). All these studies share the claim that participants learn more about structural properties of the tasks than they are able to verbalize. To explain these findings, an implicit mode of learning has been distinguished from an explicit mode. Berry and Broadbent (1995) characterize the implicit mode as

[…] a process whereby a person learns about the structure of a fairly complex
stimulus environment without necessarily intending to do so, and in such a way that the resulting knowledge is difficult to express.

In opposition to this characterization they refer to explicit learning as involving […] deliberate attempts to solve problems and to test hypotheses, and people are usually aware of much of the knowledge that they acquired.

The distinction between two learning modes has not remained unchallenged (c.f. Perruchet & Amorim, 1992; Perruchet & Pacteau, 1990; Buchner, 1994) but is cited frequently as evidence against the conception of declarative knowledge as the source for the acquisition of procedural knowledge as is assumed in the ACT-framework. Broadbent (1989) argues that the study of Berry and Broadbent (1984) contradicts the ACT claim since participants seem to learn rules for successfully operating a complex system without being able to consciously state these rules. Berry and Broadbent (1984) even found negative correlations between task performance and the ability to answer specific questions about the system’s behavior.

In section 6.3 we propose an explanation for the reported dissociation between knowledge and performance by analyzing instance-based learning in an ACT-R model and comparing it to Logan’s instance theory.

**Learning production rules**

In the previous chapter some strategies for learning task-specific rules were discussed. We will now extend those methods to a general scheme for procedural learning in ACT-R. Both the property-retrieval and the find-fact-on-feedback strategy have the desirable property that they can be used for several different tasks. The implementation of these strategies in terms of production rules is, however, rather ad hoc. This becomes an issue if the question of how these strategies themselves are learned is raised. In this chapter we will, therefore, propose a more general approach to learning new production rules. The idea is to have a standard method to construct a dependency, the declarative memory structure needed for a new production rule. Explicit learning strategies can extend this standard method. The advantage is that a learning strategy no longer has to take care of the whole process of creating a dependency, but only modifies some of the details.

It is important to note that the method of learning new productions presented here has two aspects. On the one hand, some principled decisions are made that have psychological relevance. On the other hand, there is a “programming” aspect involved: the method must produce the right rules. As a consequence, some, but not all steps in the production learning process are defensible in psychological terms.
The many constraints ACT-R poses on production rules actually simplify the problem of finding this basic method of production rule learning. Consider the most common type of production rule: a rule that matches the goal, retrieves a fact from declarative memory, and modifies the goal:

```
IF the goal has a certain type and satisfies certain properties
AND there is a fact in declarative memory that satisfies certain constraints
THEN modify one or more slots of the goal
```

The dependency necessary to learn this rule requires four principal components: the dependency itself, an example goal before the desired rule is executed, an example solution after the desired rule is executed, and the fact that is retrieved. Let us examine these four components and investigate how they may be derived.

The easiest component is the example goal. Assuming rules are derived at the point they are needed, the example goal is actually the current goal at the moment the assembly of a dependency is started. The next component is the dependency itself. Since ACT-R requires that all elements in declarative memory are former goals themselves (apart from chunks acquired through perception), the dependency must be pushed onto the goal stack at some point. The best time to do this is right at the beginning, in order to change the context from normal processing to a production learning setting. Since any goal setting may be appropriate for learning new rules, a rule is needed that pushes a dependency as a subgoal regardless of the current goal. As we already mentioned, the current goal is one of the four components needed, so we immediately stick it into its rightful place: the goal-slot of the dependency:

```
IF the goal is anything
THEN push as a subgoal a dependency with the original goal in the goal slot of the dependency
```

This rule always matches, and can interrupt normal information processing at any moment. The rule has a high cost associated with it, since it will be followed by extra processing that is not directly necessary for normal performance. The rate at which this rule will fire is directly related to the rules it competes with. If competing rules have high expected gain values, this rule will fire rarely. If competing rules have low expected gains, due to the fact that they are inaccurate or costly, this rule will fire more often. So the frequency with which dependencies are produced depends on the amount and quality of the knowledge that is already available. This is the same mechanism as the search-reflection trade-off discussed in the previous chapter.

