Learning and revising task-specific rules in ACT-R

Niels Taatgen
Cognitive Science and Engineering
University of Groningen
Grote Kruisstraat 2/1
9712 TS Groningen
email: niels@tcw2.ppsw.rug.nl
9 July 1996

Abstract

This paper discusses the problem of how to learn and revise rules for a new task. It will use the framework of the ACT-R theory, which uses analogy to learn new productions. As an example, two general rules to learn new productions will be discussed that can learn two different tasks, a beam-task and a card-classification task.

1 Introduction

If, in an experimental situation, some new task is presented to a subject, he or she is, after brief instructions, almost always capable of doing the task. This means that in a short period of time the subject has acquired the necessary task-specific knowledge to be able to get started on the task. Cognitive architectures based on production rules often ignore this phase of the experiment and just assume that subjects somehow produce this knowledge while reading instructions or studying an example.

When the task involves complex problem-solving, the set of rules the subject initially comes up with is often insufficient to reach the goal. In that case the initial rules need to be replaced or refined later on. Since revising task-knowledge also involves the acquisition of new rules, the mechanism that is responsible for the initial rules might also be responsible for the revised rules.

The hypothesis that will be explored in this paper is, that a set of general production rules is responsible for learning both initial and revised rules for a new task.

2 New production rules in ACT-R

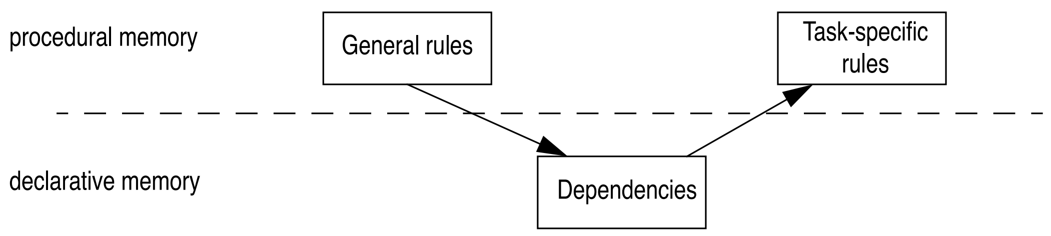
In the ACT-R architecture, new production rules can be learned by the analogy mechanism (Anderson, 1993; Lebiere, 1996). It involves the generalisation of examples in declarative memory whenever a goal turns up that resembles the example. The examples are stored in specialised chunks, dependency chunks, that contain all the information needed: an example goal, an example solution, facts (called constraints) that need to be retrieved from declarative memory to create the solution, and sometimes additional subgoals that must be satisfied before the solution applies. Adding two to three to obtain five using an addition fact is an example of a dependency:

```
dependency2+3
      isa dependency
      goal example-goal1
      modified example-solution1
      constraints fact2+3
example-goal1
      isa addition-problem
      arg1 two
      arg2 three
      answer nil
example-solution1
      isa addition-problem
      arg1 two
      arg2 three
      answer five
fact2+3
      isa addition-fact
      addend1 two
      addend2 three
      sum five
```

Analogy will produce the following production rule based on this dependency:

```
(p addition-problem-production1
=example-goal-variable>
      isa addition-problem
      arg1 =two-variable
      arg2 =three-variable
      answer nil
=fact2+3-variable>
      isa addition-fact
      addend1 =two-variable
      addend2 =three-variable
      sum =five-variable
==>
=example-goal-variable>
      answer =five-variable)
```

Although the ACT-R theory specifies how new production rules are generated from examples, it does not specify where the examples come from. But since examples are just chunks in declarative memory, they can be created by production rules. So the schema to produce task-specific production rules will be as follows:

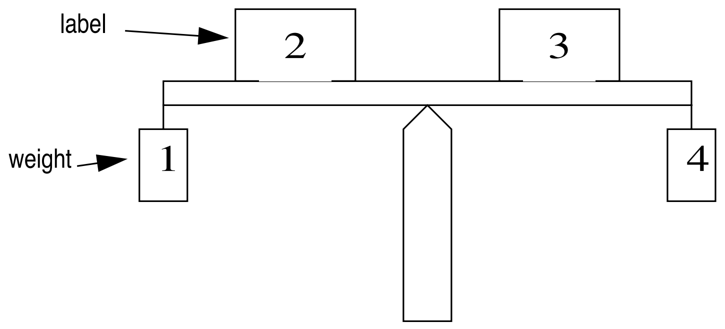


The general rules themselves need of course information to work with. Several sources of information may be available, which must be present in declarative memory, since production rules cannot directly inspect other production rules. Possible sources of information are:

- Task instructions and examples
- Relevant facts and biases in declarative memory
- Feedback
- Old goals and dependencies for the same problem

3 The beam task

To explore the above ideas we will use the task of solving beam problems. The problem is relatively easy: a beam is given, with weights on the left and the right arm. Attached to the arms of the beam are labels, each with a number on it. The task is to predict whether the beam will go left, right or remain in balance. The number on the labels have no influence on the outcome. The figure below shows an example of a beam:



Although the task is easy if we know something about weights and beams, it is much more difficult if we know nothing at all.

