

Cognitive Modeling

Lecture 3: Basic Features of Cogent

Sharon Goldwater

School of Informatics
University of Edinburgh
sgwater@inf.ed.ac.uk

January 18, 2010

1 The Cogent Environment

- Principal Features
- Data Visualization
- Model Testing

2 Modeling Language

- Overview
- Syntax
- Unification

3 Example Task

- Free Recall
- The Modal Model
- Long Term Store
- Decay, Time, and Rehearsal

Based on the Cogent tutorial held by Rick Cooper at the Cognitive Science Conference, Philadelphia, 2000.

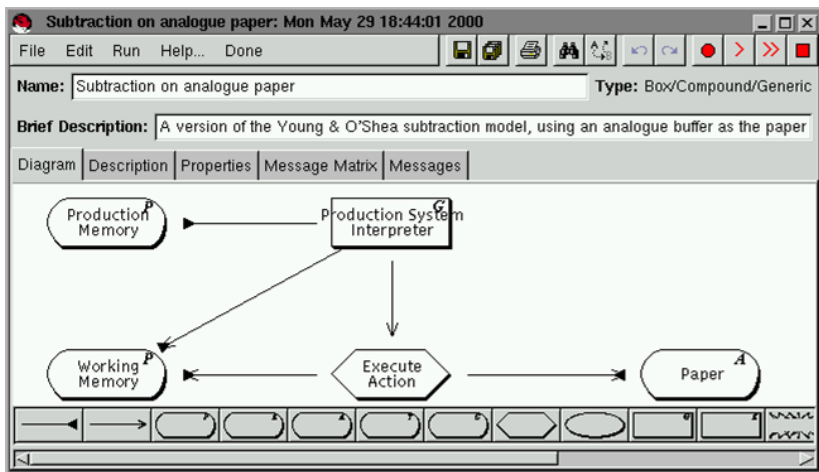
Reading: Cooper (2002: Ch. 2).

Cogent: Principal Features

Cogent offers the following features:

- a visual programming environment;
- research program management tools;
- a range of standard functional components;
- an expressive rule-based modeling language and implementation system;
- automated data visualization tools;
- a powerful model testing environment.

Visual Programming in Cogent



Research Program Management

COGENT Research Programme Manager

File Preferences... What's New... About... Help... Quit

Projects

- Arithmetic
- AttentionalBlink
- Cells
- Chapter3
- ContentionScheduling
- Graph
- Hanoi
- JDM2
- JDM2_Analysis
- JDM2_Assoc_Positive
- JDM2_Assoc_Unbiased
- JDM2_Assoc_Unbiased_Rep
- JDM2_Bayes
- JDM2_HT
- JDM2_Replications_Analysis
- LearningToAct
- Missionaries
- TMP
- ToL
- ToL_NEW
- TreeBuffer
- domino

Current Project

Project: Arithmetic Inception date: Sun Jun 11 14:28:00 1995

Brief description: Production system models of multicolumn addition and subtraction

History Description

New Root Scaling

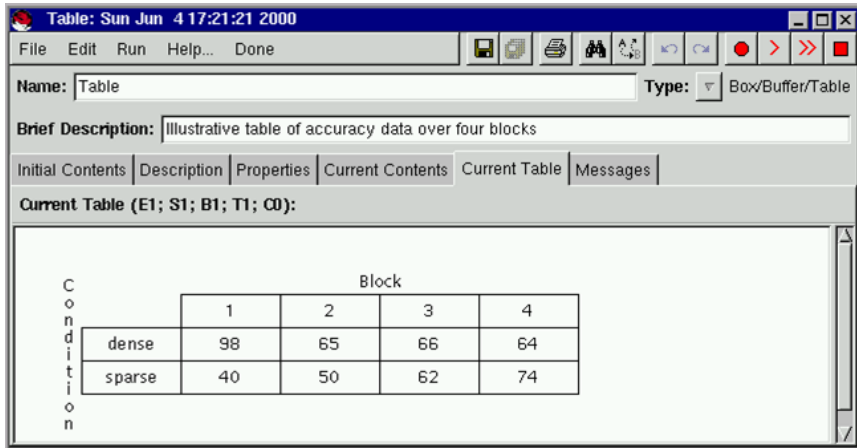
Diagram nodes:

- Addition: WM Intensive
- Addition: PM Intensive
- Addition: Goals & Subgoals
- Addition: Rando Rule Firing
- Addition: PB Intensive
- Subtraction (Young & O'Shea)
- Subtraction on analogue paper
- Subtraction (conflict resoluti...