After the dependency-pushing rule has fired, we end up with a dependency on top of the goal stack. This is illustrated in figure 6.2a: on top of some arbitrary task goal X, a dependency has been pushed as a subgoal. Only one slot of the dependency is
Figure 6.2. General method to create dependencies on the fly. (a) a dependency is pushed. (b) the copy of the original goal is pushed with a place holder for the retrieved fact. (c) the goal is modified using some retrieved fact. (d) both the modified goal and the dependency are popped, leaving a completed dependency structure.
Learning strategies

filled: the goal slot. The next step is to fill in the remaining slots of the dependency, as far as necessary. The main two slots to fill are the modified slot and the constraints slot. Some way has to be found to propose some modified goal. At this point we need some explicit learning strategy that can reason out the next step, take a guess or whatever. In order to take this next step, however, we need to restore the original goal context. This is accomplished by pushing a copy of the original goal as a new subgoal, and creating a placeholder for the retrieved fact in that subgoal.

\[
\text{IF } \text{the goal is a dependency and the modified slot is nil and } G \text{ is in the goal slot of the dependency} \\
\text{THEN push a copy } GC \text{ of } G \text{ as a subgoal, set the learn flag of } GC \text{ to true, and create a place holder in the retrieved slot of } GC \\
\text{AND put } GC \text{ in the modified slot of the dependency and set the constraints slot of the dependency to the place holder}
\]

After this rule has fired, the goal stack contains three items: the original goal, a dependency, and a copy of the original goal (figure 6.2b). The copy of the original goal has its learn flag set to true, so rules that implement explicit learning strategies are allowed to fire. The next step is that the copy of the goal is modified. This may be due to explicit learning strategies, but may also be ‘regular’ problem-solving steps (figure 6.2c). Once the goal is modified using some fact that is retained in the retrieved slot, it is popped while removing the learn flag:

\[
\text{IF the goal is has its learn-flag set to true and the retrieved slot the goal is not nil} \\
\text{THEN set the learn-flag to nil and pop the goal}
\]

At that point, further slots of the dependency may be filled, the dependency itself is popped, and ACT-R’s production compilation mechanism creates a new production rule. Now we are back in the original situation in the original goal (figure 6.2d), but with a new production rule that can modify it.

The advantage of the method outlined above is that learning strategies do not have to handle dependencies themselves, which is a big hassle. A learning strategy now only needs to recognize the learn-flag, and modify the goal while putting some fact in the retrieved slot of the goal. The method can also be modified slightly to produce production rules that push a subgoal instead of retrieving a fact. This is accomplished by simply using the stack slot of the dependency instead of the constraints slot.

The important thing to note in the method above is that procedural learning is part of normal processing, in the sense that it can be initiated at any moment. The fact that the goal needs to be copied in the subgoal, and some of the manipulations in this subgoal, are a bit awkward from a cognitive perspective. In the next chapter we will
6. Examples versus Rules

pull out all knowledge-based processing from the dependency subgoal. In that way, the actual process of learning a production rule becomes more like an implicit-learning mechanism.

6.3 Sugar Factory

In contrast to rule-based approaches that conceptualize skill acquisition as learning of abstract rules, theories of instance-based learning argue that the formation of skills can be understood in terms of the storage and deployment of specific episodes or instances (Logan, 1988; 1990). According to this view, abstraction is not an active process that results in the acquisition of generalized rules, but rule-like behavior emerges from the way specific instances are encoded, retrieved and deployed in problem solving. While ACT-R has traditionally been associated with a view of learning as the acquisition of abstract production rules (Anderson, 1983; 1993), we present a simple ACT-R model that learns to operate a dynamic system based on the retrieval and deployment of specific instances (i.e. chunks) which encode episodes experienced during system control. The ACT-R model will be compared to a model by Dienes and Fahey (1995). This comparison will involve both the accuracy of the predictions and the assumptions made by each of the models.