The general rules used to learn this task are the following:

Property-retrieval

If there is a task that has a number of objects, create a dependency that contains an example of retrieving a certain property of each of the objects.

In the case of the beam task, the objects are the arms of the beam, and weight and label are possible properties.

Find-fact-on-feedback

If feedback indicates that the answer is incorrect, and also contains the correct answer, set up a dependency that uses the goal and the answer as examples. Also, retrieve some fact that serves as a constraint in the dependency.

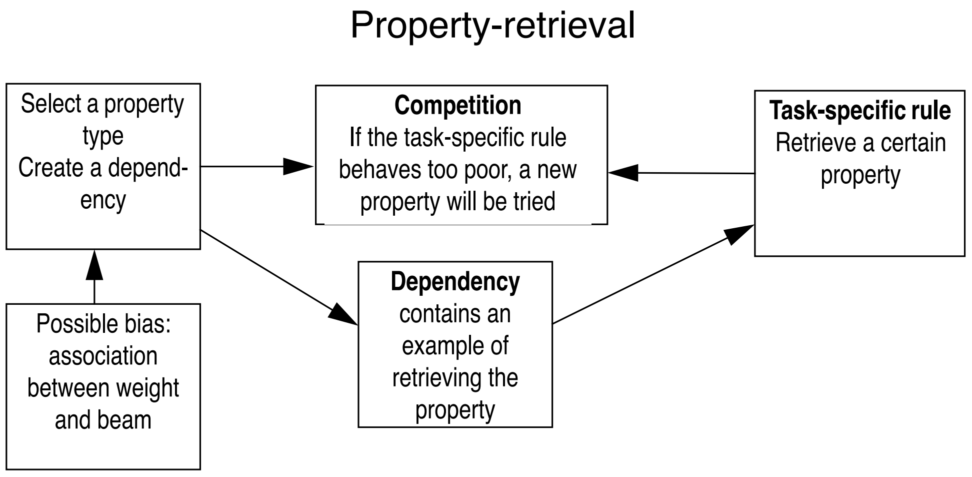
To be able to generate correct rules for the beam task, we need to retrieve the fact that a certain number is greater than another number, to predict correctly whether the beam will go left or right.

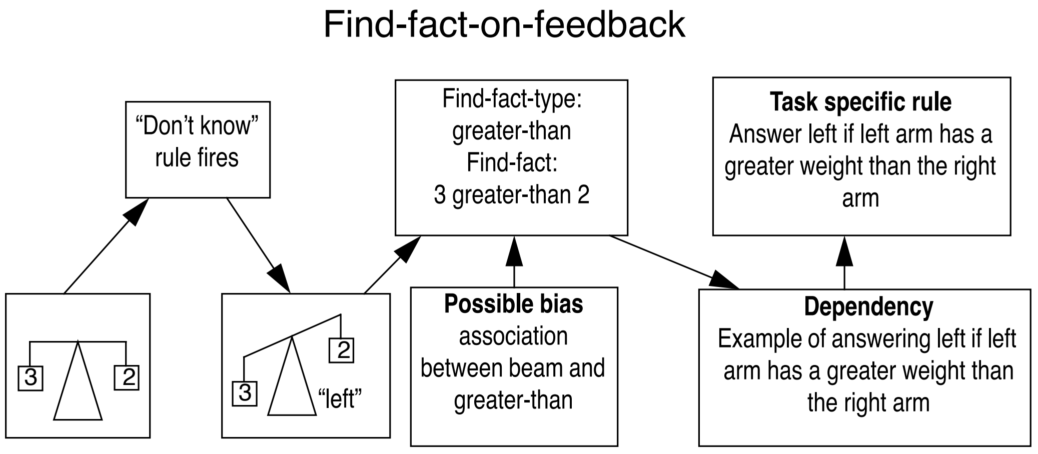
Since the general rules are just production rules, they are in direct competition with the task-specific rules they generate. So if **property-retrieval** generates a rule X to retrieve the label, X will compete with **property-retrieval**. So if X is doing a bad job, which it will if it retrieves the label that has no relevance at all to solving the problem, its evaluation will decrease, and it will eventually lose the competition, in which case **property-retrieval** will create an example of retrieving weight.

Although **find-fact-on-feedback** is only activated if feedback indicates an incorrect answer, the rules it produces are in competition with each other. The rule with the highest success rate will eventually win.