Standard Functional Components

- A library of components is supplied:
 - memory buffers;
 - rule-based processes;
 - simple connectionist networks;
 - data input/output devices.
- Components can be configured for different applications.

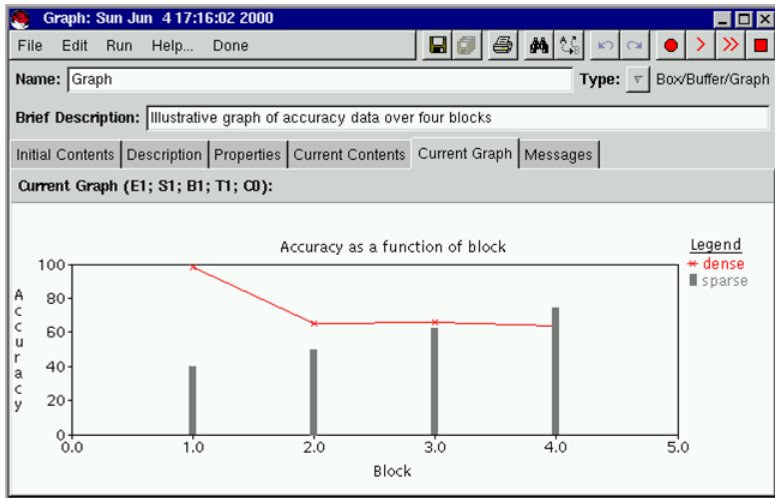
Data Visualization Tools: Tables



The screenshot shows a window titled "Table: Sun Jun 4 17:21:21 2000". The window has a menu bar with "File", "Edit", "Run", "Help...", and "Done". Below the menu bar is a toolbar with icons for file operations and navigation. The "Name" field contains "Table" and the "Type" dropdown is set to "Box/Buffer/Table". The "Brief Description" field contains "Illustrative table of accuracy data over four blocks". Below the description are tabs for "Initial Contents", "Description", "Properties", "Current Contents", "Current Table", and "Messages". The "Current Table (E1; S1; B1; T1; C0):" section displays a table with the following data:

C o n d i t i o n	Block			
	1	2	3	4
dense	98	65	66	64
sparse	40	50	62	74

Data Visualization Tools: Graphs



Data Visualization Tools: Pictures

The screenshot displays a software window titled "Current State: Mon Apr 10 11:57:11 2000". The window has a menu bar with "File", "Edit", "Run", "Help...", and "Done". Below the menu bar is a toolbar with icons for file operations (save, print, zoom) and navigation (back, forward, stop, refresh). The main area contains a form with the following fields:

- Name:** **Type:**
- Brief Description:**

Below the form is a tabbed interface with tabs for "Initial Contents", "Description", "Properties", "Current Contents", "Current Image", and "Messages". The "Current Image" tab is selected, showing a diagram labeled "Current Image (E1; S1; B1; T1; C22):".

The diagram depicts a horizontal grey bar representing a track. Three vertical grey lines (posts) are positioned on the track. Each post has a white rectangular base on the track. The leftmost post has a white rectangular block on top of its base. The middle post has a white rectangular block on top of its base. The rightmost post has a white rectangular block on top of its base.

The Model Testing Environment

- Visualization tools are dynamically updated;
- facilities are included to trace inter-component communication;
- a flexible “scripting” environment allows:
 - models to be run over multiple blocks of trials;
 - multiple “subjects” to be run over multiple blocks;
 - automated parameter varying “meta-experiments”.

Rule-Based Modeling Language

Processes may contain rules such as:

```
IF minuend(X) is in Working Memory  
   subtrahend(X) is in Working Memory  
THEN add equal(minuend, subtrahend) to Working Memory  
   send difference(0) to Write Answer
```

Cogent's representation language is based on **Prolog**.

We will use teletype for terms and **boldface** for buffers.

