COGENT: An Environment for the Development of Cognitive Models

Richard Cooper[*], Peter Yule[*], John Fox[⌖] & David Sutton[*]

[*]Department of Psychology
Birkbeck College
Malet Street,
London WC1E 7HX

[⌖]Advanced Computation Laboratory
Imperial Cancer Research Fund
Lincoln's Inn Fields
London WC2A 3PX

Abstract

COGENT[1] is a modelling environment which aims to improve the methodology of computational modelling by providing an integrated approach to model development, description, and evaluation. The environment is explicitly designed to allow psychologists with a range of computational expertise (from very little to a great deal) to develop their own computational models. It achieves these aims by providing the user with a graphical programming language based on the familiar box/arrow notation of informal cognitive psychology. This language draws on parallels between the psychological concept of functional modularity and the computational concept of object-orientation to provide a sound modelling tool in which models may be evaluated through methodologically rigorous computational experiments. COGENT has been used by a number of researchers to develop models in the domains of memory, reasoning, decision making, problem solving, the performance of complex tasks, and concept combination. This chapter presents an overview of COGENT, drawing upon two models to illustrate the system. The first, a production system model of multi-column addition, demonstrates many of the fundamental features of the system. The second, a model of Allen inferencing, uses the new "analogue buffer" features of the system to reimplement the metrical algorithm described by Berendt (1996).

1. Introduction

When attempting to develop a cognitive model, a researcher is faced with many issues. Perhaps the first to be addressed concerns the technology to be adopted. There are a bewildering array of alternatives: connectionism (in all its forms), the use of a symbolic cognitive architecture (such as Soar (Newell, 1990) or ACT-R (Anderson, 1993)), or the use of a general purpose programming language (e.g., Lisp or Prolog). If connectionism is inappropriate (because, for example, it is believed that the task being modelled involves overt symbol manipulation), the choice devolves to one between architecture and general programming language. Although the use of a cognitive architecture has unquestionable benefits (e.g., allowing the unitary application of constraints deriving from a range of tasks and domains), it is also fraught with difficulties (cf. Cooper & Shallice, 1995). Given the necessary complexities of cognitive architectures, can, for example, particular behaviour be attributed to specific architectural properties or mechanisms? How should work proceed if a particular behaviour proves intractable in the chosen architecture? Indeed, can any possible behaviour be shown to be intractable in a given architecture? And how should theorists proceed if they adhere to most but not all assumptions underlying their preferred architecture? If one accepts the argumentation behind these rhetori-

[1]COGENT is an acronym for Cognitive Objects within a Graphical EnviroNmenT. For further information, see `http://www.psyc.bbk.ac.uk/research/projects/cogent/`. We may be contacted by email at: `cogent@psyc.bbk.ac.uk`.

cal questions, then one is left with the general purpose programming language option. Although this option has great flexibility, it is generally unconstrained and suffers from a lack of sound methodology. (The most obvious methodological problem is the lack of any principled way of distinguishing implementation detail from theoretical commitment: cf. Cooper, Fox, Farringdon & Shallice, 1996.) In addition, the cognitive modeller requires considerable expertise if the general purpose programming language option is to be adopted. It therefore seems that there is a place in cognitive modelling for a special purpose modelling language which 1) embodies general principles of cognitive psychological theorising, 2) is relatively easy to learn/program, and 3) is underpinned by a sound methodological framework.

These concerns have led to the development of COGENT (Cooper & Fox, in submission), a modelling environment which aims to improve modelling methodology by addressing the three key areas of model description, model evaluation, and model development. In brief, COGENT addresses model description by providing a graphical interface to a well-defined object-oriented modelling language. The graphical interface allows researchers to specify their models in terms of interconnected boxes. Such box/arrow diagrams are one of the principal means of theory specification in cognitive psychology, and their use in COGENT is intended to allow psychologists to work with an established theory specification language. The use of box/arrow diagrams in cognitive psychology is, however, generally undermined by the lack of a well-defined operational semantics for the notation: although the diagrams are often annotated with text, the computational assumptions behind the diagrams are rarely, if ever, specified in sufficient detail to allow an objective assessment of the corresponding theory's behaviour. The box/arrow notation embodied within COGENT is supported by a well-defined operational semantics, allowing COGENT box/arrow diagrams to be executed and their behaviour thereby determined.

The basic building blocks of a COGENT model are low-level processing objects (or modules), such as buffers and rule-based processors. These objects are configurable: different processing capabilities can be assigned to different instances of the same class of object. Thus, different buffers can be specified as having different decay characteristics or capacity limitations. This configurability gives the primitive objects a great deal of flexibility, but also encourages a systematic approach to model evaluation. In particular, the sensitivity of a model's behaviour to different values of a given parameter can be assessed by running the model with those parameter values and comparing the resultant behaviours. In this way one can evaluate claims concerning the theoretical relevance of parameters, and thereby distinguish implementation detail (which should not affect behavioural measures) from theory (which, assuming theory falsifiability, will have an effect on behavioural measures).

Model development is supported within COGENT via the notion of a research programme. A research programme is grounded in a specific domain and generally attempts to answer a specific question through the development of a family of related computational models (possibly in association with a series of standard laboratory experiments).

The remainder of this chapter begins with a more detailed presentation of the facilities provided by COGENT. A simple production system model of multi-column addition is used to illustrate this presentation. Section 3 provides a more detailed exploration of the use of COGENT, focusing on the Allen inference task (Knauff, Rauh & Schlieder, 1995) and, in particular, the metrical algorithm of Berendt (1996). We conclude with a general discussion about the role of COGENT and its relation to other approaches to cognitive modelling.

2. The Basics of COGENT

The complete COGENT environment currently provides four distinct facilities: a set of well-defined configurable object classes (corresponding, intuitively, to types of cognitive module); a graphical interface for specifying models in terms of interconnected instances of these classes (i.e., in terms of box/arrow diagrams); a computational engine for running such models (i.e., a means of executing box/arrow diagrams to determine their behaviour); and a tool for maintaining sets of models within distinct research programmes. We describe and illustrate each of these in turn.

2.1 The Primitive Object Classes

Much of the power of COGENT comes from the set of standard object classes built into the system. There are currently five major classes of object: buffer, rule-based process, network, data, and compound. These classes, which are derived from the kind of functional modules used within cognitive psychological theorising, provide the building blocks with which individual models are constructed.

2.1.1 Buffers

A buffer is an information store: a place where items of information may be put for later retrieval. Buffers are appropriate for both short-term storage (e.g., modelling working memory) or long-term storage (e.g., a knowledge base). They may be configured, via a large set of properties, so as to behave in a variety of different ways, allowing, for example, element decay and/or capacity limitations.

In its default form, a buffer will have unlimited capacity, no decay and random access. This means that an indefinite number of elements may be added to the buffer, all elements will be available for later recall unless explicitly deleted from the buffer, and order of recall will be non-deterministic. If capacity limitations are applied (via selection of the “Limited Capacity” property), only a specified number of elements will be allowed in the buffer. The actual number is given by a second property. A third property determines the behaviour of a limited capacity buffer once its capacity is reached. This property can be set, for example, so that when a further element is added the oldest element is deleted, thus providing the functionality of a push-through store.

Spontaneous decay of buffer elements can also be specified through a number of properties. If selected, decay may be deterministic (such that elements will disappear after a given number of processing cycles) or probabilistic (such that elements have a “half life”, by which time there is a 50% chance that they will have disappeared).

Yet more properties specify the access characteristics of buffers. Access may be random or based on the order in which elements were added to the buffer (either FIFO, that is, least recent first, or LIFO, most recent first).

The graphical interface to COGENT, as described below, presents the user with all possible buffer properties in such a way that the user may click and select to set the properties as required (see Figure 3 for an example property panel from the multi-column addition model). This approach greatly simplifies the programmatic aspects of modelling storage devices. There are additional benefits, however. The range of properties applicable to buffers makes explicit a range of possible buffer behaviours, and shows that a module’s behaviour is not specified merely be labelling that module as a buffer.

2.1.2 Rule-Based Processes

Rule-based processes are devices whose behaviour is determined by a set of symbolic rules. Rules are just condition/action pairs, where the conditions typically involve matching elements in buffers (though advanced users may also include arbitrary Prolog) and the actions allow passing of messages to other boxes. There are two basic types of rule: triggered rules (which are activated when the process containing them receives a message that matches the rule's triggering pattern); and autonomous rules (which test their conditions on each processing cycle and hence do not require a triggering signal to fire). An example of each type of rule is shown in Figure 1. In these rules, `Conds`, `Acts` and `X` are variables. The first of the rules is autonomous. It continuously monitors *Match Memory* for a term of the form `production(Conds, Acts)` that is not also in *Refractory Memory*. If and when it finds such a term, it adds it to *Refractory Memory* and sends a message of the form `execute(Acts)` to the box named *Fire Production.* The second rule (which might be found in the process named *Fire Production*) is triggered by receipt of a message of the form `execute(Acts)`. `Acts` is understood by this rule to be a list of terms, and, when the process containing the rule receives a message of the form `execute(Acts)`, each term `X` for which `add(X)` is a member of the `Acts` list will be added to *Working Memory*.

