Cognitive Architectures

ACT-R
Outline

- Short glance on the history of ACT-R
- What is ACT-R?
- Mapping ACT-R onto the brain
- ACT-R 5.0 Architecture
- Components of ACT-R
- What is ACT-R used for?
- General discussion
History of the ACT-framework

- 1976: first ACT theory came out
- 1982: first ACT implementation appeared
- Since then both, the **Theory** and the **Implementations** were further developed (ACT, ACT-R, ACT-R2.0, ACT-R3.0, ACT-R4.0)
- 2001: release of ACT-R5.0 (Theory and Implementation) which is since then the state of the art ACT-R
What is ACT-R

ACT-R is a cognitive architecture

Researchers working on ACT-R strive to understand how people organize knowledge and produce intelligent behaviour.
What is ACT-R

ACT-R is a programming language
Models are written in ACT-R
During runtime of a model, ACT-R provides the runtime environment.

Due to its special design as a cognitive architecture, models in ACT-R can mirror human behavior on a cognitive psychology task.
What is ACT-R

- Based on facts derived from psychology experiments, ACT-R is a framework.
- Models in ACT-R reflect a certain aspect of cognition.
Framework?

ACT-R

Environment
ACT-R Architecture

ART-R claim: cognition as the interaction between specific units of knowledge:

- **Declarative knowledge**
  - Unit: **Chunks**
  - E.g. facts, goals, …

- **Procedural knowledge**
  - Unit: **Production rules**
  - E.g. action rules, behavior rules, …
ACT-R Architecture

Chunk       Chunk

Production rules

Chunk       Chunk

Environment
**ACT-R Architecture**

Chunks are created by specific modules
- visual module produces chunk “Christian is in visual field”
- Motor module produces “pressure on left hand”

Chunks set modules to action
- “search Christian” said to visual module

Modules transmit and retrieve information only out of buffers

Each module has a specific buffer for his chunks
ACT-R Architecture

Modules

Buffers

Production rules

Buffers

Modules

Chunk

Environment
Mapping ACT-R onto the brain

Question: How is ACT-R related to newest studies in neurobiology and neuroimaging?

Answer: all parts of ACT-R are designed to reflect certain brain areas!
Mapping ACT-R onto the brain

Modules

In a few examples we will try to give you a scratch of how ACT-R is designed.

Visual system: there are two build in visual modules in ACT-R referring to:

- The dorsal “where” pathway (locations)
- The ventral “what” pathway
Mapping ACT-R onto the brain

- Modules
  - Buffers
  - Production rules

- Visual Buffer (Parietal)
- Visual Module (Occipital/etc)

- Environment

- Visual cortex
- Auditory cortex
As for the visual system, other modules have been designed to match specific brain areas:

- **Manual buffer** = motor and somatosensory cortical areas
- **Goal buffer** = dorsolateral prefrontal cortex DLPFC
- **Retrieval buffer** = ventrolateral prefrontal cortex VLPFC (long-term declarative memory)
Mapping ACT-R onto the brain

**Production rules**

- Intentional Module (not identified)
- Visual Buffer (Parietal)
- Visual Module (Occipital/etc)
- Manual Buffer (Motor)
- Manual Module (Motor/Cerebellum)
- Goal Buffer (DLPFC)
- Declarative Module (Temporal/Hippocampus)
- Retrieval Buffer (VLPFC)

**Environment**

Motor

VLPFC

Posterior parietal
Mapping ACT-R onto the brain

Production rules

The basal ganglia are thought to implement production rules in ACT-R:

- Striatum: corresponding with cortical areas, responsible for pattern recognition
- Palladium: inhibitory component, performs conflict-resolution function
- Thalamus: projects to all major cortical areas, controls execution of production actions
Mapping ACT-R onto the brain
Production rules

Intentional Module
(not identified)

Declarative Module
(Temporal/Hippocampus)

Goal Buffer
(DLPFC)

Retrieval Buffer
(VLPFC)

Matching (Striatum)

Selection (Pallidum)

Execution (Thalamus)

Visual Buffer
(Parietal)

Manual Buffer
(Motor)

Visual Module
(Occipital/etc)

Manual Module
(Motor/Cerebellum)

