Unit 8 Code Description

The example models for this unit either have no experiment code or are driven mostly by code that has been described in a previous unit, and the assignment task is similar to code from other unit tasks. Thus there isn't really much to discuss about those. There are a few new commands used which will be described, and there is a new mechanism used in this unit's Building Sticks model which allows it avoids using a !bind! to do a calculation which will be described.

Extending chunks from code

One of the new commands used in the assignment task code is **extend-possible-slots**, and is found in the create-example-memories function in Lisp and create_example_memories function in Python. Those functions are responsible for adding the chunks which represent the studied items to the model's declarative memory using slot names for the features from a globally defined list of slots. Since those slot names can be specified arbitrarily by the modeler, the code needs to make sure that the model will accept them as valid slot names before creating the chunks since they may not have been specified in any of the model's chunk-types. That is what the extend-possible-slots and extend_possible_slots functions do. They have one required parameter and one optional parameter. The required parameter is the name of a slot to add to those which can be used in any chunk without having to create a new chunk-type to provide the slot. The optional parameter indicates whether or not to print a warning if the slot which is provided has already been used to name a slot. If the optional parameter is specified as non-true (nil for Lisp or False/None for Python) then no warning is provided when a previously named slot is specified, otherwise it will print such warnings.

Calling commands from the model definition

You may also notice looking at the code for the assignment that the create-example-memories and create_example_memories functions are added as ACT-R commands. That is done so that they can be called in the model's definition to generate those initial chunks for the model every time that it is reset.

In the starting model file for the assignment you will find this line:

```
(call-act-r-command "create-example-memories")
```

The call-act-r-command command can be used in a model definition to call any command which has been added to ACT-R. It requires one parameter which is the string that names the command and any number of additional parameters can be provided which will be passed to that command when it is called. In this case the command requires no parameters and thus none are given.

Instead of using call-act-r-command, it is also possible when initially adding a command to specify a name which can be used directly in the model definition like the built-in ACT-R commands, but there are some additional complications with doing that and how to do so will not be described in the tutorial.

The Imaginal-action buffer

The imaginal module has a second buffer called **imaginal-action** which can be used by the modeler to make requests that perform custom actions. Those actions are typically used to modify the chunk in the **imaginal** buffer, replace the chunk in the **imaginal** buffer with a new one, or clear the **imaginal** buffer and report an error, but may perform any other arbitrary calculation desired. Those requests can also take time during which the imaginal module will be marked as busy. Note however, the **imaginal-action** buffer is not intended to be used for holding a chunk. The **imaginal** buffer is the cognitive interface for the imaginal module and the **imaginal-action** buffer exists for the purpose of allowing modelers to create new operations which can manipulate the **imaginal** buffer.

There are two types of requests which can be made to the **imaginal-action** buffer which are referred to as a generic action and a simple action. The model for the Building Sticks task in this unit uses a simple action with no extra information to create a new chunk for the **imaginal** buffer. The generic action is more powerful in terms of what it can do and for either the generic or simple action it is possible to provide additional details in the request. Those capabilities however require more care and programming from the modeler in handling the action and are beyond the scope of the tutorial. Users interested in using those capabilities should consult the reference manual for details.

Here is the production from the model which uses a simple action request to the **imaginal-action** buffer:

```
(p encode-line-current
    =goal>
      isa      try-strategy
      state    attending
    =imaginal>
      isa      encoding
      goal-loc =goal-loc
    =visual>
      isa      line
      width    =current-len
    ?visual>
      state    free
    ?imaginal-action>
      state    free
  ==>
    =imaginal>
      length     =current-len
    +imaginal-action>
      action     "bst-compute-difference"
      simple     t
    =goal>
      state      consider-next
    +visual>
      cmd        move-attention
      screen-pos =goal-loc)
```

For this task, the “bst-compute-difference” command in the code is creating a chunk which is a copy of the chunk currently in the **imaginal** buffer (which is done using the new command copy-chunk described below) and then modifying that new copy to have a slot named difference which holds the length difference between the current line and the goal line.

A simple action request to the **imaginal-action** buffer requires specifying a slot named action which must contain a string that names a valid command or a symbol naming a Lisp function, and a slot named simple with any value (note that **nil** is not a value since it indicates the absence of a value). When a simple action request is made to the **imaginal-action** buffer the imaginal module performs the following sequence of actions:

- the imaginal module is marked as busy
- if the imaginal module is currently signaling an error that error is cleared
- the named command is called with no parameters
- the **imaginal** buffer is cleared

Then after the current imaginal action time has passed (default of 200ms and set with the :imaginal-delay parameter) the following actions will happen:

- the imaginal module will be marked as free
- if the call to the action command returned the name of a chunk then that chunk will be copied into the **imaginal** buffer
- If the call to the action command returned any other value the **imaginal** buffer will remain empty, the imaginal module's error state will become true, and the **imaginal** buffer's failure query will be true.

Here is the segment from a trace showing the actions related to the simple-action request when the encode-line-current production fires:

```
     2.148   PROCEDURAL             PRODUCTION-FIRED ENCODE-LINE-CURRENT
...
     2.148   PROCEDURAL             MODULE-REQUEST IMAGINAL-ACTION
...
     2.148   PROCEDURAL             CLEAR-BUFFER IMAGINAL-ACTION
...
     2.148   IMAGINAL               CLEAR-BUFFER IMAGINAL
...
     2.348   IMAGINAL               SET-BUFFER-CHUNK IMAGINAL IMAGINAL-CHUNK0-0
```

Except for the additional clearing of the **imaginal-action** buffer, which should not hold a chunk anyway, it performs the same actions as an **imaginal** buffer request to create a new chunk would.

In the conditions of the encode-line-current production a query is made to test that the **imaginal-action** buffer has state free. That query will return the same state as the **imaginal** buffer does. Both buffers pass their requests to the same module which can only perform one action at a time regardless of which of its buffers was used to make the request. Thus it does not matter which buffer is used to test the state for the performance of the model, but to avoid a style warning testing the buffer for which a request is being made is preferable.

One important thing to note about a simple action request is that it will always clear the **imaginal** buffer. That means that the chunk currently in the buffer will become an element of the model's declarative memory at that time. In this model that does not matter because it is not retrieving those chunks later. However, in models where later retrieval is important, having intermediate chunks added to memory like that could cause problems. In those cases, one would probably

want to use the generic action request to extend the imaginal capabilities because it does not clear the buffer automatically and can be used to modify the chunk it contains.

Copy-chunk

The last new command used in this unit is copy-chunk (using its functional form in Lisp of copy-chunk-fct). The copy-chunk command takes one parameter which is the name of a chunk. It creates a new chunk which has all of the same slots and values as the chunk provided, and returns the name of that new chunk. This command is not often needed because buffers will automatically copy chunks and there are ways to set buffers without creating a chunk in advance. In this case, because the model is using a simple imaginal-action request, the command called to handle that action needs to create a new chunk to return and since that chunk needs all of the same information as the chunk that is in the **imaginal** buffer it uses copy-chunk to do so.