COGENT provides a rule-editor which simplifies the process of specifying rules and minimises the risk of syntax errors. (See Figure 4.)

2.1.3 Networks

Network objects provide COGENT with a primitive facility for the development of hybrid and connectionist models. Networks are fully specified via a series of properties, whose values determine input and output vector width, initialisation procedure, activation function and learning rule. At present only simple two-layer feedforward networks are supported, and no special facilities are provided for visualisation of network weights. However, future versions of COGENT are likely to incorporate more complex network objects, together with appropriate visualisation tools.

2.1.4 Data

COGENT provides two subsorts of data box to allow input and output from models. Data sources are intended to allow control over the input to a model. They contain a sequence of message-destination pairs (analogous to the "THEN" side of rules). As processing proceeds, elements are removed from sources and sent to the specified destination. Data sinks are the complement of data sources. They collect messages sent by other boxes and store them in a file. They fulfil no essential role in a cognitive model, but allow output to be collected for later

```
IF:       production(Conds, Acts) is in  Match Memory
          not production(Conds, Acts) is in Refractory Memory
THEN:     add production(Conds, Acts) to  Refractory Memory
          send execute(Acts) to  Fire Production

TRIGGER:  execute(Acts)
IF:       add(X) is a member of  Acts
THEN:     add X to  Working Memory
```

Figure 1: Autonomous and Triggered Rules

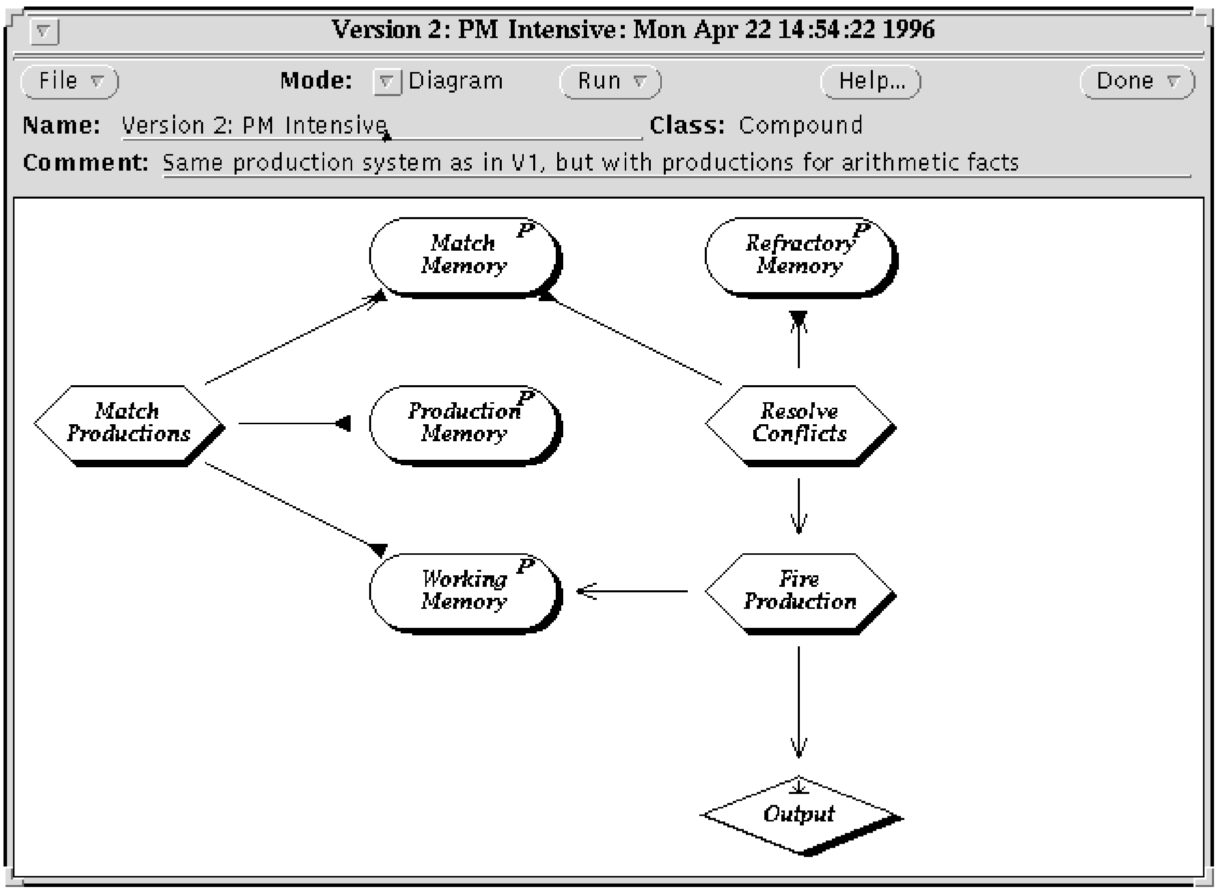
analysis.

2.1.5 Compounds

Compound objects provide COGENT with a bracketing facility, such that other sorts of object can be grouped into higher-order functional subcomponents. In box/arrow terms, a compound is just a box that contains further boxes, and the behaviour of a compound box is entirely determined by the behaviour of its subboxes and their interconnections.

2.2 The Graphical Interface

It is possible to develop models using COGENT's set of object classes and a standard text editor, and this in fact is how the first model was developed in the embryonic system (Cooper, 1995). However, the usability of the raw class set is hampered by the requisite syntax. COGENT's graphical interface circumvents this potential problem. The interface allows models to be drawn as box/arrow diagrams, which each box in the diagram corresponding to an instance of an object class. Figure 2 shows one such box/arrow diagram. In this (and all COGENT) diagrams, rule-based processes are depicted as hexagonal boxes, buffers are depicted as oblong boxes, and data sinks are depicted as diamonds. Two different types of arrow on the diagram indicate two forms of communication. Pointed arrows correspond to message passing (including buffer modification). Arrows with an blunt triangular head correspond to buffer reading. Thus, the process *Match Productions* reads from the buffer *Production Memory*. Superimposed arrows indicate both forms of communication: *Resolve Conflicts* reads from and

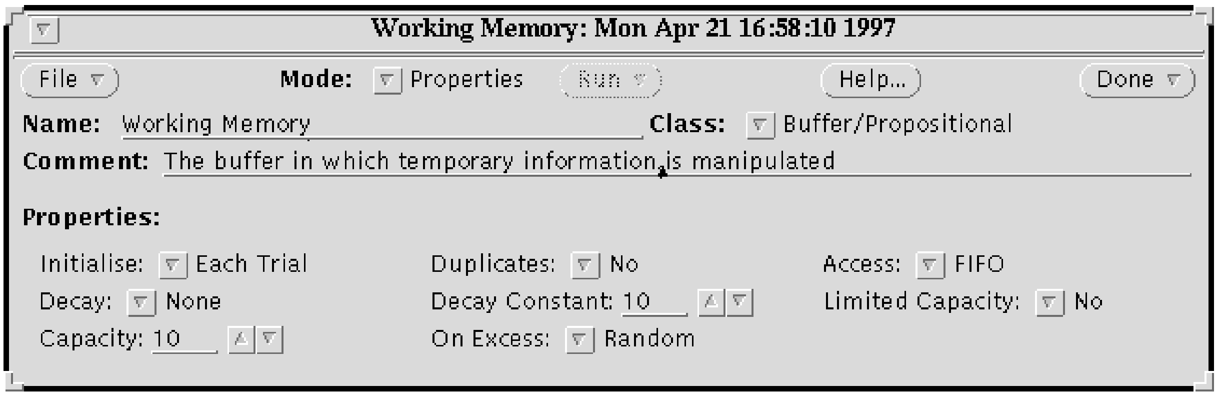

Version 2: PM Intensive: Mon Apr 22 14:54:22 1996

Name: Version 2: PM Intensive

Class: Compound

Comment: Same production system as in V1, but with productions for arithmetic facts

Figure 2: A COGENT Box/Arrow Diagram


Working Memory: Mon Apr 21 16:58:10 1997

Mode: Properties

Name: Working Memory Class: Buffer/Propositional

Comment: The buffer in which temporary information is manipulated

Properties:

