ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Programmiermethodik und Compilerbau

A Rule-Based Implementation of ACT-R Using Constraint Handling Rules

Masterarbeit an der Universität Ulm

Vorgelegt von:
Daniel Gall
daniel.gall@uni-ulm.de

Gutachter:
Prof. Dr. Thom Frühwirth
Prof. Dr. Slim Abdennadher

Betreuer:
Prof. Dr. Thom Frühwirth

2013

“A Rule-Based Implementation of ACT-R Using Constraint Handling Rules”
Version of July 18, 2013

Typesetting: PDF-LaTeX 2ε
Druck: **FIXME: Druck**

Abstract

This is the abstract of my master thesis.

Contents

1 Introduction

2 Description of ACT-R

In computational psychology, the approach to explore human cognition is to implement detailed, computational models that enable computers to execute them and simulate human behaviour [Sun08]. By conducting the same experiments with humans and with simulations of the suggested underlying cognitive models, the plausibility of models can be checked and models can be improved gradually.

On the other hand, psychology is experiencing a movement towards specialization [ABB+04], ie. there are a lot of independent, highly specialized fields that lack a more global view.

To implement consistent models of cognition, it is necessary to develop a theory that tries to put all those highly specialized components together and allows modelers to build their models on the basis of this theory. Cognitive architectures try to explain **FIXME: definition cognitive architecture from book** **FIXME: move to introduction and motivation**

Adaptive Control of Thought-Rational (ACT-R) is a cognitive architecture, that "is capable of interacting with the outside world, has been mapped onto brain structures, and is able to learn to interact with complex dynamic tasks." [TLA06, p. 29]

On top of the provided cognitive architecture, one can specify models for specific tasks. The cognitive architecture constrains the modeling to facilitate the modeling process. Thereby it ensures cognitive plausibility to some degree [TLA06, p. 29].

When talking about ACT-R, one can refer to the theory or the implementation. The theory gives a view which abstracts from implementational details that may be concerned when talking about implementation **FIXME: source**. In this work, implementation always refers to the vanilla Lisp implementation that can be downloaded from [act].

In this chapter, a short overview over the theory of ACT-R is given. First, the description is informal to provide a general image of how ACT-R works. Then, some important parts of the system are defined more formally in chapter **??**, as soon as it is needed in the implementation. All of the information in this chapter refers to the theory. Implementation is discussed in chapter **??**.

2.1 Procedural and Declarative Knowledge

A central idea of ACT-R is the distinction between *declarative* and *procedural knowledge.* The declarative knowledge consists of simple facts, whereas the procedural knowledge contains information on what to do with those facts.

2.1.1 Modular organization

This approach leads to a modular organization of ACT-R with modules for each purpose needed to simulate human cognition. Figure **??** provides an overview of some of the default modules of ACT-R. For example, the declarative module stores the factual information (the declarative knowledge), the visual module perceives and processes the visual field, the procedural module holds the procedural information and controls the computational process.

Each module is independent from the other modules and computations can be performed massively parallel within one module. The visual module, for example, can process the entire visual field at once. Additionally, modules can perform their computations parallel to other modules, for instance: The declarative module can search a specific fact while the visual module processes the visual field.

However, each module can perform its computation only locally and has no access to computations of other modules. To communicate, modules have associated buffers, where they can put a limited amount of information – one primitive knowledge element – and the procedural module can access each of these buffers. The information in a buffer could be one single fact retrieved from declarative memory or one visual object from the visual field perceived by the visual module. Information between modules is exchanged by the procedural module taking information from one buffer and putting it into another (with an optional computation on the way). This leads to a serial bottleneck in the computation, since every communication between modules has to go its way through the procedural module.

In figure **??** the general computational process is illustrated by showing the *recognize-act-cycle*: The procedural information is stored as rules that have a *condition* and an *action*. The condition refers to the so-called *working memory*, which basically is the content of all the buffers. In the recognize-phase of the cycle, a suitable rule that matches the current state of the working memory is searched. If the condition of a rule holds, it *fires* and performs its actions – this is the act-phase of the cycle. Those actions can cause changes on the buffers

that may lead to the next rule matching the current state in the next recognize-part of the cycle.