Rule-Based Modeling Language

The screenshot shows a window titled "Match Productions: Fri Jun 9 12:56:22 2000". The interface includes a menu bar (File, Edit, Run, Help..., Done), a toolbar with icons for file operations and navigation, and a main text area containing rule definitions. A "Name:" field contains "Match Productions" and a "Type:" dropdown is set to "Box/Process". A "Brief Description:" field contains "Update match memory on quiescence with all production matches". Below this are tabs for "Rules & Condition Definitions", "Description", "Properties", and "Messages". The main text area displays the following content:

```
Rule 1 (unrefracted): Add new matching productions to match memory  
TRIGGER: system_quiescent  
IF: rule(R, C, A) is in Production Memory  
    match_conditions_in_wm(C, Matches)  
THEN: add rule(R, C, A, Matches) to Match Memory  
  
Condition Definition: match_conditions_in_wm/2: Match the LHS of a production  
match_conditions_in_wm([], []).  
match_conditions_in_wm([H|T], [H|Matches]) :-  
    H is in Working Memory  
    match_conditions_in_wm(T, Matches).
```

At the bottom, there is a table with columns: Data, Element, If... then..., f(X):-g(X), Comment, and a Summarise button.

Basic Syntax

Cogent's representational unit is the *term*. Terms include

- Numbers: reals or integers. Ex: 6, 6.0.
- Atoms: any string of letters, digits, or '_' beginning with a lower-case letter. Ex: apple, b0, myName, response_count.
- Variables: any string of letters, digits, or '_' beginning with an upper-case letter or '_'. Ex: Apple, B0, MyName, _count.
- Lists: for representing sequences; consist of comma-separated terms. Ex: [a, b, c], [X], [].
- Compound terms: for representing structured information; consist of a *functor* and *arguments*. Ex: word(apple), date(6, jan).

Unification

The power of Cogent's representational system comes from *unifying* terms, binding variables to values (or to other variables).

```
IF word(Word) is in Stimuli  
  the current cycle is Cycle  
THEN send memorize(Word) to Subject  
  delete word(Word) from Stimuli  
  add presented(Word, Cycle) to Stimuli
```

Rules of Unification

- Variables match anything.
- Compound terms must have matching functors and arity.
- Lists must have matching lengths.

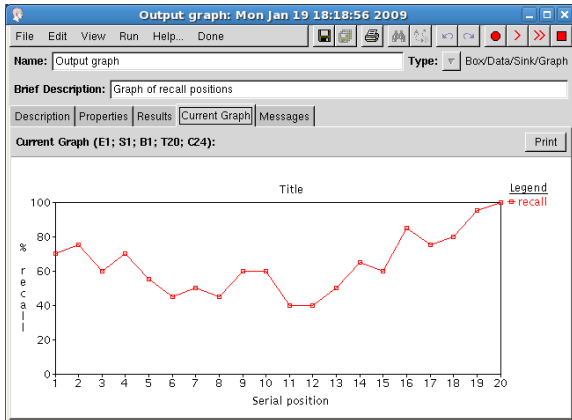
Terms	Unifies as	Bindings
$X, \text{word}(\text{Word})$	$\text{word}(\text{Word})$	$X \rightarrow \text{word}(\text{Word})$
$f(a,B,c), f(X,Y,Z)$	$f(a,B,c)$	$X \rightarrow a, Y \rightarrow B, Z \rightarrow c$
$f(X), g(a)$	fails	
$f([a,B], []), f([B,B], B)$	fails	

The Task: Free Recall

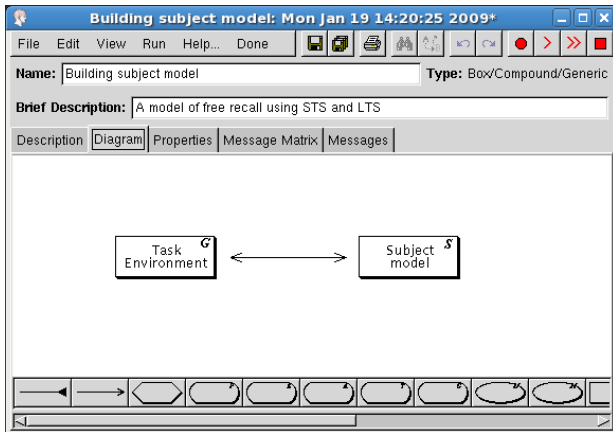
Classical experiment on word learning:

- on each trial, the subject is presented with a list of 25 words;
- the subject is told to try to memorize the words;
- after an interval, the subject must recall as many words as possible.