The Task

Berry & Broadbent (1984) used the computer-simulated scenario Sugar Factory to investigate how subjects learn to operate complex systems. Sugar Factory is a dynamic system in which participants are supposed to control the sugar production $sp$ by determining the number of workers $w$ employed in a fictional factory. The behavior of Sugar Factory is governed by the following equation:

$$sp_t = 2w_t - sp_{t-1} + \text{random component} (-1, 0, \text{or } 1)$$

The number entered for the workers $w$ can be varied in 12 discrete steps $1 \leq w \leq 12$, while the sugar production changes discretely between $1 \leq sp \leq 12$. To allow for a more realistic interpretation of $w$ as the number of workers and $sp$ as tons of sugar, these values are multiplied in the actual computer simulation by 100 and 1000, respectively. If the result according to the equation is less than 1, $sp$ is simply set to 1. Similarly, a result greater than 12 leads to an output of 12. Participants are given the goal to produce a target value of 9000 tons of sugar (so $sp=9$) on each of a number of trials. They are given no information at all about the relationship between present output, number of workers and previous output.
The models
Based on Logan’s instance theory (1988; 1990) Dienes & Fahey (1995) developed a computational model (the D&F model) to account for the data they gathered in an experiment using the Sugar Factory scenario. According to instance theory, encoding and retrieval are intimately linked through attention: encoding a stimulus is an unavoidable consequence of attention, and retrieving what is known about a stimulus is also an obligatory consequence of attention. Logan’s theory postulates that each encounter of a stimulus is encoded, stored and retrieved using a separate memory trace. These separate memory traces accumulate with experience and lead to a “gradual transition from algorithmic processing to memory-based processing” (Logan, 1988, p. 493). The ACT-R model is also based on Logan’s ideas, but differs in the way they are worked out.

Both models assume some algorithmic knowledge prior to the availability of instances that could be retrieved to solve a problem. Dienes & Fahey (1995, p. 862) observed that 86% of the first ten input values that subjects enter into Sugar Factory can be explained by the following rules:

1. If the sugar production is below (above) target, then increase (decrease) the amount of workers with 0, 100, or 200.
2. For the very first trial, enter a work force of 700, 800 or 900.
3. If the sugar production is on target, then respond with a workforce that is different from the previous one by an amount of -100, 0, or +100 with equal probability.

While this algorithmic knowledge is encoded in the D&F model by a constant number of prior instances that could be retrieved in any situation, ACT-R uses simple production rules to represent this rule-like knowledge. The number of prior instances encoded is a free parameter in the D&F model that was fixed to give a good fit to the data reported below. There is no equivalent parameter in the ACT-R model.

Logan’s instance theory predicts that every encounter of a stimulus is stored. The D&F model, however, only stores instances for those situations in which an action successfully leads to the target. All other situations are postulated to be forgotten immediately by the model. ACT-R, on the other hand, encodes every situation, irrespective of its result. The following chunk is an example of an instance stored by the ACT-R model:

```
transition1239
  ISA transition
  STATE 3000
  WORKER 800
  PRODUCTION 12000
```

The chunk encodes a situation in which an input of 800 workers, given a current production of 3000 tons, led to subsequent sugar production of 12000 tons.
The assumption that only successful instances are stored is not problematic in itself. The problem is that the D&F model uses a “loose” definition of what is successful. Due to the random component in the equation the outcome may be 1000 more or less than expected. Therefore an output of between 8000 and 10000 was considered successful by the model. This generous scheme of success was not available to participants: for them only an outcome of 9000 meant success.

Retrieving instances

In the D&F model each stored instance “relevant” to a current situation races against others and against prior instances representing algorithmic knowledge. The fastest instance determines the action of the model. An instance encoding a situation is regarded to be “relevant”, if it either matches the current situation exactly, or does not differ from it by more than 1000 tons of sugar in either the current output or the desired output, analogous to the loose range discussed above. Retrieval in the ACT-R model, on the other hand, is governed by similarity matches between a situation currently present and encodings of others experienced in the past (see Buchner, Funke & Berry, 1995 for a similar position in explaining the performance of subjects operating Sugar Factory). On each trial, a memory search is initiated based on the current situation and the target state ‘9000 tons’ as cues in order to retrieve an appropriate intervention or an intervention that belongs to a similar situation. The following production rule is used to model the memory retrieval of chunks based on their activation level:

IF the goal is to find a transition from the current state with output current to a state with new output desired AND there is a transition in declarative memory, with current output current and new output desired and a number of workers equal to number
THEN set the number of workers in the goal to number

This rule will normally only retrieve an old situation that exactly matches the current situation. However, ACT-R can also match chunks that do not exactly match the rule by a process called partial matching, which was mentioned briefly in chapter 2. This means that an old situation may also be retrieved if it is slightly different from the current situation. Instances which only partially match the retrieval pattern, i.e. which do not correspond exactly to the current situation will be penalized by lowering their activation proportional to the degree of mismatch. Activation noise is introduced to allow for some stochasticity in memory retrieval.

