Cognitive Modeling (2009–2010)

School of Informatics, University of Edinburgh
Lecturer: Sharon Goldwater

Assignment 0: Solutions

Question 1
What is the maximum number of words that the `STS` buffer can store? Why does this limit make sense? How is it implemented in Cogent?

Solution: `STS` can store a maximum of seven words. Seven chunks is also approximately the capacity of human short term memory. This is implemented in Cogent by setting `properties/capacity` to `7`.

Question 2
Does the `LTS` buffer also have a limited capacity? If it doesn't, then why don't all words in `STS` simply end up in `LTS`? Describe the role of the `Rehearsal` process in achieving this behavior.

Solution: `LTS` doesn't have a limited capacity; however, not all words in `STS` end up in `LTS` because only one word is transferred from `STS` to `LTS` in each cycle by the `Rehearsal` process. As the capacity of `STS` is limited, a word might have already expired from `STS` by the time `Rehearsal` gets round to transferring it to `LTS`.

Question 3
Select the `Current Graph` tab of the `Output Positions` element. describe the graph you obtain (or provide a screen shot). Why does the graph look this way? How does it change if you run the model 10 times?

Solution: The graph looks very uneven; it oscillates between perfect (100%) and no (0%) recall. The graph is like this because it only graphs the data of one trial; this means that a word is either recalled (100%) or not (0%); only one word has been presented for each position. With each trial, the graph becomes smoother, it approximates a U-shape. This is the effect of averaging; the graph now displays the average percentage (over 10 trials) with which a word at a certain position has been recalled. The graph should show the two main effects of serial position on recall: the recency effect and the primacy effect.

Question 4
If you run the model multiple times (as in the last question), you will realize that its behavior varies randomly from one run to the other, i.e., it doesn't always recall the same words. Explain which parts of the model are responsible for the randomness.

Solution: There are three sources of randomness in the model: items decay randomly for the `STS` buffer if it reaches its capacity limit; items are picked randomly for the `Rehearsal` process; and items are recalled in a random order.

Question 5
Set `decay` to `half-life` and the `decay constant` to 20. Reset the model by pressing the '`o`' button. Now run the model to completion 20 times. How does the recall graph look now? Explain why. Repeat this for the other two types of `decay`, i.e., `linear` and `fixed`. Does this change the graph?

Solution: With the `decay` set to `half-life` the model produces a graph with a much less pronounced primacy effect. The primacy effect is generated by LTS: items that are presented early in the experiment (especially before STS is full) are more likely to be transferred from STS to LTS (only one item per cycle can be transferred by Rehearsal). They are then available for recall later. If there is decay in LTS, some of the items will already have decayed by the time they are recalled, thus the primacy effect becomes weaker.

If decay is set to linear or fixed, the primacy effect disappears completely. This is because with linear or fixed decay, earlier items decay faster than later ones, while with half-life decay, all items have the same probability of decaying.

Question 6
What happens if you uncheck the once box in the rule's properties so that the rule is just refracted? Describe any differences you see in the output graph and the messages being sent from I/O Process to Collate Responses.

Solution: Unticking the once box means that the recall rule can fire multiple times on a single cycle (in parallel). All words in STS and LTS will be recalled simultaneously, which can be seen in the messages from I/O Process to Collate Responses: all messages are sent on the same cycle, instead of being sent one at a time on separate cycles. The difference in the graph is subtle, but recall should be slightly better when in parallel than in serial, because during serial recall, items have more time to decay from LTS while other items are being recalled.

Question 7
What does it mean for a rule to be refracted? Set the properties of the Recall rule to unrefracted. What happens when you run the model now, and why? (To stop a simulation, click on the red square button.) Finally, modify the rule by removing the recursive call: click on the button to the left of the send recall to I/O Process clause, and choose Delete this action. Recall is now in parallel. Examine the messages sent from I/O Process to Collate Responses. Now make the rule refracted and examine the messages again. What is the difference, and why? Which version do you think is more cognitively plausible?

Solution: A refracted rule only fires once with every instantiation of a predicate in the if-clause of the rule. An unrefracted rule can fire several times for a given instantiation. If you make the Recall rule unrefracted, then when it recurses, there is nothing to prevent it from recalling the same item again and recursing again. This results in an infinite loop with every item being recalled an infinite number of times. When the recursive call is removed, the refracted version recalls each word that is in STS or LTS exactly once, whereas the unrefracted version may recall a word more than once. This is because words may be in both STS and LTS, and if they are, then recall(Word) will be true for both instances and the unrefracted version will recall both of them. This is probably less plausible than recalling each word once, since the subject would typically not repeat recalled words (although one could interpret this as bringing a word to mind several times while trying to remember other words, which is arguably more plausible than only recalling each word exactly once).

Question 8
Give the box-and-arrow diagram of the new model (subject part only) and list the rules and conditions for your new version of Rehearsal. What is the recall graph you get now if you run the model 10 times?

