ulm university universität ulm

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 16, 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 (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. 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 and modules can perform their computation while other modules are working.

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 to 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 2.1.1. *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)
```

2.1.3 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*.

2.1.4 Procedural Knowledge

2.1.5 Goal Module

2.1.6 Other Modules/Outside World

2.1.7 Example

counting inspired by tutorial more examples can be found there

2.2 Serial and Parallel Aspects of ACT-R

2.3 Subsymbolic layer

2.3.1 Activation of Chunks

motivation and effects of activation activation equation base level learning latency

2.3.2 Production Utility

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/

[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