In the following sections, some of the modules and their precise interaction will be described in more detail.

2.1.2 Declarative Knowledge

The declarative module organizes the factual knowledge as an associative memory. Ie., it consists of a set of concepts that are connected to each other in a certain way.

Such elementary concepts are represented in form of chunks that can be seen as basic knowledge elements. They can have names, but they are not critical for the description of the facts and just for readability in the theory. The real description of a concept comes from its connections.

Chunks can have slots that are connected to other chunks or elements. Such an element can be regarded as a chunk without any slots. For instance, the fact that five plus to equals to seven can be modeled as a chunk that is connected to the numbers 5, 2 and 7 (see **??**). Notice that in the figure each slot has a individual name. This is necessary to distinguish the connections of the chunks, otherwise the summands were indistinguishable from the sum in the example.

Each chunk is associated with a chunk-type that determines which slots a chunk can have. For example, the fact in figure **??** has the type `addition-fact`. All chunks of this type must provide the slots `arg1`, `arg2` and `sum`.

For the chunk types there is no upper limit of slots they can define. However, Anderson et al. suggested to limit the number of slots to Miller's Number of 7 ± 2, for the reason of plausibility [**?**]. **FIXME: Find cite**

Textual Representation of Chunks

In the following, chunk-type and chunk definitions are given in a textual way which is based on the syntax of the standard implementation of ACT-R.

Definition 1. *The term* `chunk-type(name slot1 slot2 ... slotn)` *defines a chunk-type with name* `name` *and slots with names* `slot1` *to* `slotn`*.*

The term `chunk(name isa type slot1 val1 ... slotn valn)` *defines a chunk of type* `type` *with name* `name` *and corresponding slot-value-pairs, where* `slot1 val1` *signifies that the value of the slot* `slot1` *is* `val1`*. The slot-value-definitions must match the chunk-type-definition of* `type`*.*

Example 2.1. *The addition-fact chunk in figure* **??** *and its chunk-type are defined as follows:*

```
chunk-type(addition-fact arg1 arg2 sum)
chunk(a isa addition-fact arg1 5 arg2 2 sum 7)
```

Buffers

As mentioned before, modules communicate through buffers by putting a limited amount of information into their associated buffers. More precisely, each buffer can hold only *one chunk at a time*.

For example, the declarative module has the retrieval buffer associated with it, which can hold one specific declarative chunk. The declarative module can put chunks in the buffer that can be processed by the procedural module, which is described in the next section.

2.1.3 Procedural Knowledge

Procedural Knowledge in ACT-R is formulated as a set of condition-action rules. Each rule defines in its condition-part the circumstances under which it can be applied. Those conditions refer to the current chunks in the buffer. In the condition-part of a rule it is defined which kind of chunk with certain slot values must be present in which buffer for the rule to fire. For example, one rule in the process of adding the numbers 5 and 2 could have the conditions that there is a chunk of type `addition-fact` in the retrieval buffer with 5 and 7 in its argument-slots and specify certain actions if this is the case.

If the chunks in the buffers match all the conditions stated in a rule, it can be applied (“fired”), which leads its action-part to be performed. Actions can be changes of values in the chunks of a buffer, the clearing of a buffer or a buffer request, which leads the corresponding module to put a certain chunk into the requested buffer. Buffer requests are also stated in form of a chunk description where chunk-type and slots can have a special meaning. The actual semantics of a request is dependent on the module. For example, the declarative module will search a chunk that matches the chunk in the description of the request. One production rule, for instance, in the process of adding the numbers 5 and 2 could be, if the wrong `addition-fact` chunk is stored in the retrieval buffer, a retrieval request will be performed,

which states that the declarative module should put a chunk into the retrieval buffer, that has 5 and 2 in its argument slots and is of type `addition-fact`. After the successful performance of the request, a chunk with 5 and 2 as its argument will be stored in the retrieval buffer, that also has a value for the sum.