Both **property-retrieval** and **find-fact-on-feedback** can be influenced by prior knowledge. If there is an association strength between beam and weight, indicating knowledge that a beam has something to do with weight, **property-retrieval** will choose weight in favour of label. If there is an association strength between beam and greater-than, a greater-than fact will be retrieved by find-fact-on-feedback.

The following figures summarise the general rules:


Property-retrieval
Select a property type Create a dependency
Competition
If the task-specific rule behaves too poor, a new property will be tried
Task-specific rule
Retrieve a certain property
Possible bias: association between weight and beam
Dependency
contains an example of retrieving the property


Find-fact-on-feedback
"Don't know" rule fires
Find-fact-type: greater-than
Find-fact: 3 greater-than 2
Task specific rule
Answer left if left arm has a greater weight than the right arm
3
2
3
2
"left"
Possible bias
association between beam and greater-than
Dependency
Example of answering left if left arm has a greater weight than the right arm

Apart from the general rules, the model contains lisp functions to generate random beams, and production rules to give feedback. When the model produces an incorrect answer, it will try the same beam again until it can predict the right outcome.

4 Simulation results

The general rules turned out to be sufficient to learn the task. The following rules are examples of (correct) rules learned by the model. The rule generated by property-retrieval is:

```
(P GEN-GOAL-PRODUCTION10
     =OLDGOAL10-VARIABLE>
        ISA GEN-GOAL
        TYPE SOLVE-BEAM
        OB1 =O6-VARIABLE
        OB2 =O7-VARIABLE
        PROP1 NONE
        PROP2 NONE
     =P7-VARIABLE>
        ISA PROPERTY
        OF =O6-VARIABLE
        TYPE WEIGHT
        VALUE =ONE-VARIABLE
     =P8-VARIABLE>
        ISA PROPERTY
        OF =O7-VARIABLE
        TYPE WEIGHT
        VALUE =SIX-VARIABLE
  ==>
     =OLDGOAL10-VARIABLE>
        PROP1 =ONE-VARIABLE
        PROP2 =SIX-VARIABLE
        PROPTYPE WEIGHT)
```

One of the rules generated by find-fact-on-feedback is:

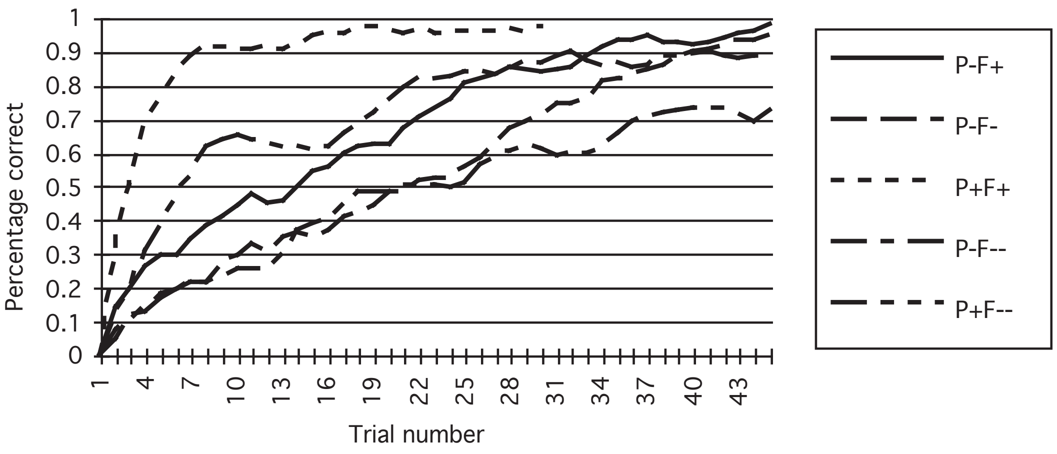
```
 (P GEN-GOAL-PRODUCTION12
     =OGOAL11-VARIABLE>
        ISA GEN-GOAL
        TYPE SOLVE-BEAM
        OB1 =O6-VARIABLE
        OB2 =O7-VARIABLE
        PROP1 =ONE-VARIABLE
        PROP2 =SIX-VARIABLE
        ANSWER NONE
        PROPTYPE WEIGHT
     =F61-VARIABLE>
        ISA GEN-FACT
        TYPE GT-FACT
        SLOT1 =SIX-VARIABLE
        SLOT2 =ONE-VARIABLE
  ==>
     =OGOAL11-VARIABLE>
        ANSWER RIGHT)
```

The model was tested in several conditions, differing in the bias given for the properties and the fact-type. The following table summarises the conditions:

P+	Association between beam and weight
P-	Association between beam and label, so a bias for the wrong property
F+	Association between beam and greater-than
F-	Association between both beam and greater-than, and beam and number, so two possible fact-types.

