Unit 7 Code Description

The paired associate model for this unit uses the same experiment code as the paired associate model from unit 4. The past-tense task is another example of a task and model that do not use the perceptual and motor components of ACT-R. It is not that different from previous instances of such tasks, and other than some more new ACT-R commands to describe there is not much to discuss with this unit's code.

Reinforcing Declarative Knowledge

One thing the code for the past-tense task does is provide the model with examples for declarative memory. To do so, it performs the same operations that would happen if the model were doing it through productions. It places a chunk into the **imaginal** buffer with the information in its slots, and then it clears the buffer so that it can be merged into declarative memory. That is not the only way that process can be handled without running the model, but it is an easy approach if the model is not currently using the buffer involved. Alternatively, one could adjust the base-level parameters to update the activations or use one of the commands for merging that are provided by the declarative memory module (see the reference manual for more information).

New Commands

set-buffer-chunk and **set_buffer_chunk** can be used to set a buffer to hold a copy of a particular chunk. It takes two parameters. The first must be the name of a buffer. The second can be the name of a chunk or a list which would be sufficient for creating a chunk (as would be provided to add-dm or define-chunks). If a chunk name is provided, it copies that chunk into the specified buffer, and if a chunk description is provided it copies that information into a chunk which is placed into the buffer. In either case, it clears any chunk which is currently in the buffer first. It returns the name of the chunk which is now in the buffer.

This acts very much like the goal-focus command, except that it works for any buffer. An important difference however is that the goal-focus command actually schedules an event which uses set-buffer-chunk to put the chunk in the **goal** buffer. That is important because scheduled events display in the model trace and are changes to which the model can react. There is another command which works more like goal-focus that is often more appropriate called schedule-set-buffer-chunk. That was not necessary for this task, but schedule-set-buffer-chunk will be used in the next unit.

clear-buffer and **clear_buffer** can be used to clear a chunk from a buffer the same way a *–buffer>* action in a production will. It takes one parameter which should be the name of a buffer and that buffer is emptied and the chunk is merged into declarative memory. It returns the name of the chunk that was cleared. Like set-buffer-chunk, the clear-buffer command does not create an event to which the model can respond, but there is a corresponding schedule-clear-buffer command which could be used to do that.

chunk-slot-value and **chunk_slot_value** can be used to get the current value of a slot in a chunk. It takes two parameters which are the name of a chunk and the name of a slot and it returns the current value of that slot. It is very similar to buffer-slot-value which was used in a previous unit when working with chunks in the buffers.

chunk-p and **chunk_p** can be used to determine if the provided name is actually the name of a chunk in the current model. It returns a true result if it is (t/True) and a false value if not (nil/None).

push-last is a macro for Lisp which adds an item to the end of a list. It takes two parameters, any item and a list. It destructively adds the item to the end of the list. This is often more convenient than the standard Lisp function push which adds to the beginning of a list.

trigger-reward and **trigger_reward** were not actually used in the code, but it was monitored to determine if the model received a reward. It was mentioned in the last unit, and it is the command which provides the rewards to a model for utility learning. It is available for a modeler to use to provide a reward value to the model at any time. It requires one parameter, and it schedules an event to provide that value as a reward at the current time. If that reward value is numeric then it is used to update the utilities of those productions which have been selected since the last reward. If it is not a numeric value then it only serves to mark the last reward time for preventing further propagation of the next reward signal.