

Getting started with ACT-R

A primer to the language primitives of ACT-R 6

Leon Urbas, 1. März 2006

Introduction

The purpose of this document is to help you to get started with ACT-R as a “cognitive programming language”. Thus it explains on a rather superficial level the main structures of the run-time engine and the main language elements. For an in-depth understanding of the mechanisms of ACT-R we recommend the tutorials and the book Atomic Components of Thought (Anderson & Lebiere 1999).

Elements of ACT-R

The core of ACT-R is a rule based production system or more precisely a physical symbol system (PSS): It has memories for symbols and structures of symbols and defines some processing functions that may create symbol structures from perception, change them in the memory or initiate motor action. Thus the steps that are necessary to construct a user model are as follows:

- 1) Define data structures that are capable to represent the knowledge about the system.
- 2) Write a set of productions that operate on this data structures
- 3) Run the model

The following sections go along this sketch of steps. Each section starts with drawing an overall picture with key-elements set in *italic letters*, then continues with a description of the syntax of the constituting elements and finally illustrates the concept with an example. In the syntax description CAPITAL LETTERS stand for place-holders (you have to fill in the appropriate values for the context) and **bold lettered words** depict key words that have to be typed as given.

Defining Data Structures

Declarative knowledge contains facts or intermediate results of information processing. The core element of the declarative memory is called *CHUNK*. A *CHUNK* basically is a typed data structure that has a unique name and may contain named references to other *CHUNKS* or terminal *SYMBOLS* like numbers. The places where these references go are called *SLOTS*. Which *SLOTS* a *CHUNK* may contain is defined by the corresponding a *CHUNK-TYPE* definition. The *CHUNK-TYPE* of a *CHUNK* is stored in a special *SLOT* named **isa**.

Definition of a data structure

The first step to set up a model is to define the *CHUNK-TYPES* according to the following syntax. Please note that you need not define the isa slot.

(**chunk-type** NAME-OF-TYPE NAME-OF-SLOT-1 NAME-OF-SLOT-2 ...)

Adding elements to the declarative memory

After having defined the data types you can add a set of chunks to the declarative memory with the add-dm command:

```
(add-dm
  ( NAME-OF-CHUNK-1
    isa NAME-OF-TYPE
    NAME-OF-SLOT-1 VALUE-OF-SLOT-1
    NAME-OF-SLOT-2 VALUE-OF-SLOT-2
    . . .
  )
  ( NAME-OF-CHUNK-2 isa ... )
)
```

VALUE-OF-SLOT is a reference to another CHUNK by NAME.

Example

A heavily used example is the definition of an addition fact, e.g. $7 + 2 = 9$. This fact may be represented by three slots: addend1, addend2 and finally the sum. In ACT-R you would write this as

```
(chunk-type number)
(chunk-type addition-fact addend1 addend2 sum)
(add-dm
  (seven isa number)
  (two isa number)
  (nine isa number)
  (fact-7+2=9 isa addition-fact addend1 seven addend2 two
  sum nine)
)
```

Defining Information Processing Procedures

The procedural knowledge about information processing of the user model is a set of production rules, each of them having a CONDITION and an ACTION-Part¹. The CONDITION-Part defines a set of TESTs on the data structures of the memories, the ACTION-Part defines a set of manipulative actions on the data structures. The complexity of the access to different memories and subsystems is encapsulated by means of BUFFERS. Currently ACT-R implements four buffers:

GOAL	This buffer holds information persistent over a sequence of productions
RETRIEVAL	This buffer moderates the access to the long term memory
VISUAL	This buffer is used to access the visual subsystem
MOTOR	This buffer is used to issue commands to the motor sub-system

Production Rules

All productions follow the same schema:

```
(P NAME-OF-PRODUCTION
```

¹ Because the condition is the precursor of the action and the syntax is often IF CONDITION THEN ACTION, the CONDITION Part is often named Left-Hand Side (LHS) and the Action Right Hand Side (RHS)

```

        CONDITION ( s )
    ==>
        ACTION ( s )
    )

```

Testing Buffers

The CONDITION-Part consists of a set of TESTs on buffers (depicted by =)

```

=BUFFER-1 >
    NAME-OF-SLOT-1 VALUE-OF-SLOT-1
    NAME-OF-SLOT-2 VALUE-OF-SLOT-2
=BUFFER-2 >
    NAME-OF-SLOT-3 VALUE-OF-SLOT-3
...

```

This could be read as “if BUFFER-1 contains an element where the slot NAME-OF-SLOT-1 equals VALUE-OF-SLOT-1 and slot NAME-OF-SLOT-2 equals VALUE-OF-SLOT-2 and BUFFER-2 contains an element where slot NAME-OF-SLOT-3 equals VALUE-OF-SLOT-3”

Defining Actions

The ACTION-Part consists of a set of actions on buffers in a comparable manner. If the BUFFER-NAME is preceded with = the current element is modified, a preceding + stands for a query to that buffer

```

=BUFFER-1 >
    NAME-OF-SLOT-1 VALUE-OF-SLOT-1
    NAME-OF-SLOT-2 VALUE-OF-SLOT-2
+BUFFER-2 >
    NAME-OF-SLOT-3 VALUE-OF-SLOT-3
...

```

This could be read as “then change BUFFER-1’ slot NAME-OF-SLOT-1 to VALUE-OF-SLOT-1 and NAME-OF-SLOT-2 to VALUE-OF-SLOT-2 and send a query to BUFFER-2 where NAME-OF-SLOT-3 equals VALUE-OF-SLOT-3”

Usage of Variables

The third usage of = is to denote variables. Variables are used to enforce slots of two buffers to contain the same value and to copy the value of a slot of one buffer to the slot of another one.

Testing two slots

The schema to write an IS-EQUAL CONDITION on two slots of different buffers is as follows:

```

=BUFFER-1 >
    NAME-OF-SLOT-1 =NAME-OF-VARIABLE-1
=BUFFER-2 >
    NAME-OF-SLOT-2 =NAME-OF-VARIABLE-1

```

This could be read as “if BUFFER-1 contains an element where the value of slot NAME-OF-SLOT-1 is equal to the value of slot NAME-OF-SLOT-2 of the element in BUFFER-2.

Copying between slots

To copy between two buffers you use variables to transport information between the CONDITION and ACTION part of the production

```
(P NAME-OF-CHUNK-1
  =BUFFER-1>
  NAME-OF-SLOT-1 =NAME-OF-VARIABLE-1
==>
  =BUFFER-2>
  NAME-OF-SLOT-2 =NAME-OF-VARIABLE-1
)
```

This could be read as “if BUFFER-1 contains an element that has a slot named NAME –OF-SLOT-1 then copy the value of this slot into slot NAME-OF-SLOT-2 of the element in BUFFER-2.

Running the Model

Now you are almost ready to run your model. Running means, that the run-time engine of ACT-R interprets the productions of your user model in a repeated Recognize-Act Cycle:

RECOGNIZE which productions could apply by evaluating their CONDITION-Part. Please not that ALL productions of your model are tested! The order of occurrence in the file is not significant here

ACT on the selected procedures, that is execute all or a subset of the ACTION-Parts of the recognized productions.

The set of productions that could apply on a given cycle is called the conflict set. ACT-R selects in an intermediate step called conflict resolution the “best” of this set due to a utility calculus that allows to learn and e.g. switch strategies in a changing environment.

In practice there are still some things to do to complete the model – define an initial state, code supporting lisp code, connect to a real world or simulation, a.s.o.