SYMBOLOIC PROCESSING FOR INTELLIGENCE

Fourth Lecture
On unified theories of cognition
The William James Lectures
Harvard University

Allen Newell
Computer Science and Psychology
Carnegie-Mellon University

11 March 1987
REPRISE: HUMAN COGNITIVE ARCHITECTURE

What aspects are dictated by the nature of its world?

The real-time constraint on cognition

From neural technology, to get mind-like behavior

Only two small system levels available

The yield —

1. Neural, cognitive, rational, social timescales

2. Computational symbolic systems

3. Four levels of the cognitive band
   
   Architecture — symbolic access $\Theta 10$ ms

   Elementary deliberation (automatic) $\Theta 100$ ms

   Selection of prepared operators (controlled) $\Theta 1$ s

   Composed operators (full cognition) $\Theta 10$ s

4. Recognition-based architecture

5. Continual shift to recognition (learning)
PLAN OF THE LECTURE

Present a specific architecture for cognition — Soar

The basis for a unified theory of cognition

Focus is on functionality (for this lecture)

How Soar attains intelligent behavior

How the requirements dictate its structure

The architectural features derived in Lecture 3

Also the details of making it be intelligent

1. Architecture for central cognition

2. Learning from experience

3. The total cognitive system

4. Functionality and ability

5. Qualitative aspects of human cognition
MAJOR FEATURES OF CENTRAL ARCHITECTURE
Cognition, but not perception or motor behavior

1. Problem spaces to represent all tasks
   Little knowledge yields search, lots yields direct path
   Problem-solving architecture (no process substrate)

2. Productions for all long-term memory (symbols)
   Search control, operators, declarative knowledge

3. Attribute/value representation medium for all things

4. Preference-based procedure for all decisions
   Preference language: accept/reject, better/worse

5. Goals (and goal stack) to direct all behavior
   Goals are created by the system itself
   At performance time from impasses, not in plans
   Operators perform the function of deliberate goals

6. Chunking of all goal-results occurs continuously
PROBLEM SPACE ARCHITECTURE

Blocks World
Problem Space

Task
Implementation +
Search-control
Knowledge

Primitive functions

Select a problem space

Select a state from those directly available

Select an operator

Apply the operator to obtain new state

The deliberative acts of architecture
PRODUCTION SYSTEM

Familiar view — Collection of condition-action rules

Better — Content-addressed memory, recognition system

Soar production system (OPS5-like)

\[ C, C \rightarrow A \]

Conditions are patterns

\[ C, C, C \rightarrow A \]

\[ C, C \rightarrow A, A \]

Obtain all instantiations \[ [W, W, W, \ldots] \]

Actions only add elements to working memory

Elements leave when no longer accessible

No conflict resolution — Entirely parallel

Example Soar production

(sp propose-operator*comprehend

(goal <g> ↑problem-space <p> ↑state <s>))

(problem-space <p> ↑name base-level-space)

(state <s> ↑object <b> ↑input <i>)

(box <b> ↑on table ↑on-top nothing)

― (signal <i> ↑attention yes)

―

(operator <o> ↑name comprehend)

(preference <o> ↑role operator ↑value acceptable

↑goal <g> ↑problem-space <p> ↑state <s>))
Elaboration phase produces preferences

(S13 acceptable for supergoal state)

(S13 rejected for supergoal state)

(Q2 acceptable for operator)

(Q7 acceptable for operator)

(Q7 better than Q2 for operator)

(Q9 indifferent to Q6 for operator)
IMPASSES AND PROBLEM SPACES
ALL THE WAY DOWN

Long-term
Task-implementation and search-control knowledge
EXAMPLE OF OPERATION

Blocks-world space

Initial state

\[
\begin{array}{c}
A \\
BC
\end{array}
\quad \begin{array}{c}
A \rightarrow T \\
B \rightarrow C \\
A \rightarrow C \\
C \rightarrow A
\end{array}
\]

Desired state

\[
\begin{array}{c}
A \\
B \\
C
\end{array}
\quad \begin{array}{c}
B \\
C \\
A \rightarrow B
\end{array}
\]

tie

impasse

selection

space

[ ] \quad \begin{array}{c}
E(A \rightarrow C) \\
E(C \rightarrow A) \\
E(A \rightarrow T)
\end{array}
\quad \begin{array}{c}
[ - ] \\
E(C \rightarrow A) \\
[ -- ]
\end{array}

nochange

impasse

nochange

impasse

evaluation

space

\[
\begin{array}{c}
A \\
BC
\end{array}
\quad \begin{array}{c}
A \rightarrow C \\
B \rightarrow A
\end{array}
\]
CHUNKING — LEARNING FROM EXPERIENCE

Converts goal-based problem solving into productions

Actions — Based on the results of the subgoal

Conditions — Based on the pre-impasse situation

The aspects necessary to produce the actions

1. Chunks are productions — processes not data

2. A form of permanent goal-based caching

3. Chunks generalized implicitly

   Ignore whatever the problem solving ignored

4. Learning occurs during problem solving

5. Chunking applies to all subgoals

   Search control, operator implementation, ...