Initialise: Each Trial Duplicates: No Access: FIFO

Decay: None Decay Constant: 10 Limited Capacity: No

Capacity: 10 On Excess: Random

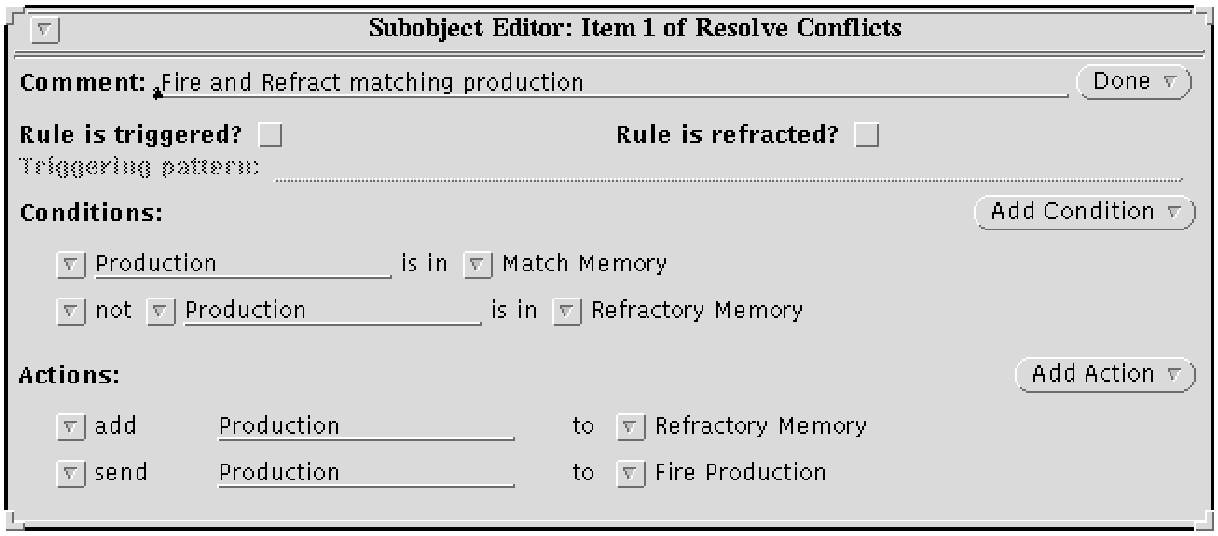
Figure 3: The Properties Panel of Working Memory

writes to *Refractory Memory*.

The graphical interface provides a palette of object classes, instances of which may be selected and positioned on the canvas. The resulting boxes can then be named and joined by the appropriate arrows as necessary.

In order to fully specify a COGENT model, it is necessary to flesh out the specifications of the boxes which constitute the box/arrow diagram. Double-clicking on any box opens the box to allow setting of class-specific properties and other instance-specific information. Figure 3 shows the properties panel for a typical buffer. As noted above, each property can be set by selecting values from menus, minimising the computational expertise required.

The specification of other instance-specific information (e.g., the rules within processes, or the initial states of buffers) is performed through a series of graphical editors. Each type of box element (e.g., rules, buffer elements, data source messages) has its own structured editor. Figure 4 shows the rule editor with one of the rules from Figure 1 loaded. Note that, as with property specification, most options are provided by menu buttons. Thus, a button to the left of the first condition hides a menu of editing commands which may be applied to that condition (which includes options to delete the condition, add a qualifier (such as a negation), insert a new condition before or after the current condition, or change the condition from a buffer


Subobject Editor: Item 1 of Resolve Conflicts

Comment: Fire and Refract matching production

Rule is triggered? ☐ Rule is refracted? ☐

Triggering pattern:

Conditions:

Production is in Match Memory

not Production is in Refractory Memory

Actions:

add Production to Refractory Memory

send Production to Fire Production

Figure 4: The Rule Editor

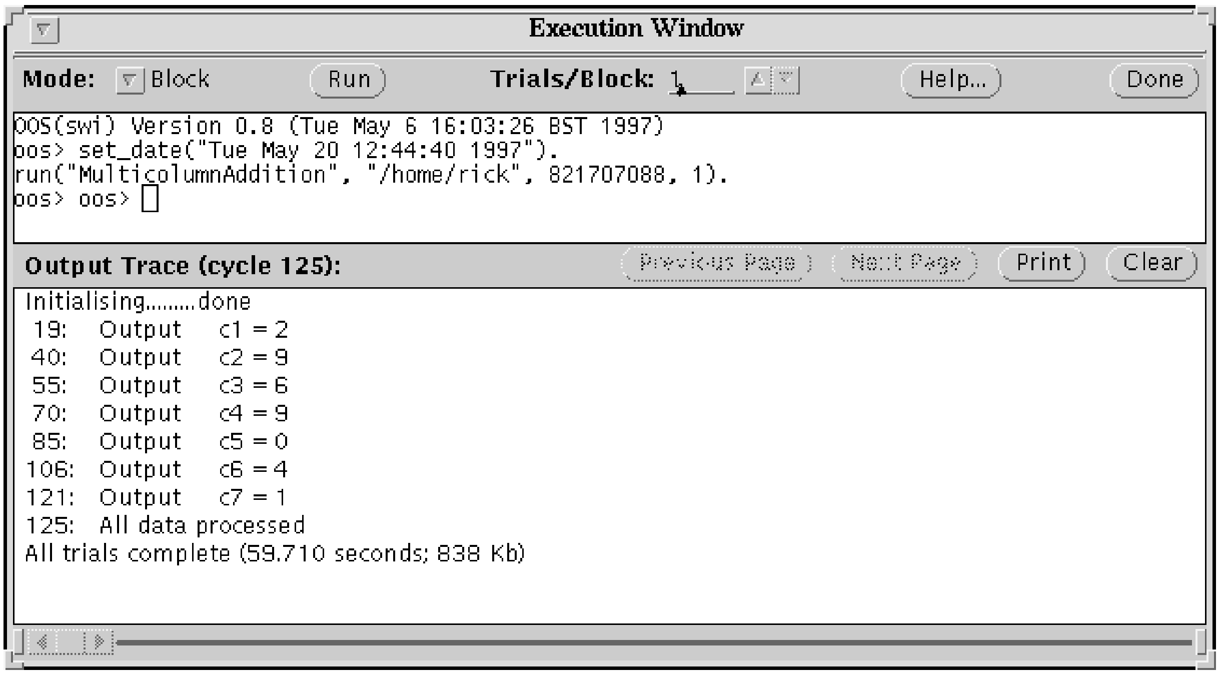
match to some other kind of condition). Similarly, the menu button to the left of "Match Memory" in the first condition allows the selection by menu of the buffer against which the specified term (`Production`) should be matched. In this (and all) editors, the user's textual input is limited to terms from a general purpose knowledge representation language (which is based on Prolog).

2.3 The Execution Model

COGENT's graphical interface maps between the user's representation of a model (expressed in terms of boxes, arrows, properties and rules) and a computationally complete executable representation of the model. COGENT also provides an interface to an execution engine on which the computational representation can be run.

The execution model underlying COGENT is cyclic and based on the parallel operation of subprocesses, with each box operating as a separate subprocess. Communication between subprocesses is effected by message passing. A global data channel is used to store all messages in transit and each processing cycle involves processing and updating this data channel. In brief, each box is considered on every processing cycle. Any messages for that box are removed from the data channel and processed. In the case of buffers, this processing may result in modifications to buffer contents (e.g., the addition or deletion of buffer elements). In the case of processes, messages on the data channel may trigger rules leading to the creation of new messages for processing on the next cycle. Autonomous rules within processes are also checked and, if their conditions are satisfied, the messages corresponding to their actions are added to the message channel for processing on the next cycle. Any messages on the channel destined for data sinks are consumed by copying them to the data sinks' output files and removing them from the channel. Data sources are also checked for the generation of messages, which are moved from the data source onto the global data channel. This processing is repeated for each box (effectively in parallel) on every processing cycle. See Cooper (1995) for more details.

Access to the execution engine is via an interface window (see Figure 5), which provides the



```
OOS(swi) Version 0.8 (Tue May 6 16:03:26 BST 1997)
oos> set_date("Tue May 20 12:44:40 1997").
run("MulticolumnAddition", "/home/rick", 821707088, 1).
oos> oos>
```

```
Initialising.........done
 19:   Output   c1 = 2
 40:   Output   c2 = 9
 55:   Output   c3 = 6
 70:   Output   c4 = 9
 85:   Output   c5 = 0
106:   Output   c6 = 4
121:   Output   c7 = 1
125:   All data processed
All trials complete (59.710 seconds; 838 Kb)
```

Figure 5: The Interface to the Execution Engine

user with a series of controls allowing execution of single processing steps, multiple processing steps, complete trials, or blocks of multiple trials. A complete trial corresponds to the notion of a trial within standard experimental psychology, and a block corresponds to a sequence of trials, also as in standard experimental psychology. The use of language rooted in experimental psychology reflects our commitment to empirical computational research, in which computational experiments and standard laboratory-based experiments go hand in hand.

