

ACT-R: A cognitive architecture

Terese Liadal

liadal@stud.ntnu.no

Seminar AI Tools

Universität des Saarlandes, Wintersemester 06/07

Dozenten: A. Ndiaye, M. Kipp, D. Heckmann, M. Feld

"If the brain were so easy we could understand it, we would be too stupid to."

Lyall Watson

Table of Contents

0. Abstract.....	3
1. Introduction.....	4
2. Cognitive modelling.....	4
3. The architecture: an overview.....	5
3.1 The modules.....	6
3.2 The buffers.....	6
3.3 The production system.....	6
3.3.1 A single step of cognition	7
3.4 Brain regions and ACT-R.....	7
4. The Programming Language ACT-R.....	8
5. Retrieving productions and chunks.....	10
5.1 Productions: Utility.....	10
5.2 Chunks: Activation	10
5.3 Production Compilation.....	10
6. Evolution of ACT-R.....	10
7. The GUI.....	11
8. Conclusion.....	17

0. Abstract

The aim of this paper is to give a brief introduction to ACT-R (Adaptive Control of Thought - Rational), a hybrid cognitive architecture, realized as a symbolic goal-oriented production system where some components are subsymbolic. ACT-R is not only a cognitive theory but also a general-purpose programming language written in lisp. ACT-R helps us write programs that try to explain human cognitive processes. ACT-R's architecture and design have been made to reflect certain assumptions about the human cognitive processes.

1. Introduction

ACT-R (Adaptive Control of Thought - Rational)[1] is a cognitive architecture that is both a programming language and a theory about how the mind works. In this paper I will try to present the need for a cognitive architecture, why they exist, and give an overview of how ACT-R is realized. ACT-R has both symbolic and subsymbolic components and features and is thus an example of a *hybrid* cognitive architecture. On the surface it is a symbolic goal-oriented production system [2], but the retrieval of facts and rules for the pattern matching and rule execution in the production system, as well as the speed of retrieval is governed by subsymbolic components that can be summarized in mathematical equations. The work of cognitive modelling, that is, to write a program or in ACT-R terminology, a *model*, is twofold. On the one side there is the task of using the right tool, and on the other side, there is the work of using this tool to write psychological plausible code. In this paper I will not address how one solves the latter task, but try to give an overview of what features ACT-R provides when it comes to write code that simulates cognition with a production system.

2. Cognitive modelling

What is understood by the term 'cognition'? It has several definition, ranging from meaning merely 'thought' [3] to 'Pertaining to the mental processes that include knowing, thinking, learning, judging, and problem solving[4] to mention a few. For the purposes of this paper I will interpret the term towards the latter definition. According to Wikipedia cognition is 'used in several loosely related ways to refer to a faculty for the human-like processing of information, applying knowledge and changing preferences'[5]. So cognitive modelling can be understood as some systematic way to imitate or simulate the processes involved in human learning, problem-solving, planning, or in the widest sense; what we think of as intelligence.

So cognitive modelling is to make a computational process that "thinks as a human", the focus is not so much on if the program solves a given task, but how it solves it. The goal of any cognitive modelling is to give added insight into the human thought processes.

Cognitive modelling lies in the middle of a huge interdisciplinary field,

including psychology and artificial intelligence, connectionism and computer science. It is used to try to make systems exhibit real human-like intelligent behavior, to model how we think a human goes about to solve a problem, and compare the data with real-life experiment data. Traditional methods like measuring time to complete task and measuring error rate ("competence") can be used, and lately also data from brain scans.

Newell proposed the Unified Theory of Cognition, a unified way to view all cognitive processes[6]. We seek to understand our brain and the way we think, and it is believed that to figure out what the common underlying methods for cognitive processes are will help us a good step further on the quest of decomposing and understanding ourselves. Other cognitive architectures, such as SOAR and EPIC have also been suggested, so ACT-R's role in UTC is not unique.

3. The architecture: an overview

The architectural components are the *modules*, where the most important ones are the goal module, the declarative memory module and the perceptual-motor modules. For each of these modules there is a corresponding buffer. Information fragments, so-called *chunks*, are made ready for use in the buffers, where pattern matching is performed by the central production system, which is made up of rules, so-called production rules or simply *productions*, which can access the content of the buffers and take appropriate action.