   Whenever knowledge is incomplete or inconsistent

6. Learns only what system experiences

7. General mechanism for moving up the P-D isobar
Chunk1:
If the problem-space is simple-blocks-world and the state is one proposition different than the goal and the state has block1 and block2 clear and block1 is on the table and the desired state has block1 on block2 then make a best preference for the operator that moves block1 onto block2.
CHUNKING IN THE BLOCKS WORLD
TOTAL COGNITIVE SYSTEM

Brief overview now, more later

Basic concern — To get interface right

Long term memory

Encoding productions \[\uparrow \downarrow\] Cognitive productions \[\uparrow \downarrow\] Decoding productions \[\uparrow \downarrow\]

[Goal stack]

Working memory

\[\uparrow \downarrow\] Perceptual Systems \[\uparrow \downarrow\] Motor Systems

---[Senses]---[Muscles]---

\[\uparrow \downarrow\] External environment

In terms of performance

\[[P \longrightarrow E] \longrightarrow C \longrightarrow [D \longrightarrow M]\]

In terms of structure and learning

\[[P] \longrightarrow [E \longrightarrow C \longrightarrow D] \longrightarrow [M]\]
R1-SOAR: CONFIGURATION TASK

R1 expert system (McDermott, 1980; DEC)

Input: An order for a Vax computer (a dozen items)

Processor, bus, primary memory, disks, graphics, ...

Output: Information to assemble the system (ten pages)

Filled out and verified order

Spatial layout in cabinets with all connections

Take into account many factors

Cost of components, power demands, cable lengths, ordering on bus, component compatibilities, ...

R1 characteristics (1984)

3300 Ops5 productions

10000 components (data base)

About 1000 production cycles for a typical task
R1-SOAR: PROBLEM SPACES
Second version of R1-Soar

Configure modules and backplaces

Configure a backplane(type)

order backplane

put backplane in box

put backplane in box section

consider next section

get backplane from order

put module in backplace

put board in slot

put modules in backplace

decrement power

next module

consider next slot
# PERFORMANCE AND LEARNING ON R1-SOAR

<table>
<thead>
<tr>
<th></th>
<th>No Learning</th>
<th>During Learning</th>
<th>After Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>1731</td>
<td>485</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>[232]</td>
<td>[+59]</td>
<td>[291]</td>
</tr>
<tr>
<td>Partial</td>
<td>243</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>[234]</td>
<td>[+14]</td>
<td>[248]</td>
</tr>
<tr>
<td>Full</td>
<td>150</td>
<td>90</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>[242]</td>
<td>[+12]</td>
<td>[278]</td>
</tr>
</tbody>
</table>

**Tasks**
- **Base**: No search-control knowledge
- **Partial**: Two key search-control rules
- **Full**: Search control equivalent to R1's

**Units**
- Decision cycles (e.g., select operator)
- [numbers of rules]
DESIGNER-SOAR: ALGORITHM DESIGN
Steier (1986)

Original system: Designer (Kant, Newell, Steier)

Designer-Soar is to complete and extend Designer

Target is design of convex hull

Major problem spaces:

Algorithm Design (top level)

States: data flow descriptions of algorithms

Operators: modify descriptions, focus attention

Developmental Evaluation

States: algorithm descriptions with data

Operators: execute descriptions on data

Application Domain

States: domain objects (sets, figures)

Operators: modify domain objects
DESIGNER-SOAR: SIMPLE EXAMPLE

Intersection
Given two sets, produce set of common elements

Memory to hold one input set

\[ \rightarrow \{M1\} \rightarrow \]

Generator to generate set elements

\[ \rightarrow \{M1\} \rightarrow [G1] \rightarrow \]

Test to check if element is in other set

\[ \rightarrow \{M1\} \rightarrow [G1] \rightarrow [T1] \rightarrow \]

Memory to hold second input set for test

\[ \rightarrow \{M1\} \rightarrow [G1] \rightarrow [T1] \rightarrow \]
\[ \rightarrow \{M2\} \]

Memory to build output set

\[ \rightarrow \{M1\} \rightarrow [G1] \rightarrow [T1] \rightarrow \{M3\} \rightarrow \]
\[ \rightarrow \{M2\} \]
Design-level of Cypress (D. Smith, 1986)

Algorithm design space (partial algorithms)