Each function (i.e., step, trial, or block execution) is accessed via menu or panel buttons, eliminating the requirement that the user be familiar with a complex command language and associated syntax.

During model execution, output from data sinks is printed in the lower half of the window shown in Figure 5. The left column of numbers indicates the cycle on which the sink message was received. This is followed by the sink's name and the actual message.

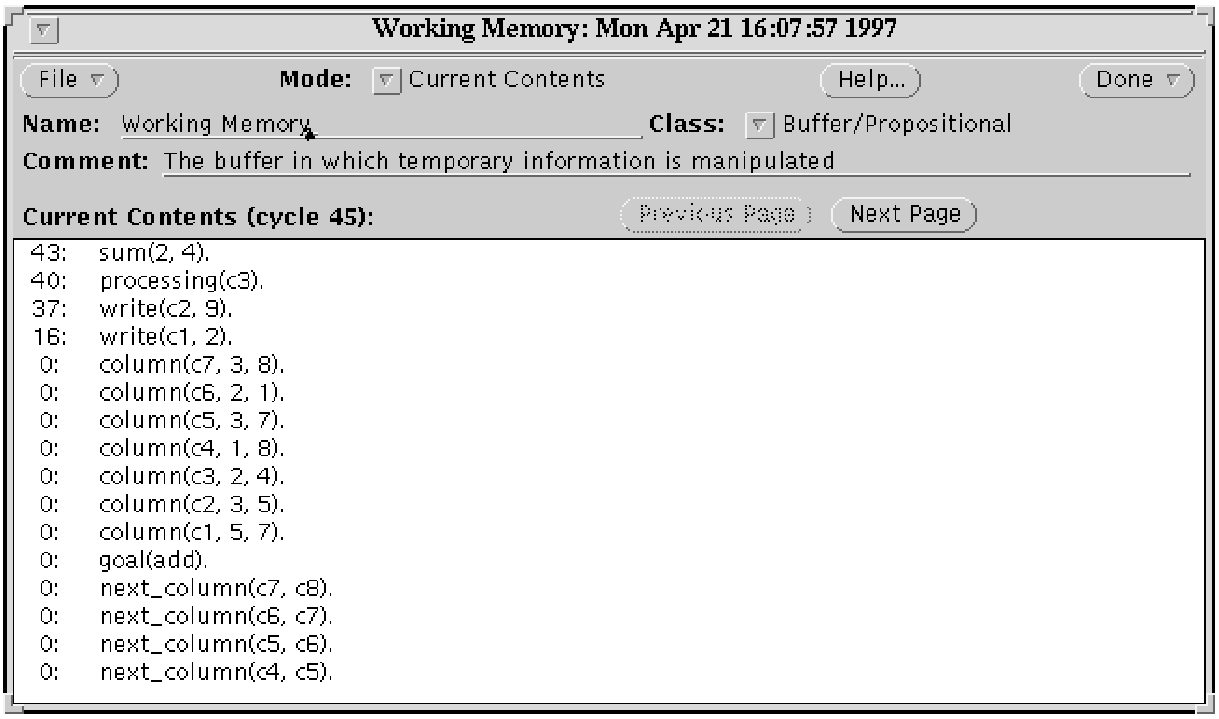
Boxes such as buffers and data sources/sinks, whose contents change during processing, may also be opened during model execution to show their evolving contents. (See Figure 6.)

A recent extension to the execution interface is the inclusion of a scripting facility. This facility allows the user to create extended sequences of processing commands. In this way it is possible to run, for example, three blocks of 25 trials, each with different decay properties set on some buffer in order to determine the criticality of the decay properties.

2.4 Support for Research Programmes

As noted in section 1, we consider model development (i.e., the evolution of models through time) to be an essential element of computational modelling, and further that support for model development should be embodied in the computational tools which support modelling itself.

COGENT supports model development through the concept of a research programme. A COGENT research programme consists of a set of computational models. There is a single



Mode: Current Contents

Name: Working Memory

Class: Buffer/Propositional

Comment: The buffer in which temporary information is manipulated

Current Contents (cycle 45):

```
43:   sum(2, 4).
40:   processing(c3).
37:   write(c2, 9).
16:   write(c1, 2).
 0:   column(c7, 3, 8).
 0:   column(c6, 2, 1).
 0:   column(c5, 3, 7).
 0:   column(c4, 1, 8).
 0:   column(c3, 2, 4).
 0:   column(c2, 3, 5).
 0:   column(c1, 5, 7).
 0:   goal(add).
 0:   next_column(c7, c8).
 0:   next_column(c6, c7).
 0:   next_column(c5, c6).
 0:   next_column(c4, c5).
```

Figure 6: The Contents of Working Memory During Processing

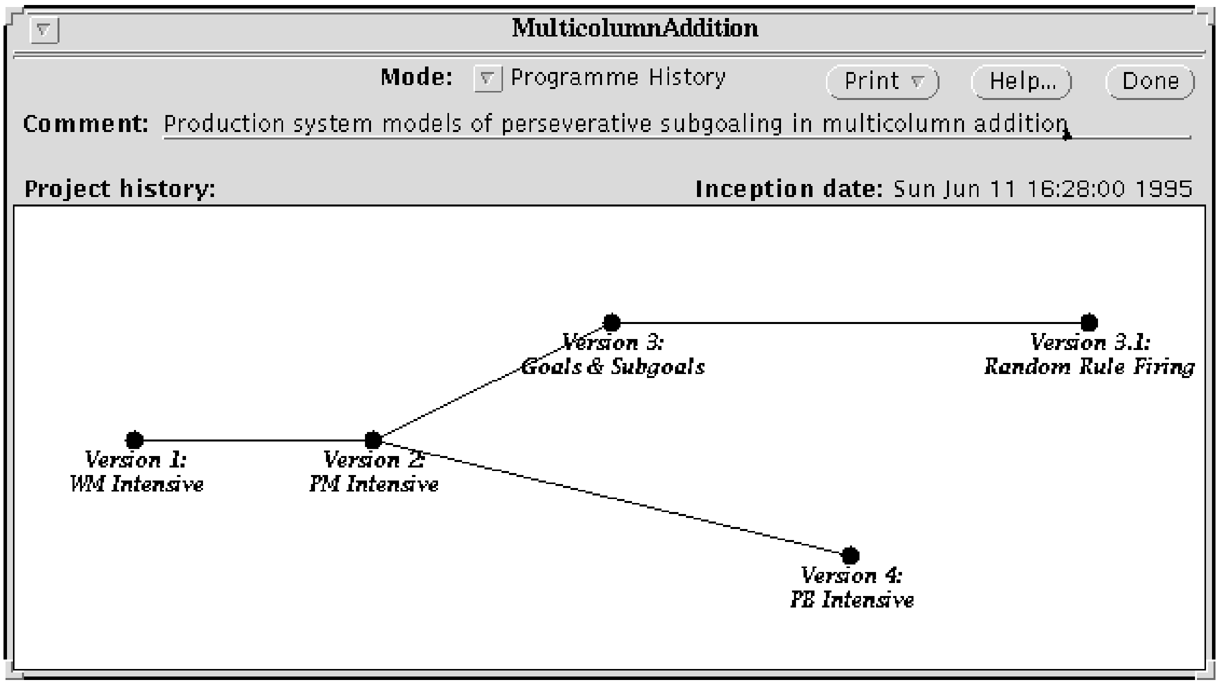

MulticolumnAddition
Mode: Programme History
Print
Help...
Done
Comment: Production system models of perseverative subgoaling in multicolumn addition
Project history:
Inception date: Sun Jun 11 16:28:00 1995
Version 1: WM Intensive
Version 2: PM Intensive
Version 3: Goals & Subgoals
Version 3.1: Random Rule Firing
Version 4: PB Intensive

Figure 7: A Research Programme Consisting of Five Models

"root" model (the first model in the programme), and each other model has a parent model (i.e., the model on which it is based). Multiple models may share a parent, allowing tree-structured programmes. The underlying model of research is that as a programme develops, a number of models will be explored, with possibly many variants on any single model being tested.