Although the term *module* is used for the procedural system, it differs a lot from the other modules: In contrast to other modules, the procedural module has no own buffers, but can access the buffers of all the other modules. “It really is just a system of mapping cortical buffers to other cortical buffers” [And07, p. 54].

The procedural system can only fire one rule at once and it takes 50 ms for a rule to fire [And07, p. 54]. After firing the selected rule, the next recognize cycle starts and a suitable rule will be detected and caused to fire. While that time, other modules may perform requests triggered in the action of the first rule. Sometimes, rules have to wait for results of certain modules and they cannot fire before those results are available. Those two facts illustrate how the procedural module can become a serial bottleneck in the computation process.

2.1.4 Goal Module

An essential part of human cognition is the ability to keep track of the current goal to achieve and to subordinate all actions to the goal [ABB+04, p. 1041]. For complex cognitive tasks, several rules have to be applied in series and intermediate results must be stored (without changing of the environment). Another important aspect is, that complex tasks can have different subgoals that have to be achieved to achieve the main goal. For instance, if one wants to add two multi-digit numbers, he would add the columns and remember the results as intermediate results.

In ACT-R, the goal module with the goal buffer is used for this purpose.

Working memory

The goal module and buffer are often referred to as *working memory* [ABB+04, p. 1041], but actually it can have another meaning as stated in [ARL96]: As usual in production systems, everything that is present to the production system and can match against the production rules is part of the working memory. With this definition, all chunks in the buffers form the working memory.

In this work, the term working memory will be used in this second meaning, since it discusses the topic from a computer science view and the second definition is related to production rule systems. When talking about the content of the goal buffer, this will be remarked explicitly.

2.1.5 Other Modules/Outside World

Since human cognition is embodied, there must be a way to interact with the outside world to simulate human cognition in realistic experiments. Therefore, ACT-R offers perceptual/motor modules like the manual module for control of the hands, the visual module for perceiving and processing the visual field or the aural module perceiving sounds in the environment.

Like with every other module, communication is achieved through the buffers of those modules.

FIXME: describe visual module

2.1.6 Example

FIXME: finish counting inspired by tutorial more examples can be found there

2.2 Serial and Parallel Aspects of ACT-R

In the previous sections there were some remarks on the serial and parallel aspects of ACT-R. According to [And07, p. 68], four types of parallelism and seriality can be distinguished:

Within-Module Parallelism: As mentioned above, one module is able to explore a big amount of data in parallel. For example, the visual module can inspect the whole visual field or the declarative module performs a massively parallel search over all chunks.

Within-Module Seriality: Since modules have to communicate, they have a limited amount of buffers and each of those buffers can only hold one chunk. For example, the visual module only can concentrate on one single visual object at one visual location, the declarative module only can have one single concept present, the production system can fire only one rule at a time, ...

Between-Module Parallelism: Modules are independent of each other and their computations can be performed in parallel.

Between-Module Serialism: However, if it comes to communication, everything must be exchanged via the procedural module that has access to all the buffers. Sometimes, the production system has to wait for a module to finish, since the next computation relies on this information. So, modules may have to wait for another module to finish its computation before they can start with theirs triggered by a production rule that states a request to those modules.

The procedural module is the central serial bottleneck in the system, since the whole communication between modules is going through the production system and the whole computation process is controlled there. The fact that only one rule can fire at a time leads to a serial overall computation. Another serial aspect is that some computations need to wait for the results of a module request. If no other rule matches in the time while the request is performed, the whole system has to wait for this calculation to finish. After the request, the module puts the result in its buffer and the rule needing the result of the computation can fire and computation is continued.

2.3 Subsymbolic layer

The previously discussed aspects of the ACT-R theory are part of the so-called symbolic layer. This layer only describes discrete knowledge structures without dealing with more complex questions like:

- How long does it take to retrieve a certain chunk?
- Forgetting of chunks
- If more than one rule matches, which one will be taken?

Therefore, ACT-R provides a subsymbolic layer that introduces “neural-like activation processes that determine the availability of [. . .] symbolic structures” [AS00].

