Tutorial 1
Introduction to COGENT

1 What COGENT is about

In the first part of the tutorials, we are going to use COGENT, a visual programming environment designed specifically for implementing symbolic models of human cognitive skills, such as for example parsing. Models consist of a combination of buffers and processes, linked by arrows indicating the flow of information. Buffers contain factual information and represent memory, while processes manipulate and transform information according to rules the modeller defines.

2 COGENT Setup and References

See tutorial web page and the slides at:
`http://www.coli.uni-saarland.de/~gparis/Cours/CP_Tutorials/`

3 A Model of Self-Paced Reading

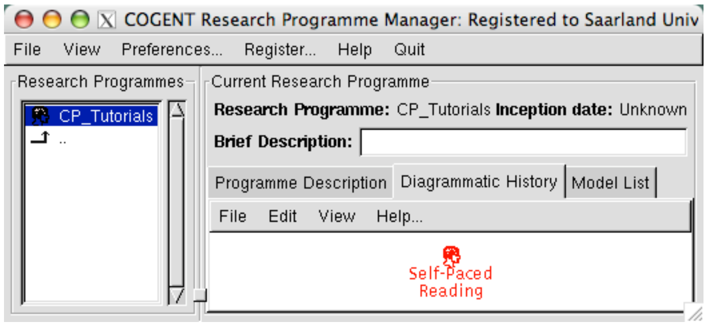
In these tutorials, we are going to focus on parsing written sentences. Our models will correspond to what we think happens inside a reader's mind as they read and analyze individual sentences.

In order to explore the COGENT environment, learn about some of its features and about the syntax it uses, today we are going to build a simple model of a self-paced reading experiment. Imagine you are running a self-paced reading experiment to investigate human parsing: You have a **list of stimuli** which is presented word-by-word on a **computer display**. The **participants read** the sentences, and **press the space bar** each time they want the next word to appear. Each time they do so, the **time** they take from one word to the next is **recorded to a file**, so that it can later be used for analysis.

3.1 Creating the Model and Adding Components

1. Let us start building our first model. Create a new research program in COGENT, e.g. *"`CP_Tutorials`" (File → New → Project)*. This will contain all the models we will develop over the course of term.

2. In the right window panel, in the *Diagrammatic History* tab of your research program, create your first model (Right-click on the canvas, *File → New Model)*. Give the model a name, e.g. *`“Self-Paced Reading”`*.

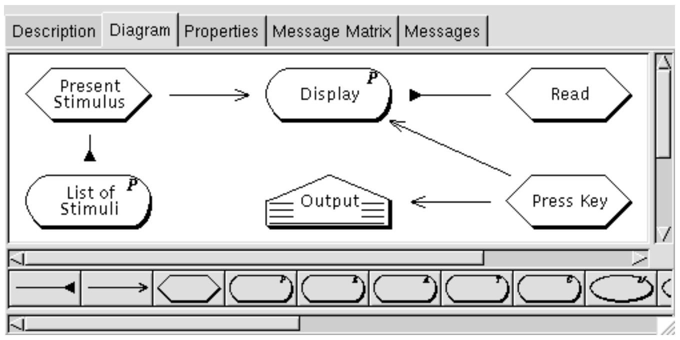

COGENT Research Programme Manager: Registered to Saarland Univ
File View Preferences... Register... Help Quit
Research Programmes
CP_Tutorials
Current Research Programme
Research Programme: CP_Tutorials Inception date: Unknown
Brief Description:
Programme Description Diagrammatic History Model List
File Edit View Help...
Self-Paced Reading

3. If your model isn't highlighted in red, click on it once to select it.

4. Then open it by double-clicking on the head symbol, and switch to the *Diagram* tab. This is where the components of a COGENT model are specified, using drag & drop. The main four types of components we will use are:
   - **Propositional buffers** (rectangles with rounded ends and a *“P”* in the upper right corner): To store any kind of factual information.
   - **Processes** (stretched hexagons): To manipulate and transform information according to rules.
   - **Text sinks** (rectangles with a pointed top and horizontal lines on the sides): To save data to files.
   - **Compound boxes** (plain rectangles): Boxes that contain other components, grouping them into functional units.

