

H24/3300

MONASH UNIVERSITY
THESIS ACCEPTED IN SATISFACTION OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ON..... 20 December 2002

.....

for Sec. Research Graduate School Committee

Under the copyright Act 1968, this thesis must be used only under the normal conditions of scholarly fair dealing for the purposes of research, criticism or review. In particular no results or conclusions should be extracted from it, nor should it be copied or closely paraphrased in whole or in part without the written consent of the author. Proper written acknowledgement should be made for any assistance obtained from this thesis.

**Encoding and Parsing of
Algebraic Expressions by
Experienced Users of Mathematics**

**Anthony Robert Jansen
B.Sc., B.Comp.(Hons.)**

School of Computer Science and Software Engineering
Monash University

SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

JANUARY, 2002

Contents

Abstract	xiii
Declaration	xv
Acknowledgements	xvi
Publications Arising from this Thesis	xvii
1 General Introduction	1
1.1 Notations	2
1.1.1 Natural Language	2
1.1.2 Writing Systems	8
1.1.3 Visual Languages	10
1.1.4 Mathematical Notations	14
1.2 Language Processing	18
1.2.1 Processing Natural Language	18
1.2.2 Processing Mathematical Notations	22
1.3 Motivation for the Thesis	24
2 Encoding of Algebraic Expressions	26
2.1 Syntactic Encoding of Algebraic Expressions	27
Experiment 1	28
Method	28
Results and Discussion	31
2.2 The Role of Visual Processes	33

Experiment 2	37
Method	37
Results and Discussion	40
2.3 Masked and Visible Priming	43
2.4 The Role of Lexical Processes	45
Experiment 3	45
Method	45
Results and Discussion	48
2.5 The Role of Post-Lexical Processes	49
Experiment 4	50
Method	50
Results and Discussion	51
2.6 Syntactic Encoding with Phrasal Structure	53
Experiment 5	55
Method	55
Results and Discussion	57
2.7 Summary	58
3 Restricted Focus Viewer	62
3.1 Restricted Focus Viewer Tool	65
3.1.1 Description of the RFV	66
3.1.2 Data Replayer	70
3.2 A Qualitative Comparison with Eye-Tracking	71
Experiment 6	71
Method	71
Results and Discussion	75
3.3 Validation of the RFV	78
Results and Discussion	81

<i>CONTENTS</i>	iii
3.4 Summary	85
4 Parsing of Algebraic Expressions	86
4.1 Processing Expressions a Piece at a Time	87
Experiment 7	92
Method	92
Results and Discussion	97
4.2 Markov Chains and Model Analysis	102
4.2.1 Transition Data and Markov Chains	104
4.2.2 Scanning Models	105
4.2.3 Model Parameters and Maximum Likelihood	109
4.2.4 Comparing Models	112
4.3 Developing Scanning Models	114
Experiment 8	114
Method	114
Results and Discussion	118
4.4 Verifying the Scanning Models	137
Experiment 9	139
Method	139
Results and Discussion	141
4.5 Summary	148
5 General Discussion	151
5.1 Implications	154
5.2 Future Work	156
5.3 Conclusion	156
A Transition Data for Experiment 8	158

B Transition Data for Experiment 9	167
C Restricted Focus Viewer	
Version 2.1	
User's Manual and Tutorial	174
C.1 System Requirements	175
C.1.1 Java 2	175
C.1.2 Program and Source Files	175
C.2 Description of the RFV	175
C.2.1 The Focus Window	175
C.2.2 Stimulus Images and Blurring	176
C.2.3 Motion Blur	178
C.2.4 Guidelines for Setting Parameters	178
C.3 RFV Input File	179
C.4 Running the RFV	179
C.4.1 RFV Errors	180
C.4.2 Guidelines for Running the RFV	180
C.4.3 Participant Interaction with the RFV	182
C.5 Tutorial	182
C.5.1 The Visual Elements	182
C.5.2 Layout of Visual Elements	193
C.5.3 Feedback and Time Limits	205
C.6 RFV Output Data File	211
C.7 The Replayer	212
C.7.1 Running the Replayer	213
C.7.2 Keyboard Commands	215
C.8 Input File Parameter Specifications	217
References	226

