

**Title**

Niclas Andreas Dobbertin  
Technische Universität Darmstadt

**Abstract**

ABSTRACT

## Title

## Introduction

Cognitive Architectures, modeling learning, production systems, ACT-R

## Productions

Productions are Rules with a condition and an action. Example production. can interact with various modules (memory, vision, motor)

## Learning

Retrieval(activation) strength, utility learning, production compilation, ...

## Task

Modified Frensch/Elio Task. 7 mathematical procedures, learning differently based on presentation order

**Table 1**

*Experiment Procedures.*

Procedures
$(Sandstein_4 - Sandstein_2) * Mineralien$ $(2 * Algen) + Sandstein_{min}$ $Gifte_{max} + Gifte_{min}$ $(Mineralien * 2) - Gifte_4$ <i>Das Höhere von</i> $(Gifte_3 - Gifte_2), (Sandstein_3)$ <i>Das Kleinere von</i> $(Sandstein_1 + Gifte_1), (Algen)$ $100 - \text{dem Höchstem aller Ergebnisse}$

*Note.* The seven translated procedures used in this experiment. The bottom procedure is always included as it calculates the total water quality.

## Model

chunktypes, pre-knowledge

The model was made using the ACT-R architecture through the pyactr implementation. It has the subsymbolic system, production compilation and utility learning enabled.

The model works with four different types of chunks specified. Number chunks hold the number, its digits and the number one higher. Math operation chunks hold an operation, two arguments and a result. Procedure chunks hold the operations, variables and values that make up a procedure in the experiment. The math goal chunk is used in the goal buffer and hold various slots used for operations, like the current operation, arguments, counters and flags.

The model gets some basic knowledge that does not have to be learned in the form of chunks set at model initialization. It knows each procedure already and can retrieve its operations and values with an key. It knows all numbers from 0 to 999 through the number chunktype. It has math operation chunks for all greater/less comparisons for numbers between 0 and 10. It has math operation chunks for addition of numbers between 0 and 21.

All trials are generated before the simulation starts and ordered depending on condition. The model uses an environment to simulate a computer screen. Elements are

specify parameters

specify that it still has to find the correct procedure to use?

currently has even more chunks for some reason, check if necessary

arranged in columns with the values in rows below their column header. Everytime the user inputs an answer or the variables change, the environment variables are directly updated. User input and trial change is detected from the model trace.

get the  
pyactr tk  
working  
and put  
screenshot

The model works through the tasks with a set of productions, which perform mathematical operations, search the screen, input answers and organize order of operations.

## Greater/Less-than Operation

Maybe better as figure note or in appx. and simpler/shorter description

This pair operations compares two multi-digit numbers and sets the greater/less number as answer. For each digit (hundreds, tens, ones) there is a set of productions comparing that digit of the two numbers. Each production set for a digit requires all higher-significant digits to be equal. That means that the productions comparing the tens can only fire if the hundreds are equal and the productions comparing the ones can only fire if both the hundreds and the tens are equal. The selected production now retrieves a comparison of the two digits from declarative memory. Depending on the result, either number 1 or number 2 will be written into the answer slot.

## Addition Operation

This operation adds two numbers through column-addition. The first production retrieves the sum of the ones digits of the two numbers. The sum is put into the ones digit of the answer. Next it tries to retrieve an addition operation from memory, where ten plus any number equals the previously found sum. If the retrieval fails, the result of the ones addition was less than ten and no carry-over is necessary. If the retrieval succeeds, a carry flag is set and the second addend of the retrieved operation (the part over ten) is set as the ones digit answer. Now the sum of the tens digits of the numbers is retrieved. If the carry flag is set, add one to the sum. Again check for remainder and set a carry flag if necessary. Then the same repeats for the hundreds digits.

maybe 10  
instead  
ten

## Multiplication Operation

This operation multiplies two numbers through repeated addition. Multiple productions handle cases in which one of the arguments is one or zero and directly set the answer accordingly. First, it tries to retrieve the sum of the second argument plus itself and sets a counter to one. If the retrieval succeeds, set the answer to the sum and increment the counter by one. While the counter is not equal to argument 1, retrieve the sum of argument 2 plus the result and increment counter. If the counter is equal to argument 1, the operation is finished. If the retrieval of the sum fails, save arguments and counter in different slots and change the current operation to addition, as well as the next operation to multiplication. When the current operation is multiplication again and there are values in the saved argument slots, restore arguments and continue.

## Subtraction Operation

Subtraction column-wise austrian method

## Motor System

The motor module is used to input the answers and to press continue. When the current operation is to type the answer, the first production requests the tens digit to

be pressed on the keyboard. When the action is finished, the ones digit and spacebar to continue in turn are requested to be pressed.

## Visual System

not really feasible to describe each production, more general overview

The visual module is used for various operations to find the current task or to replace variables in a task with the values shown on the screen. The screen is organized in columns with headers, so the visual module first searches for the correct column by keyword. Now different kinds of searches will be performed dependent on what is requested.

To find the next task, the search goes down the column of tasks and saves the task at the current row. If there is nothing in the answer column at the same y-coordinate, the currently saved task was not answered, the search is done. To find a variable value by index, the search travels down the column while counting and stops at the desired index. To find the max/min value of a variable, the search travels down all values in the column, checks for each one if it is greater/less than the currently saved value and replaces it if necessary. Once all values are checked, the search is finished.

## Utility Operations

Several productions dictate in what order operations are executed. When the operation slots are empty, the visual search for the next unanswered task is started. When a task is found, productions check if the argument are already numbers and if not, request the visual search for substitution with the values on screen. When a task is finished, the result is saved in a slot and other slots are reset, starting task search again. If the second task is finished, start the motor input of the answer.

One production detects if the current operation is finished and another operation is queued, and sets the next operation.

Since operations use both the full numbers and their digits, a set of productions fills digit slots with the digits of a number and vice versa.

## Figure 1

*Logic Flow of Addition*

*Note.* When each production is executed depending on state. Either example for one operation or figures for all?

## Figure 2

*Screenshot of experiment display*

*Note.* Example water sample as shown to a subject.

## Results

### Figure 3

*Mean solution time in training and transfer phase*

*Note.* Mean solution time of all six procedures of a water sample in blocks of five samples.

### Figure 4

*Comparison with human experiment*

## Discussion

### Confirmation/Contradiction to Experiment

#### Model Improvements

More in-depth modeling of all operations

track working memory usage

#### Working with ACT-R/pyactr

no basic productions given, everything has to be implemented from scratch, papers using act-r very rarely publish their model code

this model uses many different operations and modules of ACT-R and has to model each from scratch and handle task switching

vis: relative positions are not implemented, the visual search loops had to be unrolled to the required number of iterations and is not general