5. According to the above description of a self-paced reading experiment, the components we need are:
   - A propositional buffer to store the list of stimuli,
   - Another to represent the display,
   - A process in-between them to present the stimuli one-by-one on the display,
   - Another process to represent the participant reading the input from the display,
   - A process to represent key pressing, which will trigger clearing of the display,
   - And an output sink to store the reading times.

6. Create the components in your model:
   a) Use the scroll bar at the bottom to find each type of component,
   b) Click on the component,
   c) Then click on the canvas where you want the box to appear, positioning the elements as in the screenshot below.



7. Open each component with a double-click and rename them *List of Stimuli*, *Present Stimulus*, *Display*, *Read*, *Press Key*, and *Output*.

8. Then add the arrows. There are two kinds of arrows in COGENT, which determine what information can flow from one component to another:

   - Blunt arrows with a triangle at the end are **"read" arrows** and indicate that a process can read from a given component,
   - Whereas **"write" arrows** with an arrowhead at the end indicate what components a process can modify (add to or delete from).

   Here is how to draw arrows in COGENT:

   a) Click on the arrow in the components palette,

   b) Then click on the source box,

   c) Drag to the destination box,

   d) And release.

   Note that you can delete arrows as well change their direction and type by right-clicking on them and choosing from the context menu that opens.

3.2 Entering the Stimuli

You are now going to add a few sentences to use as stimuli. The syntax used in COGENT is inspired from the programming language Prolog:

- Sentences are represented by Prolog lists: They are enclosed between square brackets and the words (the elements of the list) are separated by commas.

- Capital letters denote variables, so in *"John loves Mary"* the words `"john"` and `"mary"` should NOT be CAPITALIZED.

1. Add the sentence `[john, loves, mary]` to *List of Stimuli*.

   a) Open the buffer,

   b) Click on the *Initial Contents* tab,

   c) Add an empty element by clicking on the *Element* component, then on the canvas,

   d) Open it with a double-click and modify it.

2. Then add a few more simple sentences of your own.

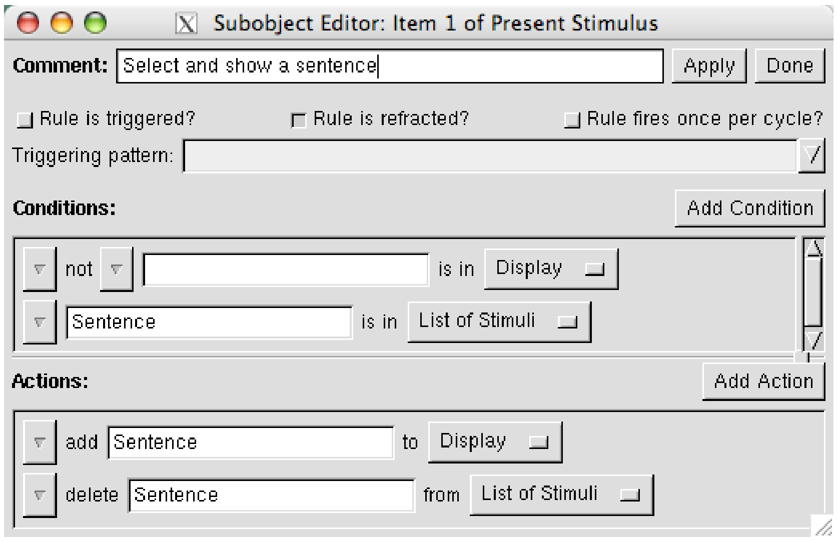
3.3 Setting up the Process *Present Stimulus*

We are now going to set up the necessary rules in *Present Stimulus* to diplay the stimuli one-by-one. In our first version of the model, the sentences are going to be presented as a whole to the participant. For this, a single rule is sufficient, which will state that whenever the display is empty, a sentence should be chosen from the list of stimuli and shown on the display.

3.3.1 Specifying Rules

1. Rules are added to processes the same way components are added to buffers. Add an empty rule to the process by clicking on the *"If... then..."* tab and then on the canvas.

2. Double-click on it to open the rule editor. You can see that COGENT rules are made up of a *Condition* part and an *Action* part (you can ignore for now the the check boxes at the top of the window). Basically, conditions specify when the rule should fire, and actions how the state of the system is changed by the rule.



Subobject Editor: Item 1 of Present Stimulus

Comment: Select and show a sentence

Rule is triggered? Rule is refracted? Rule fires once per cycle?