Logical-inference space (assertions) — Not incorporated

<table>
<thead>
<tr>
<th>Search control</th>
<th>Insertion sort</th>
<th>Merge sort</th>
<th>Quicksort full spec</th>
<th>Quicksort bad spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal</td>
<td>303</td>
<td>342</td>
<td>476</td>
<td>1132</td>
</tr>
<tr>
<td>Full</td>
<td>140</td>
<td>140</td>
<td>140</td>
<td>266</td>
</tr>
<tr>
<td>Minimal</td>
<td>222</td>
<td>236</td>
<td>226</td>
<td>238</td>
</tr>
<tr>
<td>Full</td>
<td>135</td>
<td>140</td>
<td>130</td>
<td>188</td>
</tr>
</tbody>
</table>

Across task transfer (minimal search control)

<table>
<thead>
<tr>
<th>Prior learning</th>
<th>Insertion-sort</th>
<th>Mergesort</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.7%</td>
<td>296.86%</td>
<td>421.88%</td>
<td></td>
</tr>
<tr>
<td>269.89%</td>
<td>20.6%</td>
<td>417.87%</td>
<td></td>
</tr>
<tr>
<td>273.90%</td>
<td>292.85%</td>
<td>20.4%</td>
<td></td>
</tr>
</tbody>
</table>
SUMMARY OF TASKS

Many small and modest tasks (21), many methods (19)
Eight puzzle, Tower of Hanoi, Waltz labeling
Dypar (NL parsing), Version spaces, Resolution TP
Generate & test, Hill climbing, Means-ends analysis
Constraint propagation

Larger tasks (some in progress)
R1-Soar: 3300 rule industrial expert system (25%)
Neomycin-Soar: Revision of Mycin
Designer-Soar: Algorithm discovery
Cypress-Soar: Divide-&-conquer algorithm designer
(Coder-Soar: Algorithms to code )
(Weaver-Soar: VLSI router)

Learning (chunking)
  Learns on all tasks it performs
    Learns search control, operators, spaces
    Improves with practice, transfers to other tasks
  Explanation-based generalization
  Outside guidance (by chunking)
  Abstraction planning (by chunking)
  Constraint compilation (by chunking)

Task acquisition
  Builds spaces from external specs (by chunking)
HOW DOES SOAR APPROXIMATE A KL SYSTEM

1. Computationally universal

   Necessary, but not deal with real time constraint

2. Production systems

   Real time via recognition

   Abandon fixed conflict resolution

3. Decision process

   Open to quiescence to get all that is available

3. Impasses

   Seek knowledge whenever it is not available

   Never rest on apriori fixed finite mechanism

   Errors are due to knowing the wrong thing

4. Chunking

   Continually convert slow processes to fast ones

Issues — Sharing knowledge, scope of chunking
MAPPING SOAR INTO HUMAN COGNITION

Productions \( \Theta 10 \) ms
Symbol system (access and retrieve)
Recognition system (content addressed)
Parallel operation
Involuntary
Unaware of individual firings
Duration: Match a function of complexity
(Should be simpler match than Ops5, possibly)

Decision cycle \( \Theta 100 \) ms
The smallest deliberate act
Accumulates knowledge for an act and decides
The smallest unit of serial operation
Involuntary (exhaustive)
Awareness attends decision (products, not process)
Duration: Longest production chain (to quiescence)

Primitive operators \( \Theta 1 \) s
Serial operation
Primitive observable thinking acts
Duration: Sequence of decision cycles (2 minimum)
Goal-oriented

Goal attainments \( \Theta 10 \) s
Smallest unit of goal attainment
Smallest non-primitive operators
Smallest unit of learning (chunking)
SOAR AND THE SHAPE OF HUMAN COGNITION #1

Does Soar have the right qualitative shape?

1. Has general features derived from real-time constraint

   Symbol system, automatic/controlled behavior,
   recognition-based, fast-read/slow-write,
   continual shift to recognition (learns from experience)

2. Behaves intelligently

   But is not completely rational (only approximates KL)

3. Goal oriented

   But not just because it has learned goals

   Goals arise out of its interaction with environment

4. Interrupt driven

   Depth-first local behavior, progressive deepening

5. Default behavior is fundamentally adaptive

   Does not have to be programmed to behave
6. Serial in midst of parallelism

   Autonomous behavior (hence an unconscious)

7. Recognition is strongly associative

   Does not have access to all that it knows

   Remembering can be a problem — can work at it

8. Not know how it does things

   Learned procedures are non-articulatable

   Chunking accesses WM trace, not productions

   Can work interpretively from declarative procedures

9. There is meta-awareness or reflection

   Can step back and examine what it is doing

10. Indefinitely large knowledge

11. Aware of large amounts of immediate detail

   But focused, with a penumbra
ISSUES AND LACUNA IN SOAR

1. Things not in Soar 4.4, but coming in Soar 5

   P-E-C-D-M

   Full development of perceptual mechanisms

   Full development of motor system

   Single state principle

   Less powerful match (no equality testing)