List of Figures

1.1	Parse tree for the sentence <i>The man will wash the red car.</i>	5
1.2	Two valid parse trees for the sentence <i>The boy saw the man with the telescope.</i>	7
1.3	Valid examples of two visual languages: music notation and chemical structural formulae.	10
1.4	Invalid examples of two visual languages: music notation and chemical structural formulae.	11
1.5	Example of a graph grammar for describing the structural formulae of single chain hydrocarbons (left), and an example derivation (right).	12
1.6	Example of an electronic circuit diagram of a Practical Voltage Source (left) and its corresponding production rule from an attributed multiset grammar (right).	13
1.7	Examples of stimuli used by Hayes (1973).	22
2.1	Examples of the laws of perceptual organization: (A) the law of similarity, (B) the law of proximity, (C) the law of closure, (D) the law of good continuation, and (E) the law of familiarity.	34
2.2	Parse tree for $3x + 7(8 - 6y)$	54
3.1	Example of the type of problem used by Suppes et al. (1983).	63
3.2	Example of a visual stimulus and its corresponding blurred image.	66
3.3	Regions of the stimulus used to achieve the graded blurring effect.	67
3.4	Two examples of the focus window on different regions of the stimulus.	68

3.5	Example of a data replayer scan-path.	70
3.6	Examples of eye-tracking scan-path data which shows good calibration of the eye-tracking equipment.	76
3.7	Examples of eye-tracking scan-path data which shows bad calibration of the eye-tracking equipment.	77
3.8	Examples of RFV scan-path data on the same algebraic expressions shown in Figures 3.6 and 3.7.	78
3.9	The two pulley systems used.	79
3.10	Two examples of the focus window on different regions of pulley system 1.	80
3.11	Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the proportion of errors at different causal chain positions.	81
3.12	Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the mean response times for different trial types.	82
3.13	Example of an RFV fixation protocol for the statement <i>The lower pulley turns counterclockwise</i>	84
3.14	Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the breakdown of gaze duration on different components of the pulley systems.	84
4.1	Parse tree for the expression $3y^2 + 7x$	88
4.2	Grammatical units for the algebraic notation considered in this thesis, and the predicted order in which the elements of the units are scanned.	91
4.3	Mean viewing time per character for each sub-expression in Experiment 7, as a function of sub-expression type and presentation order.	100
4.4	Examples of the focus window over two different regions of the expression shown on the left.	117

4.5	Example RFV scanpaths for two of the algebraic expressions used in Experiment 8.	119
4.6	Example RFV fixation protocols for the two scanpaths shown in Figure 4.5.	119
4.7	Parse tree for the expression $7(x^4 + 2z)^2 - 3$	134
4.8	Parse trees for $x + 3 + y$	145
4.9	Example RFV scanpaths from two participants over an expression with a double summation used in Experiment 9.	147
4.10	Example RFV scanpaths from two participants over an expression with a double integral used in Experiment 9.	148
C.1	Regions of the stimulus used to achieve the graded blurring effect.	176
C.2	Example of a visual stimulus and its corresponding blurred image.	176
C.3	An example of the five images needed to present a stimulus.	177
C.4	Two examples of the focus window on different regions of the stimulus.	177
C.5	The RFV setup window.	181
C.6	Setting up an experiment.	181
C.7	The steps to take in the RFV setup window.	185
C.8	The RFV window after loading the input file block.	186
C.9	The end of experiment block.	187
C.10	A different font and colour for the line of text.	187
C.11	The result of changing the default text font and colour.	190
C.12	An example of a button.	190
C.13	Changing the button colours.	192
C.14	Changing the RFV window background colour.	192
C.15	An example of a static image.	194
C.16	An example of a stimulus with the focus window near the centre.	194
C.17	A block containing a line of text and a static image.	196
C.18	Modifying the distribution of vertical space.	196

C.19 Further modifying the distribution of vertical space.	198
C.20 Using a fixed vertical space in the layout of the elements.	198
C.21 Adding a row to a block.	200
C.22 Manipulating the extra horizontal space in a row.	200
C.23 Further manipulating the horizontal space in a row.	202
C.24 Example of columns defined within rows.	202
C.25 Layout of various rows and columns within the RFV window.	204
C.26 Forcing a gap between the two buttons in the top row.	204
C.27 A simple two choice block.	206
C.28 Feedback for an incorrect response.	208
C.29 Feedback that includes the time taken to respond.	208
C.30 Feedback to indicate that the time limit expired.	210
C.31 Example of the Replayer in Scanpath mode.	213
C.32 The Replayer program's block selection window.	214
C.33 Selecting a block to be replayed.	214