Triggering pattern:

Conditions:

not is in Display

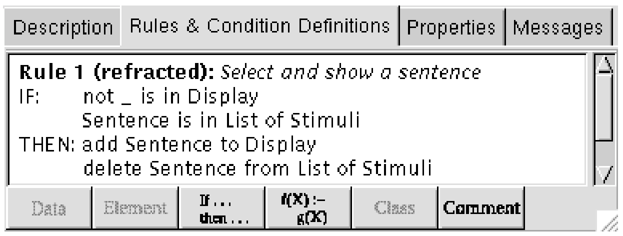
Sentence is in List of Stimuli

Actions:

add Sentence to Display

delete Sentence from List of Stimuli

3. If you now close the rule editor, you will see that on the canvas, the conditions are preceded by the key word `IF:` and the actions by `THEN:` (the meaning of *"refracted"* will be explained later).



Rule 1 (refracted): *Select and show a sentence*

```
IF:     not _ is in Display
        Sentence is in List of Stimuli
THEN:   add Sentence to Display
        delete Sentence from List of Stimuli
```

Conditions Since sentences are going to be presented whole to the participant, the only condition for the current rule is that the display should be empty.

a) Open the rule editor and try to enter the condition: Click on *Add Condition*, select *match* from the drop-down menu, and try to select *Display* from the drop-down menu on the right of the empty box that appears.

b) You will notice that the *Display* buffer is not available. This is because *Present Stimulus* cannot read from the display.

c) Go to top level of your model and add the necessary read arrow between *Present Stimulus* and *Display* by clicking first on the process and then on the buffer, so that the blunt triangle points towards *Display*. Both arrows will be displayed on top of each other.

d) Then come back to the rule editor: *Display* is now available from the list of buffers the rule can read from.

e) Now for the left side of the *match*: What does it mean for the display to be empty? In COGENT, you specify that a buffer should be empty by stating that it should not contain anything, or rather *"any thing"*.

   - *"Any thing"* can be represented by the Prolog anonymous variable, the underscore. It will be entered automatically by COGENT in the box if you leave it empty (see screenshots on page 4).
   - This needs to be negated: Click on the triangle in front of the empty box to open the drop-down menu, and select *Add qualifier → not*.

Actions Now for the rule's *Actions*. Rules in COGENT can do two main types of things, namely add or delete buffer elements.

a) First we need to select a sentence from the *List of Stimuli*. This is achieved by adding an extra *match* in the *Conditions*.

b) Enter a variable name for the sentence to be selected, e. g. `Sentence` (note the capital).

c) In order to add the sentence to the *Display* buffer, click on *Add action → add* and then choose which buffer to add the sentence to in the drop-down list.

d) Then enter the variable you chose under b) in the empty box on the right.

e) It is also important to remember to delete the variable from the list of stimuli, otherwise the sentence will be shown indefinitely over and over when we simulate an experiment. Repeat the two previous steps, but this time click on *Add action → delete*.

f) You will notice that you cannot choose *List of Stimuli* from the list of buffers.

g) This is because deleting from a buffer in COGENT requires write access to that buffer. Add the necessary arrow to the model, then choose *List of Stimuli* from the list of buffers and specify the variable in the field on the left.

You are now done specifying the rule. Before you close the editor though, add a comment at the top describing what the rule does, e. g. `"Select and show`

a sentence". Documenting your code this way will make it much easier to understand once our models get more complex.

