Learning in ACT-R: Chunking Revisited

Richard L. Lewis

Department of Psychology
University of Michigan

March 22, 2003
Acknowledgements

- NASA Ames Research Center
- Roger Remington, Alonso Vera, Bonnie John, Mike Matessa
- ACT-R research group
  - particularly Niels Taatgen, John Anderson, Christian Liebere
Overview

- Overview of current version of ACT-R (5.0)
  - How it works
  - Highlight major new developments
  - With some editorial comments and comparisons to Epic and Soar

- Model of learning a hierarchically structured task
  - A kind of *learning from instruction*

- Summarize interesting properties and general implications
**ACT-R 5.0: Buffers and modules**

- **Environment**
- **Productions**
- **Retrieval Buffer**
- **Goal Buffer**
- **Selection**
- **Execution**
- **Visual Buffer**
- **Manual Buffer**
- **Visual Module**
- **Manual Module**
- **Declarative Module**

Diagram shows the flow of information from the environment to the various modules and buffers, including matching, selection, and execution processes.
ACT-R 5.0: Buffers and modules

- Declarative Module
  - Goal Buffer
  - Retrieval Buffer

- Productions
  - Matching
  - Selection
  - Execution

- Visual Buffer
- Manual Buffer

- Visual Module
- Manual Module

- Environment

Keeps track of where one is in task; intermediate results.
ACT-R 5.0: Buffers and modules

- Declarative Module
  - Goal Buffer
  - Retrieval Buffer
  - Productions
    - Matching
    - Selection
    - Execution
  - Visual Buffer
  - Manual Buffer
    - Visual Module
    - Manual Module

Long term declarative store (contains chunks)

Environment
ACT-R 5.0: Buffers and modules

- Environment
- Productions
- Retrieval Buffer
- Matching
- Selection
- Execution
- Visual Buffer
- Manual Buffer
- Visual Module
- Manual Module

Declarative Module

Goal Buffer

Retrieval Buffer

Holds retrieved chunk from declarative memory
ACT-R 5.0: Buffers and modules

Separate location, object identity buffers

Environment

- Declarative Module
- Goal Buffer
- Retrieval Buffer
- Visual Buffer
- Manual Buffer
- Visual Module
- Manual Module
ACT-R 5.0: Buffers and modules

- Declarative Module
  - Goal Buffer
  - Retrieval Buffer
- Productions
  - Matching
  - Selection
  - Execution
- Visual Buffer
- Manual Buffer
- Visual Module
- Manual Module

Environment

Key-strokes, mouse clicks, mouse movements
ACT-R 5.0: Buffers and modules

Match and modify buffers

Declarative Module

Goal Buffer

Retrieval Buffer

Productions

Matching

Selection

Execution

Visual Buffer

Manual Buffer

Visual Module

Manual Module

Environment
~ 100 Published Models in ACT-R 1997-2002

I. Perception & Attention
   1. Psychophysical Judgements
   2. Visual Search
   3. Eye Movements
   4. Psychological Refractory Period
   5. Task Switching
   6. Subitizing
   7. Stroop
   8. Driving Behavior
   9. Situational Awareness
  10. Graphical User Interfaces

II. Learning & Memory
   1. List Memory
   2. Fan Effect
   3. Implicit Learning
   4. Skill Acquisition
   5. Cognitive Arithmetic
   6. Category Learning
   7. Learning by Exploration and Demonstration
   8. Updating Memory & Prospective Memory
   9. Causal Learning

III. Problem Solving & Decision Making
   1. Tower of Hanoi
   2. Choice & Strategy Selection
   3. Mathematical Problem Solving
   4. Spatial Reasoning
   5. Dynamic Systems
   6. Use and Design of Artifacts
   7. Game Playing
   8. Insight and Scientific Discovery