Solution: The box-and-arrow diagram is given in Figure 1, the list of rules and condition in Figure 2. Crucially, there is no read access to the buffer Forgotten in this model (as stipulated in the question); solutions in which the presence of an item in Forgotten is checked before adding it to LTS (in order to prevent duplication in both places) require read access to Forgotten. Note that the deletion of an item also counts as read access: it can't be plausibility assumed that forgotten

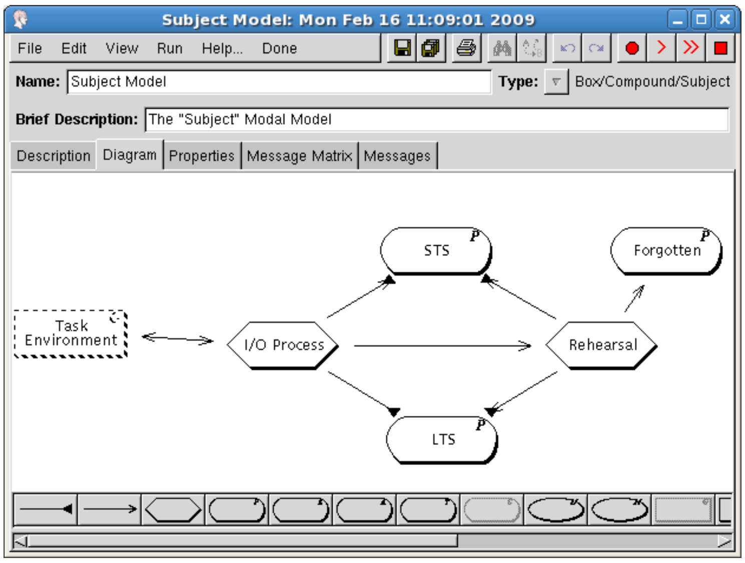
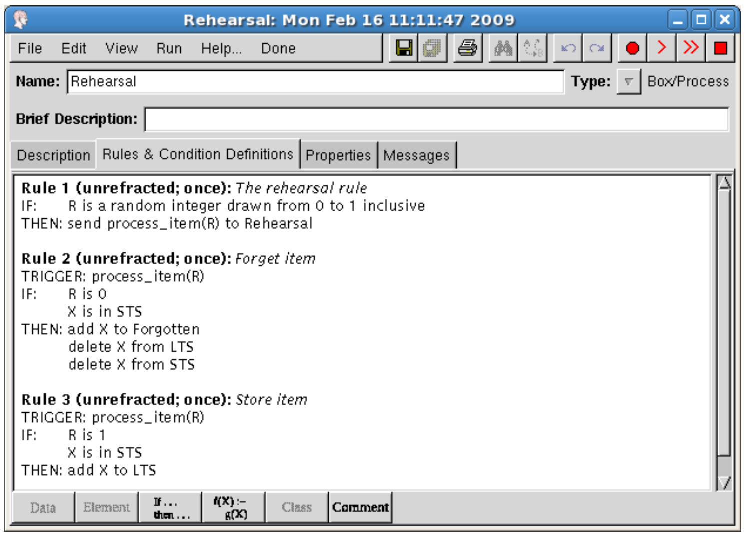

Subject Model: Mon Feb 16 11:09:01 2009
File Edit View Run Help... Done
Name: Subject Model
Type: Box/Compound/Subject
Brief Description: The "Subject" Modal Model
Description Diagram Properties Message Matrix Messages
STS
Forgotten
Task Environment
I/O Process
Rehearsal
LTS

Figure 1: Box-and-arrow diagram for the Modal Model with forgetting.


Rehearsal: Mon Feb 16 11:11:47 2009
File Edit View Run Help... Done
Description Rules & Condition Definitions Properties Messages
Data Element If... then... f(X):- g(X) Class Comment

Name: Rehearsal

Type: Box/Process

Brief Description:

```
Rule 1 (unrefracted; once): The rehearsal rule
IF:     R is a random integer drawn from 0 to 1 inclusive
THEN: send process_item(R) to Rehearsal

Rule 2 (unrefracted; once): Forget item
TRIGGER: process_item(R)
IF:     R is 0
        X is in STS
THEN: add X to Forgotten
        delete X from LTS
        delete X from STS

Rule 3 (unrefracted; once): Store item
TRIGGER: process_item(R)
IF:     R is 1
        X is in STS
THEN: add X to LTS
```

Figure 2: Rule set of the `Rehearsal` process of the Modal Model with forgetting.

items can be deleted later on. The other important aspect is that items that are added to `Forgotten` are simultaneously deleted from `LTS` and `STS`. This is the only way of preventing an item from ending up both in `Forgotten` and `LTS` (as stipulated in the question). If an item is deleted only from `LTS` when it is added to `Forgotten`, then it could later be rehearsed again and end up in `LTS` as well.

The recall graph for this model exhibits worse performance than the original model except at the very end of the recall graph, and little or no primacy effect. This is similar to adding a large decay to the `LTS` buffer of the original model.

A similar solution that almost works is to choose the item to be processed in Rule 1 and pass it as an additional argument to Rules 2 and 3. However, note that Rule 2 (which deletes items from `STS` that are to be forgotten) only fires on the cycle after Rule 1 (which chooses items from `STS` to rehearse). This means that an item could be slated for deletion on one cycle, then rehearsed again on the following cycle before being deleted, and end up in `LTS` because of the second rehearsal.

Alternative solutions are possible, e.g., involving an additional buffer that keeps track of the random number associated with each word so that later rehearsals always send the word to the same buffer as before. However this solution is not as clean: it doesn't technically have access to words in the `Forgotten` buffer, but it does need to store and access the fact that they have been forgotten.

Question 9
Discuss the cognitive plausibility of the new model that you have developed. Sketch an alternative approach to modeling hippocampal damage that doesn't require a `Forgotten` buffer.

Solution: This depends on the solution for Question 7. In the solution given here, there are at least two flaws. First, one would not want to assume that items can be deleted from `LTS` or `STS` by the `Rehearsal` process, given that the brain damage should only affect the process of transferring information from short term to long term memory. And of course there is no evidence for the existence of a structure in the human brain that corresponds to the `Forgotten` buffer.

One alternative to the above model would be to simply remove the `Forgotten` buffer, and not bother to delete items from `STS` and `LTS` if R is 0. In other words, for each item, choose at random between two possibilities: transfer the item to `LTS`, or do nothing. This would simply slow down the rate of rehearsal and result in fewer items in `LTS`.