3.4 Setting up *Press Key*

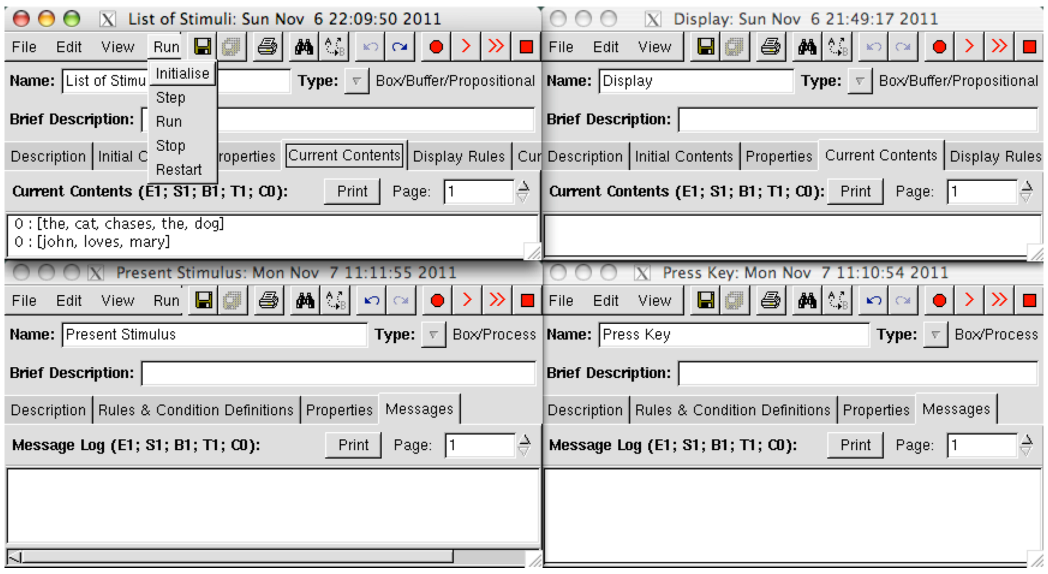
In the present model, no real "processing" will be done—we will leave that until next week. Therefore, the process *Read* will remain empty. However, you need to define a rule in *Press Key* which will represent the participant pressing the space bar when they are done reading a stimulus, thereby clearing the display and subsequently causing the next stimulus to appear. Write this rule and add a comment to it to explain what it does (e. g. ***"Clear the display"***).

3.5 Running the Model

We are now going to observe the model in action.

1. Open the following windows next to each other on your desktop:
   - The *Current Contents* tab of the *List of Stimuli* buffer,
   - The *Messages* tab in *Present Stimulus*,
   - The *Current Contents* tab in *Display*,
   - and the *Messages* tab in *Press Key*.

   Your other COGENT windows can be hidden underneath or reduced.


List of Stimuli: Sun Nov 6 22:09:50 2011
File Edit View Run
Initialise
Step
Run
Stop
Restart
Name:
Type:
Brief Description:
Description Initial C roperties Current Contents Display Rules Cur
Print Page:
Display: Sun Nov 6 21:49:17 2011
File Edit View
Description Initial Contents Properties Current Contents Display Rules
Present Stimulus: Mon Nov 7 11:11:55 2011
Description Rules & Condition Definitions Properties Messages
Press Key: Mon Nov 7 11:10:54 2011

Name: List of Stimu Type: Box/Buffer/Propositional

Current Contents (E1; S1; B1; T1; C0): Page: 1

```
0 : [the, cat, chases, the, dog]
0 : [john, loves, mary]
```

Name: Display Type: Box/Buffer/Propositional

Current Contents (E1; S1; B1; T1; C0): Page: 1

Name: Present Stimulus Type: Box/Process

Message Log (E1; S1; B1; T1; C0): Page: 1

Name: Press Key Type: Box/Process

Message Log (E1; S1; B1; T1; C0): Page: 1

2. Click on the red round button at the top right of any window to initialize the model. All your sentences should be displayed in the *Current Contents* tab of the *List of Stimuli* buffer. The `0` at the begining of each line indicates that they were added to the buffer the last time it was re-initialized.

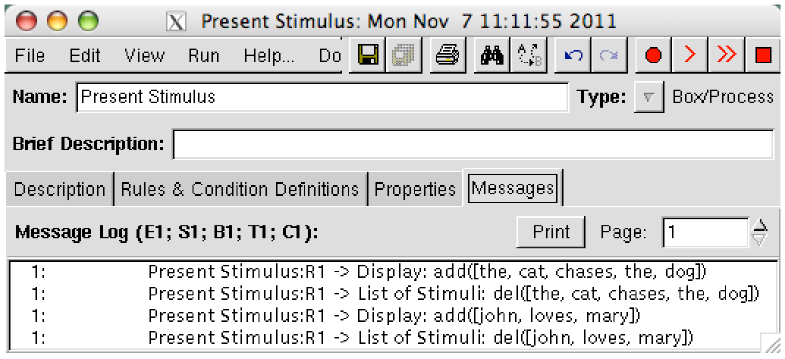
3. At same time, notice the newly initialized counter that appears below the tab bar in every COGENT window: This indicates the time in COGENT since the model

was last run. *E* stands for *"Experiment"*, *S* for *"Subject"* (same as Participant), *B* for *"Block"*, *T* for *"Trial"*, and *C* for *"Cycle"*. Model cycles are the smallest time unit in COGENT, during which all rules whose conditions were fulfilled at the previous step are executed in parallel.