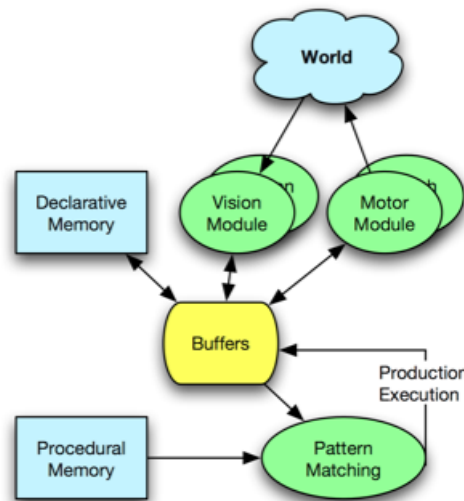


Figure 1: overview of core ACT-R components

ACT-R distinguishes between procedural and declarative knowledge, the procedural knowledge is the set of all the productions, and the declarative knowledge is the set of all chunks in the declarative memory. Procedural knowledge is the knowing how, and declarative knowledge is knowing that. That Paris is the capital in France is declarative knowledge, to know that to make tea you must first fill the kettle with water, and then heat the water is procedural knowledge. The sum of all

chunks does not equal the declarative memory, as chunks can be made available from other modules than the declarative memory.

3.1 The modules

The most common modules are the goal module, the declarative module and the perceptual-motor modules. Traditionally the focus was on the higher order brain functions, because the extent of the system had to be limited somehow [1]. As ACT-R has developed, it has become easier for the eager brain modeller to add modules, but a certain knowledge of lisp is required.

3.2 The buffers

Various parts of the brain are involved in cognition. Not only remembering facts located in the long term memory, but our visual processing, motoric competence (imagine learning to ride a bike by reading a book). But not all of this information is available at all times. We don't remember all we know, we don't attend to every object in our visual field and we don't always move consciously. The buffers are an attempt to model that, as it only offers a small part of the knowledge the various modules can offer at a time. The buffers are 'chunk creation scratch pads', and the only place where information is made available to the pattern matching of the production system. As of ACT-R 6, chunks contain copies of the content of the declarative memory or chunks retrieved from other modules. The declarative memory can only be modified by the clearing of a buffer, when the chunks previously located in the buffer are merged into declarative memory. ACT-R assumes that production rules are basic representations of cognitive ability, and that we can represent complex cognitive behavior as a sequence of production rule applications. Anderson et al. points out in [1] that this is indeed an approximation.

3.3 The production system

A production system is an environment that enables us to define what we want to happen when certain prerequisites are fulfilled, without having to explicitly take into consideration in what order these actions should be matched, checked or executed. "Productions consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be *triggered*. If a production's action is executed, it is said to have *fired*. A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered." [12] In ACT-R the IF-THEN is formalized in the productions, a state is whatever happens to be in the buffers, and the subsymbolic processes chooses which production to fire when more than one match.

3.3.1 A single step of cognition

ACT-R operates with simulated time steps where rule selection, pattern matching and execution are thought to happen simultaneously. Within one of these time steps, usually set to 50 ms for practical empirical reasons, a single production is executed. These series of production application make up the execution of complex cognitive behavior, and forces the strict sequential execution of production rules that characterized an ACT-R program.

3.4 Brain regions and ACT-R

As mentioned above, the architecture in ACT-R is chosen to reflect certain assumptions about the way human cognition works. In the latest version, namely ACT-R 5 and ACT-R 6 Anderson et al.[1] believes that the various modules in ACT-R can be directly mapped to the regions of the brain where learning, reasoning and other higher order brain functions take place (see Figure 2.). Experiments with brain scans show activation in these cortical regions when they are expected, which adds to the assumption that this mapping is at least partly correct[8].

The declarative module is believed to represent the functions of the long term memory, which is situated in the hippocampus. The buffers reflect the fact that we can only remember and retrieve some facts at a time. We are not simultaneously aware of everything we know, and some facts are harder to retrieve than others. The brains own production system is in the basic ganglia and the cortex. This is where we believe that new knowledge is produced, through reasoning and interaction of the outside world and things we can retrieve from memory. Anderson et al. [1] point out that the assumption that every single piece of information passes through the basic ganglia is simply false. Later extensions of ACT-R are planned to reflect this assumption.

