Unit 8 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 uses two new ACT-R commands (one in the experiment file and one in the model definition), but otherwise is similar to code from other unit's tasks. Therefore only those new commands will be described here without describing all of the experiment code. In addition to that, this text is going to explain how the new Building Sticks model avoids using a !bind! to do a calculation and instead uses a request to the imaginal module to do so, which relies on code in the updated bst experiment files for this unit.

Extending chunks from code

The one new command used in the assignment task code is found in the create-example-memories function in Lisp and create_example_memories in Python:

```
(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))))))


def create_example_memories():

    for s in slots:
        actr.extend_possible_slots(s,False)
        actr.define_chunks([s,"isa","chunk"])

    for c in cat1:
        chunk = ["isa","example","category",1]
        for slot,value in list(zip(slots,c)):
            chunk.append(slot)
            chunk.append(value)
        actr.add_dm(chunk)

    for c in cat2:
        chunk = ["isa","example","category",2]
        for slot,value in list(zip(slots,c)):
            chunk.append(slot)
            chunk.append(value)
        actr.add_dm(chunk)
```

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 chunks (using a symbol in Lisp and a string in Python). 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.

You may also notice looking at the code that those functions are added as ACT-R commands:

```
(add-act-r-command "create-example-memories" 'create-example-memories
  "Categorize task function to add the initial example chunks to simulate the training process.")

actr.add_command("create-example-memories",create_example_memories,
  "Categorize task function to add the initial example chunks to simulate the training process.")
```

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.

Calling commands from the model definition

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)
```

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 true value. At the bottom of the Building Stick experiment code files for this unit we find the function which implements the "bst-compute-difference" command and it being added:

```
(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)))))))))

(add-act-r-command "bst-compute-difference" 'compute-difference
                   "Imaginal action function to compute the difference between sticks.")


def compute_difference():
    c = actr.buffer_read('imaginal')
    n = actr.copy_chunk(c)
    actr.mod_chunk(n,'difference',
                   abs(actr.chunk_slot_value(c,'length') - actr.chunk_slot_value(c,'goal-length')))
    return n

actr.add_command("bst-compute-difference",compute_difference,
                 "Imaginal action function to compute the difference between sticks.")
```

Those functions create a copy of the chunk in the **imaginal** buffer and then modify that copy to contain a slot named difference which holds the difference between the values in the length and goal-length slots of that chunk.

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 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 placed 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:

```
     2.191   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.