4. Now click once on the single red arrow to step through one cycle. The counter will go up to *"...T1; C1"*, meaning that you are on trial 1, cycle 1. Notice that the red buttons are shortcuts for what is the menu *Run*.

5. Have a look at what is printed in *Present Stimulus*, in the *Display* and in *Press Key*: These messages inform you what actions just took place.

   - In *Present Stimulus*, you can see that on cycle `1` (the number on the far left), `R1` (Rule 1) in *Present Stimulus* fired, carrying out several actions: All your sentences were deleted from the list of stimuli, and added to the display.



Message Log (E1; S1; B1; T1; C1):

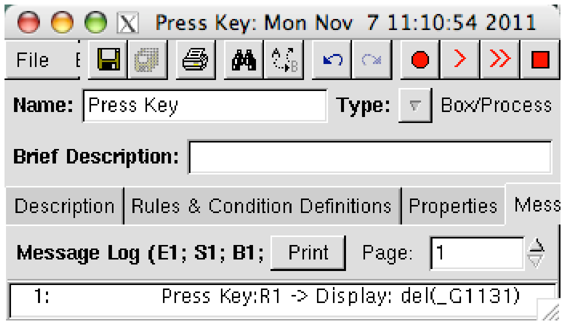
```
1:        Present Stimulus:R1 -> Display: add([the, cat, chases, the, dog])
1:        Present Stimulus:R1 -> List of Stimuli: del([the, cat, chases, the, dog])
1:        Present Stimulus:R1 -> Display: add([john, loves, mary])
1:        Present Stimulus:R1 -> List of Stimuli: del([john, loves, mary])
```

   The result of this can be seen in the *Current Contents* tab of *Display*.

   This, of course, was not the intended behavior. The reason is the inherently parallel nature of processing in COGENT: Since there were more than one possible bindings for *Sentence* in Rule 1, the rule was applied once for each set of bindings.

   - Moreover, notice what is printed in *Press Key*: Rule 1 has fired, deleting... "something" from the display. You can see that COGENT uses a Prolog unbound variable to represent what it deleted—or rather *"attempted"* to delete, because there was nothing in the display to be deleted.

   Such unbound variables should in general be avoided because of the unpredicted effects they can have.



Message Log (E1; S1; B1;

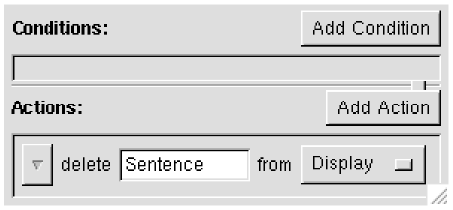
```
1:        Press Key:R1 -> Display: del(_G1131)
```

3.6 Correcting the Model

In the previous section, we observed two problems with our model. We are now going to correct this, starting with the unbound variables.

3.6.1 Unbound Variables

The simplest possible approach to clearing the display in section 3.4 is to create a rule with a single action in *Press Key* as shown below:



Rule 1 (refracted): *Clear the display*
IF: True
THEN: delete Sentence from Display

However, this leads to an unbound variable because in COGENT, all variables in a rule's actions must first be instantiated in the conditions.

Actually, had you wanted to add an element to a buffer instead of deleting one, COGENT would have complained, and you would have got an error at run time about trying to ``*add an ungrounded element to a grounded buffer"*. *"Ungrounded elements"* is COGENT's way of calling unbound variables.

The solution to this problem is simple: Using *match*, add "*Sentence is in Display*" to the rule's conditions. *Sentence* will be properly bound before being used in the actions, and therefore no unbound variable will be shown in the messages when you run the model.

3.6.2 Experimental Flow

Now let us turn to the experimental flow.

1. In a self-paced reading experiment, stimuli are presented one-by-one on the display, not all at the same time. This can be achieved by adding an additional constraint to the rule in *Present Stimulus*. Open the rule editor and check the box at the top of the window that says *"Rule fires once per cycle"*.