As figure 6.3 shows, the use of instances instead of the initial algorithmic knowledge increases over time, resulting in the gradual transition from algorithmic to memory-based processing as postulated by Logan (1988, p. 493).
Theoretical Evaluation

While the two models of instance-based learning share some striking similarities, the D&F-model makes unrealistic assumptions with respect to the storage and the retrieval of instances. Dienes & Fahey (1995) found out that these critical assumptions are essential to the performance of the D&F model (p. 856f):

The importance to the modeling of assuming that only correct situations were stored was tested by determining the performance of the model when it stored all instances. (...) This model could not perform the task as well as participants: the irrelevant workforce situations provided too much noise by proscribing responses that were in fact inappropriate (...) If instances entered the race only if they exactly matched the current situation, then for the same level of learning as participants, concordances were significantly greater than those of participants.

Since the ACT-R model does not need to postulate these assumptions, this model can be regarded as the more parsimonious one, demonstrating how instance-based learning can be captured by the mechanisms provided by a unified theory of cognition.

Empirical Evaluation

While the theoretical analysis of the assumptions underlying the two models favors the ACT-R approach, we will briefly discuss the empirical success of the models with respect to empirical data reported by Dienes and Fahey (1995). Figure 6.4 shows the trials on target when controlling Sugar Factory over two phases, consisting of 40 trials each. ACT-R slightly overpredicts the performance found in

Figure 6.3. Relative use of instance retrieval per trial by the ACT-R model
6: Examples versus Rules

After the participants had controlled the Sugar Factory for 80 trials, they had to do a slightly different task. Again they had to determine the work force in 80 situations, but now they did not receive feedback, but just moved on to a new, unrelated situation. The 80 situations presented were the last 40 situations from the first part of the experiment mixed with 40 new situations.

Figure 6.5 shows how the percentage of times (concordance) participants chose the same work force in this second task as they did in the first. The baseline level...
represents the chance that both choices are equal due to random choice. This chance is higher than 1/12, because some choices are made more often during the experiment than others. The correct column shows how often the same work force is chosen if this leads to a correct output, and the wrong column shows the same for the incorrect outputs. Again, both models seem to do a similarly good job in explaining the data, with neither model being clearly superior.

6.4 The Fincham task

An example of a task in which both rule learning and instance learning are viable strategies is described by Anderson & Fincham (1994). In this task, participants first have to memorize a number of facts. These facts look like this:

“Hockey was played on Saturday at 3 and then on Monday at 1.”

We will refer to these facts as “sports-facts” to prevent confusion with facts and rules in the model. A sports-fact contains a unique sport and two events, each of which consists of a day of the week and a time. After having memorized these facts, participants were told they really are rules about the time relationships between the two events. So in this case “Hockey” means you have to add two to the day, and subtract two from the time. In the subsequent experiment, participants were asked to predict the second event, given a sport and a first event, or predict the first event, given the sport and the second event. So participants had to answer questions like: “If the first game of hockey was Wednesday at 8, when was the second game?” Figure 6.6 shows an example of the interface used in the experiment. In this paradigm, it is possible to investigate evidence for both rule-based learning and instance-based learning.

Directional asymmetry, evidence for rule-based learning, can be tested for by first training participants to predict events in one direction for a certain sports-fact, and then reverse the direction and look how performance in the reverse direction relates to performance on the trained direction. Evidence for instance learning can be
gained by presenting specific examples more often than other examples. Better performance on these specific examples would indicate instance learning. Anderson & Fincham (1994), and later Anderson, Fincham & Douglass (1997) performed five variations on this basic experiment, three of which we will discuss here. But before discussing the specific experiments, we will first take a look at the ACT-R model we have developed.

The ACT-R model
The central assumption of our model of the Fincham task is that the data can only be explained by multiple strategies. We will use the four strategies discussed by Anderson, Fincham & Douglass (1997): analogy, abstraction, rule and instance. These strategies have different cost-success profiles (summarized in figure 6.9), which determine at what stage of the learning process they will be most prominent. Figure 6.7 shows schematic representations of each of the strategies. Since each problem involves calculating a day and a time, two separate sub-problems have to be solved. Each of these strategies corresponds to one of the boxes in figure 6.1.