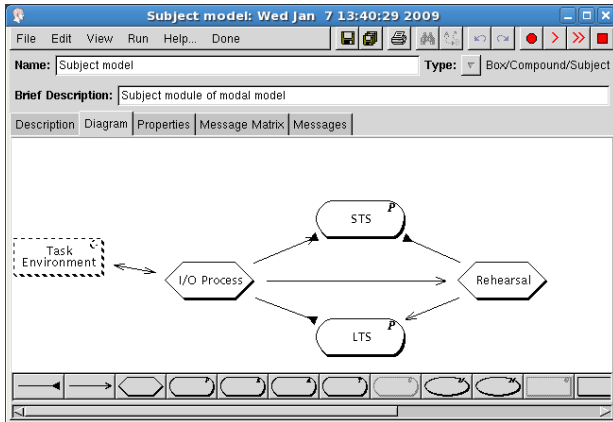
Free Recall: Empirical Findings



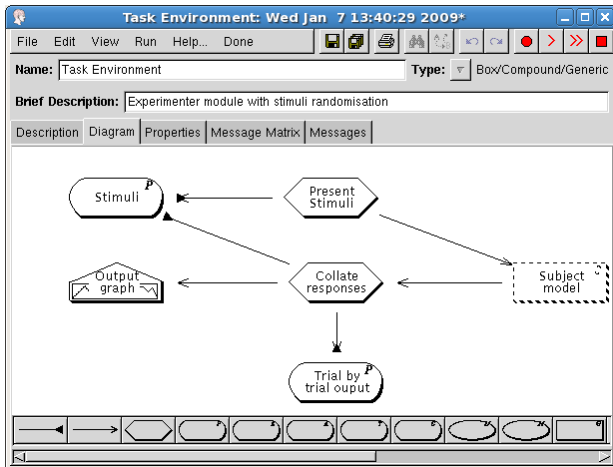
The Modal Model: Top Level



The Modal Model: Subject



The Modal Model: Task environment



The Modal Model: Stimuli

Stimuli: Fri Jan 9 11:28:19 2009*

File Edit View Run Help... Done

Name: Stimuli Type: Box/Buffer/Propositional

Brief Description: List of words to memorize

Description Initial Contents Properties Current Contents Display Rules Current Display Messages

Element: Nothing
word(black)

Element: Nothing
word(white)

Element: Nothing
word(red)

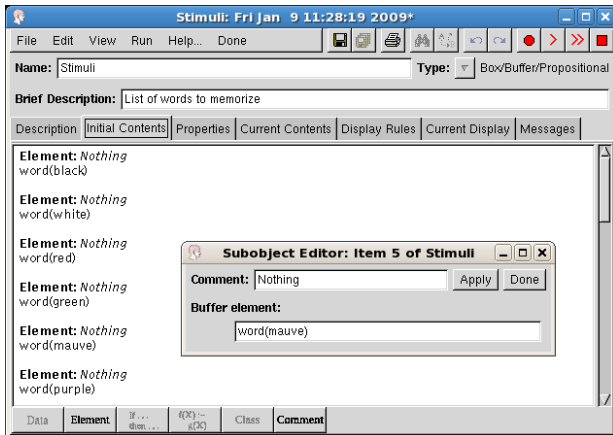
Element: Nothing
word(green)

Element: Nothing
word(mauve)

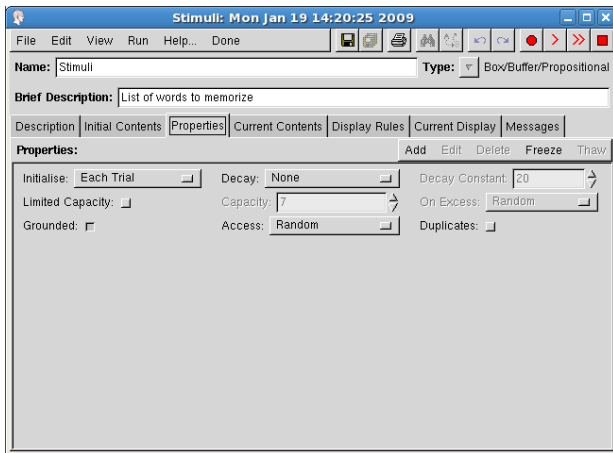
Element: Nothing
word(purple)