This concept of a research programme is supported by COGENT's Research Programme Manager, which provides file management facilities for organising research programmes and a research programme viewer. The viewer (see Figure 7 for an example) provides a graphical depiction of a research programme in terms of a family tree. Each node in the tree corresponds to a model, with the left/right axis being used to represent time. Thus, in Figure 7 the horizontal spacing indicates that work on "Version 4: PB Intensive" began before work on "Version 3.1: Random Rule Firing". This graphical representation provides a simple way of depicting the complexity (or otherwise) of particular research programmes, and the place of individual models within research programmes. Double-clicking on any node in the diagram invokes the above described box/arrow viewing and editing facilities. Individual models may also be copied or deleted.

3. Modelling Allen Inference: An Extended Example

In this section we provide a more detailed example of the use of COGENT — a model of performance on the Allen inference task as discussed by Berendt & Schlieder (1997). The purpose of this discussion is threefold. Firstly, it provides a concrete demonstration of how COGENT can be used to implement a model that was originally developed by other researchers (Berendt, 1996). Secondly, it links COGENT to models of reasoning (in general) and other work in this volume (in particular). Thirdly, it demonstrates COGENT's flexibility — our initial implementation of the model lead to the development of a new object type (an analogue buffer) which was then incorporated into COGENT's class hierarchy and which is used in the version of the model presented here.

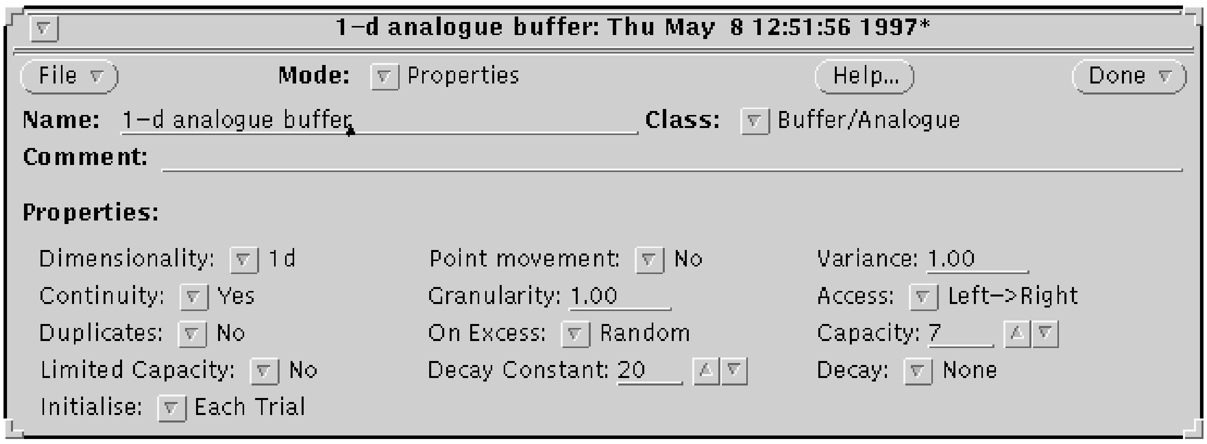
3.1 Analogue Buffers

Although the standard buffer type in COGENT is versatile enough for a wide range of models, it does not discriminate between different types of content. Any well-formed Prolog terms can serve as content, and buffer properties such as decay and capacity limitations apply indiscriminately to all types of content. However, there are occasions when psychologists wish to model buffers with content-specific properties.

For example, theories of "mental imagery" typically postulate a specialised imagery store, with properties which apply on the level of metrical representations. There are suggestions (Berendt, 1996) that one such property of the human imagery buffer is a lack of precision in representing location. As a step towards handling such types of models, we have begun developing a specialised buffer type for imagery.

In its present form, the *analogue buffer* is a specialisation of the normal buffer type, which we can call a *propositional buffer*. Analogue buffers impose restrictions on the types of content which they can contain — the only permitted content items are drawn from a restricted set of Prolog structures which specify *graphical objects*, such as points, lines, polygons and circles, in terms of Cartesian co-ordinates. This restriction ensures that each object has a coherent geometrical interpretation, so the contents of the buffer, taken together, specify a picture, which can be displayed using a built-in viewer. It should be emphasised that there is no "surface" or bitmapped representation like that assumed by Kosslyn (1980), but rather the graphical representation is specified and addressable in terms of significant units. So if conventional buffer decay is enabled, whole objects are lost at once, rather than regions which cut across objects indiscriminately.

Figure 8 shows the Properties view of an analogue buffer. Analogue buffers have all the configurable properties of propositional buffers, as well as *dimensionality* (presently one-dimensional and two-dimensional buffers are supported), *continuity* (which determines whether points can be represented with arbitrary precision or not) and *granularity* (which specifies grain size when continuity is not selected). In addition to these, another two properties permit a kind of imagistic decay, which we can simply call *point movement*. When point movement is selected, points can move on each cycle. Point movement is random but constrained by a *variance* parameter which specifies a normal probability distribution, so if a value of 1 is specified for variance, there is a 0.68 probability that any given point location will not deviate by more than 1 scale unit on each cycle. Finally, there are extra access options (in addition to the usual FIFO, LIFO and Random access options) which permit retrieval of items in orders based on



1-d analogue buffer: Thu May 8 12:51:56 1997*

Mode: Properties

Name: 1-d analogue buffer

Class: Buffer/Analogue

Comment:

Properties:

Dimensionality: 1d	Point movement: No	Variance: 1.00
Continuity: Yes	Granularity: 1.00	Access: Left->Right
Duplicates: No	On Excess: Random	Capacity: 7
Limited Capacity: No	Decay Constant: 20	Decay: None
Initialise: Each Trial		

Figure 8: Analogue Buffer Properties

their spatial layout: these are Top → Bottom, Bottom → Top, Left → Right and Right → Left. So when Left → Right is selected, the leftmost object is retrieved first, and the rightmost last.

At present the only permissible object types are points, lines, text and markers (in either one-dimensional or two-dimensional buffers), intervals (one-dimensional only), polygons, circles and boxes (two-dimensional only). Each type is represented as a Prolog structure, with two arguments, a name and a geometric specification. In keeping with the restriction that analogue buffer contents must have a graphical interpretation, a special graphical viewer is provided, which depicts the buffer contents pictorially, with a key based on the name fields of the objects in the buffer.

3.1.1 Allen Inferences

A very simple application of COGENT's analogue buffer facility uses a one-dimensional buffer to implement Berendt's model of Allen inferences using metrical representations (Berendt, 1996; Berendt & Schlieder, 1997).

To explain the Allen inference task briefly, participants are presented with inference problems comprising two sentences. Each sentence relates two intervals, one of which is mentioned in both sentences, so the participant's goal is to produce a conclusion about the relation between the remaining two intervals. It is, then, a three-term series task, similar to syllogistic reasoning, but since there is no requirement to produce a conclusion which holds in all possible models of the premises, there is no need for a model revision process as proposed by Johnson-Laird (1983). The set of possible Allen relation types is shown in Figure 9.

For half of the problems, there is only one valid conclusion; however, for the remainder, any of several different configurations of intervals are possible. As Knauff *et al.* (1995) observe, human reasoners show very specific preferences, usually for only one of the possible conclusions to these problems. If one assumes that human reasoners solve these problems via the construction of some kind of mental model (as is assumed here), then this places a strong constraint on the set of acceptable models. Moreover, although many problems can be related by symmetry

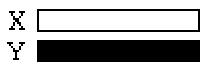
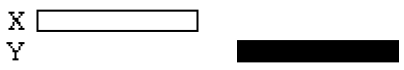
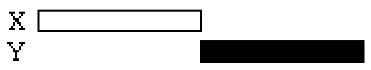
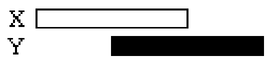
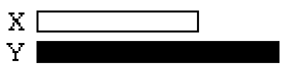
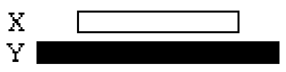
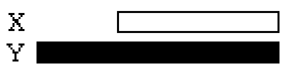
X equals Y	
X before Y	
X meets Y	
X overlaps Y	
X begins Y	
X during Y	
X ends Y	