The analogy strategy (figure 6.7a) has the highest cost, but only needs the sports-facts learned initially. Starting at the top goal, a subgoal is pushed onto the goal stack to either find the day or the time. To be able to do this, the original example must first be retrieved, and the appropriate elements (days or times) must be extracted. Another subgoal takes care of this stage. After retrieving the example, this second subgoal is popped, and a new subgoal is pushed to make an analogy between the example and the current problem. First the relation in the example is determined, for example the fact that two has to be subtracted from the day. Most of the time, this
The Fincham task

Figure 6.7. Schematic representation of the four possible strategies used in the model. Note that two strategies (possibly the same) are needed to solve the whole problem: one for the day of the week and one for the time of the day.
6: Examples versus Rules

relationship can be determined directly by retrieval, for example the relationship between four and six. But sometimes, as in the case of days of the week, this has to be done by counting. To determine the relationship between Sunday and Friday, one has to count two steps back from Sunday. Counting is taken care of by an additional subgoal, with the advantage that this subgoal is added to declarative memory and can be retrieved during later trials to determine the relation directly. After determining the relationship in the example, this relation is applied to the current problem. This can again be direct, or through a counting subgoal.

The analogy strategy requires prior knowledge. The model assumes that people already know how to make simple analogies, how to memorize and recall strings of words, and that they know relationships between numbers and days of the week, and are able to calculate these relations if they cannot be retrieved from memory. The rest of the necessary knowledge, mainly involving perceptual-motor operations like reading the information on the screen and entering the answers, has to be learned by the participants during the instructions. This aspect of the task is not modeled.

The abstraction strategy (figure 6.7b) assumes knowledge about the relation between the two days or two times for a certain sport. For example, “Hockey” means “add two to the days”. An abstraction in the model is a declarative fact that stores this information, for example:

```
ABSTRACTION234
  ISA ABSTRACTION
  SPORT HOCKEY
  TYPE DAY
  RELATION PLUS2
```

Using an abstraction to find the answer only requires two steps: retrieve the abstraction and apply it to the current problem. The second step, application, may involve another counting subgoal, similar to the analogy strategy. Although the abstraction strategy is more efficient than the analogy strategy, it requires knowledge participants initially do not have: abstractions.

The rule strategy (figure 6.7c) uses production rules to find the answer. Each of the rules has two versions, one that retrieves the answer, and one that calculates the answer. An example of a retrieve rule is:

```
IF the goal is to find the day of the second event, the sport is hockey and the day of the first event is day1
  AND day1 plus two days equals day2
THEN put day2 in the second event slot of the goal
```

The calculate version pushes this calculation as a subgoal, which is handled by the same production rules that determine and apply the relations in the analogy strategy. An example of this second version is:
The Fincham task

IF the goal is to find the day of the second event, the sport is hockey and the day of the first event is \( \text{day}1 \)
THEN push as a subgoal to find the answer to \( \text{day}1 \) plus two days
AND put the answer in the second event slot of the goal

The advantage of the rule strategy is that its costs are much lower than those of the analogy strategy, and also slightly lower than the costs of the abstraction strategy, since the answer can be found in a single step. However, in order to use it, the necessary production rules must be learned. Furthermore, the two example rules given only calculate the second event given the first. To calculate the first event given the second, two additional rules are needed.

The strategy with the lowest costs is the instance strategy (figure 6.7d). It can be applied to the top-goal, since it retrieves the answer from past subgoals directly. This strategy will only work if the appropriate instance is available. An example of an instance is:

```
ITEM434
  ISA ITEM
  SPORT HOCKEY
  TYPE DAY
  LEFT SUNDAY
  RIGHT TUESDAY
```

To be able to fully depend on this strategy, all possible examples have to be learned. For each sports-fact, seven to nine examples are needed.

The abstraction, rule and instance strategy are actually short-cuts for the original analogy strategy. The abstraction and rule strategy make short-cuts at the subgoal level of the analogy strategy, and the instance strategy directly at the top level. The knowledge needed for the instance short-cut is gained automatically, since the popped subgoals serve as examples. To be able to use an example, its activation must be high enough, so it has to be repeated a number of times before it can successfully be retrieved. Abstractions and rules, on the other hand, have to be learned more explicitly.