IV. Language Processing
   1. Parsing
   2. Analogy & Metaphor
   3. Learning
   4. Sentence Memory

V. Other
   1. Cognitive Development
   2. Individual Differences
   3. Emotion
   4. Cognitive Workload
   5. Computer Generated Forces
   6. fMRI
   7. Communication, Negotiation, Group Decision Making
Knowledge representation: Procedural vs. declarative

- This has long been a feature of ACT theories
  - Cognition emerges as interaction between procedural and declarative knowledge

- **Declarative memory** contains chunks
  - Structured configurations of small set of elements
  - Sometimes described as containing “facts”; but real issue is not content, but how they are accessed

- **Procedural memory**: production rules
  - Asymmetric condition-action pairs
  - Match on buffers, modify buffers
“Chunks” in declarative memory

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

(three
  isa integer
  value 3)

(four
  isa integer
  value 4)

(seven
  isa integer
  value 7)
“Chunks” in declarative memory

THREE isa INTEGER

FACT3+4 isa VALUE

SEVEN isa VALUE

FOUR isa VALUE

ADDEND1 isa ADDITION-FACT

ADDEND2 isa ADDITION-FACT

INTEGRER isa VALUE

SUM isa VALUE

3 isa VALUE

7 isa VALUE

3 isa ADDEND1

4 isa ADDEND2

37 isa VALUE
“Chunks” in declarative memory
More “chunks”

Declarative memory contains partial products as well (thus serves as a WM)

(NP28
  isa syn-obj
  word Dog
  spec-of IP27
  spec D28
  cat N
  case Nom
  number Sing
  finite nil
  attached Yes-Attached)

(saw-v isa major-cat-entry
 word saw
 cat v
 finite yes-finite
 tense past
 number sing-plural
 looking-for-case acc
 looking-for-cat N)
Productions: Match and modify buffers

- Productions match against and modify buffers
  - Modifying the *goal buffer* means (a) keeping track of *intermediate results* of computation or (b) changing momentary *control state*, or (c) *replacing the goal chunk*
  - Modifying other buffers means issuing a *request* to the corresponding module to do something

- Productions do *NOT* match directly against declarative memory
Productions: Matching and modify buffers

- Productions in ACT-R 5.0 often come in pairs

(P retrieve-answer
 =goal>
 ISA comprehend-sentence
 agent =agent
 action =verb
 object =object
 purpose test

==> 
 =goal>
 purpose retrieve-test
 +retrieval>
 ISA comprehend-sentence
 action =verb
 purpose study
)

← sentence processing complete

← update state

← retrieve sentence involving verb
Generating a response

(P answer-no
   =goal>
   ISA comprehend-sentence
   agent =agent
   action =verb
   object =object
   purpose =object
   =retrieval>
   ISA comprehend-sentence
   - agent =agent
   action =verb
   - object =object
   purpose study
   ==> 
   =goal>
   purpose done
   +manual>
   ISA press-key
   key "d"
)
## Summary of ACT-R performance and learning

### Performance

<table>
<thead>
<tr>
<th></th>
<th>Declarative</th>
<th>Procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbolic</strong></td>
<td>Retrieval of Chunks ✔</td>
<td>Application of Production Rules ✔</td>
</tr>
<tr>
<td><strong>Subsymbolic</strong></td>
<td>Noisy Activations, Control Speed and Accuracy</td>
<td>Noisy Utilities, Control Choice</td>
</tr>
</tbody>
</table>

### Learning

<table>
<thead>
<tr>
<th></th>
<th>Declarative</th>
<th>Procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symbolic</strong></td>
<td>Encoding Environment and Caching Goals</td>
<td>Production Compilation</td>
</tr>
<tr>
<td><strong>Subsymbolic</strong></td>
<td>Bayesian Learning</td>
<td>Bayesian Learning</td>
</tr>
</tbody>
</table>
Activation-based retrieval: 
**Focus, decay, & interference**

Only the contents of the goal buffer and retrieval buffer and are available for processing (production match)
Activation-based retrieval: Focus, decay, & interference