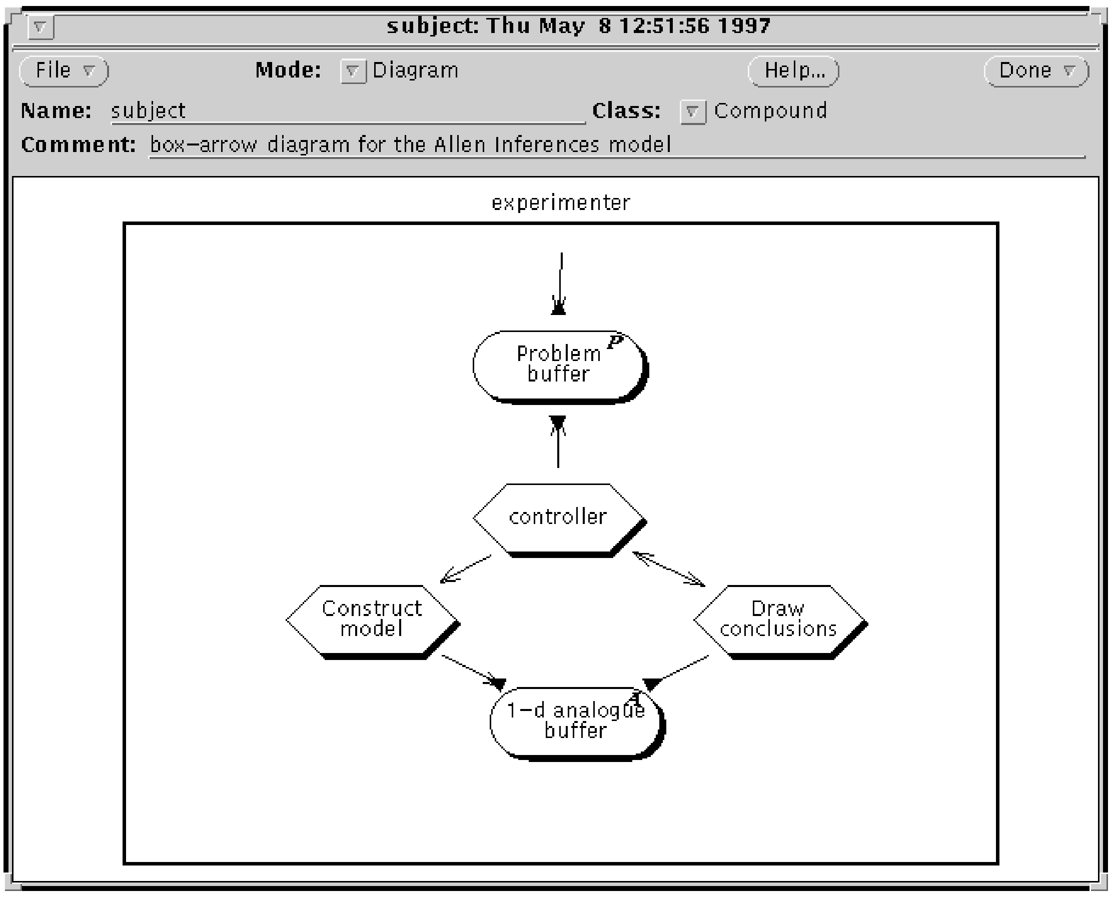
Figure 9: The Set of Allen Relations

transformations, human reasoners' responses do not reflect these symmetries. These asymmetry results rule out a class of simple, length-parameterised metrical models, and entail that the model-construction process must be sensitive to the order in which it employs the premises — the representation of a given relation occurring as the second premise will be different from the representation of the same relation occurring as the first premise.

Berendt's solution to the problem takes as its starting point the assumption that the mental imagery system cannot represent position accurately. Reasoners are assumed to be aware of this limitation, so they use strategies to circumvent it, for example, avoiding (where possible) models which represent points as having identical position. By making small length adjustments when representing intervals, it is possible to construct models which are robust with respect to small changes in point location. Since the length adjustments must be sensitive to the pre-existing contents of the imagery store, relations are represented differently if they occur as the second premise than if they occur as the first, so the model accounts for the empirical asymmetry results in a natural fashion, giving it a high degree of both descriptive and explanatory adequacy.

3.1.2 The COGENT Implementation

The COGENT implementation of Berendt's metrical model of Allen inferencing uses the analogue buffer facilities described above to clarify and simplify the processes assumed to be operating on the imagery store. In order to implement the model, we developed a set of symbolic rules to represent the Allen relations as configurations of metrical intervals in a one-


subject: Thu May 8 12:51:56 1997

Mode: Diagram

Name: subject

Class: Compound

Comment: box–arrow diagram for the Allen Inferences model

Figure 10: Box/Arrow Diagram for the Allen Inferences Model

dimensional analogue buffer. A further set of rules reads a conclusion off from the resulting total configuration. Since the order in which relations are constructed is an important variable, model construction must be implemented as a serial process, so there is a third set of rules to schedule the construction and inspection of models.

Figure 10 shows the COGENT box/arrow diagram for the inferencing model. It consists of two buffers (one analogue buffer for the metrical image, and one propositional buffer for linguistic representations of premises and conclusions), as well as three distinct processes (a controller, and two slave systems for constructing and interrogating the metrical image).

After giving an overview of the time-course of problem solution below, we present details of the construction of representations in the analogue buffer, and the drawing of conclusions on the basis of the representation.

3.1.3 Overview of Processing within the Model

A trial begins when the problem sentences are input to the *Problem buffer*. The *Problem buffer* can be considered as the simulation of a sheet of paper, containing the premises, and eventually the conclusion of the problem.

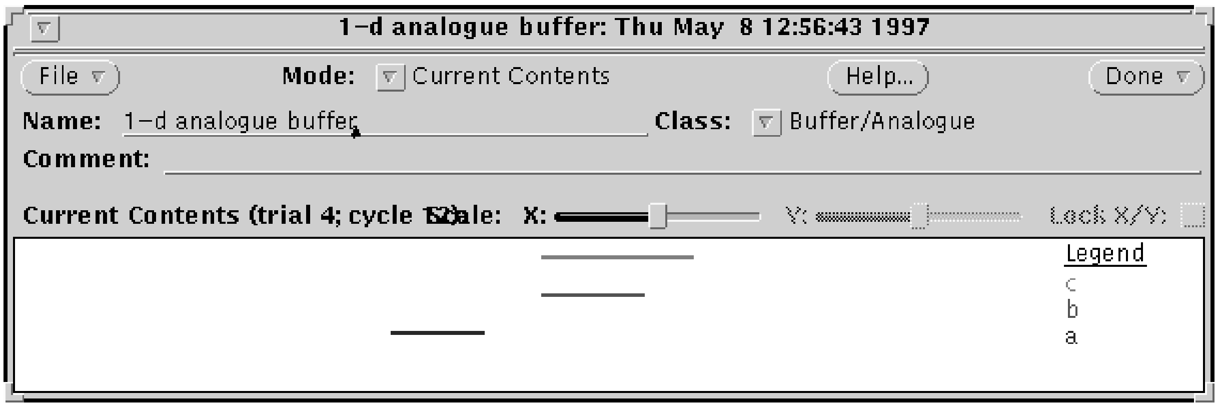
The *Controller* has read/write access to the *Problem buffer*, and waits for the premise pair to appear. When this occurs, a refracted rule (i.e., a rule which fires just once for any instantiation of its variables) annotates the premises, builds a model construction plan and triggers the *Construct model* process to initialise the *1-d analogue buffer*. The process also recurrently triggers itself (by sending itself a message) with the model construction plan. On subsequent cycles the *Controller* reads each annotated premise in turn according to the plan, and sends the premise to the *Construct model* process for integration into the model.

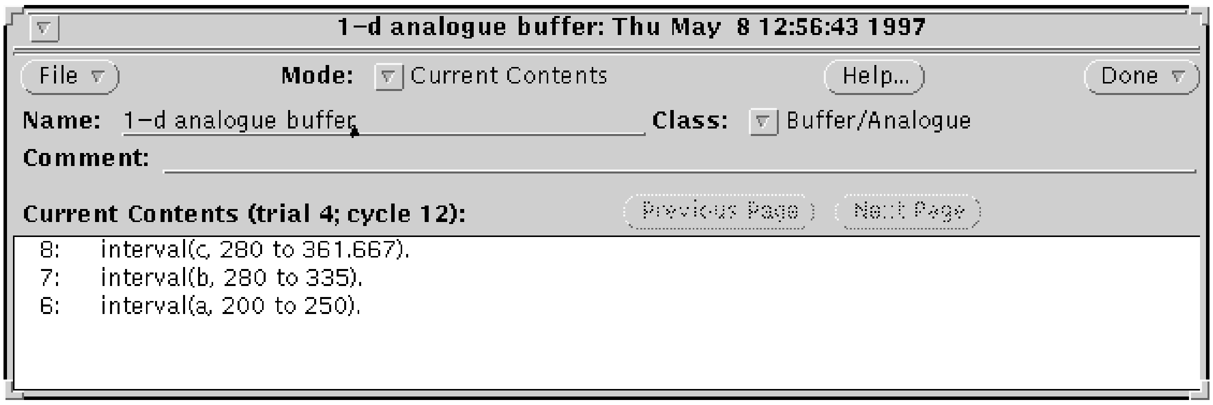
The *Construct model* process contains rules for each forward and inverse Allen relation. To integrate a premise, it selects the interval which forms the subject of the premise and reads the corresponding interval specification from the analogue buffer. It then computes the position and length of the predicate interval, and adds (a representation of) this interval to the *1-d analogue buffer*. The initialisation rule (which is triggered before integration of the first premise) ensures that the presupposition of this process — that the subject interval for the premise is already in the buffer — is satisfied.