To create an abstraction and use it for later problems, information from different levels of the goal stack has to be used. The relation is determined in the analogy subgoal, while the name of the sport is stored higher in the goal stack. As a consequence, old goals created by the analogy strategy cannot be used as abstractions. An explicit goal is necessary to assemble it. An appropriate moment to do this is at the end of the analogy strategy, as illustrated in figure 6.8a. The goal is not popped, but is replaced by a goal to build an abstraction. Alternatively, the abstraction could be derived first and be subsequently applied. Since this alternative will produce the same predictions, it is not further investigated.
Learning a new production rule presupposes a dependency that must be created explicitly. As discussed earlier in this chapter, a dependency and a copy of the goal may be pushed as a subgoal to accomplish this (figure 6.8b). The subgoal that calculates a day or a time is replaced by a dependency. Further processing is done on a copy of the original subgoal. Assuming some other strategy has found the answer, the subgoal is popped and the dependency is completed. After the dependency has been popped from the goal stack, ACT-R’s production compilation mechanism will compile the dependency into a production rule. In this particular model, pushing a dependency can only be successfully completed if it is followed by the abstraction strategy, since only the abstraction strategy can provide for the necessary constraint (for example, the appropriate plus2 fact in the hockey case). In the case of the analogy strategy, this constraint is buried deeper in the goal-structure, and cannot easily be recovered.

For both abstraction and rule learning, additional steps in the reasoning process are necessary that are irrelevant to the immediate solution. The production rule that proposes to create an additional abstraction goal has to compete with the rule that proposes to just pop the goal and be done. Similarly, the rule that proposes to replace the original goal with a dependency has to compete with rules that try to solve the problem immediately. Since the rules that propose additional processing imply
additional costs, they will only occasionally win the competition. Building up abstractions and production rules may therefore be a slow process, and may well be a source of individual differences. Figure 6.9 summarizes cost and learning aspects of the four strategies.

In the Fincham task, learning of abstractions, instance learning and rule learning are all viable strategies from the viewpoint of rational analysis. Abstraction and rule learning will lead to quicker results but need more effort initially, since rules are not learned automatically. Instance learning is eventually the best strategy, but requires much more training to be fully effective.

**Empirical evaluation of the model**
In order to test the predictive power of the model, three experiments conducted by Anderson, Fincham and Douglass have been modeled. The first experiment was used to determine all the parameters, so the second and the third experiment can be considered as predictions based on the first. Each of the experiments tries to gain insights into the learning process by seeking evidence for the use of rules and the use of instances. The data discussed in the experiments all come from Anderson, Fincham and Douglass (Anderson & Fincham, 1994; Anderson, Fincham & Douglass, 1997), the model outputs are produced by 100 runs of our model.

**Experiment 1**
In the first experiment (experiment 2 in Anderson & Fincham, 1994), participants had to learn eight sports-facts. In the first three days of the experiment, four of these sports-facts were tested in a single direction: two from left to right and two from right to left. On each day 40 blocks of trials were presented. In each block, each of
the four sports-facts was tested once. On the fourth day all eight sports-facts were tested in both directions. On this day 10 blocks of trials were presented, in which each of the eight sports-facts was tested twice, once for each direction.

The model uses the following parameters: base-level decay is set to 0.3, in accordance with the findings in the Tulving-model in chapter 4, both permanent activation noise and normal activation noise are set to 0.05, the expected gain noise is set to 0.2, the retrieval threshold is set to 0.3 and both the latency factor and latency component are set to their default values of 1.0. Except for the base-level decay, all these values are close to their recommended values. Furthermore, the same parameter values will also be used for experiment 2 and 3.

Figure 6.10 shows the latencies in the first three days of the experiment, both the data from the experiment and from the model. Although the results of the model are the product of four interacting strategies, this produces no discontinuities: the learning curve of the model resembles a power-function, except for a slight decrease in performance at the beginning of each new day. The fit between the model and data is quite good: $R^2=0.94$. Figure 6.11 shows the results for day 4. Both in the data and in the experiment there is a clear directional asymmetry, since items in the practiced direction are solved faster than reversed items. Items that are completely new and have been practiced in neither direction, however, are performed even more slowly than the reversed items, indicating rule learning cannot be the whole explanation for all of the learning in the first three days of the experiment.