Data	Element	If ... then ...	i(x) - s(x)	Class	Comment

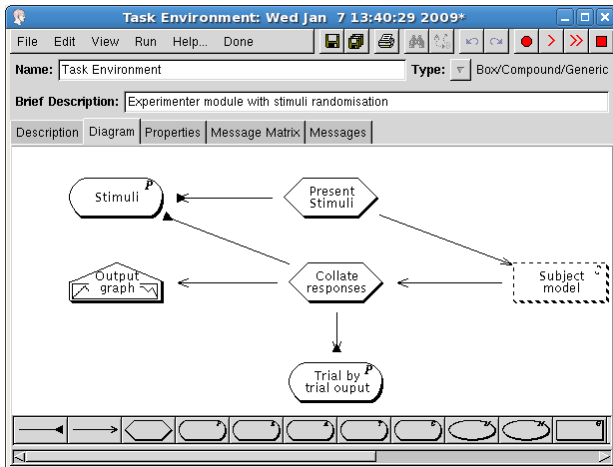
The Modal Model: Editing stimuli



The Modal Model: Stimuli properties



The Modal Model: Task environment



The Modal Model: Presenting stimuli to subject

The screenshot shows a software window titled "Present Stimuli: Mon Jan 19 14:20:25 2009*". The window has a menu bar (File, Edit, View, Run, Help..., Done) and a toolbar with icons for file operations and execution. Below the menu bar, there is a "Name:" field containing "Present Stimuli" and a "Type:" dropdown menu set to "Box/Process". A "Brief Description:" field contains the text "Presents stimuli in random order".

The main area of the window is divided into tabs: "Description", "Rules & Condition Definitions", "Properties", and "Messages". The "Rules & Condition Definitions" tab is active, displaying two rules:

```

Rule 1 (refracted; once): Present one stimulus per cycle
IF: word(X) is in Stimuli
    the current cycle is C
THEN: send memorise(X) to Subject model:I/O Process
      delete word(X) from Stimuli
      add presented(X, C) to Stimuli

Rule 2 (refracted): Trigger recall when no more stimuli
IF: not word(X) is in Stimuli
THEN: send recall to Subject model:I/O Process
    
```

At the bottom of the window, there is a table with the following headers: "Data", "Element", "If... then...", "f(X) :- g(X)", "Class", and "Comment".

The Modal Model: Messages

Present Stimuli: Mon Jan 19 14:36:45 2009

File Edit View Run Help... Done

Name: Present Stimuli Type: Step one cycle class

Brief Description: Presents stimuli in random order

Description Rules & Condition Definitions Properties Messages

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

Print Page: 1

```

1: Present Stimuli:R1 -> I/O Process: memorise(white)
1: Present Stimuli:R1 -> Stimuli: add(presented(white, 1))
1: Present Stimuli:R1 -> Stimuli: del(word(white))
    
```

I/O Process: Mon Jan 19 14:34:02 2009

File Edit View Run Help... Done

Name: I/O Process Type: Box/Process

Brief Description: Accesses the memory processes

Description Rules & Condition Definitions Properties Messages

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

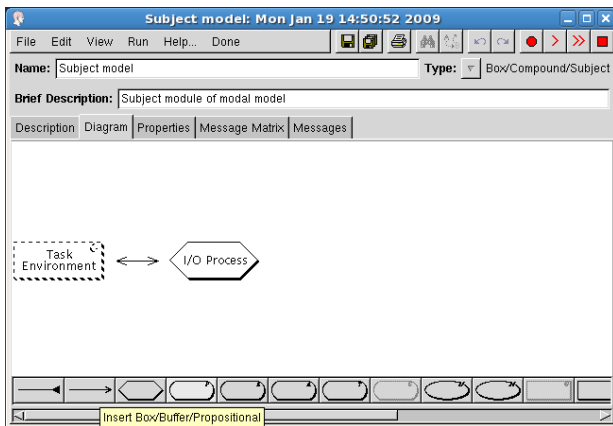
Print Page: 1

```

1: Present Stimuli:R1 -> I/O Process: memorise(white)
    
```