Figure 11 shows a graphical view of the contents of *1-d analogue buffer* after both the premises *A before B, B begins C* have been integrated into the model. The underlying representation is shown underneath, illustrating how graphical objects, specified as Prolog structures, are interpreted graphically. Note the COGENT interface uses colour to allow the three intervals to be distinguished.

Because the *Controller* module is operating in parallel according to a construction plan, and the *Construct model* process takes a single cycle to integrate each premise, the *Controller* can predict when the model will be complete and when it should trigger the *Draw conclusion* process (i.e., there is no need for an acknowledgement message from the *Construct model* process back to the *Controller*). Thus, on the cycle after the second premise is sent to the construction process, the *Controller* triggers a conclusion-drawing process to read off any valid conclusions from the completed model.

Like the construction process, the *Draw conclusions* process contains a rule for each distinct conclusion type. These test for satisfaction of truth conditions relating to point ordering. Each candidate conclusion is tested in parallel, and any rules whose truth conditions are satisfied send the appropriate conclusion back to the controller module.


1-d analogue buffer: Thu May 8 12:56:43 1997
File
Mode: Current Contents
Help...
Done
Name: 1-d analogue buffer
Class: Buffer/Analogue
Comment:
Current Contents (trial 4; cycle 12)
Scale: X:
Y:
Lock X/Y:
Legend
c
b
a


1-d analogue buffer: Thu May 8 12:56:43 1997
File
Mode: Current Contents
Help...
Done
Name: 1-d analogue buffer
Class: Buffer/Analogue
Comment:
Current Contents (trial 4; cycle 12):
Previous Page
Next Page

```
8:   interval(c, 280 to 361.667).
7:   interval(b, 280 to 335).
6:   interval(a, 200 to 250).
```

Figure 11: Graphical (above) and Textual (below) Views of Analogue Buffer Contents

On receiving any conclusions, the *Controller* writes them to the *Problem buffer*. At this point there are no outstanding triggers, so the model halts.

3.1.4 Model Construction and Inspection Details

To illustrate the use of the analogue buffer, it is worthwhile going into some detail about how the model construction and conclusion drawing processes operate. Figure 12 shows one of the model construction rules. This rule constructs a representation for the "overlaps" relation.

The rule is triggered by receipt of a message consisting of a Prolog structure, `premiss/2`, whose arguments are a list representing the premise and a number representing the construction number (i.e., first or second premise). This number is used for calculating the length adjustment, `epsilon`. When triggered, the antecedent conditions are tested; if they satisfy (in

```
TRIGGER:  premise([X, normal(overlaps), Y], Construction)
IF:       interval(X, Sx to Ex) is in 1-d analogue buffer
          Length is Ex - Sx
          delta_l(Delta)
          epsilon(Construction, E)
          Sy is Sx + Delta + E
          Ey is Sy + Length
THEN:     add interval(Y, Sy to Ey) to 1-d analogue buffer
```

Figure 12: The Construction Rule for *X overlaps Y*

this example they always will), then the production fires, and executes its consequent actions.

The first condition retrieves the interval representation corresponding to the grammatical subject of the premise from the analogue buffer, incidentally instantiating its start- and end-point, `Sx` and `Ex`. These are used to calculate the interval's length, then the `delta_l` constant is retrieved, specifying the distance between the subject and object intervals' start-points. This must be adjusted with the parameter `epsilon`, which is calculated on the basis of the construction number. Having retrieved all relevant parameters, the start- and end-points are calculated, exhausting the conditions. The consequent action adds the new interval, corresponding to the grammatical object of the premise, to the analogue buffer.

Figure 13 shows a conclusion-drawing rule. This rule draws the conclusion *X overlaps Y* if its conditions are satisfied. When triggered with the appropriate trigger pattern, it retrieves the interval representations for `X` and `Y` from the analogue buffer, then tests for the appropriate relations among the intervals' start- and end-points. If these conditions are satisfied, the conclusion is sent to the controller.

It should be evident that minor modifications of each of these rule types suffice to cover all the possible Allen relations, both in the model construction and conclusion drawing processes.

3.1.5 The effect of point movement

Provided the analogue buffer property controlling Point Movement is switched off, the model produces valid conclusions for almost all possible pairs of Allen relations, and the conclusions it produces are the ones that Berendt's model predicts; however, if Point Movement is switched on, the range of valid conclusions produced by the model increases, and more invalid conclusions are also produced, depending on the setting of the Variance parameter.

For example, given the premises *A before B, B before-inverse C*, the model will ordinarily produce only the conclusion *A overlaps-inverse C*. But with Point Movement enabled, the model will sometimes produce alternative conclusions: for example, in a sample of 15 trials with these premises, the model produced *A overlaps-inverse C* 12 times, and the alternative valid conclusion *A during C* 3 times. This illustrates how Berendt's model is robust with respect to point movement, since the overwhelming majority of conclusions were still the predicted ones, but also that occasionally the amount of degradation of the metrical representation is great enough to result in an alternative conclusion.

Unfortunately, space limitations preclude a more thorough analysis of the model's sensitivity to Point Movement and Variance here; see Berendt & Schlieder (1997) for more discussion of the subject.

```
TRIGGER:  conclusion([X, normal(overlaps), Y])
IF:       interval(X, Sx to Ex) is in 1-d analogue buffer
          interval(Y, Sy to Ey) is in 1-d analogue buffer
          Sx < Sy
          Sy < Ex
          Ex < Ey
THEN:     send conclusion([X, normal(overlaps), Y]) to controller
```

Figure 13: Rule for Drawing the Conclusion *X overlaps Y*

3.1.6 Discussion of the Allen Inferences Model

In its present form the model implements Berendt's (1996) metrical algorithm quite straightforwardly, for a task in which natural language conclusions are drawn on the basis of natural language presentation of premise pairs. This is similar to the task employed by Knauff *et al.* (1995), except that in that task participants were required to specify their conclusions using a graphical computer interface.

The present implementation draws natural language conclusions since it was constructed with a view to modifying the model to investigate "Figural effects" in the Allen inference task, by analogy with the syllogistic reasoning task (Johnson-Laird, 1983, Johnson-Laird & Bara, 1984, Stenning & Yule, in press). In the Knauff *et al.* paradigm, premises are always presented with the same term arrangement (or "Figure") in the premises — notated as ab/bc, where the terms a, b and c denote the three intervals — whereas in the syllogistic reasoning task, four different term arrangements can be used — ab/bc, ba/cb, ab/cb and ba/bc. These Figures are known to affect human reasoners' preferences among valid conclusions in the syllogistic reasoning task — this is known as the Figural effect (Johnson-Laird & Bara, 1984).

It is possible that if Allen inference problems were presented using these Figures that similar effects would be observed. There is a straightforward mapping between the set of problems used by Knauff *et al.* and the set generated from the uninverted Allen relations and the four Figures: inverting term order in a premise corresponds to transforming the semantics of the relation from normal to inverse. In terms of the COGENT model, premise integration must be modified by changing the rules in the *Controller* process to be sensitive to the four possible term arrangements in the premises, permitting integration of a semantic inverse of the premise where term order is inverted, but also, permitting integration of the second premise first when appropriate. This manipulation permits new predictions of the model to be derived and tested.

We are currently investigating human performance on a Figural variant of the Allen inferences task, in tandem with the development of appropriate COGENT models.

3.1.7 Other Applications of Analogue Buffers

As well as the Allen Inferences model described here, analogue buffers can be used to model a wide variety of reasoning and mental imagery processes. We anticipate that the one-dimensional analogue buffer will be useful for modelling a range of temporal reasoning and seriation tasks. The two-dimensional buffer type has already been used for a model of mental rotation of random two-dimensional polygons (cf. Cooper, 1975), and for a model of syllogistic reasoning using Euler Circles (cf. Stenning & Yule, in press).

The analogue buffer is a recent addition to the COGENT object hierarchy, and we anticipate that the range of analogue buffer functionality will be substantially increased in the near future; as well as adding more graphical object types, for example icons, we intend to introduce configurable limits on image size, and modify the handling of buffer access to give greater flexibility in directional scanning. Moreover, it is likely that we will introduce a more general set of configurable buffer properties governing activation levels for buffer contents. Used in an analogue buffer, this should permit construction of imagery models using "fading" decay, a common feature of theories of human imagery processes (e.g. Kosslyn, 1980).