Figure 6.12 shows how the model uses the four strategies in the course of the experiment. At the start of the experiment, analogy is used most of the time, but both the abstraction and the instance strategy gain in importance after a few blocks of trials. The rule strategy appears later, and only plays a minor role during the first day. At the start of the second day, there is a large shift toward using rules at the expense of instances. This can be explained by the fact that the activation of a large portion of the instances has decayed between the two days, so that they cannot be retrieved anymore. Since only a few rules are needed for successful performance, they receive more training on average and are less susceptible to decay. Note that the abstraction strategy remains relatively stable between the days since it also less susceptible to decay than the instance strategy. This pattern is repeated at the start of the third day, although the instance strategy loses less ground due to more extended training of the examples. At the start of the fourth day, the frequency of use of the analogy strategy goes up again, since there are no production rules for the new four sports-facts. The abstraction strategy can take care of the reversed items though, so in that case the expensive analogy strategy is not needed. This explains the fact that reversed items are still faster than completely new items.
The Fincham task

![Graph showing latency data for days 1 to 3 in experiment 1]

Figure 6.10. Latencies for day 1 to 3 in experiment 1

<table>
<thead>
<tr>
<th>Direction and Practice</th>
<th>Data (in seconds)</th>
<th>Model (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same direction, practised</td>
<td>8.9</td>
<td>8.4</td>
</tr>
<tr>
<td>Reverse direction, practised</td>
<td>10.9</td>
<td>9.3</td>
</tr>
<tr>
<td>Not practised</td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>

Figure 6.11. Effect of direction of practice and whether a rule has been trained on time to respond (in seconds) from day 4 of experiment 1.

![Graph showing proportion of trials using different strategies]

Figure 6.12. Proportion of the trials a certain strategy is used by the model in experiment 1
6: Examples versus Rules

Experiment 2

In experiment 2 (experiment 1 in Anderson, Fincham & Douglass, 1997) the directional asymmetry was explored further. Instead of having only a single transfer day, two rules were reversed each day of the experiment. This requires quite a complicated experiment, since on each day a rule has to be presented in two directions that was presented in one direction previously. So, on day 1 of the experiment, two out of eight rules were presented in two directions, while the remainder was only tested in one direction, on day 2 four out of eight rules, up to day 4 where all rules were presented in both directions. On each day participants had to do sixteen blocks of ten to sixteen trials, ten trials on day 1, twelve trials on day 2, fourteen trials on day 3, and sixteen trials on day 4. To further investigate the difference between rule and instance based performance, participants were asked after each trial whether they solved it using a rule or an example. Finally, on each day one of the sports-facts studied originally was offered as a trial somewhere between block 7 and 10. If performance on this original sports-fact is better than on other sports-facts, this indicates the participant retrieves the answer instead of calculating it.

The latencies for day 1 to 4 are shown in figure 6.13 for both the data and the model. Although the model is slightly slower than the participants, the learning curves are parallel. Directional asymmetries are calculated using the two rules that are presented in two directions for the first time that day. The solution time for the practised direction is subtracted from the solution time for the reversed direction. The result is the extra time needed for the reversal, and is shown in figure 6.14. Both the data and the model show a gradual increase in asymmetry over the days, although asymmetry for the model is slightly larger than for the data. To be able to map the participants’ reports of using either a rule or an example onto the model, we first have to decide when the model uses a rule or an example. The most logical choice is to assume that both the analogy and the instance strategy are strategies that use examples, and that the abstraction and the rule strategy are strategies that use rules. Figure 6.15 shows the results of both the model and the data on this aspect of the task. Since the “solve by example”-category includes both the slowest (analogy) and the fastest (instance) strategy, it eventually becomes faster than the rule strategy.
as analogy is not used anymore. Both the data and the model show this phenomenon.

The latencies for the original sports-fact that was presented between block 7 and 10 are shown in figure 6.16, and are compared with the average latencies between block 7 to 10. Performance on original examples is clearly superior to other examples, indicating instance learning. Figure 6.17, finally, shows the strategies that were used by the model in the course of the experiment. It shows a pattern that is similar to the pattern in experiment 1.