Base level activation is function of usage history; yields both power law decay & power law learning

\[ A_i = B_i + \ldots \]
Activation-based retrieval: Focus, decay, & interference

A set of probes $P$ provides additional activation to memory elements that match the probes (and reduced activation to elements that mismatch). Result is a soft match.

$$A_i = B_i + \sum_k P_k M_{ki}$$
Activation-based architecture: Focus, decay, & interference

Both retrieval time and probability of retrieval are a function of the activation of the target and its competitors. Thus, interference depends on the number, activation, and similarity of competitors.

\[ A_i = B_i + \sum_k P_k M_{ki} \]
Base level learning
Example: Sentence processing

1. New word arrives and is encoded

2. Set retrieval cues based on new word, via production rule

3. Cues boost activation of prior constituents

4. Attach to retrieved constituent (most highly active), via production rule
A pipelined architecture
A pipelined architecture

**Visual buffer/processor**

**Cognitive processor**

**Retrieval buffer**

**Visual processor** [executes saccades, delivers encoded visual items]

**Cognitive processor** [production system operating on 50ms cycle; issues retrieval requests, perceptual/motor commands]

**Retrieval buffer** [receives requests in form of memory probes (features to match against); delivers result of retrieval]
A pipelined architecture

Considerable available parallelism
  - Production rules fire in parallel while retrievals in process, while visual system programming a saccade, while motor system executing command, etc.
Trace of the model in action

Time 0.687: Module :VISION running command FIND-LOCATION
Time 0.687: Attend-Word-Saw Selected
Time 0.737: Attend-Word-Saw Fired
Time 0.737: Module :VISION running command MOVE-ATTENTION
Time 0.737: Project-Ip-From-Nominative-Noun Selected
Time 0.787: Module :VISION running command FOCUS-ON
Time 0.787: Project-Ip-From-Nominative-Noun Fired
Time 0.787: Lexical-Retrieval-Request Selected
Time 0.837: Lexical-Retrieval-Request Fired
Time 0.844: Saw-V Retrieved
Time 0.844: Set-Retrieval-Cues-Based-On-Tensed-Verb Selected
Time 0.894: Set-Retrieval-Cues-Based-On-Tensed-Verb Fired
Time 0.896: Ip22 Retrieved
Time 0.923: Match-Ip-Expectation1 Selected
Time 0.946: Match-Ip-Expectation1 Fired
Production choice and utility learning

- Only a single production can fire at a time (a serial bottleneck); the production with the highest utility is selected.

- The parameters $P$ and $C$ are incrementally adjusted as function of experience.

Expected Utility = $PG - C$

- $P = \frac{\text{Successes}}{\text{Successes + Failures}}$
- $C = \text{cost of achieving goal if production selected}$
- $G = \text{value of current goal}$
Production composition
(Taatgen & Anderson)

IF reading the word for a paired-associate test
and a word is being attended
THEN retrieve the associate of the word
and note trying to recall

Recall Vanilla-7

IF recalling for a paired-associate test
and an associate has been retrieved with response N
THEN type N
and note the answer is being typed

Results in:
IF reading the word for a paired-associate test
and “vanilla” is being attended
THEN type “7”
and note the answer is being typed
Some composition principles

1. **Perceptual-Motor Buffers:** Avoid compositions that will result in jamming when one tries to build two operations on the same buffer into the same production.

2. **Retrieval Buffer:** Except for failure tests proceduralize out and build more specific productions.