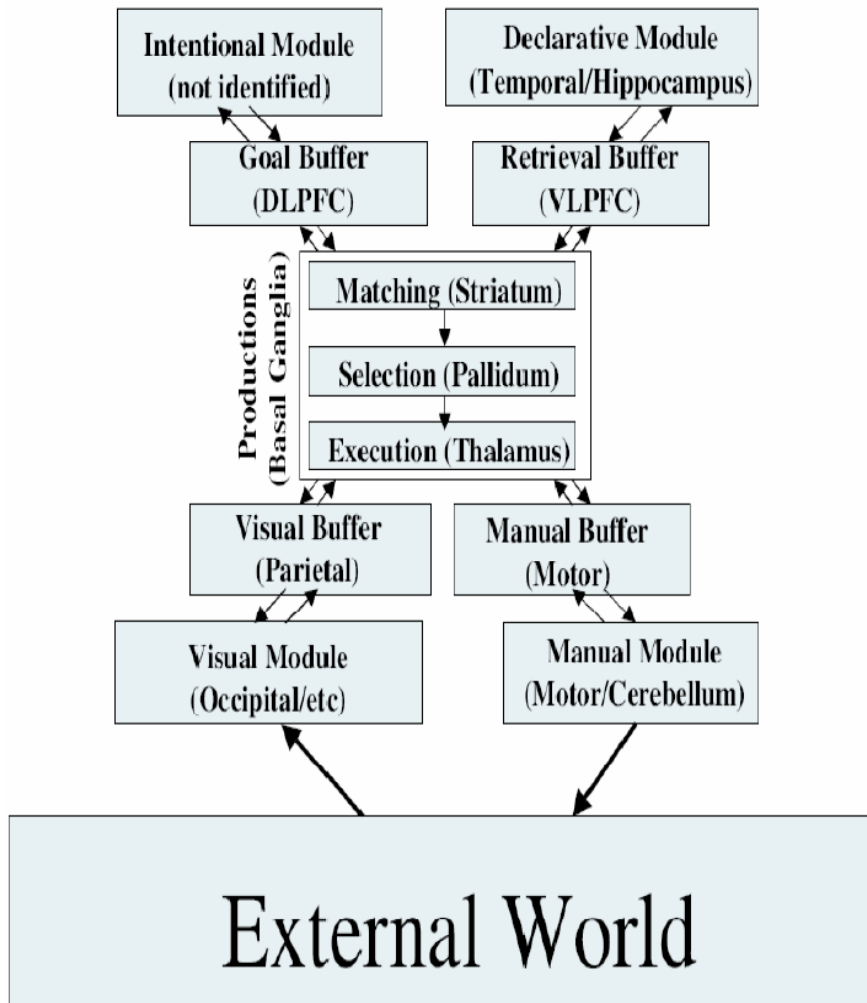


Figure 2: The mapping to the brain

4. The Programming Language ACT-R

Productions are written as (p {pattern to be matched} ==> {actions}), variables are written as =var-name, the "==" divides the pattern and actions of the production system rules.

Lets have a look at one of the simplest ACT-R programs possible, a model (ACT-R programs are called models) that counts.

```

(define-model count
(sgp :esc t :lf .05 :trace-detail high)
(chunk-type count-order first second)
(chunk-type count-from start end count)
(add-dm
 (b ISA count-order first 1 second 2)
 (first-goal ISA count-from start 1 end 2)
 )
)

```

```

(p start
  =goal>
    ISA      count-from
    start    =num1
    count    nil
  ==>
  =goal>
    count    =num1
  +retrieval>
    ISA      count-order
    first    =num1
)
(p increment
  =goal>
    ISA      count-from
    count    =num1
    - end    =num1
  =retrieval>
    ISA      count-order
    first    =num1
    second   =num2
  ==>
  =goal>
    count    =num2
  +retrieval>
    ISA      count-order
    first    =num2
    !output! (=num1)
)
(p stop
  =goal>
    ISA      count-from
    count    =num
    end      =num
  ==>
  -goal>
    !output! (=num)
)
(goal-focus first-goal)
)