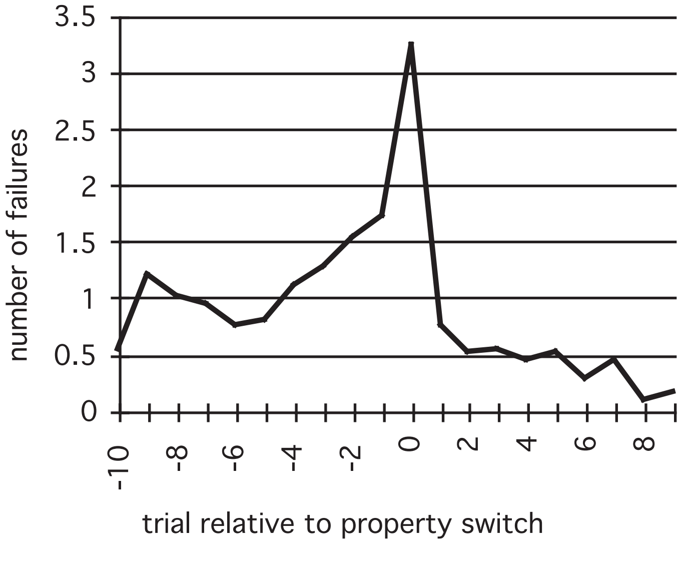
F-- No associations between beam and fact-types, four fact-types are possible.

Each experiment has a P and an F-condition. Each experiment has been run 30 times for 50 trials. The following table shows the results:



As can be seen in the graph, the P+F+ condition learns to solve the task quite rapidly, and the fact that the model doesn't reach a 100% score within a few trials is only due to the fact that beams are generated randomly, only occasionally producing a beam in which balance is the correct answer. Performance decreases if the model has less initial information. In the case of the P-F-- condition, the model often fails to find the correct rules for the task.

The results in the figure above suggest a gradual increase of performance. This is however not the case, but a result of averaging 30 runs. If individual runs are examined, each has a certain point where performance increases dramatically. To make this clear the following graph depicts the average number of incorrect tries for each trial in the P-F+ condition, averaged with respect to the point where the model switches from examining the label property to examining the weight property. So at x=0 the model creates a dependency that contains an example of retrieving weight.



The dependency is created at the moment that the model has failed several times to predict the right answer. As a result, the evaluation of the rule that retrieves the labels drops and the general rule can propose a new dependency.

5 The card task

General rules are of course only general if they can be used for different tasks. So the same rules were used for a new task, a card-classification task. In this task, cards with pictures must be sorted into two categories. The pictures are either one or two squares or circles, which are either red or green and either large or small. The criterion to sort on is the colour (red=yes; green=no), which the subject has to discover. The same general rules can be used to learn this task. First, a property must be selected, so either colour, shape, size or number. After that, the relevant aspect is tied to the answer. The following rules are examples of rules learned by the model:

```
(P GEN-GOAL-PRODUCTION167
     =OLDGOAL1167-VARIABLE>
        ISA GEN-GOAL
        TYPE SOLVE-CAT
        OB1 =O164-VARIABLE
        PROP1 NONE
     =P165-VARIABLE>
        ISA PROPERTY
        OF =O164-VARIABLE
        TYPE COLOUR
        VALUE =GREEN-VARIABLE
  ==>
     =OLDGOAL1167-VARIABLE>
        PROP1 =GREEN-VARIABLE
        PROPTYPE COLOUR)

 (P GEN-GOAL-PRODUCTION169
     =OGOAL167-VARIABLE>
        ISA GEN-GOAL
        TYPE SOLVE-CAT
        OB1 =O164-VARIABLE
        PROP1 =GREEN-VARIABLE
        ANSWER NONE
        PROPTYPE COLOUR
     =GREEN-VARIABLE>
        ISA GEN-FACT
        TYPE COLOUR
        SLOT1 GREEN
  ==>
     =OGOAL167-VARIABLE>
        ANSWER NO)
```

6 Conclusions and future work

The beam and card task show that different tasks can be learned by the same general rules. The rules produce both initial rules and revised rules. Also, the beam-task can be learned under several conditions: if the model has some knowledge about beams, it can learn the task quite rapidly, and if the model has little or no knowledge, learning is much slower. So this type of learning strikes a nice balance between one-shot learning methods like explanation-based learning and learning using huge amounts of examples, as in neural networks and genetic algorithms.

Future work will consist of the extension of the set of general learning rules, and the application of these rules to a larger set of tasks. Also, the general schema allows for the generation of new general rules, possibly contributing to a solution to the bootstrapping problem of learning.

References

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

Lebiere, C. (1996). *ACT-R 3.0: A users manual*. Report, Carnegie Mellon University, Pittsburgh.