**Experiment 3**

In experiment 3 (experiment 3 in Anderson, Fincham & Douglass, 1997), the effect of repeated examples is further explored. The same experimental setup as in experiment 2 was used, except that the experiment now took five days and each day consisted of 32 blocks of trials. On the first day eight rules were tested in only one direction. On each subsequent day, a new pair of rules was also tested in the reversed direction. So, on day 2 eight rules were tested in the practiced direction,
6: Examples versus Rules

and two rules in the reverse direction, on day 3 eight rules were tested in the practiced direction and four rules in the reverse direction, etcetera. To see if instances that are repeated more often than others are solved faster, half of the instances presented for a certain sport were identical, while the other half were generated in the usual way.

Figure 6.18 shows the results for both the data and the model. Repeated instances have a clear advantage over unique instances, further evidence for instance-based learning. Figure 6.19 shows the directional asymmetry results. After a steady increase between day 2 and 4, it decreases on day 5, both in the model and the data. On day 5 however, both the data and the model show a decline in asymmetry, indicating that instance-based reasoning, which has no asymmetry, takes over from rule-based reasoning.
6.5 Discussion

The two models discussed in this chapter demonstrate that understanding skill acquisition is not just a matter of answering the question whether skills are represented by rules or examples. People apparently have the capacity to store previous results and the capacity to generalize rules. Whether or not both types of learning show up in the results of experiments depends on their successfulness. In the Sugar Factory experiment, rules were very hard to generalize, so behavior can be explained by learning examples only. The Fincham experiment, on the other hand, shows clear evidence for both types of learning, since there is a balance between the usefulness of learning rules and examples.
6: Examples versus Rules

A theory that just states that skill learning is a matter of both instance learning and
rule learning is rather weak, and will certainly not end the debate. That is why
cognitive modeling is so useful: the theory proposed in this chapter is not just the
conjunction of two existing theories, but adds the constraint that the structure of the
task is a main determinant of which types of learning will have an impact on
performance.

The subject of the previous chapter, implicit versus explicit learning, is also tied to
the discussion of rule versus instance learning. If we consider a rule as a
generalization of one or more examples, creating abstractions is the most important
step of rule learning. Proceduralizing this abstraction later on is just an efficiency
improvement. This brings up another issue, namely whether the proceduralization
of abstractions is a form of explicit or implicit learning. Technically, it is explicit in
ACT-R, since a dependency has to be pushed on the top of the goal stack, so is the
focus of attention for a while. But is learning a production rule really an intentional
act? This is at odds with our intuitions about production rules, especially since we
have no conscious access to production rules. How can we intentionally learn things
we cannot directly access?

An alternative is to suppose production learning is a more or less automatic
mechanism, along the lines sketched in section 6.2. The assumption that production-
rule learning is an implicit learning mechanism implicates another stance towards
explicit learning strategies. Instead of depicting explicit knowledge as dependency
manipulators, explicit strategies are clever abstraction builders and interpreters.
Although the Fincham model needs an abstraction before a production rule can be
compiled, we might imagine more simple situations in which a rule can be learned
without explicit declarative intervention (for example as in the child model of
discrimination-shift learning in chapter 5). Eventually it may be possible to develop
a learning mechanism that does not need the dependency structure at all. It must be
noted that ACT-R’s developers still consider production compilation as a tentative

If learning a production rule is an implicit learning process, the explicit part of
learning rules lies in constructing abstractions, which can be considered declarative
rules. Now that rules and instances are both declarative representations in ACT-R,
we might ask the question whether there really is a distinction between the two.
Instances in the Sugar Factory model are used for situations that are different from
the situation in which they were created, so some sort of generalization occurs at the
moment an instance is applied. Abstractions in the Fincham model are used as rules,
but are just declarative facts in memory. The main difference is not their
representation, but the way in which they were learned, either implicitly or
explicitly.
Again a traditional distinction in cognitive psychology is not what it seems when analyzed in detail within a cognitive architecture. The mapping from implicit memory to procedural memory and explicit memory to declarative memory turned out to be invalid, and now the mapping from procedural memory to rules and declarative memory to examples is not valid either. Although the concepts themselves are quite meaningful, we have to learn to live with the fact that there are no direct mappings between them and the underlying cognitive architecture. But this may just make them more interesting.
6: Examples versus Rules