List of Tables

2.1	Example expressions and sub-expressions used in examining syntactic well-formedness.	29
2.2	Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 1.	32
2.3	Example expressions and sub-expressions used in examining visual versus syntactic properties.	39
2.4	Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 2.	40
2.5	Example expressions and primes used in Experiments 3 and 4.	46
2.6	Mean correct response times (ms) and error rates (%) as a function of prime type for Experiment 3	48
2.7	Mean correct response times (ms) and error rates (%) as a function of prime type for Experiment 4	51
2.8	Example expressions and sub-expressions used in examining phrasal properties.	56
2.9	Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 5.	57
3.1	Guidelines for setting RFV parameters.	69
3.2	Examples of the expressions used in Experiment 6, along with their corresponding statements and whether those statements were correct.	72
4.1	Example expressions and sub-expressions used in Experiment 7.	94

4.2	Examples of the expressions and corresponding questions used in Experiment 7, and the correct answers to the questions.	95
4.3	Mean total memorization times (ms) and question error rates (%) as a function of sub-expression type and presentation order for Experiment 7.	98
4.4	Transition table based on the example data using the expression $3x + 7$	105
4.5	Markov Chain based on the transition frequencies from the example data using the expression $3x + 7$	106
4.6	Markov Chain based on a scanning model favouring left-to-right transitions.	106
4.7	Markov Chain based on a left-to-right scanning model that uses parameters.	110
4.8	Predictive ability of the model, M , shown in Table 4.7, given the transition data, D , shown in Table 4.4, for various parameter settings.	111
4.9	The expression form used for each group in Experiment 8, along with example expressions.	115
4.10	Examples of the expressions and corresponding statements used in Experiment 8, and whether those statements were correct.	116
4.11	Mean symbol fixation times (ms) as a function of symbol type for Experiment 8.	120
4.12	The proportion of trials for which the initial symbol (as determined by the syntactic structure) was one of the first three symbol fixations, as a function of expression group for Experiment 8.	122
4.13	Markov Chain for the example expression $8x^2 + 5$, showing the role of the three parameters, α , β and γ , in the syntax based models.	127
4.14	Parameter values for Syntax Models I to IV, inferred using the training data, $D_{training}$, for Experiment 8.	128
4.15	Number of parameters, predictive accuracy and model selection scores for Syntax Models I to IV, for Experiment 8.	129
4.16	Markov Chain for the example expression $9(y^3 + 1)$, showing the role of the four parameters, α , β , δ and ϵ , in the syntax based models.	130

4.17	Parameter values for Syntax Models V to VII, inferred using the training data, $D_{training}$, for Experiment 8.	131
4.18	Number of parameters, predictive accuracy and model selection scores for Syntax Models V to VII, for Experiment 8.	131
4.19	Mean fixation times (ms) for symbols that are the last element of a syntactic constituent (end symbols) and those that are not (non-end symbols), as a function of expression group for Experiment 8.	135
4.20	Mean fixation times (ms) for symbols that are the last element of a syntactic constituent (end symbols) and the non-operator symbols that are not (non-end symbols, excluding operators), as a function of expression group for Experiment 8.	136
4.21	Examples of the expressions used in Experiment 9.	140
4.22	Parameter values for the Experiment 9 models, inferred using the training data, $D_{training}$, along with the values inferred for the corresponding model in Experiment 8.	142
4.23	Number of parameters, predictive accuracy and model selection scores as a function of model type for Experiment 9.	143
A.1	Symbol mapping example for the expression $(x + 3)^2$	158
A.2	Transition Table for Expression Group 1.	159
A.3	Transition Table for Expression Group 2.	160
A.4	Transition Table for Expression Group 3.	161
A.5	Transition Table for Expression Group 4.	162
A.6	Transition Table for Expression Group 5.	163
A.7	Transition Table for Expression Group 6.	164
A.8	Transition Table for Expression Group 7.	165
A.9	Transition Table for Expression Group 8.	166

B.1 Two example transition tables which contain the exact same data, but are presented using different symbol orders. 167

B.2 Transition Table for Expression 1. 168

B.3 Transition Table for Expression 2. 169

B.4 Transition Table for Expression 3. 170

B.5 Transition Table for Expression 4. 171

B.6 Transition Table for Expression 5. 172

B.7 Transition Table for Expression 6. 173

Abstract

Mathematical notations are a way of representing mathematical concepts in written form, and their use has become vital in many fields ranging from Engineering to Commerce. Yet, despite centuries of use, surprisingly little is known about how mathematicians perceive mathematical expressions. In this thesis, the comprehension of algebraic expressions by experienced users of mathematics is investigated. Also of interest are possible similarities with how readers process natural language text.