2. Default behavior is not quite all in the architecture

   Currently default productions avoid impasse pits

3. Not demonstrated yet — although consonant

   Flexibility to the point of non-brittleness

   Full scope of learning

4. Missing major aspects (Require structural additions?)

   Emotion, dreaming, imagery, ...
SUMMARY:
SYMBOLIC PROCESSING FOR INTELLIGENCE

A specific architecture for cognition — Soar

The central construct of a unified theory of cognition

Focus is on functionality — being intelligent

1. Architecture for central cognition

   Problem spaces, productions, goals

   Decision cycle, impasses

2. Learning from experience

   Chunking, at the production level

3. The total cognitive system (P-E-C-D-M)

   Encoding & decoding — Uncontrolled productions

4. Functionality and ability

   Incorporates most mechanisms of intelligence

5. Qualitative aspects of human cognition
SOAR DESIGN AND THE MULTIPLE CONSTRAINTS

1. Behave flexibly — Yes

2. Adaptive (rational, goal-oriented) behavior — Yes

3. Operate in real time — Yes

4. Rich, complex, detailed environment
   Perceptual detail — Interface only
   Use vast amounts of knowledge — Yes
   Motor control — Interface only

5. Symbols and abstractions — Yes

6. Language, both natural and artificial — No

7. Learn from environment and experience — Yes

8. Acquire capabilities through development — No

9. Live autonomously within a social community — No

10. Self awareness and a sense of self — No

11. Be realizable as a neural system — No

12. Arise through evolution — No
REFERENCES, LECTURE 4

On Soar

Copies of tech report in Harvard Psychology Library


General references for Lecture 4


SOAR AND ACTIVATION

1. Multiple roles of activation
   Determines access path (Quillian, many others)
   The representational media (connectionism)
   Determines processing rate (Anderson)

2. General concern about doing real tasks
   Activation systems still good only for analysis

3. Specific theoretical concern about learning
   Crypto-knowledge constraint
   Tried activation-based productions in Xaps2

4. Yielding interesting new forms of representation media
   Properties of continuity, coarse coding

5. The issue of approximation
   Activation cannot be critical as duration increases
   Soar as approximation to an activation-based system
SOAR AND SCHEMAS

1. Knowledge is organized — A basic truth

2. Schemas are a **data-structure** solution to this
   "Real schemas" — The kind we program
   They are rigid and unadaptive
   Large-grain-size argument is misplaced
   Because it confuses structure with behavior

3. Knowledge organization in Soar
   The declarative representation
   Attributes and values (as opposed to lists)
   No inheritance, defaults, attached procedures
   Productions provide dynamic, complex semantic net
   Inheritance comes in the elaboration phase
   Attached procedures realized by impasses

---

Key: Problem spaces are an action-oriented encoding
SOAR AND ACT*

Differences

<table>
<thead>
<tr>
<th>ACT*</th>
<th>Soar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory</strong></td>
<td><strong>Procedural</strong></td>
</tr>
<tr>
<td>Declarative procedural</td>
<td></td>
</tr>
<tr>
<td><strong>Higher organization</strong></td>
<td><strong>Problem spaces</strong></td>
</tr>
<tr>
<td>None</td>
<td></td>
</tr>
<tr>
<td><strong>Goals</strong></td>
<td><strong>Impasse created</strong></td>
</tr>
<tr>
<td>Deliberate learned</td>
<td></td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td><strong>All-or-none cycles</strong></td>
</tr>
<tr>
<td>Activation variable rate</td>
<td></td>
</tr>
<tr>
<td><strong>Learning</strong></td>
<td><strong>Chunking</strong></td>
</tr>
<tr>
<td>Compilation composition proceduralization</td>
<td></td>
</tr>
<tr>
<td>Tuning</td>
<td></td>
</tr>
<tr>
<td>strengthening</td>
<td></td>
</tr>
<tr>
<td>generalization</td>
<td></td>
</tr>
<tr>
<td>discrimination</td>
<td></td>
</tr>
</tbody>
</table>
USING KNOWLEDGE FOR CONTROL

Standard AI scheme: Methods + Selection

Method = Procedure + Deliberate-subgoals

Weak methods are basic to intelligent action

Generate-test, hill climbing, progressive deepening,
means-ends analysis, minimax, constraint propagation

Soar uses implicit methods for the weak methods

Implicit method = Conjoining independent heuristics

Major implications

Knowing leads directly to doing

No need to learn method control structure — Emerges

Permitted by two conditions

Soar — Problem spaces and production systems

Weak method — Search related and simple