```

Figure 3 : A simple ACT-R Program

The '*define-model*' command is part of every ACT-R model. The '*sgp*' command stands for "set global parameters", and this is where you adjust the numerous parameters ACT-R provides user control over. '*chunk-type*' is a command that defines the various chunks, it is roughly equivalent to a class definition, where the actual chunk instantiations would be analogous to objects. '*add-dm*' adds chunk to the declarative memory, and is the a priori knowledge of the system. New chunks can get added to the system during runtime. A chunk is instantiated by (*chunk-name isa chunk-type slot 1 value1 slot2 value2 ... slotN valueN*) where chunk-name can be any unique name, isa is part of the syntax and chunk-type has to be predefined by the '*chunk-type*' command. An example from the code above is (b ISA count-order first 1 second 2), chunk-name is b, chunk-type is count-order and slot1 is first and value1 is 1. Any slots that are not explicitly mentioned will have a default value of nil.

This program has three productions, defined by the command p, (p {*production body*}). The productions are called start, increment and

stop. Every left hand side (LHS) of every production accesses the goal module. To access a buffer, one must use the following syntax: =buffer-name>, +buffer-name> or -buffer-name>. = modifies the buffer, + requests a chunk from that buffer and indirect clears what happens to be in it, - clears a buffer. Every time a buffer is cleared, the chunk that happens to be in the buffer gets merged into the declarative memory.

5. Retrieving productions and chunks

5.1 Productions: Utility

A production is chosen among all the production that match the current chunks in the buffers because of its *utility*. Utility of a production is defined as:

$$U = PG - C + noise \quad [1]$$

Where U stands for utility, P is the probability that this production will lead to the desired goal ($P = \text{successes} / (\text{successes} + \text{failures})$), G is the goal value, a global parameter usually set to 20, and C is the cost of using this production, calculated in time units or similar. The noise adds some nondeterminism to the conflict resolution, since if we have many similar productions with about the same utility, different runs will choose different productions, and not always the one with slightly higher probability. Utility can be turned off with a global parameter, and also set manually for each production [10].

5.2 Chunks: Activation

A chunk is chosen in a similar way as the production, and the subsymbolic process that takes care of that is called activation. The activation can be summarized in the following equation:

$$A = \text{recency of use} + \text{usefulness} + \text{noise}$$

5.3 Production Compilation

For a pair of productions, the mechanism of production compilation can combine these specific executions into one production [1] p.1045. This is not possible for all the pairs of productions, specifically there are difficulties with the modules associated with input from the outside world. Initially the utility of this new production is lower than the first of the old productions of the pair, but if this is an frequently used combination, the new production will eventually replace the old ones (its utility will increase), and the execution of whatever these production does will be faster, simply because they now require just one production time step.

6. Evolution of ACT-R

The first version of ACT-R came in 1993, which was a neural network implementation of Anderson's theory. In 1998 came ACT-R 4.0 [9], later an extension to include visual and motorical modules were added, and in 2002 the ACT-R 5 came which integrated the motoric and visual modules. In 2005 a complete rewrite of ACT-R 5 was proposed; ACT-R 6. It is supposed to clear up some issues ACT-R 5 was dealing with, and makes a few other design decisions that make the two versions incompatible; aka ACT-R 6 is not backwards compatible. Some of the differences between ACT-R 5 and ACT-R 6 is that as of ACT-R 6 the set of all chunks does not equal the declarative memory [7].

7. The GUI

Figure 4 shows a screen-shot taken from program execution in the windows standalone environment. There are currently standalone versions for Windows XP and Mac PPC. On other systems you need to run ACT-R inside some version of lisp.

