Cognitive Modeling
Cognitive Architectures

Ute Schmid supported by Michael Siebers

Kognitive Systeme, Angewandte Informatik, Universität Bamberg

last change: 28. Oktober 2010
Cognitive Architectures

- “unified theory of cognition”
  - explicit definition of basic mechanisms of information processing
  - assumption that these mechanisms are constant over all domains (problem solving, language understanding, pattern recognition etc.)
  - basic mechanisms: control of interaction with environment, representation of information in memory, strategy to select rules
  - Advantage: different models realized in the same architecture get comparable
  - Disadvantages: Modeling language often over-constrains what is modelled, modelling can easily get “unnatural” and “clumsy”

- Examples: GPS, EPIC, SOAR, ACT

Cognitive Architectures are basically symbolic systems based on the concept of production systems.
Production System

A production system is defined by

- A working memory containing (symbolic) expressions
- A set of production rules (productions) of the form $LS \rightarrow RS$
  A rule can be applied to data in the working memory if the LS matches with these data.
  Application means execution of the RS.
- An interpreter which controls information processing with a fixed strategy. The interpreter works in recognize-act-cycles
  - Match: LS against data
  - Select one of the matching productions by some strategy
  - Apply RS of the selected rule

Interpreter

Working Memory

match

select

apply

Production Memory
Rule Selection Strategies

- Production Rules can be associated with strengths.
- Strengths can be modified: decay when not used, increment when successfully used.
- Selection of the rule with the highest strength value.
- Compare to rule selection in Prolog, in AI Planning.
History

- Post Tag Systems (Post, 1943)
- Formal grammers (Chomsky, 1957)
- General Purpose Production System as a theory of the control structure of human information processing (PSG, Newell, 1973) in which the General Problem Solver (GPS) is realised
- Cognitive architectures in cognitive modeling: Epic, ACT-R, Soar, Clarion ...
A Simple Example

Productions:

\[ R_1 \quad 1 \times x \quad \implies \quad x \]
\[ R_2 \quad x \times (y + z) \quad \implies \quad x \times y + x \times z \]
\[ R_3 \quad 2 \times x \quad \implies \quad x + x \]
\[ R_4 \quad x \times y \quad \implies \quad y \times x \]
A Simple Example

(0) Working Memory: $2 \times (1 + 1 \times 2)$

**Match:** Applicable Rules

- $R_1$ for $1 \times 2$
- $R_2$ for complete expression
- $R_3$ for complete expression
- $R_4$ for $1 \times 2$
- $R_4$ for complete expression

**Select:** $R_1$

**Apply:** results in $2 \times (1 + 2)$
A Simple Example

(1) Working Memory: $2 \times (1 + 2)$

**Match:** Applicable Rules

$R_2$ for complete expression

$R_3$ for complete expression

$R_4$ for complete expression

**Select:** $R_2$

**Apply:** results in $2 \times 1 + 2 \times 2$
A Simple Example

(2) Working Memory: $2 \times 1 + 2 \times 2$

**Match:** Applicable Rules

- $R_3$ for left part
- $R_3$ for right part
- $R_4$ for left part
- $R_4$ for right part

**Select:** $R_3$

**Apply:** results in $1 + 1 + 2 \times 2$
A Simple Example

(3) Working Memory: $1 + 1 + 2 \times 2$

**Match:** Applicable Rules
- $R_3$ for right part
- $R_4$ for right part

**Select:** $R_3$

**Apply:** results in

(4) Working Memory: $1 + 1 + 2 + 2$

**Match:** no rule

Halt
Comments

- Model of procedural knowledge needed to solve a problem, based on a fixed strategy of knowledge application
- If knowledge about $x \times 2 = x + x$ ($R_3$) would be missing (which could be the case for a grammar school kid), the problem could not be solved
- Would the fixed strategy prefer e.g. $R_4$ in step (2), the rule applied afterwards would have been $R_1$ resulting in $2 + 2 + 2$
- System would have needed a step more (pupil a longer solution time)
- A control strategy preferring always $R_4$ (no removing of multiplication) would result in that the system would find no solution (or a pupil)
- Intelligent Tutor Systems for elementary algebra use production system models to explain errors (e.g. Anderson’s Algebra Tutor, e.g. Kurt van Lehn’s work)
Learning and Production Systems