2. Re-initialize the model. The counter is reset to *"E1; S1; B1; T1; C0"*.

3. Now run the model again, slowly, step by step. You should observe that on odd-numbered cycles (1,3,5…) a sentence is added to the display (check the *Current Contents* tab in *Display* and the *Messages* tab in *Present Stimulus*), and then that it is deleted on even-numbered cycles (check the *Current Contents* tab in *Display* and the *Messages* tab in *Press Key*).

4. However, if you check the counter, you will notice that it is not incremented correctly: COGENT has not understood that the first trial was over after the sentence was deleted from the display. The problem is that COGENT waits for the model to be quiescent (i. e. no rules are applicable, therefore all processing within the model is complete) before assuming a trial is over.

5. In order to correct this, we need to resort to COGENT's "special triggers", and the solution in this case is not an obvious one. In *Present Stimulus*, open the rule and check the box at the top for *Rule is triggered*, then add `system_end(trial)` as a *Triggering pattern* (do not worry about the automatic change from *"refracted"* to *"unrefracted"*—this will be explained later).

6. Try to run the model again. What happens when the trial counter is incremented? You should see that the *List of Stimuli* buffer is re-initialized: All items you entered in the *Initial Contents* are re-added to the buffer. This, of course, should not happen: The buffer should be initialized once for every participant, not for every trial.

7. This can be changed by opening the *Properties* tab of the *List of Stimuli* buffer and choosing *Each Subject* instead of *Each Trial* in the drop-down menu next to *Initialise*. You may also want to do the same for all other components of the model. In this case, log messages will not be erased at the end of a trial any more. In some cases, this setting affects the experiment flow, in others, it is just a matter of whether you prefer all messages to be kept throughout the experiment or whether you would like the list shortened to the output of a single trial, especially when our models get more complex and more messages occur.

8. Now go through the model slowly again. Both item flow and counter should now be correct. Make sure you understand what is printed in the various buffers and processes.

9. As a final touch to *List of Stimuli*, we are going to make one more change so that sentences are always presented in the same order. In the *Properties* of *List of Stimuli*, change *"Access: Random"* to *"Access: FIFO"* so that buffer elements are accessed *"first-in-first-out"*, i. e. from the top of the list to its bottom.

4 Word-by-Word Presentation of the Stimuli

In self-paced reading experiments, sentences are not often presented as a whole to the participant. Usually, they are shown word-by-word, and the reading times from one word to the next are measured. We are now going to refine our previous model by building a second one on top of it.

4.1 Creating a Second Model on Top of the First

1. Close all the windows of your model.

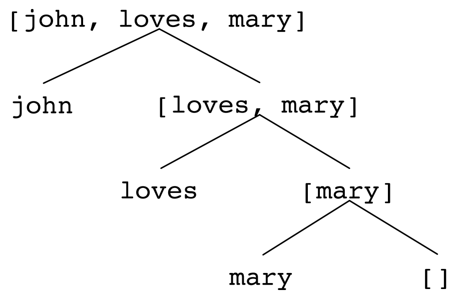
2. In the main COGENT window, create a copy of your model in the *Diagrammatic History* tab, and rename it something like *`"Word-by-Word Presentation"`*.

3. Note the link COGENT draws between both models, indicating steps in model development.

4.2 Taking Lists Apart

In order to display words one-by-one on the screen, we are going to have to "take apart" the Prolog list representing each sentence. Remember the explanations about lists:

- In many programming languages, lists are actually made up of a first element, the *"head"* of the list, and the rest, its *"tail"*, containing all other elements.
- The **head** of the list is a **single element**, whereas the **tail** of a list is **itself a list** which contains all elements but the first one.
- In Prolog, there is a special in-built operator to separate the head of a list from its tail, written as a vertical bar, "`|`" (the vertical line or pipe symbol). This is the notation which COGENT uses to take lists apart.
- At the end of a list, there is a special element, the empty list, in Prolog "`[]`". The empty list cannot be split apart.
- For example, the sentence "`[john, loves, mary]`" could also be written as "`[john | [loves, mary]]`" and could be taken apart in three steps as follows:



