Sep 5th, 2023

1. Ask about chatgpt
2. Feedback of act-r grammar

The command for creating a production in ACT-R is called **p**, and the general format for creating a production is:

(p Name "optional documentation string"

*buffer tests*

==>

*buffer changes and requests*

)

1. Flow chart of model translation

Aug. 18th Siyu Wu

Me: You are going to do a code translation task for cognitive architectures between Soar and ACT-R, To start with, may you tell me what you know about ACT-R?

ChatGPT:

Certainly! ACT-R, which stands for Adaptive Control of Thought - Rational, is a cognitive architecture that models human cognition and behavior. It was developed by John R. Anderson and his colleagues. ACT-R is designed to simulate a wide range of cognitive processes, including perception, memory, attention, problem-solving, and decision-making.

Here are some key features and concepts of ACT-R:

1. \*\*Production Systems:\*\* ACT-R is based on the idea of production systems, where cognitive processes are modeled using production rules. Production rules consist of conditions (matching patterns) and actions (what to do when the conditions are met). These rules are used to represent cognitive tasks and behaviors.

2. \*\*Declarative and Procedural Memory:\*\* ACT-R separates memory into declarative and procedural components. Declarative memory stores factual knowledge, while procedural memory stores production rules and sequences of actions.

3. \*\*Chunks:\*\* Chunks are the fundamental units of knowledge in ACT-R. They represent pieces of information and can be thought of as mental representations. Chunks contain slots (attributes) that hold values, and they can be used to model various cognitive elements, including concepts, perceptions, and goals.

4. \*\*Buffers:\*\* ACT-R uses buffers to represent the current state of cognitive processing. Buffers hold chunks of information and provide a way to interact with external environments or other cognitive modules.

5. \*\*Goals and Subgoals:\*\* Goals in ACT-R represent the cognitive tasks that the system is currently working on. These goals can be hierarchical and have subgoals. The system's behavior is driven by the goals it's pursuing.

6. \*\*Production Matching and Execution:\*\* ACT-R uses a pattern matching process to find applicable production rules based on the contents of buffers. When a rule's conditions match the contents of the buffers, the rule's actions are executed, which may involve modifying buffers, creating or updating chunks, and changing the focus of attention.

7. \*\*Learning:\*\* ACT-R includes mechanisms for learning. It can adjust production rule strengths based on the frequency of rule firing and the outcomes of actions.

8. \*\*Parameter Tuning:\*\* ACT-R models require parameter values to adjust the speed of various cognitive processes. These parameters can be tuned to match human performance in specific tasks.

ACT-R is widely used for simulating and explaining human performance in various cognitive tasks, from simple reaction time experiments to complex problem-solving scenarios. It has been applied in fields such as psychology, cognitive science, human-computer interaction, and education to gain insights into how humans think and behave in different situations.

M: What are the differences between declarative memories and chunks?

C: Declarative memory and chunks are closely related concepts within the ACT-R cognitive architecture, but they serve slightly different purposes and represent different aspects of cognitive processing. Here are the key differences between declarative memory and chunks:

\*\*Declarative Memory:\*\*

- \*\*Definition:\*\* Declarative memory is a memory system that stores factual knowledge, such as facts, concepts, and associations between items.

- \*\*Content:\*\* Declarative memory contains general knowledge that is not tied to specific cognitive tasks or actions.

- \*\*Representation:\*\* It stores information in a distributed network or semantic network form, where concepts are connected by links that represent relationships between them.

- \*\*Example:\*\* Knowing that Paris is the capital of France is stored in declarative memory.

- \*\*Usage:\*\* Declarative memory provides the background knowledge that the cognitive system can draw upon when performing tasks. It is not directly involved in production rule processing or action execution.

- \*\*Role:\*\* Declarative memory supplies the information that chunks and production rules can use to make decisions and generate behavior.

\*\*Chunks:\*\*

- \*\*Definition:\*\* Chunks are the fundamental units of knowledge in ACT-R. They represent pieces of information and serve as mental representations.

- \*\*Content:\*\* Chunks contain specific pieces of information, including perceptual features, goals, current task states, and items from declarative memory.

- \*\*Representation:\*\* Chunks consist of slots (attributes) that hold values. The values in the slots can be drawn from declarative memory or generated during cognitive processing.

- \*\*Example:\*\* A chunk might represent a perceived object, a goal state, or a step in a problem-solving task.

- \*\*Usage:\*\* Chunks play a central role in production rule processing. They are manipulated and compared in production rules' conditions and actions. Chunks also serve as the contents of different buffers that represent the cognitive system's current state.

- \*\*Role:\*\* Chunks provide a way to represent and manipulate the current cognitive state and allow for pattern matching and decision-making in the production system.

In summary, declarative memory stores general knowledge and concepts, whereas chunks are specific mental representations used for current cognitive processing. Declarative memory supplies the information that chunks use to represent current states, goals, and perceptions, and production rules use these chunks to generate behavior. Chunks are a more task-specific and dynamic representation compared to the more stable and general knowledge stored in declarative memory.

M: If you are going to creat a model named count. the model can count from 1 to 5, may you generate a the code with the decarative memories and the start production rule of intitilizing the start of the count?