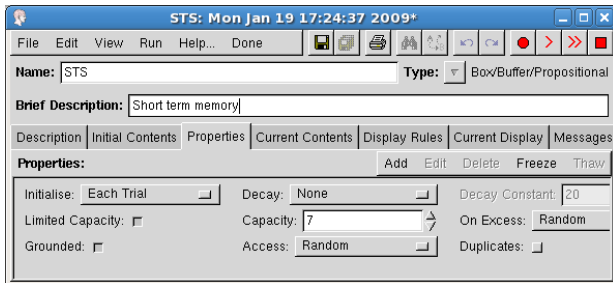
Building the Short Term Store

Create propositional buffer by clicking on button:



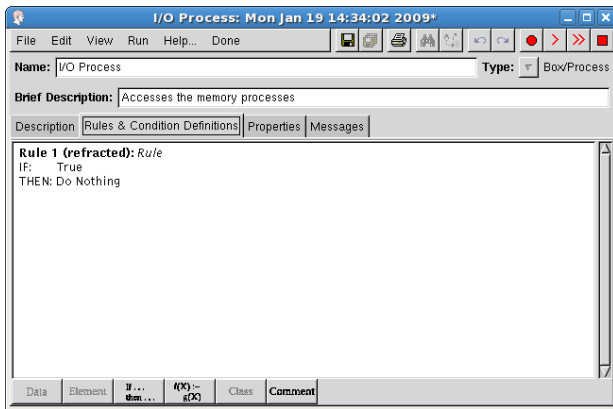
Building the Short Term Store

Double-click new buffer to name it and edit properties. **STS** has Limited Capacity of 7:



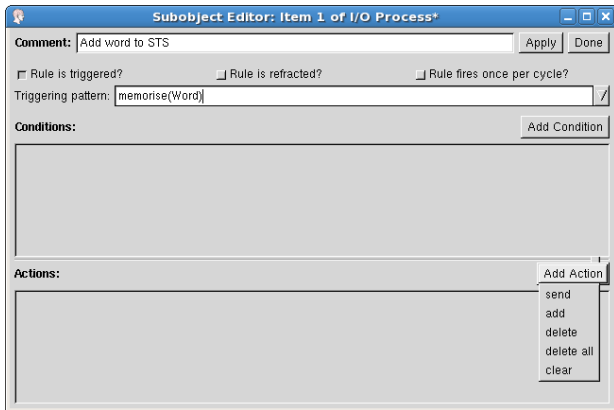
Building the Short Term Store

Open **I/O Process** and add an If...Then... rule:



Building the Short Term Store

Double-click rule to edit:



Building the Short Term Store

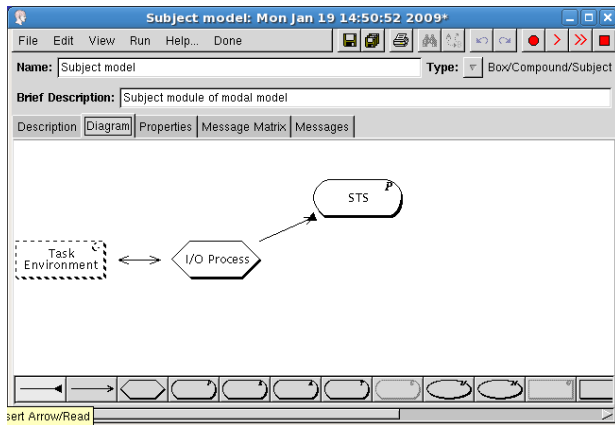
The rule to transfer words to **STS**:

The screenshot shows a window titled "Subsubject Editor: Item 1 of I/O Process*". The window contains the following fields and controls:

- Comment:** A text box containing "Add word to STS" with "Apply" and "Done" buttons to its right.
- Options:** Three checkboxes: "Rule is triggered?" (checked), "Rule is refracted?" (unchecked), and "Rule fires once per cycle?" (unchecked).
- Triggering pattern:** A text box containing "memorise(Word)" with a search icon to its right.
- Conditions:** A section with an "Add Condition" button and an empty list area.
- Actions:** A section with an "Add Action" button and a list area containing one action: "add Word" in a text box, followed by "to STS" in a button.