- Typically only changes of values of strength associated with productions
- In addition: composition of rules to larger units (utility problem!, i.e., swamming the memory with lots of composites)
- Very seldomly considered: construction of new rules from example experience
- Even more seldom: learning new control strategies

ALL these levels of learning are realised by humans!
ACT-R

Adaptive Control of Thought – Rational

- Main Predecessor: ACT
- Integration of HAM (human associative memory) as declarative memory
- Developed by John Anderson and his group at Carnegie-Mellon University
- Based on ideas of Newell (GPS)
- Implemented in Lisp
- Current version ACT-R 6, relates to neurological data such as those obtained from FMRI
- Captures aspects of learning and memory, problem solving and decision making, perception and attention
- Applications: Cognitive Tutors for Mathematics, which is used in many schools
- Lots of information, including tutorials at act-r.psy.cmu.edu
Basic Structure of ACT-R
ACT-R Components

- Memory modules:
  - **Declarative memory:** consisting of facts such as Washington, D.C. is the capital of United States, France is a country in Europe, or \(2 + 3 = 5\). *know what, accessible, verbalizable*
  - **Procedural memory:** made of productions. Productions represent knowledge about how we do things: for instance, knowledge about how to type the letter “Q” on a keyboard, about how to drive, or about how to perform addition. *know how, skills, control of cognition*

- Perceptual-motor modules: interface with the real world; most well-developed perceptual-motor modules in ACT-R are the visual and the manual modules.
ACT Components cont.

- Buffers: contents of the buffers at a given moment in time represents the state of ACT-R at that moment
  - Special buffer: goal buffer, holding current goals
- Pattern Matcher: searches for a production that matches the current state
- Execution: Match-Select-Apply; heuristics for selection
Declarative Units

Organized in chunks with slots

Action023:
  isa chase
  agent dog
  object cat

Fact3+4:
  isa addition-fact
  addend1 three
  addend2 four
  sum seven
Declarative Memory

- Chunks are interconnected, e.g. via isa, building a network
- Spreading Activation Network
- Mechanisms for activation of chunks, spreading activation are called subsymbolic
- Learning addresses mainly modifications of base activations

\[ A_i = B_i + \sum_j W_j S_{ji} \]

with

- \( A_i \): activation of chunk \( i \)
- \( B_i \): base level activation of chunk \( i \)
- \( W_j \): attentional weighting of chunks \( j \) which are part of the current goal
- \( S_{ji} \): strength of association between chunk \( j \) and chunk \( i \)
Spread of Activation (1)

- Study sentences, where people and locations occur in one, two, or three of the sentences (fan)

1. A hippie is in the park.
2. A hippie is in the church.
3. A hippie is in the bank.
4. A captain is in the park.
5. A captain is in the cave.
6. A debutante is in the bank.
7. A fireman is in the park.
8. A giant is in the beach.
9. A giant is in the dungeon.
10. A giant is in the castle.
11. A earl is in the castle.
12. A earl is in the forest.
13. A lawyer is in the store.
Spread of Activation (2)

- Judge whether the facts were seen before, mixed with new sentences (foils)
- When fan increases, the time to respond increases. Foil sentences take longer to respond to than the targets.