Production rules

Environment
ACT-R Architecture

- **Intentional Module** (not identified)
- **Declarative Module** (Temporal/Hippocampus)
- **Goal Buffer** (DLPFC)
- **Retrieval Buffer** (VLPFC)
- **Visual Buffer** (Parietal)
- **Manual Buffer** (Motor)
- **Visual Module** (Occipital/etc)
- **Manual Module** (Motor/Cerebellum)
- **Matching (Striatum)**
- **Selection (Pallidum)**
- **Execution (Thalamus)**

**Buffers**

- **Declarative memory**
- **Procedural memory**
  - **“pattern-matcher”**

**Environment**
The modules

There are two types of modules:

- **memory modules.**
  - declarative memory
  - procedural memory

- **perceptual-motor modules**
  - take care of the interface with the simulation of the real world (visual and the manual modules).
chunks

chunks =

- units of declarative knowledge
- represent things remembered or perceived

example:

- 2+3=5
- Boston is the capital of Massachusetts
- there is an attended object in the visual field
- ...

chunks: examples

one way to model the fact: $2 + 3 = 5$

**DEFINITION**

(CHUNK-TYPE integer value)
(CHUNK-TYPE addition-fact addend1 addend2 sum)

**INSTANCE**

(three
isa integer
value 3)

Chunk

NAME

TYPE

{ATTRIBUTES}
chunks: examples

(CHUNK-TYPE integer value)
(CHUNK-TYPE addition-fact addend1 addend2 sum)

(three
  isa integer
  value 3)

(four
  isa integer
  value 4)

(seven
  isa integer
  value 7)

(fact3+4
  isa addition-fact
  addend1 three
  addend2 four
  sum seven)

reference to other chunks
chunks: examples
Fact: The cat sits on the mat.

Encoding: (Chunk-Type proposition agent action object)

(Add-DM
  (fact007
    isa proposition
    agent cat007
    action sits_on
    object mat)
)

chunks: examples
Fact: The black cat with 5 legs sits on the mat.

Chunks:

(Chunk-Type proposition agent action object)
(Chunk-Type cat legs color)

(Add-DM
  (fact007 isa proposition
    agent cat007
    action sits_on
    object mat)
  (cat007 isa cat
    legs 5
    color black)
)
chunks: examples

- animal
  - skin
  - moves

- fish
  - swims
  - gills

- bird
  - wings
  - flies

- shark
  - swims
  - dangerous

- salmon
  - swims
  - edible

- canary
  - sings
  - yellow

- ostrich
  - tall
  - can't fly
productions

**Procedural knowledge** to achieve a given goal:

- processes
- skills

**production** =

- unit of procedural knowledge
- condition-action rule that “fire” when the conditions are satisfied and execute the specified actions.
productions

conditions can depend on

- the current goal to be achieved,
- the state of declarative knowledge (i.e. recall of a chunk)
- the current sensory input from the external environment.

actions can:

- alter the state of declarative memory
- change goals
- initiate motor actions in the external environment
Structure of Productions

( P name

condition part Specification of Buffer Tests

. 

. 

delimiter ==> 

action part Specification of Buffer Transformations 

. 

)
Example of productions

(P increment

operation & buffer

=goal>
ISA count-from
number =num1

=retrieval>
ISA count-order
first =num1
second =num2

==> 

=goal>
number =num2

+retrieval>
ISA count-order
first =num2

(If the goal is to count from =num1

and a chunk has been retrieved of type count-order where the first number is =num1 and it is followed by =num2

Then

change the goal to continue counting from =num2

and request a retrieval of a count-order fact for the number that follows =num2)
Example of productions

(P find-next-word
  =goal>
    ISA comprehend-sentence
    word nil
  ==>
  +visual-location>
    ISA visual-location
    screen-x lowest
    attended nil
  =goal>
    word looking
)

← no word currently being processed.

← find left-most unattended location

← update state
Example of productions

(P attend-next-word

=goal>
  ISA comprehend-sentence
  word looking

=visual-location>
  ISA visual-location

=>

=goal>
  word attending

+visual>
  ISA visual-object
  screen-pos =visual-location

← looking for a word
← visual location has been identified
← update state
← attend to object in that location
Discussion

The \textit{atomic components} of thought?