The first part of the thesis focuses on the encoding of algebraic expressions. Five experiments were constructed, three of which involved memory recognition tasks, with the other two making use of the masked and visible priming paradigms. These experiments examined the role of visual, "lexical" and syntactic processes in the encoding of algebraic expressions. The results of these experiments indicate that the encoding of algebraic expressions by experienced users of mathematics is based primarily on mathematical syntax, with syntactic processes playing a more important role than processes that occur at the visual or "lexical" level. Algebraic expressions are also shown to be encoded into components that represent the phrasal constituents of the expression.

The focus of the thesis then shifts to the parsing of algebraic expressions. To analyse parsing strategies, a tool is needed that can track the visual attention of a person as they are reading such an expression. While eye-tracking equipment has proven useful in tracking visual attention for many types of visual stimuli, it has several limitations which make it unsuitable for use with algebraic expressions. Thus, an alternative tool, the Restricted Focus Viewer (RFV), is introduced.

Three final experiments were then constructed to analyse how experienced users of mathematics parse algebraic expressions. Experiment 7 involved an expression construction task that requires algebraic expressions to be read a piece at a time. Experiments 8 and 9 involve tasks that use the RFV to track the order in which the symbols of an expression are scanned. The results of these experiments indicate that scanning of algebraic

expressions by experienced users of mathematics is also based primarily on mathematical syntax. Among the analysis techniques used to examine the scanning models is Markov Chain model analysis. The use of Markov Chain modelling in concert with the RFV data provided a unique avenue for obtaining a detailed account of the important features for the parsing of algebraic expressions.

Overall, the results of this thesis indicate that the encoding and parsing of algebraic expressions has marked similarities with the way in which sentences of natural language are processed. This suggests that there may be certain core processes that are common to the processing of both algebraic notation and natural language.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university and, to the best of my knowledge and belief it contains no material previously published or written by another person, except where due reference is made.

A solid black rectangular box redacting the signature of the author.

Anthony R. Jansen.

Monash University, 2002.

Acknowledgements

First and foremost I would like to thank my supervisors, Kim Marriott and Greg Yelland whose experience, patience, enthusiasm and encouragement have made this thesis possible. My PhD has been both a rewarding and enjoyable experience and it has been a privilege to work closely with these two exceptional researchers.

I am also grateful for the opportunity of collaborating with Alan Blackwell, and I thank him for the stimulating discussions and his valuable input on the development of the Restricted Focus Viewer. Special thanks also goes to Mary Hegarty for the opportunity of working with her at UCSB, and especially for allowing me to make use of her eye tracking laboratory. Her feedback on the development of the RFV was also very helpful.

To all the participants who generously gave their time to participate in my experiments, I give a big thank you. Thanks especially to all those who regularly helped out by participating in my pilot studies (Torst, Dave, Ian, Ricky, Linda, Bernie and many others).

It has been a privilege during my PhD to attend several international conferences and meet and discuss my work with many different people; I thank them all for their interest and feedback. Also during my travels, I have been fortunate to share the hospitality of many people, and particular thanks must go to Greg Badros, Mary Hegarty, Alan Blackwell and Helen Arnold, and Warwick Harvey.

A crucial aspect of surviving my PhD has been the company and support of my fellow postgrad students. In particular I would like to thank Linda, Iman, Brendan, Toby, Sarah, Stewie, Steve Bethune and Steve Welsh for trying to keep me sane. Thanks also to Claire Byrne, in particular for her help with understanding many of the programs that were needed for analysing my data. Further thanks for social outlets during my PhD go to the coffee crowd (especially the PEGlets) and the volleyball people, among many others (thanks Joel for the chess games, and Warwick for inspiring me to learn to juggle).

Finally, I would like to thank my family, both for supporting my decision to do a PhD, and for their support during my PhD.

Publications Arising from this Thesis

Conference Papers

Jansen, A.R., Marriott, K. & Yelland, G.W. (1999). Perceiving Structure in Mathematical Expressions. In M. Hahn & S. C. Stoness (Eds.), *Proceedings of the Twenty First Annual Conference of the Cognitive Science Society* (pp. 229-233). Lawrence Erlbaum Associates.

Jansen, A.R., Marriott, K. & Yelland, G.W. (2000). Constituent Structure in Mathematical Expressions. In L. R. Gleitman & A. K. Joshi (Eds.), *Proceedings of the Twenty Second Annual Conference of the Cognitive Science Society* (pp. 238-243). Lawrence Erlbaum Associates.