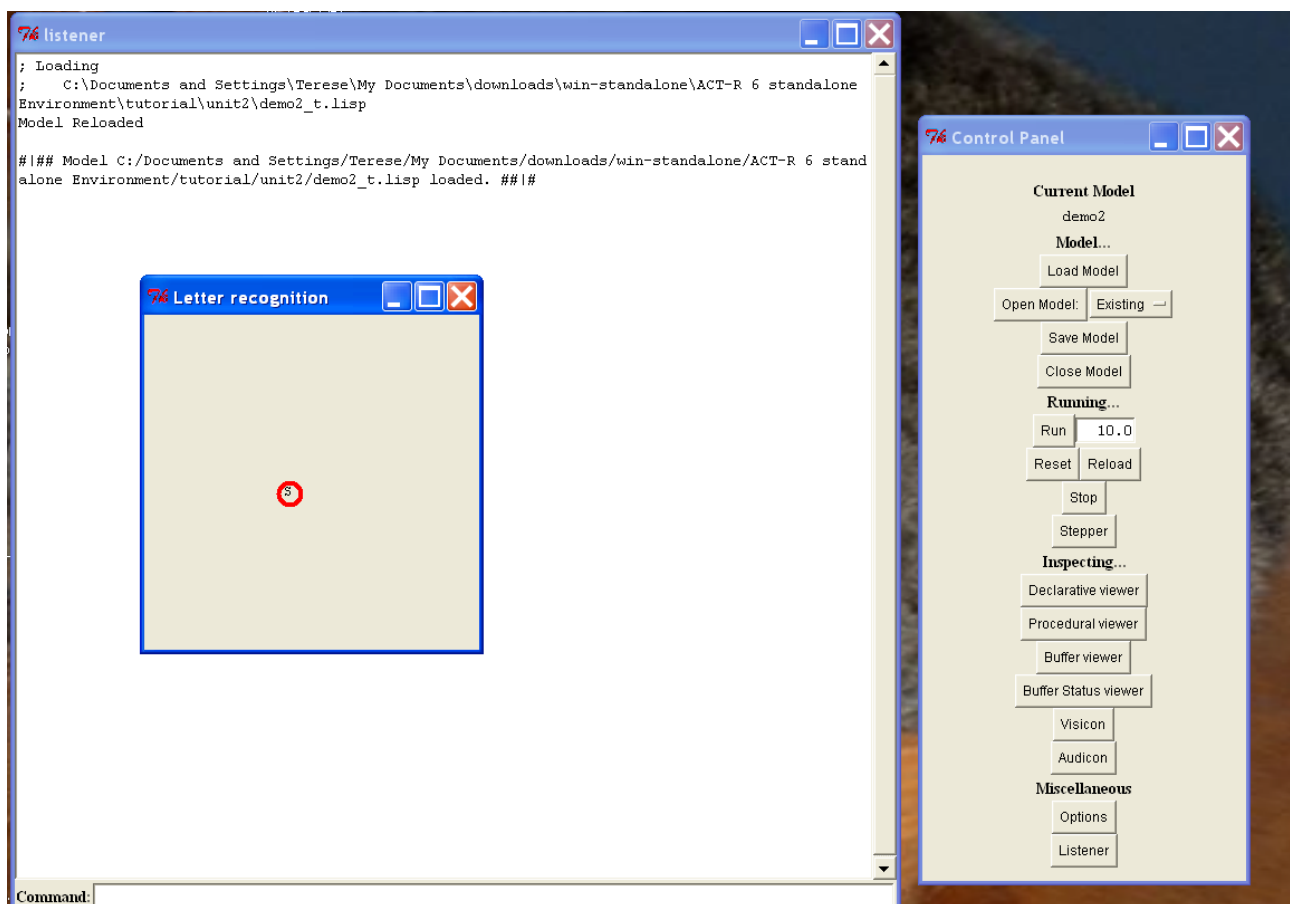


Figure 4: The ACT-R interface

Figure 4 is taken from a program in the tutorial units, where either the user or an ACT-R model is prompted to type in the key shown in the "Letter recognition" window. The red circle is the models way to let us know where it is looking. It is possible to switch between user and model execution by using the `setf` command, `(setf *actr-enabled-p* nil)` says the user gives the input, and `(setf *actr-enabled-p* t)` says the ACT-R

model gives it.

The ACT-R interface provides several features helpful for understanding of program execution and for debugging. At any time the declarative viewer (Figure 5) and the procedural viewer (Figure 7) will tell you what productions and declarative chunks you have, for easier inspection than looking at the source code, in addition to declarative chunks that are being created at run time when the buffers are cleared and merged into declarative memory (Figure 6).

