ACT-R

- theoretical and practical basis of the Cognitive Tutors, building on:
  - Lisp Tutor
  - production system
    - rule-based programming language

- Overview:
  - A simple example
  - A more complex example (addition)

Components of a production rule system

- Working memory -- the database
- Production rule memory
- Interpreter that repeats the following cycle:
  1. Match
     - Match "if-parts" of productions with working memory
     - Collect all applicable production rules
  2. Conflict resolution
     - Select one of these productions to "fire"
  3. Act
     - "Fire" production by making changes to working memory that are indicated in "then-part"

An example production system

- You want a program that can answer questions and make inferences about food items
- Like:
  - What is purple and perishable?
  - What is packed in small containers and gives you a buzz?
  - What is green and weighs 15 lbs?

First cycle of execution

<table>
<thead>
<tr>
<th>WORKING MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM = (green, weighs-15-lbs)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CYCLE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Productions whose condition parts are true: P1</td>
</tr>
<tr>
<td>2. No production would add duplicate symbol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RULE MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1. IF green THEN produce</td>
</tr>
<tr>
<td>P2. IF packed-in-small-container THEN delicacy</td>
</tr>
<tr>
<td>P3. IF refrigerated OR produce THEN perishable</td>
</tr>
<tr>
<td>P4. IF weighs-15-lbs AND inexpensive AND NOT perishable</td>
</tr>
<tr>
<td>THEN staple</td>
</tr>
<tr>
<td>P5. IF perishable AND weighs-15-lbs THEN turkey</td>
</tr>
<tr>
<td>P6. IF weighs-15-lbs AND produce THEN watermelon</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INTERPRETER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Find all productions whose condition parts are true</td>
</tr>
<tr>
<td>2. Deactivate productions that would add a duplicate symbol</td>
</tr>
<tr>
<td>3. Execute the lowest numbered production (or quit)</td>
</tr>
<tr>
<td>4. Repeat</td>
</tr>
</tbody>
</table>

Do this yourself before reading on!

- Hand simulate the execution of the production rule model.
- For each cycle, write down the following information:
  - Activate rules:
  - Deactivate rules:
  - Execute rule:
  - WM = ( ... )
- What is in working memory when the production rule model finishes?
- Are there any mistakes in the production rules?
A second production rule model example

- Think about how you would write production rules to do multi-column addition?

\[ \begin{align*}
264 & + 716 \\
\end{align*} \]

- What if-then rules would you use to perform this task in a step-by-step fashion?

Production rule model for addition

**FOCUS-ON-FIRST-COLUMN**
- If the problem is a multi-column problem
- And there is no pending subgoal
- And there is the leftmost column of the problem
- Set a subgoal to process the leftmost column
- Add them as the result is column C
- Add the carry into column C (if any) has been added
- And then set a subgoal to write carry as the result
- Then set a subgoal to write carry as the result
- Next set a subgoal to write carry as the result
- And remove the goal to process column C

**FOCUS-ON-NEXT-COLUMN**
- Set a subgoal to process column C
- Set them as the sum of the addition in column C
- And a subgoal to write sum as the result
- And remove the goal to process column C

**ADD-CARRY**
- Set a subgoal to write sum as the result
- And there is a carry into column C
- And the carry has not been added
- Then change the goal to write sum + 1 as the result
- And mark the carry as added

**WRITE-CARRY**
- Then set a subgoal to write carry as the result
- And remove the goal

**WRITE-SUM**
- Then set a subgoal to write sum as the result
- And remove the goal

A Trace of Production Rule Firings

**Step 1**
1. **FOCUS-ON-FIRST-COLUMN**
   - Goal: Process column 1
   - C = column 1
   - Sum = 0
   - Action: Write 0 as result in column 1
   - Done

2. **ADD-CARRY**
   - Goal: Process column 2
   - C = column 2
   - Sum = 7
   - Action: Write 7 as result in column 2

3. **WRITE-SUM**
   - Goal: Process column 2
   - C = column 2
   - Sum = 7
   - Action: Write 7 as result in column 2

**Step 2**
1. **FOCUS-ON-NEXT-COLUMN**
   - Goal: Process column 3
   - C = column 3
   - Sum = 0
   - Action: Write 0 as result in column 3
   - Done

2. **ADD-CARRY**
   - Goal: Process column 3
   - C = column 3
   - Sum = 0
   - Action: Write 0 as result in column 3
   - Done

3. **WRITE-SUM**
   - Goal: Process column 3
   - C = column 3
   - Sum = 0
   - Action: Write 0 as result in column 3

4. **WRITE-CARRY**
   - Goal: Process column 3
   - C = column 3
   - Sum = 0
   - Action: Write 0 as result in column 3
   - Done

5. **WRITE-CARRY**
   - Goal: Process column 3
   - C = column 3
   - Sum = 0
   - Action: Write 0 as result in column 3
   - Done

A Trace of Production Rule Firings (ctnd.)

**Step 3**
1. **FOCUS-ON-NEXT-COLUMN**
   - Goal: Process column 4
   - C = column 4
   - Sum = 0
   - Action: Write 0 as result in column 4

2. **ADD-CARRY**
   - Goal: Process column 5
   - C = column 5
   - Sum = 0
   - Action: Write 0 as result in column 5

3. **WRITE-SUM**
   - Goal: Process column 5
   - C = column 5
   - Sum = 0
   - Action: Write 0 as result in column 5

4. **WRITE-CARRY**
   - Goal: Process column 5
   - C = column 5
   - Sum = 0
   - Action: Write 0 as result in column 5
   - Done

5. **WRITE-CARRY**
   - Goal: Process column 5
   - C = column 5
   - Sum = 0
   - Action: Write 0 as result in column 5
   - Done

**Step 4**
1. **FOCUS-ON-NEXT-COLUMN**
   - Goal: Process column 6
   - C = column 6
   - Sum = 0
   - Action: Write 0 as result in column 6

2. **ADD-CARRY**
   - Goal: Process column 6
   - C = column 6
   - Sum = 0
   - Action: Write 0 as result in column 6

3. **WRITE-SUM**
   - Goal: Process column 6
   - C = column 6
   - Sum = 0
   - Action: Write 0 as result in column 6

4. **WRITE-CARRY**
   - Goal: Process column 6
   - C = column 6
   - Sum = 0
   - Action: Write 0 as result in column 6
   - Done

5. **WRITE-CARRY**
   - Goal: Process column 6
   - C = column 6
   - Sum = 0
   - Action: Write 0 as result in column 6
   - Done

**Step 5**
1. **DONE**

How could you model students that don’t carry?

- Instead of doing the addition correctly:

\[ \begin{align*}
264 & + 716 \\
\end{align*} \]

- Can you model a student who writes:

\[ \begin{align*}
264 & + 716 \\
\end{align*} \]

- How can you change the production rule model?