C: Certainly! Here's an example of how you might create an ACT-R model named "count" that can count from 1 to 5. The model will use declarative memories to store the numbers and a production rule to initialize the count:

```lisp

; Define declarative memories for numbers 1 to 5

(chunk-type number

value

)

(chunk number-1

(isa number)

(value 1)

)

(chunk number-2

(isa number)

(value 2)

)

(chunk number-3

(isa number)

(value 3)

)

(chunk number-4

(isa number)

(value 4)

)

(chunk number-5

(isa number)

(value 5)

)

; Define the goal structure

(goal-focus start-counting)

; Initialize the count

(p start

=goal>

ISA start-counting

==>

=goal>

ISA counting

=count>

ISA count

value 1

)

```

Let's break down this code:

1. \*\*Declarative Memories:\*\* We define a chunk-type named "number" to represent numbers. Then, we create individual chunks for numbers 1 to 5 using the `number` chunk-type. Each chunk has an `isa` attribute set to "number" and a `value` attribute representing the respective number.

2. \*\*Goal Structure:\*\* We set the initial focus of the goal module to the goal "start-counting." This means the model will start with the goal of counting numbers.

3. \*\*Initialize Count Production Rule (`start`):\*\* This production rule fires when the goal "start-counting" is focused. It initializes the count to 1 by updating the goal to "counting" and creating a `count` chunk with a value of 1.

This example provides a simple structure for an ACT-R model that can count from 1 to 5. Please note that this is just a basic example, and you can expand upon it to include additional production rules for counting and transitioning between numbers.

Test the code in elisp mode:

#|Warning: chunk-type called with no current model. |#

; file: /Users/siyuwu/Desktop/count\_phase\_1

; in: CHUNK NUMBER-1

; (CHUNK NUMBER-1 (ISA NUMBER) (VALUE 1))

;

; caught STYLE-WARNING:

; undefined function: COMMON-LISP-USER::CHUNK

; (ISA NUMBER)

;

; caught STYLE-WARNING:

; undefined function: COMMON-LISP-USER::ISA

;

; caught WARNING:

; undefined variable: COMMON-LISP:NUMBER

; (CHUNK NUMBER-1 (ISA NUMBER) (VALUE 1))

;

; caught WARNING:

; undefined variable: COMMON-LISP-USER::NUMBER-1

; (VALUE 1)

; caught STYLE-WARNING:

; undefined function: COMMON-LISP-USER::VALUE

;

; compilation unit finished

; Undefined functions:

; CHUNK ISA VALUE

; Undefined variables:

; NUMBER NUMBER-1

; caught 2 WARNING conditions

; caught 3 STYLE-WARNING conditions

While evaluating the form starting at line 6, column 0

of #P"/Users/siyuwu/Desktop/count\_phase\_1":

The warning shows the model has mistakes with the chunks, didn’t define the model. first the command it use to define the chunk is wrong, should be define-chunk in stead of chunk, second it put chunk-type as a slot which is wrong. I debugged the first output with the attempt to minimal change the model.

I degugged the model and the model is modified as this

; Define declarative memories for numbers 1 to 5

(clear-all)

(define-model newcount

;; chunk-type

(chunk-type number order

value

)

;; declarative memories

(define-chunks

(count isa number order number-1 value 1)

(number-1)

(number-2)

(number-3)

(number-4)

(number-5)

(1)

(2)

(3)

(4)

(5)

)

; Define the goal structure

(goal-focus count)

; Initialize the count

(p start

=goal>

ISA number

order number-1

==>

=goal>

ISA number

order number-2

value 1

!bind! =x (chunk-slot-value count value)

!output! (=x)

)

)

Then I reput the model into the chat-gpt and ask it to generate the next production rule(s) to count to 5

M: I debugged the model, and this is the debugged the model based on the general framework you give to me. and based on this debugged model, may you generate the next production rule of increment count that the model can count up to 5? "; Define declarative memories for numbers 1 to 5

C:It looks like you've already made progress on the ACT-R model and have provided a "start" production rule to initialize the count. Now, you're looking to generate the next production rule to increment the count up to 5. Here's the additional production rule to achieve this:

```lisp

(p increment-count

=goal>

ISA number

order ?o

value ?v

?count <number

=order ?o

value ?v>

==>

=goal>

ISA number

order ^number-1

value ?next-v

=count>

^number order ?o

^value ?v

-^number ?count

)

```

This production rule, named "increment-count," operates based on the following logic:

1. It matches a goal with `ISA` "number," `order` attribute `?o`, and `value` attribute `?v`.

2. It also matches a chunk named `number` with `order` attribute matching `?o` and `value` attribute matching `?v`.

3. When the conditions are satisfied, it updates the goal's `order` attribute to "number-1," effectively incrementing the count order.

4. It also updates the `count` chunk's `number` and `value` attributes to match the previously matched chunk's attributes.

5. Finally, it removes the previously matched `number` chunk from the `count` buffer using the `-^number ?count` action.

This rule will allow the model to increment the count from 1 to 5. You can place this production rule in your ACT-R code after the "start" production rule and incorporate it into your model to complete the counting behavior.

As we can see, this outp ut is off the road….