Building the Short Term Store

Add a read arrow from **I/O Process** to **STS**:



Building the Short Term Store

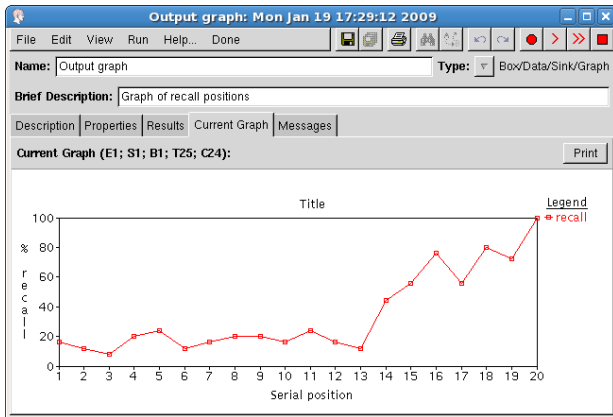
The rule to recall words from **STS** (use Add Condition → match):

The screenshot shows a window titled "Subobject Editor: Item 2 of I/O Process". It contains the following fields and controls:

- Comment:** Read from STS. Buttons: Apply, Done.
- Rule is triggered? Rule is refracted? Rule fires once per cycle?
- Triggering pattern:** recall
- Conditions:** Add Condition button. A list containing: Word is in STS.
- Actions:** Add Action button. A list containing: send Word to Task Environment:Collate responses.

Building the Short Term Store

Recall graph now shows a recency effect (output is for 20 trials):



Building the Short Term Store

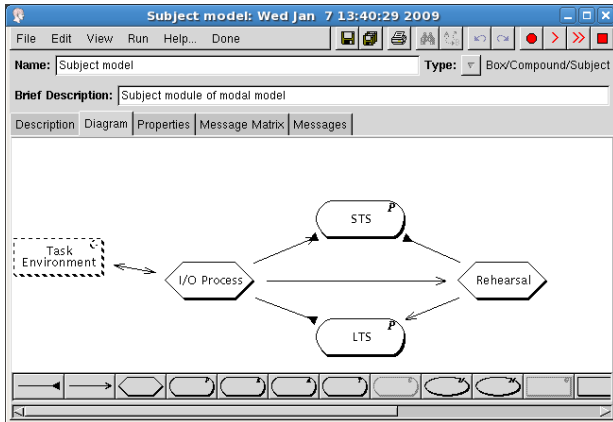
- What causes the recency effect? If we changed the properties of **STS**, could we change the shape of the graph?
- Watch the Messages view of **Input/Output**. What happens there now when you run (or single-step) through a trial?

Adding the Long Term Store

The modal model also includes:

- a long term store (LTS);
- a rehearsal process to transfer information from STS to LTS;
- the possibility to recall from either STS or LTS.

Adding the Long Term Store



Adding the Long Term Store

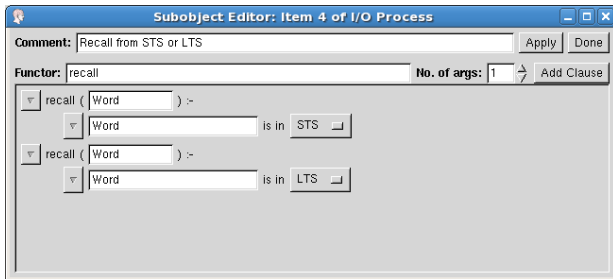
The rehearsal rule:

The screenshot shows a window titled "Subobject Editor: Item 1 of Rehearsal". It contains the following fields and controls:

- Comment:** A text field containing "Rehearse from STS to LTS". To its right are "Apply" and "Done" buttons.
- Options:** Three checkboxes: "Rule is triggered?" (unchecked), "Rule is refracted?" (unchecked), and "Rule fires once per cycle?" (checked).
- Triggering pattern:** An empty text field with a slash "/" at the end.
- Conditions:** A section with an "Add Condition" button. Below it is a list containing one condition: a dropdown menu with a downward arrow, a text field with "X", the text "is in", another dropdown menu with "STS" and a small icon, and a small icon.
- Actions:** A section with an "Add Action" button. Below it is a list containing one action: a dropdown menu with a downward arrow, the text "add", a text field with "X", the text "to", another dropdown menu with "LTS" and a small icon, and a small icon.