2.3.1 Activation of Chunks

The activation A_i of a chunk i is a numerical value that determines if and how fast a chunk can be retrieved by the declarative module. Suppose there are two chunks that encode addition facts for the same two arguments (let them be 5 and 2), but with different sums (6 and 7), for example. This could be the case, if a child learned the wrong fact, for example. When stating a module request for an addition fact that encodes the sum of 5 and 2, somehow one of the two chunks has to be chosen by a certain method, since they are both matching the request. This is determined by the activation of the chunks: The chunk with the higher activation will be chosen.

Additionally, a very low chunk activation can prevent a chunk of being retrieved: If the activation A_i is less than a certain *threshold* τ, then the chunk i cannot be found.

At last, activation determines also how fast a chunk is being retrieved: The higher the activation, the shorter the retrieval time.

Base-Level Activation

The activation A_i of a chunk i is defined as:

$$A_i = B_i + \Gamma \tag{2.1}$$

where B_i is the *base-level activation* of the chunk i. Γ is a context component that is described in chapter 2.3.1. Equation (2.1) is a simplified variant of the *Activation Equation*.

The base-level activation is a value associated with each chunk and depends on how often a chunk has been practiced and when the last retrieval of the chunk has been performed. Hence, B_i of chunk i is defined as:

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) \tag{2.2}$$

where t_j is the time since the jth practice, n the number of overall practices of the chunk and d is the decay rate that describes how fast the base-level activation decreases if a chunk has not been practiced (how fast a chunk will be forgotten). Usually, d is set to 0.5 [ABB+04,

p. 1042]. Equation (2.2) is called *Base-Level Learning Equation*, as it defines the adaptive learning process of the base-level value.

This equation is the result of a rational analysis by Anderson and Schooler. It reflects the log odds that a chunk will reappear depending on when it has appeared in the past [TLA06, p. 33]. This analysis led to the *power law of practice* [ABB+04, p. 1042]. In [AS00, pp. 8–11] equation (2.2) is motivated in more detail by describing the power law of learning/practice, the power law of forgetting and the multiplicative effect of practice and retention with some data. Shortly, it states, that if a particular fact is practiced, there is a improvement of performance which corresponds to a power law. At the same time, performance degrades with time corresponding to a power law. Additionally, they state that if a fact has been practiced a lot, it will not be forgotten for a longer time.

Activation Spreading

In ACT-R, the basic idea of activation is, that it consists of two parts: The base-level component described above, and a context component. Every chunk that is in the current context has a certain amount of activation that can spread over the declarative memory and enhance activation of other chunks that are somehow connected to those chunks in the context. The activation equation (2.1) is extended as follows:

$$A_i = B_i + \sum_{j \in C} W_j S_{ji} \tag{2.3}$$

where W_j the *attentional weighting* of chunk j, S_{ji} the *associative strength* from chunk j to chunk i and C is the *current context*, usually defined as the set of all chunks that are in a buffer [ABB+04, p. 1042], [TLA06, p. 33], [act12, unit 5]. Figure **??** illustrates the addition-fact $5 + 2 = 7$ with the corresponding quantities introduced in the last equation.

The values for W_j reflect how many sources of activation are in the current context. A source of activation is a chunk in the goal buffer or in all buffers (called current context), depending on the version of the ACT-R theory [ABB+04, p. 1042], [TLA06, p. 33], [act12, unit 5, p. 1]. To limit the total amount of source of activation, W_j is set to $\frac{1}{n}$, where n is the number of sources of activation. With this equation, the total amount of activation that can spread over declarative memory is limited, since the more chunks are in the current context, the less important becomes a particular connection between a chunk from the context with a chunk from declarative memory.