4. Discussion

The previous sections have described COGENT as a piece of technology and shown how it can be used to implement a model of reasoning. We now return to general issues concerning methodology, COGENT's role within computational modelling, and our future plans for COGENT development.

4.1 Methodology

A central theme of work on COGENT has been support for a systematic methodology within cognitive modelling (Cooper *et al.*, 1996). In contrast to empirical psychological work, models are frequently published in a poorly specified form (and hence are not replicable), bear no strict correspondence to the cognitive theory which they are intended to implement, and hide numerous potentially influential implementation decisions. COGENT embodies several methodological principles which alleviate these problems.

Firstly, COGENT achieves a degree of methodological rigour through the strict one-to-one mapping between COGENT boxes and functional models. The graphical interface ensures that a strict correspondence is maintained between the theoretically justified box/arrow diagram and the executable code (which COGENT itself generates). At the box/arrow level, all functional elements must be made fully explicit and there is no scope for hiding implementation detail.

Secondly, by providing a range of standard object classes, COGENT systematises the definitions of standard boxes (e.g., buffers, rule-based processes, networks). This standardisation assists communication by preventing ambiguity and vagueness in specification. At the same time, instances of the standard object classes may be configured for individual applications by a range of parameters which fully determine their behaviour. Thus, modellers cannot ignore or overlook the range of computational properties possible of an instance of any particular object type.

Thirdly, COGENT allows the systematic exploration of parameter spaces so that any dependencies of behaviour on parameter values can be fully determined. If the value of a parameter is found to affect simulated behaviour then clearly that parameter must have some theoretical import. Only if behaviour is independent of a parameter's value can the parameter be said to be irrelevant.

Finally, COGENT provides some support for the development of computational research programmes (paralleling the empirical research programmes common in current cognitive psychology), whereby sequences of successively more adequate models can be explored and developed.

4.2 Environments and Architectures

As emphasised in the introduction, COGENT is not an architecture like Soar or ACT-R. It does not, for example, embody any specific assumptions about control processes (see Johnson (1997) for a comparison of the architectural control processes in ACT-R and Soar), and it does not contain any pre-specified learning mechanisms. Thus COGENT imposes minimal theoretical constraints on the modeller. Proponents of architectures have argued that such theoretical constraints are necessary, and furthermore that they are one of the great strengths of the architectural approach (see, for example, Newell, 1990). We are not wholly convinced by these arguments. While there is a place for a well-defined research programme investigating possible architectures of cognition, it frequently appears that architectures are used primarily as programming environments solely in order to lend credibility to an otherwise unrelated cognitive model. The use of an architecture solely for programmatic support is not in itself dangerous, but it promotes the (false) beliefs that a) a model implemented in the architecture is somehow superior to one which is not, even if the two models are based upon the same psychological assumptions; and b) a model implemented in an architecture somehow adds support to the psychological pretensions of that architecture. These issues are discussed in more detail in Cooper & Shallice (1995).

COGENT is not susceptible to these arguments. A model implemented in COGENT truly *is* superior to an equivalent mode which is implemented in some other technology precisely because of COGENT's preoccupation with methodological concerns. And a model implemented in COGENT only adds to COGENT's psychological pretensions to the extent that any model stated in box/arrow terms adds to the psychological pretensions of the box/arrow notation. As a general cognitive modelling environment, COGENT does not embody any computational constraints over and above those embodied in the box/arrow notation itself. Our approach is to provide optional facilities, such as the various buffer properties, which facilitate exploration of alternative assumptions about a given theory's implementation, rather than to impose any overarching general theory of cognition upon the theorist.

A model implemented within COGENT gains clarity and succinctness (through the use of the familiar box/arrow notation supplemented with object-oriented concepts and a computationally sound operational semantics) and methodological rigour (through the requirement of a one-to-one mapping between components of box/arrow diagrams and computational mechanisms and the support for computational experiments and research programmes). Thus, COGENT is most appropriate for the development of cognitive models which are not clearly rooted in any specific architecture (either because the model is conceived of independently of any particular architecture or because the modeller is not willing to accept any existing architecture in full). The range of models which have to date been implemented within COGENT — including models of child memory (Barreau, 1997), multi-column addition (Cooper, 1996), medical diagnosis (Fox Cooper, in press; Cooper & Fox, 1997), as well as the reasoning models discussed above — are testament to this position.

4.3 Future Directions

Development work on COGENT is progressing in three main areas: models; methodology; and software.

Much of COGENT's development has been driven by generalising the needs of particular models. The various buffer properties, for example, have arisen through generalisations of buffers required by models of child memory, multi-column arithmetic and decision making, and analogue buffers were introduced when we turned our attention to models of imagery-related tasks. We are actively developing further models (primarily in the areas of reasoning and decision making) to provide motivation and support for further system enhancements.

It is anticipated that further methodological support will be incorporated into COGENT at both the level of individual models and the level of research programmes. At the level of individual models, it is possible to treat the experimental environment as a separate compound box. This allows greater control over the presentation of stimuli and the collection of data from the model in question. We intend to provide further support for this approach in terms of 1) extensions to the scripting language facilities so that computational experiments can be specified in the same terms as standard laboratory based experiments (i.e., by specifying trial order, block order, stimulus randomisation, etc.); 2) extensions to the capabilities of data sinks so that output can be interactively tabulated, graphed and analysed; and, ultimately, 3) capabilities for replacing models by real subjects so that the same experimental environment can be used to collect and maintain both human and computational data. Our goal here is to allow the same input data files to be used for both human and computational experiments, and for both types of experiment to generate output files which may be automatically compared and analysed.

This will provide methodological support at the level of research programmes by ensuring a close relation between computational and laboratory work.

Finally, development of the COGENT software is continuing in order to extend the potential user base. The version of COGENT described and shown here is implemented using the XView widget set and requires a Sun Workstation or similar machine running UNIX/X windows (COGENT is also available for the LINUX operating system.) A port of the software to the Microsoft® Windows™ environment is underway, and versions compatible with Windows 3.*x* and Windows 95/NT will become available shortly.

5. References

Anderson, J.R. (1993). *Rules of the Mind*. Hillsdale, NJ., Lawrence Erlbaum Associates.

Barreau, S. (1997). *Developmental Constraints on a Theory of Memory*. PhD Thesis. Department of Psychology. University College London.

Berendt, B. (1996). Explaining preferred mental models in Allen inferences with a metrical model of imagery. In G. W. Cottrell (ed.) *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. San Diego, CA. pp. 489–494.

Berendt, B. & Schlieder, C. (1997). Mental model construction in spatial reasoning: A comparison of two computational theories. This volume.

Cooper, L.A. (1975). Mental rotation of random two-dimensional shapes. *Cognitive Psychology*, **7,** 20-43.

Cooper, R. (1995). Towards an object oriented language for cognitive modelling. In J.D. Moore & J.F. Lehman (eds.) *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. Pittsburgh, PA. pp. 556–561.

Cooper, R. (1996). Perseverative subgoaling in production system models of problem solving. In G.W. Cottrell (ed.) *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. San Diego, CA. pp 396–402.

Cooper, R. & Fox, J. (1997). Learning to make decisions under uncertainty: The contribution of qualitative reasoning. To appear in *Proceedings of the 19th Annual Conference of the Cognitive Science Society*. San Fransisco, CA.

Cooper, R. & Fox, J. (in submission). COGENT: A visual design environment for cognitive modelling.

Cooper, R., Fox, J., Farringdon, J. & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence*, **85,** 3–44.

Cooper, R. & Shallice, T. (1995). Soar and the case for Unified Theories of Cognition. *Cognition*, **55**, 115–149.

Fox, J. & Cooper, R. (In press). Cognitive processing and knowledge representation in decision making. In R.W. Scholtz & A.C. Zimmer (eds.), *Qualitative Aspects of Decision Making*. Lengerich, Germany, Pabst Science Publishers.

Johnson, T. (1997). A comparison of ACT-R and Soar. This volume.

Johnson-Laird, P.N. (1983). *Mental Models*. Cambridge, UK, Cambridge University Press.

Johnson-Laird, P.N. & Bara, B. (1984). Syllogistic Inference. *Cognition*, **16**, 1–61.

Knauff, M., Rauh, R. & Schlieder, C. (1995). Preferred mental models in qualitative spatial reasoning: A cognitive assessment of Allen's Calculus. In J.D. Moore & J.F. Lehman (eds.) *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. Pittsburgh, PA. pp. 200–205.

Kosslyn, S.M. (1980). *Image and Mind*. Cambridge, MA., Harvard University Press.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA., Harvard University Press.

Stenning, K. & Yule, P. (in press) Image and language in human reasoning: a syllogistic illustration. *Cognitive Psychology*.