3. **Safe Productions:** Production will not produce any result that the original productions did not produce.
Summary of major new developments and shifts

- Introduction of perceptual motor components
  - (inspired by/taken from Epic)
- Buffer structure/constrained production form
- Factoring out retrieval
  - Productions now come in pairs; retrieval happens in parallel, can be interrupted
- Production composition
ACT-R and Soar

- Obvious differences (uniform production memory in Soar, no subsymbolic level)
- But: *Soar’s control structure is more flexible than ACT-R’s* (least-commitment run-time decision cycle supported by parallel knowledge retrieval vs. utility learning)
- Not clear how ACT-R would learn *contextually conditioned* control knowledge
  - Possible response: Soar’s control structure is layer above ACT-R
    - Seems reasonable response for Epic, but not for ACT-R
ACT-R and Epic

- Epic’s cognitive processor is completely parallel; no serial bottleneck (ACT-R has two)
- Not clear if ACT-R’s single serial control stream is fast enough for all kinds of complex real time tasks
  - Example: I have serious doubts about sufficiency for *language processing* by itself, let alone in concert with other cognitive tasks
  - Though ACT-R group is agnostic about whether language has special dedicated processors (Anderson et al 2001)
A model of learning hierarchically controlled behavior

- We’re exploring an ACT-R model that can take a **declarative specification of a task in the form of a goal-subgoal hierarchy**, interprets that specification to perform the task, and **gradually learns new task-specific production rules**
  - The interpreter is just a set of production rules that know how to traverse a task hierarchy
  - Hierarchy bottoms out in motor/perceptual primitives
Why?

(1) Subgoal hierarchies **useful descriptions** of tasks, from using ATMs to flying tactical air missions
   - So any process that converts these to productions would be useful

(2) Subgoal hierarchies have proven important in the **architecture of flexible performance systems** (e.g., TACAir-Soar)
   - TACAir-Soar success = hierarchically controlled behavior + flexible/interruptible control structure

(3) **Learning** to perform such tasks is important

(4) In particular, **instruction taking** important capability
   (Lewis, Newell & Polk 1989; Huffman, 1994; Anderson, 2001; Taatgen, 2002)
..and particularly critical for ACT-R

- Because ACT-R has just one active goal chunk available to control processing!
- No architecturally-distinguished goal-subgoal relations or processes (pushes, pops)
  - Therefore no architectural learning mechanism specialized to learn across goal/subgoal boundaries!

- Can a non-goal-based learning mechanism “chunk” arbitrary goal hierarchies?
- Can a single-goal architecture behave as flexibly as an architecture with a goal stack?
The task
Goal/subgoal decomposition

do-banking
  initiate
  transaction
    withdraw
    amount
      correct
    end

account #
  card

PIN
  4 9 0 1

withdraw
  amount
  correct
  8 0
Declarative language

- Based on PDL (procedural description language in Apex; Freed 2000)
  - Rather GOMS-like
- Important properties:
  - Defines a **hierarchical decomposition** of the task
  - Defines a **partial ordering** on subgoals/primitive steps
Examples

(Do-banking-step-a
  ISA step-definition
  step-label a
  parent-task do-banking
  task type-pin
  arg1 none
  arg2 none
  wait-for-a not-done
  wait-for-manual free
  if-failure-goto none)

(Type-PIN-step-b
  ISA step-definition
  step-label b
  parent-task Type-PIN
  task press-key
  arg1 "B"
  arg2 none
  wait-for-a done
  wait-for-b not-done
  wait-for-manual free
  if-failure-goto none)
The local control state

- Hierarchy can be arbitrarily deep, but at any given point, only following information is kept local in the goal buffer
  - Which steps in this local goal have been accomplished (done, not-done)
  - Name of parent-task (single symbol)
  - Single symbol denoting entire control state of all instantiated goals higher in the hierarchy
  - Intermediate results
The interpretive algorithm

(1) Execute the step directly if possible.