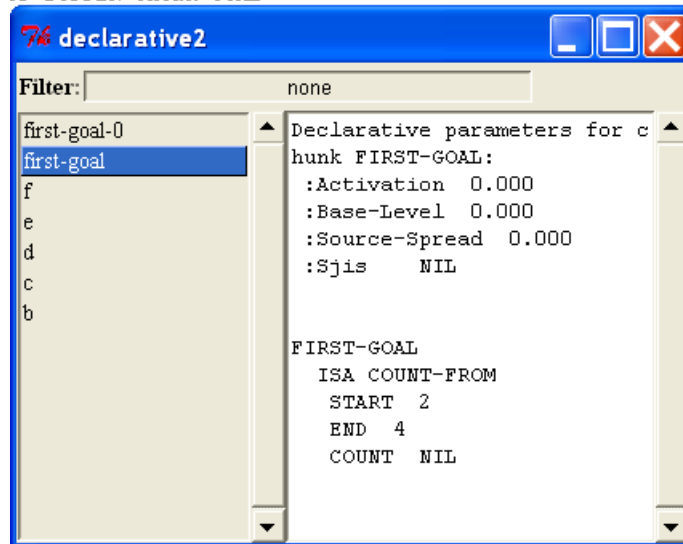


Figure 5: The declarative viewer

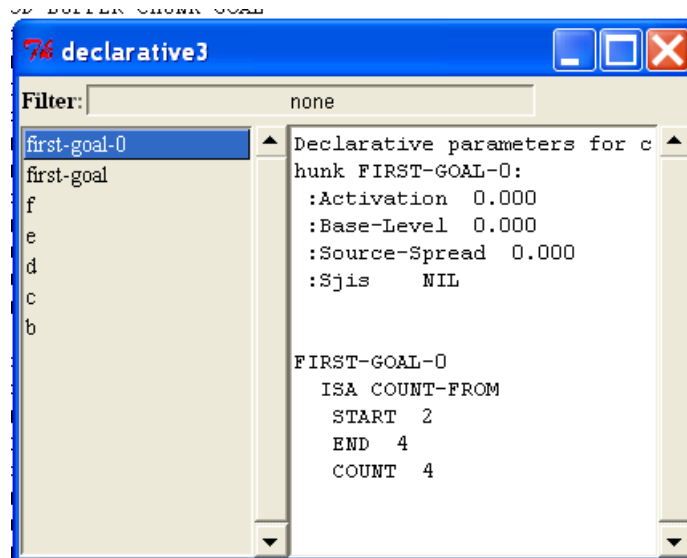


Figure 6: The creation of new chunks

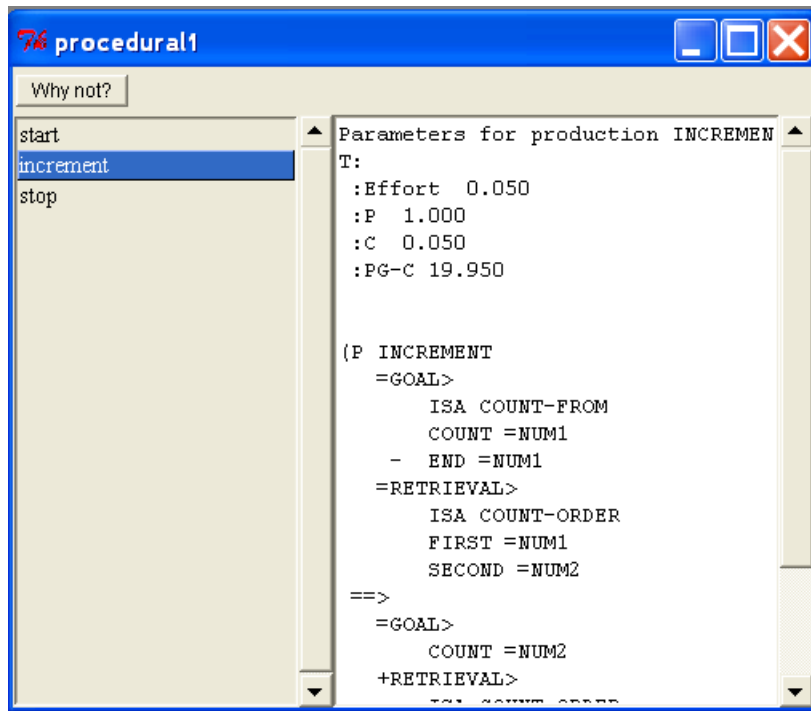


Figure 7: The procedural viewer

In trying to determine why a production does not fire, the 'why not' button (Figure 8) tells you which parts of the IF-part of the rule that is not fulfilled/matched. Most production systems provide similar functionality (for instance Jess [11]) and it is invaluable in debugging.