Strength of Association and Fan effect In equation (2.3) the strength of association S_{ji} from a chunk j to a chunk i is used to determine the activation of a chunk i. In the ACT-R theory, the value of S_{ji} is determined by the following rule: If chunk j is not a value in the slots of chunk i and $j \neq i$, then S_{ji} is set to 0. Otherwise S_{ji} is set to:

$$S_{ji} = S - \ln(\mathrm{fan}_j) \tag{2.4}$$

where fan_j is the number of facts associated to term j [AS00, p. 1042]. In more detail: "fan_j is the number of chunks in declarative memory in which j is the value of a slot plus one for chunk j being associated with itself" [act12, unit 5, p. 2].

Hence, equation (2.4) states that the associative strength from chunk j to i decreases the more facts are associated to j.

This is due to the *fan effect:* The more facts a person studies about a certain concept, the more time he or she needs to retrieve a particular fact of that concept [AR99, p. 186]. This has been demonstrated in an experiment presented in [AR99], where every participant studied facts about persons and locations like:

- A hippie is in the park.
- A hippie is in the church.
- A captain is in the bank.
- ...

For every person the participants studied either one, two or three facts. Afterwards, they were asked to identify targets, that are sentences they studied, and foils, sentences constructed from the same persons and locations, but that were not in the original set of sentences. Figure **??** represents an example chunk network of the studied sentences (based on [AR99, fig. 1].

"The term *fan* refers to the number of facts associated with a particular concept" [AR99, p. 186]. In figure **??**, some facts are shown with their fan, S_{ji} and B_i values.

The result of the experiment was, that the more facts are associated with a certain concept, the higher was the retrieval time for a particular fact about that concept.

In the ACT-R theory, this result has been integrated in the calculation of the strengths of association: In equation (2.4) the associative strength decreases with the number of associated elements. The value S is a model-dependent constant, but in many models estimated about 2 [ABB+04, p. 1042]. Modelers should take notice of setting S high enough that all associative strengths in the model are positive [act12, unit 5, p. 3].

Figure **??** illustrates the activation spreading process.

Latency of Retrieval

As mentioned before, the activation of a chunk affects if the chunk can be retrieved (depending on a threshold and the activation values of the other matching chunks). In addition, activation also has an effect on the retrieval time of a chunk:

$$T_i = F \cdot \mathrm{e}^{-A_i} \tag{2.5}$$

where T_i is the *latency* of retrieving chunk i, A_i the activation of this chunk, as defined in equation (2.3), and F the *latency factor* which is usually estimated to be

$$F \approx 0.35\mathrm{e}^{\tau} \tag{2.6}$$

where τ is the retrieval threshold as mentioned in section 2.3.1, but F can also be set individually by the modeller. Nevertheless, in [ABB+04, p. 1042] it is stated that the relationship of the retrieval threshold and the latency factor in equation (2.5) seems to be suitable for a lot of models.

2.3.2 Production Utility

For the production system, there is the subsymbolic concept of *production utilities* to deal with competing strategies. For instance, if a child learns to add numbers, it may have learned different strategies to compute the result: One could be counting with the fingers and the other could be just retrieving a fact for the addition from declarative memory. If the child now has the goal to add two numbers, it somehow has to decide which strategy it will choose, since both strategies match the context.

In ACT-R, production utility is a number attached to each production rule in the system. Just like with activation of chunks, the production rule with the highest utility will be chosen, if there are more than one matching rules. The utilities can be set statically by the modeler, but there are also ways that the system learns the utilities automatically by practice.

In the current version of the ACT-R theory, a reinforcement learning rule based on the Rescorla-Wagner learning rule [RW72] has been introduced. The utility U_i of a production rule i is defined as:

$$U_i(n) = U_i(n-1) + \alpha\left(R_i(n) - U_i(n-1)\right) \tag{2.7}$$

where α is the *learning rate* which is usually set around .2 and $R_i(n)$ is the reward the production rule i receives at its nth application [And07, pp. 160–161]. This leads to the utility of a production rule being gradually adjusted to the average reward the rule receives [act12, pp. 6–7].