1. In order to present the sentences word-by-word, we need to add an additional buffer to the model to keep track of the next word in the current sentence. Add *Current Stimulus*.
2. Instead of writing the sentence directly to the display, we are now first going to store the complete sentence in *Current Stimulus*, and then add a new rule to *Present Stimulus* to show one word at a time. Modify the first rule in *Present Stimulus* so that when a stimulus is chosen, it is added to *Current Stimulus* instead of *Display*. You will need to add the necessary arrows to the model before you can do so, too.
3. Below, you can see the second rule in *Present Stimulus*, which you should add to the process.

```
Rule 2 (refracted): Present stimulus word-by-word
IF:     not _ is in Display
        [FirstWord|Rest] is in Current Stimulus
THEN:   add FirstWord to Display
        delete [FirstWord|Rest] from Current Stimulus
        add Rest to Current Stimulus
```

4. **It is important you understand well what happens here, since this constitutes the basis of list processing in COGENT.**

   a) The second condition says that if there is currently a sentence in *Current Stimulus*, it should be split apart. The variable `FirstWord` is bound to the head of the list (`john`), and the variable `Rest` to its tail (`[loves, mary]`).

   b) In the actions, the first element of the list (`FirstWord`) is added to the display,

   c) And deleted from *Current Stimulus* in two steps:

      I. First, the whole list, complete from head to tail, is deleted,

      II. And then the tail is re-added.

      This is an important way of proceeding in Prolog and thus in COGENT, which we are going to be making heavy use of.

   d) Note in particular how `Rest` is bound with `[loves, mary]` and therefore implicitly has outer brackets around it, although you do not spell them out, but `FirstWord` does not, since it is a single element.

5. Cycle through the model and observe how the variables are bound. Make sure you understand what is going on in all buffers and processes (*Current Contents* or *Messages* tab).

4.3 Applying Finish to the Model

Now for some finishing touches:

1. First, check your rules in the processes: Which occurences of `Sentence` should be replaced by `Word`? This does not matter to the computer, but making sure your variable names make sense will make your program much easier to understand—for you and anyone else trying to understand it.

2. Then, have a look at the contents of *Current Stimulus* after the last word of a sentence is deleted. You will notice it contains the last element of the original list, the empty list.

   - Do you understand what prevents its being moved to the display?
   - What problem does the presence of the empty list cause?
   - Can you think of a way to fix it? (Hint: Add a rule to "clean up" at the end of trial)

4.4 Refraction

Add the sentence *"The cat chases the dog"* to your model and make it the first stimuli in the experiment by right-clicking on it and selecting *Move* in the context menu, or by commenting other sentences using *Ignore*.

If you now cycle once again through the model, you will notice a problem after the second *"the"* is added to the display: Instead of deleting the word from the display as expected, the model will all of a sudden jump to trial 3 instead of processing the end of the sentence, indicating that it could not find any more rules to apply at the end of trial 2. The problem has to do with refraction, a tricky concept from artificial intelligence used in COGENT.

Refraction comes from from production systems, the type of system from which COGENT is inspired (this is also where the parallel firing of rules comes from). Refraction means that **a given rule** should normally **NOT fire twice with the same variable bindings**. What happens here is that the first time *"the"* is encountered, it is bound with *`Word`* in Rule 1 in Press Key, and thus deleted from the display, but the second time, refraction prevents this.

The solution, of course, is now obvious: Make the rule "unrefracted" by unchecking the corresponding box at the top the rule editor. The model should now work as expected. In the case of **linguistic stimuli**, **most processing rules** in the models we will build will need to be **unrefracted** for the model to work correctly. By default, new rules in COGENT are refracted, but if you choose to use a trigger, they automatically become unrefracted, although you can change this if you need to.

5 Gathering and Analyzing Reading Times

5.1 Outputing Information to a File

Until now, we have not been keeping track of what the model did. However, if we want data we can analyze later, we need to output something to a file so we have a basis for our analyses. This will also make our model of self-paced reading a little more "interesting".

1. Instead of the usual add and delete operations, writing to an output sink is done with *"send"*. Add the two following operations to the appropriate rules in your model (they do not belong all in the same rule, and you may need additional arrows):

   ```
   send show(Word) to Output
   send clear(Word) to Output
   ```

   Notice the use of complex Prolog terms to encode information. You are always free to define complex terms in your code if you need them, with whatever arity and as much nesting as you want.

2. Observe what is written to the data sink under the tab *Results*. You can scroll up and down by using the arrows next to *Page* in COGENT components.

3. In order to save the output to a file, change the value of the location field in the sink's *Properties* tab to *"Input/Output Directory"* instead of *"local"*.

4. At the same time, change when the text sink is initialized to keep the results of the whole experiment, not just one trial.