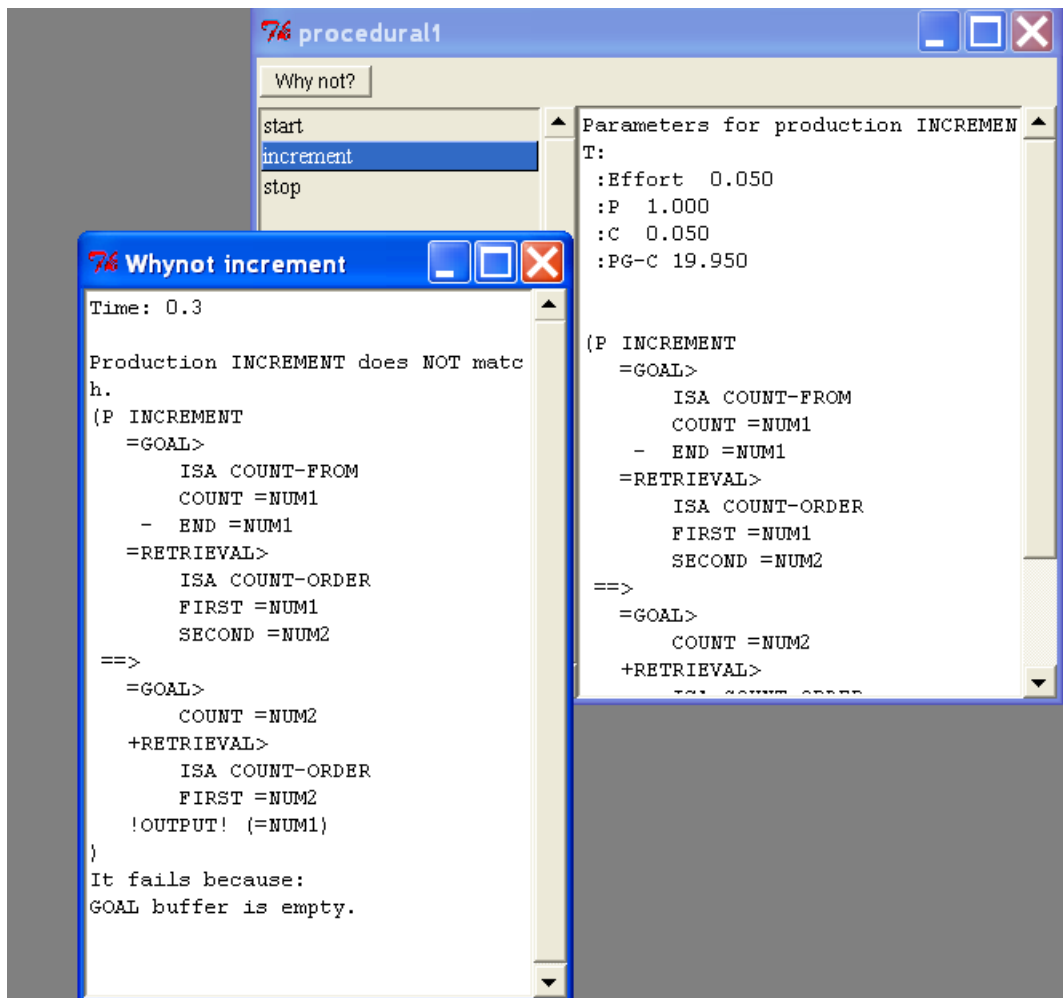


Figure 8: The 'why not' button

The buffer viewer (Figure 9) and buffer status viewer (Figure 10) tells you what is in the various buffers, and what status (busy, empty, full and so on) these buffers have. Especially when you are dealing with the buffers interfacing the outside world, it is useful to know if the fact you wished to retrieve has arrived in the buffer yet.