- Is declarative knowledge (=chunk) available in every cognitive module?
- Semantic be modeled arbitrary
- Chunks be of any granularity
  - “pixel in visual field” vs. “Chris is standing in front of me”

- Can timing of the computation be compared with humans?
- Does the division of symbolic and subsymbolic processing make sense?
- Is ACT-R just a strange kind of programming language?
Environment

Productions (Basal Ganglia)

Retrieval Buffer (VLPFC)

Goal Buffer (DLPFC)

Matching (Striatum)

Selection (Pallidum)

Execution (Thalamus)

Visual Buffer (Parietal)

Manual Buffer (Motor)

Visual Module (Occipital/etc)

Manual Module (Motor/Cerebellum)

Intentional Module (not identified)

Declarative Module (Temporal/Hippocampus)
the perceptual-motor modules

no real sensors and effectors
the output of the visual and the input to the motor system are just modeled

the visual and manual module are most important (because of many computer tasks, involving scanning the screen, typing, moving the mouse...
the Act-R visual system

visual system

visual location module
“where”
-> dorsal stream

visual object module
“what”
-> ventral stream

Request:
• constraint a
• constraint b
...

Response:
location meeting those constraints

chunks

Production P

e.g.
- screen x lowest
- color: red
- screen-y-greater-than 153
....

→ leftmost word

→ red object (among green ones; supports experimental data from visual “pop-out-effects”)
the Act-R visual system

visual system

visual location module
“where”
-> dorsal stream

visual object module
“what”
-> ventral stream

chunk: representation of the visual location

attention-shift to that location

Production P
In humans: Suppose the goal is to add 64 + 36.
Assumption: the sum is not already stored, but one has to go through a series of substeps to come up with the answer and keep track of the various partial results (e.g. sum of the ten digits).

The goal module has this responsibility of keeping track of what these intentions are so that behavior will serve that goal.
How could the goal buffer be organized?

- good example for goal-subgoal structures in problem solving: Tower of Hanoi Problem
- naive human response: move the disks to their ultimate location (greedy)
- but: goal-subgoal strategy is often discovered during practice
Tower of Hanoi - problem

Interest of Anderson et al. (Tower of Hanoi: Evidence for the Cost of Goal Retrieval1, 2002):

- not which strategy is adopted, but what does it tell about goal-subgoal interaction
- what are the “cognitive costs” for implementing a subgoaling strategy
Experimental setup:

- The strategy to solve the problem was given and trained.
- Task: solve the problem as fast as possible
  - Formulate a goal by clicking the disk on the source peg, then the destination peg.
  - Do action or post goal.
- Time between actions and accuracy of moves were measured; also eye movements were recorded.
Strategy - Algorithm

1. formulate a goal
2. decision
   - if a legal move to achieve your goal is possible, do it and skip next step (3), otherwise post it on the goal stack

3. formulate a prerequisite goal
   - if you cannot move a disk D, find the largest disk that’s blocking the move and move it to a peg which is neither the source, nor the destination peg of D

4. Try again
   - go back to #2 to see whether you can achieve your last goal posted

5. Repeat the process
   - go back to step one, until all disks are at their final position
example demo

GOAL STACK

A

B

C

Do it!

Post it!
Table 3
Cognitive operations associated with the 30 critical actions

<table>
<thead>
<tr>
<th>Category</th>
<th>Action</th>
<th>Formulate</th>
<th>Store</th>
<th>Retrieve</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under 4</td>
<td>Post 3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2.85</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.69</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.72</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.69</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td>Move 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.66</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2.84</td>
</tr>
<tr>
<td>Under 5</td>
<td>Post 3</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>3.82</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.72</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.71</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3.09</td>
</tr>
<tr>
<td></td>
<td>Move 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.86</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3.35</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.67</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2.83</td>
</tr>
<tr>
<td></td>
<td>Post 3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2.96</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.74</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.71</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2.91</td>
</tr>
<tr>
<td></td>
<td>Move 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.65</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.81</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.69</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.71</td>
</tr>
<tr>
<td>End</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.71</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2.91</td>
</tr>
<tr>
<td></td>
<td>Move 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.65</td>
</tr>
<tr>
<td></td>
<td>Post 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.81</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.69</td>
</tr>
<tr>
<td></td>
<td>Move 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>Move 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.71</td>
</tr>
<tr>
<td>Correlations</td>
<td>Store</td>
<td>0.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieve</td>
<td>0.20</td>
<td>0.51</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>0.37</td>
<td>0.67</td>
<td>0.93</td>
<td></td>
</tr>
</tbody>
</table>
results

- participants are slower at those points where they must retrieve a goal, and are more slower the longer ago it was posted
- the accuracy data suggests that participants are forgetting their goals
- The tendency to inspect the goal stack increases dramatically at those retrieval points
- goal retrieval seems to be the major factor limiting performance in this task
But what has Act-R to do with this experiment?
Act-R

Act-R was used to model this task
Act-R 4.0 had a “perfect memory” goal stack on which all goals can be stored perfectly and accessed without any retrieval time costs

BUT: data shows clear goal limitations!