(2) Recognize the current control state.
   – Result is a symbol (gensym) that denotes the current pattern of
dones/not-dones plus symbol denoting the parent control state
   – How? Attempt to retrieve an existing chunk with this pattern
     – If fail, create new chunk and use chunk ID as recognition
       symbol

(3) Retrieve a candidate step definition.
   – What are the retrieval cues? Current control state pattern!
   – But in general, could use any knowledge source here

(4) Check wait-fors, and instantiate retrieved step as
new controlling task
   – Destructively modify goal buffer

(5) If step is done, “pop” by unpacking parent
control state symbol (attempt chunk retrieval)

(6) Goto 1
**Control chunks:** Coding control state information

Control-State184

- isa CONTROL-STATE
- parent-task None
- task Task-X
- arg1 nil
- arg2 nil
- step-a Done
- step-b Not-Done
- step-c Done
- step-d Not-Done
- step-e Not-Done
- step-f Not-Done
Kinds of learning/behavior that emerge

- Learning to **traverse the goal hierarchy** via productions, without declarative retrievals
- Learning **direct associations** from goals to motor actions/results associated with deeper subgoals
- **Collapsing together** cognitive results and motor actions
- Learning to **recognize and transition** between **new control state codes**
- Frantically **trying to fill slack time** when waiting for motor processor to complete
Direct association from high level goal to response

IF task is Do-Banking
and no steps are done
and there is no parent task
and the manual processor is free

THEN

control-state is C58
parent-control-state is C41
task is Enter-PIN
press the “4” key
step-A is done
request retrieval for next step
Creation of multiple results in parallel along with motor response

IF task is Task-X
    and no steps are done
    and there is no parent task
THEN
    parent control-state is C143
    click the mouse
    produce results of step A, B, C
    task is terminate
Pop, return result, transition control state

IF  task is TASK-C
    and parent-control-state is C54
    and there is no task-c result
    and the parent task is TASK-X
THEN
    control-state is C61
    task is TASK-X
    TASK-C-RESULT is seven
    step-C is done
    request next step definition
XAPS: A blast from the past (Rosenbloom & Newell 1981)

- This sort of chunking of goal hierarchies is similar to the original Rosenbloom & Newell work on chunking, in that critical part of “chunks” are new symbols that denote hierarchical structure.

- BUT: Two big differences
  1. In XAPS chunking, symbols denoted *encoded stimuli* and *response patterns*. In the ACT-R model, symbols denote *control states*
     **CLAIM: WE NEED ALLTHREE.**
  2. XAPS chunking, like the Soar chunking that it evolved into, is a mechanism predicated over goal-subgoal relations.
Effects of learning
Four interesting properties

(1) Learns new control codes *(control chunks)*
   - Supports efficient traversal; provides additional way to characterize procedural skills

(2) Learns *within* and *across* goals/subgoals via same mechanism
   - But without architectural subgoals and therefore without a learning mechanism based on subgoals

(3) Permits *behavior conditioned on any goal/supergoal* in hierarchy
   - Not blind to context, because control state symbol denotes entire hierarchy

(4) Still *interruptible* in principle
Ok, four more...

(5) (Should) compile down to permit **as much parallelism as possible** in architecture
- Early, behavior is shaped by task-structure
- After practice, behavior is shaped by architecture

(6) System can always **fall back on explicit processing of goal structure** when needed
- This behavior evident in current model

(7) May **avoid some classic Soar chunking problems**
- Noncontemporaneous constraints
- Data chunking

(8) Step toward **Instruction taking!**
Can it do everything Soar chunking can do?

- **NO.**
  - At least, not in a straightforward way
  - What Soar’s chunking can do is a function of Soar’s control structure
    - Recall earlier remarks about relatively limited nature of ACT control structure

- But, this seems to be an issue of control structure differences, rather than learning mechanism differences
Limitations, concerns

- This is still extremely preliminary work, using early version of new learning mechanism
- Not clear it is asymptoting at optimal
- Somewhat erratic behavior; learns many useless productions
- Deliberate control-state recognition feels heavy-handed
- Works with fixed goal/subgoal hierarchy now
  - Though shouldn’t be a limitation in principle
- Will it scale to complex, dynamic tasks?
Final golden nugget: Data

All subjects: Total times by trial (smoothed)
As function of hierarchy boundary