Adding the Long Term Store

To recall from either **STS** or **LTS**, we define a new condition by clicking the $f(X) :- g(X)$ button and editing the definition:



- Read $:-$ as 'if': `recall(Word)` is true if `Word` is in **STS**.

Adding the Long Term Store

The screenshot shows a software window titled "I/O Process: Mon Jan 19 18:16:14 2009*". The window has a menu bar (File, Edit, View, Run, Help..., Done) and a toolbar with icons for file operations and execution. Below the menu bar, there are fields for "Name:" (I/O Process) and "Type:" (Box/Process). A "Brief Description:" field contains the text "Accesses the memory processes".

The main area of the window is divided into tabs: "Description", "Rules & Condition Definitions", "Properties", and "Messages". The "Rules & Condition Definitions" tab is active, displaying the following content:

```

Rule 1 (unrefracted): Add words to STS
TRIGGER: memorise(Word)
IF: True
THEN: add Word to STS

Rule 2 (unrefracted): Recall from STS or LTS
TRIGGER: recall
IF: recall(Word)
THEN: send Word to Task Environment:Collate responses

Rule 3 (unrefracted): Stop rehearsal on recall
TRIGGER: recall
IF: True
THEN: send stop to Rehearsal

Condition Definition: recall/1: Recall from STS or LTS
recall(Word) :-
    Word is in STS.
recall(Word) :-
    Word is in LTS.
    
```

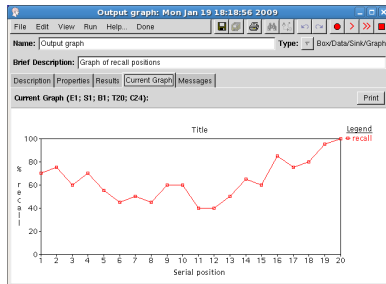
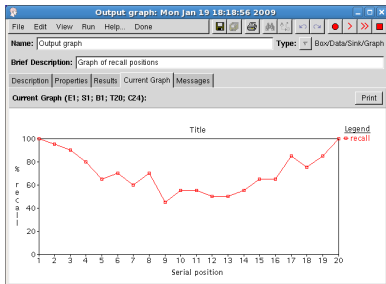
At the bottom of the window, there is a table with the following headers: "Data", "Element", "If... then...", "f(X):-g(X)", "Class", and "Comment".

Adding the Long Term Store

- What causes the Primacy Effect?
- Monitor the Messages view of the **I/O Process**. Why does the model sometimes recall the same word twice in the same trial?

Adding the Long Term Store

The current output (left) still doesn't match the output from the intro (right). What is different? Why?



Decay, Time, and Rehearsal

- Add decay to **LTS**. Explore different decay rates.
- Change the rehearsal rate by adding another copy of the rehearsal rule.
- All memorized words are currently recalled in parallel. Make the recall process serial.

Decay, Time, and Rehearsal

The serial recall rule:

The screenshot shows a software window titled "Subobject Editor: Item 2 of Input/Output". It contains the following fields and controls:

- Comment:** A text box containing "The recall rule" and a "Done" button.
- Options:** Three checkboxes: "Rule is triggered?", "Rule is refracted?", and "Rule fires once per cycle?".
- Triggering pattern:** A text box containing "recall".
- Conditions:** A section with an "Add Condition" button. Below it is a list containing one condition: "recall (Word)".
- Actions:** A section with an "Add Action" button. Below it are two actions:
 - send recalled(Word) to Experimenter:expt process
 - send recall to Input/Output

Decay, Time, and Rehearsal

- Explore the effect of the Access property of each buffer. Play with these (and other) parameters to see how they affect the model's behavior.
- The Experimenter system is written using standard Cogent. Try to discover how it works.
- Go on to develop the model into something substantial.

References

Cooper, Richard P. 2002. *Modelling High-Level Cognitive Processes*. Lawrence Erlbaum Associates, Mahwah, NJ.