5. You can use the red double arrow to run complete trials in one go. If you want to run a whole experiment, you can do so by opening the *OOS window* in the *View menu*, going to the tab *Current Script* and making modifications there. However, it is best not to overwrite COGENT's default scripts but to save your own scripts

under other names by using the ***File*** menu. Notice how scripts can call each other: In the script loaded by default in COGENT, another script is called, namely ***Trial***, which you can open to have a look at, too. This is the most basic script in COGENT. If you want to run whole experiments, you could thus create your own scripts calling the ***Trial*** script with different parameters. More information about scripts can be found in the documentation under *"Running Models"*.

6. Open the output file to have a look at it. You will find it in your COGENT directory, in the subfolder `"Output"`.

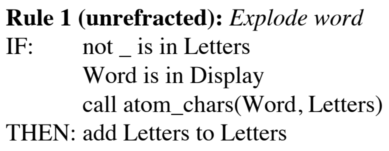
5.2 Making Reading Times Dependent on Word-Length

The run times in our current model cannot be said to be very interesting, because every word takes exactly two cycles to process. You are now going to extend the model so that reading times are dependent on word-length.

1. Create a new model based on the previous one and give it a new name.

2. Add a buffer ***Letters*** to which you can add the letters contained in a word in the form of a list. You could consider this buffer as a participant's "temporal memory" while reading words.

3. In ***Read***, add the following rule, possibly adapting the variables to your needs (note the difference between `Letters`, the variable, and ***Letters***, the buffer!):



Rule 1 (unrefracted): *Explode word*
IF: not _ is in Letters
Word is in Display
call atom_chars(Word, Letters)
THEN: add Letters to Letters

`atom_chars/2` is a call to an built-in Prolog operator which converts a string to a list of characters. For example, if `Word` is bound with `"john"`, the variable `Letters`, which in Prolog approximately corresponds to a "return variable" in other programming languages, would be bound to `[j,o,h,n]`. You can then use the list of letters in whatever way you want within the rule. Built-in Prolog operators can be called the conditions of a COGENT rule by clicking on ***miscellaneous***, ***call Prolog*** in the ***Add Condition*** drop-down menu (but they an advanced feature of COGENT you do not need to remember).

4. Add another rule to ***Read*** that does naïve "letter recognition", in which letters are deleted one at a time from the list (see the rule that presents sentences word-by-word for help with the syntax; note that you can cut and paste rules between buffers via the context menu using ***Delete*** and ***Paste***).

5. Modify the rule in ***Press Key***:

   - You will need to make sure it fires only after all letters have been read, when the end of the list is reached (that is, when only the empty list, `[]`, remains in ***Letters***).

- It should also ensure the state of the system is reset to allow the next word to be read. For example, if anything is left in ***Letters***, such as the empty list, it may need to be deleted.

6. Check the ***Output*** sink: The numbers in front of `key_press` now correspond to the moment when the participant pressed the space bar, which is whenever they are done processing the current word. Here is an example of the kind of output you should get:

```
experiment(1) , (subject(1) , (block(1) , (trial(1) , date('21:55:40 08 Nov 2011'))))
2 : show(john)
8 : clear(john)
9 : show(loves)
16 : clear(loves)
17 : show(mary)
23 : clear(mary)
experiment(1) , (subject(1) , (block(1) , (trial(2) , date('21:56:06 08 Nov 2011'))))
25 : show(the)
30 : clear(the)
31 : show(cat)
36 : clear(cat)
37 : show(chases)
45 : clear(chases)
46 : show(the)
51 : clear(the)
52 : show(dog)
57 : clear(dog)
experiment(1) , (subject(1) , (block(1) , (trial(3) , date('21:56:12 08 Nov 2011'))))
```

7. Using a spreadsheet or a script in any programming language of your choice, it is then easy to demonstrate that there is a linear relationship between word length and reading times:

	show	**clear**		**length**	**time (cycles)**	**difference**
trial(1)	2	8	john	4	6	2
	9	16	loves	5	7	2
	17	23	mary	4	6	2
trial(2)	25	30	the	3	5	2
	31	36	cat	3	5	2
	37	45	chases	6	8	2
	46	51	the	3	5	2
	52	57	dog	3	5	2

The reading time for a word is thus its length in letters plus 2 cycles for additional processing.