Altman & Trafton: memory for goals might behave like any other memory and be subject to forgetting
new Act-R model

- getting rid of the goal stack!
- relies on ACt-Rs general declarative memory to store goals
  - in Act-R each chunk has a base-level activation that increases each time the chunk is used and decreases with lack of use
  - Gaussian retrieval-probability function over the base-level activation
results

A

Latency (sec.)

- Under 4
- Theory

B

Latency (sec.)

- Under 6
- Theory

nice fit!
conclusions of this research:

- Cognitive architectures like Act-R(4.0) or SOAR are wrong in their assumption of a special goal stack.
- Goals in the subgoaling task are probably no different than other sort of intentions people set.
- Goals appear to behave like other more common kinds of declarative memory and shows the same effects in practice and retention interval.
Intentional Module (not identified)  
Declarative Module (Temporal/Hippocampus)  
Goal Buffer (DLPFC)  
Retrieval Buffer (VLPFC)  
Productions (Basal Ganglia)  
Matching (Striatum)  
Selection (Pallidum)  
Execution (Thalamus)  
Visual Buffer (Parietal)  
Manual Buffer (Motor)  
Visual Module (Occipital/etc)  
Manual Module (Motor/Cerebellum)  
Environment
the buffers

ACT-R accesses its modules (except for the procedural-memory module) through buffers.

For each module, a dedicated buffer serves as the interface with that module.

The contents of the buffers at a given moment in time represents the state of ACT-R at that moment.
the buffers

- each buffer can hold a relatively small amount of information (chunk)

- chunks that were former buffer contents are now stored in the declarative memory module

- buffers are conceptual similar to Baddley’s working memory “slave systems”
  - the central cognitive system can only sense the content of the buffers
  - the content of the chunks can only be accessed by the highly specialized modules
the buffers

the most important buffers in Act-R are:

- **Goal Buffer**
  - keeps track of one’s internal state in solving a problem
  - preserves information across production cycles

- **Retrieval Buffer**
  - holds information retrieved from long-term declarative memory
  - seat of chunk activation calculations

- **Manual Buffer**
  - responsible for control of hands

- **Visual “where” Buffer**
  - location

- **Visual “what” Buffer**
  - visual objects
  - attention shifts correspond to buffer transformations
Pattern matcher

- The pattern matcher searches for a production that matches the current state of the buffers.
- Only one such production can be executed at a given moment.
- That production, when executed, can modify the buffers and thus change the state of the system.
- Thus, in ACT-R cognition unfolds as a succession of production firings.
Production selection

Making Choices: Conflict Resolution

Expected Gain = \( E = PG - C \)

Probability of choosing \( i = \frac{\sum e^{E_i/t}}{\sum e^{E_j/t}} \)

\( t \) reflects noise in evaluation and is like temperature in the Boltzmann equation

\( P \) is expected probability of success

\( G \) is value of goal

\( C \) is expected cost

\( \alpha \) is prior successes

\( m \) is experienced successes

\( \beta \) is prior failures

\( n \) is experienced failures

Successes = \( \alpha + m \)

Failures = \( \beta + n \)
Outlook

What is ACT-R used for?
What is ACT-R used for?

ACT-R has been used successfully to create models in domains such as:

- learning and memory,
- problem solving and decision making,
- language and communication,
- perception and attention,
- cognitive development, or
- individual differences

...but not only in tasks of cognitive psychology ACT-R has applications...
What is ACT-R used for?
General Discussion

Modularity

- Fodor: “higher-level cognition is impossible to encapsulated into separate components”
- General doubts about success of function localization in brain imaging research
Reference


Anderson et al. (2002) *Tower of Hanoi: Evidence for the Cost of Goal Retrieval*

http://act.psy.cmu.edu