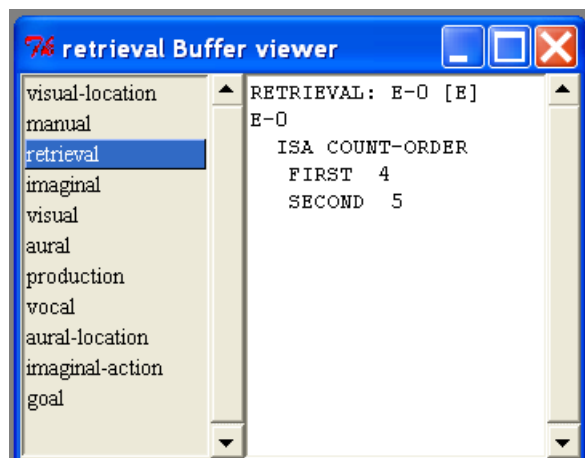


Figure 9: The buffer viewer

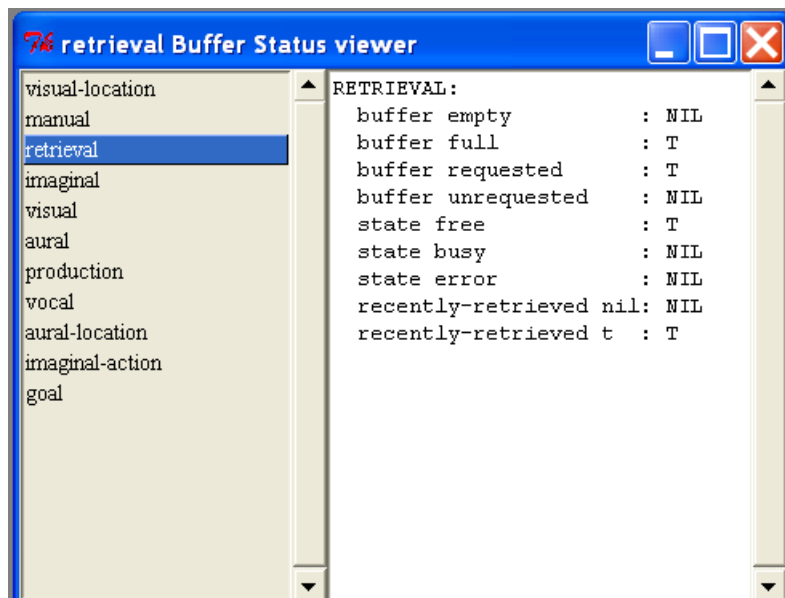


Figure 10: The buffer status viewer

8. Conclusion

In this paper I have presented the main features of ACT-R. It consists of declarative and procedural knowledge executed in a production system, and subsymbolic processes govern the retrieval of this knowledge for reasoning and execution.

ACT-R is a theory that concerns the buildup of the human cognitive processes. It is a way to make programs that take certain assumptions about the human brain into consideration when building systems that is supposed to exhibit intelligent behaviour. It is specially useful when it comes to model learning and the interaction between the various cortical regions.

References

- [1] J.R.Anderson, D.Bothell, M.D. Byrne, S.Douglass, C. Lebiere, Y. Qin. 2004. An Integrated Theory of the Mind; *Psychological Review*, 111(4). 1036-1060.
- [2] Roman V Belavkin; On Emotion, Learning and Uncertainty: A Cognitive Modelling Approach; Thesis submitted to The University of Nottingham for the degree of Doctor of Philosophy, August 2002, p.9-13
- [3] www.mentalhealthcare.org.uk/resources/glossary/, 21. March 2007
- [4] www.dphilpotlaw.com/html/glossary.html, 21. March 2007
- [5] <http://en.wikipedia.org/wiki/Cognition>, 21. March 2007
- [6] A. Newell. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press
- [7] <http://ACT-R.psy.cmu.edu/actr6/>, 21. March 2007
- [8] J.M. Fincham, C.S. Carter, V. van Veen, V.A. Stenger, and J.R.Anderson. 2002. Neural mechanisms of planning: A Computational Analysis Using Event-Related fMRI. *Proceedings of the national Academy of Sciences, USA*, 1999, p.3350
- [9] <http://en.wikipedia.org/wiki/ACT-R>, 28. April 2007
- [10] <http://act-r.psy.cmu.edu/tutorials/ACT-R5parameters.html>, 28. April 2007
- [11] *Jess in Action: Java Rule-based Systems*; Ernest Friedman-Hill; 2003
- [12] http://en.wikipedia.org/wiki/Production_system, 28. April 2007