Unit8 Model Code Description

The example models for this unit either have no experiment code, or are driven by code that has been described in previous units. The assignment task's experiment code only uses one new ACT-R command, but otherwise is similar to other unit's tasks. Therefore only that one new command will be described here without describing all of the experiment code. In addition to that, this text is going to explain how the bst-learn-ppm model avoids using a !bind! to do a calculation and instead uses a request to the imaginal module to do so.

Extending chunks from code

The new command used in the assignment task code is extend-possible-slots which is used in the create-example-memories function:

```
(defun create-example-memories ()
  (dolist (x *slots*)
    (extend-possible-slots x nil)
    (define-chunks-fct `((,x isa chunk))))
  (dolist (x *cat1*)
    (add-dm-fct `((isa example category 1 ,@(mapcan 'list *slots* x)))))
  (dolist (x *cat2*)
    (add-dm-fct `((isa example category 2 ,@(mapcan 'list *slots* x))))))
```

The create-example-memories function is responsible for creating the initial chunks in the model's declarative memory using slot names which are specified in the list *slots*. Since those slot names were not declared in the chunk-types for the model the extend-possible-slots command is used to tell ACT-R that we are dynamically adding new slot names to the model. This is the same thing that happens when we make a request in a production to modify a chunk with a slot name that doesn't yet exist. If we did not do this then we would get a warning about using slot names that are undefined when we try to create the example chunks.

The extend-possible-slots command has one required parameter and one optional parameter. The required parameter is a symbol which names a slot to add to those which can be used in chunks. 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 nil then no warning is provided when a previously named slot is specified otherwise it will print such warnings.

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. This model 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     compute-difference
      simple     t
    =goal>
      state      consider-next
    +visual>
      cmd        move-attention
      screen-pos =goal-loc)
```

A simple action request to the **imaginal-action** buffer requires specifying a slot named action which must name a Lisp function, and a slot named simple with any true value. When a simple action request is made the imaginal module performs the following actions:

- the imaginal module is marked as busy
- if the imaginal module is currently signaling an error that is cleared
- the named function 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 things will happen:

- the imaginal module will be marked as free
- if the call to the action function returned the name of a chunk then that chunk will be placed into the **imaginal** buffer
- If the call to the action function 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 compute-difference function which is called as a result of the request in the encode-line-current production:

```
(defun compute-difference ()
  (let* ((chunk (buffer-read 'imaginal))
         (new-chunk (copy-chunk-fct chunk)))
    (mod-chunk-fct new-chunk
                   (list 'difference
                         (abs (- (chunk-slot-value-fct chunk 'length)
                                 (chunk-slot-value-fct chunk 'goal-length)))))))
```

It creates a copy of the chunk which is currently in the imaginal buffer using the ACT-R copy-chunk command and then sets the difference slot of that new chunk to be the difference between the length of the current stick (as set in the length slot of the chunk from the **imaginal** buffer) and the length of the goal stick (as set in the goal-length slot of the chunk from the **imaginal** buffer) just as the !bind! did in the previous version of the model. That new chunk is returned from the function and thus will be put into the **imaginal** buffer after 200ms have passed.

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

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

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.