<table>
<thead>
<tr>
<th>Location</th>
<th>Person</th>
<th>Fan</th>
<th>Mean</th>
<th>Person</th>
<th>Fan</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Mean</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1.111</td>
<td>1.174</td>
<td>1.222</td>
<td>1.169</td>
<td>1.197</td>
<td>1.221</td>
</tr>
<tr>
<td>2</td>
<td>1.167</td>
<td>1.198</td>
<td>1.222</td>
<td>1.196</td>
<td>1.250</td>
<td>1.356</td>
</tr>
<tr>
<td>3</td>
<td>1.153</td>
<td>1.233</td>
<td>1.357</td>
<td>1.248</td>
<td>1.262</td>
<td>1.471</td>
</tr>
<tr>
<td>Mean</td>
<td>1.144</td>
<td>1.202</td>
<td>1.357</td>
<td>1.20</td>
<td>1.236</td>
<td>1.349</td>
</tr>
</tbody>
</table>
Goal buffer provides a context in which to perform a retrieval.

Chunks spread activation to the other chunks in declarative memory based on their relations to the other chunks which we call their strengths of association. This essentially results in increasing the activation of those chunks which are related to the current context.
Alternative Explanations?

- Spreading Activation Versus Compound Cue Accounts of Priming: Mediated Priming Revisited (Gail McKoon and Roger Ratcliff, 1992)
- Matching of Person – Location pair with the graph: more tests if source node has more edges
- ...
Goal-driven productions: LS contains goals or subgoals, RS can set new subgoals (compare to hierarchical planning)

LS specifies a pattern of chunks that must be present in the buffers for the production rule to apply

RS specifies some actions to take, basically modifications of chunks or requests for other chunks
Production Example

(p add-ones
   =goal>
     isa add-pair
     one1 =num1
     one2 =num2
     one-ans waiting
   =retrieval>
     isa addition-fact
     addend1 =num1
     addend2 =num2
     sum =sum

==>
   =goal>
     one-ans =sum
     carry waiting
   +retrieval>
     isa addition-fact
     addend1 ten
     sum =sum
)

English Description
If the goal is to add a pair of numbers and =num1 is the ones digit of the first
and =num2 is the ones digit of the second
and the ones digit of the answer is waiting
and the retrieved chunk is of type addition-fact stating
   =num1 plus
   =num2 equals
   =sum
Then change the goal so that
the ones answer is =sum
and note that a carry is being processed
and request a retrieval
of a chunk of type addition-fact
with addend1 being ten
and the sum being =sum
Tower of Hanoi in ACT (1)
(chunk-type disk size peg state)
(chunk-type peg name)
(chunk-type tower-task largest current goal)
(chunk-type move-disk disk to from other test at)
(chunk-type countfact first then)
(chunk-type encode-configuration size state)
(add-dm
   (disk1c isa disk size 1 peg c state current)
   (disk2c isa disk size 2 peg a state current)
   (disk3c isa disk size 3 peg b state current)
   (disk4c isa disk size 4 peg b state current)
   (disk1g isa disk size 1 peg b state goal)
   (disk2g isa disk size 2 peg a state goal)
   (disk3g isa disk size 3 peg c state goal)
   (disk4g isa disk size 4 peg c state goal)
   (fact01 isa countfact first 0 then 1)
   (fact12 isa countfact first 1 then 2)
   (fact23 isa countfact first 2 then 3)
   (fact34 isa countfact first 3 then 4)
   (fact45 isa countfact first 4 then 5)
   (a isa peg name a)
   (b isa peg name b)
   (c isa peg name c)
   (goal isa tower-task largest 4)
   (current isa chunk))
Tower of Hanoi in ACT (2)

(p start-tower
"IF goal is to solve tower task of size =size and =size greater than 1 THEN set a subgoal to move disk =size checking disk =new and change the goal to solve a tower task of size =new"

=goal> isa tower-task
    largest =size
    - largest 1
    current t
    goal t

=Fact> isa countfact
    first =new
    then =size

==> 

=goal> goal nil
    largest =new

=newgoal> isa move-disk
    disk =size
    test =new

!push! =newgoal)
Tower of Hanoi in ACT (3)

(p move
"IF the goal is move disk of size n to peg x and all smaller disks have been checked
THEN move disk n to peg x and pop the goal"
=goal>
  isa move-disk
  test 0
  from =from
  to =to
  disk =size
=disk>
  isa disk
  size =size
==> 
=disk>
  peg =to
!pop!)
}
Chunks given completely

Only two productions (of many) shown on the slides and only partially in ACT-R language

\texttt{countfact} necessary to model the knowledge that disc 1 is smaller than 2 etc.