Usually, rewards can occur at every time and it is not clear which production rule will be strengthened by the reward. In [And07, p. 161] an example is described, where a monkey receives a squirt of juice a second after he presses a button. The question now is, which production rule is rewarded, since between the reward and the firing of a rule there always is a break. In ACT-R, every production that has been fired since the last reward event will be rewarded, but the more time lies between the reward and the firing of the rule, the less is the reward this particular rule receives. The reward for a rule is defined as the amount of external reward minus the the time from the rule to the reward. This implies, that the reward has to be measured in units of time, eg., how much time is a monkey willing to spend to get a squirt of juice? [And07, p. 161]

In implementations of ACT-R, rewards can be triggered by the user at any time or can be associated with special production rules that model the successful achievement of a goal (they check, if the current state is a wanted state and then trigger a reward, so every rule that has led to the successful state will be rewarded).

It is important to mention that by the definition of the reward for a production rule, rules also can get a negative reward, if their selection was too long ago. If one wants to penalize all rules since the last reward, a rule with reward 0 can be triggered, which leads to all rules applied before being rewarded with a negative amount of reward.

2.4 Learning

3 Constraint Handling Rules

A Quelltexte

In diesem Anhang sind einige wichtige Quelltexte aufgeführt.

```
1 :- use_module(library(chr)).
2
3 a(X) <=> check(X) | b.
4
5 check(13).
6 check(X) :-
7   X <10.
```

Bibliography

[ABB+04] ANDERSON, John R. ; BOTHELL, Daniel ; BYRNE, Michael D. ; DOUGLASS, Scott ; LEBIERE, Christian ; QIN, Yulin: An Integrated Theory of the Mind. In: *Psychological Review* 111 (2004), Nr. 4, 1036–1060. http://dx.doi.org/10.1037/0033-295X.111.4.1036. – DOI 10.1037/0033–295X.111.4.1036. – ISSN 0033–295X

[act] *The ACT-R Homepage*. http://act-r.psy.cmu.edu/

[act12] *The ACT-R Tutorial*. http://act-r.psy.cmu.edu/actr6/units.zip. Version: 2012

[And07] ANDERSON, John R.: *How can the human mind occur in the physical universe?* Oxford University Press, 2007. – ISBN 978–0–19–539895–3

[AR99] ANDERSON, John R. ; REDER, Lynne M.: The fan effect: New results and new theories. In: *JOURNAL OF EXPERIMENTAL PSYCHOLOGY GENERAL* 128 (1999), 186–197. http://www.andrew.cmu.edu/user/reder/publications/99_jra_lmr_2.pdf

[ARL96] ANDERSON, John R. ; REDER, Lynne M. ; LEBIERE, Christian: Working memory: Activation limitations on retrieval. In: *Cognitive psychology* 30 (1996), Nr. 3, 221–256. http://www.researchgate.net/publication/2605305_Working_Memory_Activation_Limitations_on_Retrieval/file/d912f50d532cd825c1.pdf

[AS00] ANDERSON, John R. ; SCHUNN, Christian D.: Implications of the ACT-R learning theory: No magic bullets. In: GLASER, R. (Hrsg.): *Advances in instructional psychology: Educational design and cognitive science* Bd. 5. Hillsdale, NJ : Lawrence Erlbaum Associates, 2000, S. 1–33

[RW72] *Kapitel* 3. In: RESCORLA, R. A. ; WAGNER, A. W.: *A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement*. New York : Appleton-Century-Crofts, 1972, S. 64–99

[Sun08] SUN, Ron: Introduction to Computational Cognitive Modeling. Version: 2008. http://www.cogsci.rpi.edu/~rsun/folder-files/sun-CHCP-intro.pdf. In: SUN, Ron (Hrsg.): *The Cambridge Handbook of Computational Psychology*. New York : Cambridge University Press, 2008, 3–19

[TLA06] TAATGEN, Niels A. ; LEBIERE, C. ; ANDERSON, J.R.: Modeling Paradigms in ACT-R. Version: 2006. http://act-r.psy.cmu.edu/papers/570/SDOC4697.pdf. In: *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation.* Cambridge University Press, 2006, 29–52

Name: Daniel Gall Matrikelnummer: 645463

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Daniel Gall