Specific models exist in ACT-R for: memory for subgoals, learning with practice

Given the goal-directed rules, the knowledge represented in the rules together with the control strategy represents the same knowledge as the recursive function

But: Separation of declarative knowledge and non verbalizable productions, working memory restrictions

Assumption: If people can perform perfectly for ToH, then the rules might be verbalisable!
Tower of Hanoi in PDDL

(define (domain hanoi)
  (:predicates (on ?disk1 ?disk2)
    (smaller ?disk1 ?disk2) (clear ?disk))
  (:action MOVE :parameters (?disk ?source ?dest)
    :precondition (and (clear ?disk)
      (on ?disk ?source)
      (clear ?dest)
      (smaller ?disk ?dest))
    :effect (and (on ?disk ?dest)
      (not (on ?disk ?source))
      (not (clear ?dest))
      (clear ?source))))
(define (problem hanoi4)
    (:domain hanoi)
    (:objects D1 D2 D3 D4 P1 P2 P3)
    (:init (on D1 D2) (on D2 D3) (on D3 D4) (on D4 P1)
            (clear D1) (clear P2) (clear P3)
            (smaller D1 D2) (smaller D1 D3) (smaller D1 D4)
            (smaller D2 D3) (smaller D2 D4) (smaller D3 D4))
    (:goal (and (on D1 D2) (on D2 D3) (on D3 D4) (on D4 P3))))
Tower of Hanoi in Lisp

(defun towerofhanoi(n)
    ( move n 'A 'B 'C)
)

(defun display ( from to )
    (princ "Move Disk From Peg ")
    (princ from)
    (princ " To Peg")
    (princ to)
    (format t "~%")
    nil)
)

(defun move ( n from to via )
    (cond ((equal n 1) (display from to ))
        (t (move (- n 1) from via to )
            (display from to)
            (move (- n 1) via to from ))))
CLARION Architecture

- By Ron Sun, Rensselaer Polytechnic Institute since about 2000
- Quoted from website:
  - Aims to explore the interaction of implicit and explicit cognition, emphasizing bottom-up learning (i.e., learning that involves acquiring first implicit knowledge and then acquiring explicit knowledge on its basis)
  - Developing of artificial agents (cognitive AI) and understanding human learning and reasoning (theoretical psychology)
  - Examples: Artificial grammar learning, Categorical inference, Creative problem solving, social simulation

- see tutorial slides at
  http://www.cogsci.rpi.edu/~rsun/clarion.html
Subsystems

- ACS: action centered subsystem
- NACS: non-action centered subsystem
- MS: motivational subsystem
- MCS: meta-cognitive subsystem

Dual representation for each subsystem

- top-level explicit
- bottom-level implicit
- interaction for action and learning
The Subsystems

ACS
- Bottom: reactive actions, learned with reinforcement learning
- Top: explicit symbolic rules, different learning mechanisms

NACS
- Bottom: associative memory
- Top: Explicit representation of concepts

MS
- Bottom: drive states
- Top: Explicit representation of goals

MCS
- Regulates goal structures and cognitive processes per se
Differences to ACT-R

Quoted from Sun Tutorial

- ACT-R is not meant for autonomous learning, without a lot of a priori knowledge.
- CLARION is capable of automatic and effortless similarity based reasoning while ACT-R has to use cumbersome pairwise similarity relations.
- In ACT-R, there is no built-in modeling of motivational processes — goals are externally set and directly hand-coded.
- In ACT-R, there is no built-in metacognitive process.