Blackwell, A.F., Jansen, A.R. & Marriott, K. (2000). Restricted Focus Viewer: A Tool for Tracking Visual Attention. In M. Anderson, P. Cheng & V. Haarslev (Eds.), *Theory and Application of Diagrams. Lecture Notes in Artificial Intelligence (LNAI) 1889* (pp. 162-177). Springer-Verlag.

Journal Articles

Jansen, A.R., Marriott, K. & Yelland, G.W. (In Press). Comprehension of Algebraic Expressions by Experienced Users of Mathematics, *Quarterly Journal of Experimental Psychology (Section A)*.

Jansen, A.R., Blackwell, A.F. & Marriott, K. (In Press). A Tool for Tracking Visual Attention: The Restricted Focus Viewer, *Behavior Research Methods, Instruments, & Computers*.

Technical Reports

Jansen, A.R. (2000) *Restricted Focus Viewer (RFV) Version 2.0 User's Manual*. (Tech. Rep. No. 2000/78). School of Computer Science and Software Engineering, Monash University.

Jansen, A.R. (2001) *Restricted Focus Viewer (RFV) Version 2.1 User's Manual and Tutorial*. (Tech. Rep. No. 2001/87). School of Computer Science and Software Engineering, Monash University.

*"Nothing in the history of mathematics is to me so surprising or impressive
as the power it has gained by its notation or language."*

J. W. L. Glaisher, 1915.

Chapter 1

General Introduction

Mathematical notations are a way of representing mathematical concepts in written form. Humans have been using mathematical notations for thousands of years (Cajori, 1928), and as our understanding of mathematical concepts has expanded, so too have the notation systems we use. Over the centuries, these systems have evolved and borrowed from each other to become the mathematical notations that are considered standard today.

Many simple mathematical tasks can be carried out mentally, not requiring the use of written notations. These include everyday tasks such as calculating how much time remains before the next meeting, or how much change should be received when paying for groceries. However, for many of the technologies that are used around the world today, the development and use of mathematical notations has been a necessary precursor. Tasks ranging from calculating the load that can be supported by a bridge beam, to determining if patients who received a specific drug treatment responded significantly better than those given a placebo, would be impossible to carry out without mathematical notations. In fact, it is almost impossible to conceive of entire fields, such as Engineering, Science or Economics, existing as they do today without mathematical notations.

There has been much research that examines mathematical ability, varying in focus from basic numerical skills (for example, see Ashcraft, 1992; Frensch & Geary, 1993; Campbell, 1994) to more complex problem solving (Hegarty, Mayer, & Monk, 1995; Newell & Simon, 1972), and also examining data from patients with cognitive impairments (Jackson

& Warrington, 1986; Clark & Campbell, 1991; McCloskey, Harley, & Sokol, 1991). Yet little is known about how humans perceive and comprehend mathematical expressions. This is surprising given the vital role that mathematical notations play in complex mathematical tasks.

The main aim of this thesis is to explore how experienced users of mathematics encode and parse mathematical expressions, and in particular to examine whether mathematical expressions are processed in a manner similar to the way in which sentences of natural language are processed. The term 'experienced users of mathematics' refers to people who have not only studied mathematics at a high level but who also frequently use mathematical expressions. It should be noted that this thesis does not focus on how experienced users of mathematics solve mathematical problems or how they think about mathematical expressions at a semantic level. Of interest instead are the encoding and parsing processes which occur below the level of conscious awareness.

1.1 Notations

This section begins by examining the class of languages whose structure and features have been studied more thoroughly than any other; natural languages. Following this, the writing systems that are used to represent natural languages in a visual medium are discussed. Writing systems are then contrasted with other visual notations, specifically those that make up the class of languages known as visual languages. Key differences between the two notation types are explored. Finally, the visual languages of interest in this thesis, mathematical notations, are examined in depth.

1.1.1 Natural Language

The use of spoken language is a hallmark of human communication. By producing a stream of sounds, we can convey information about the structure and function of objects, events in the past, present or future, or abstract concepts that are independent of our sensory perception. This is possible because the sounds produced consist of systematic patterns

that can be arranged in specific ways to convey meaning. Although humans speak many different languages with diverse characteristics, all spoken languages share certain core properties, allowing them to be grouped into a class of languages referred to as natural languages.

The primary unit of communication in natural language is the word. When young children are learning a language, they begin by using single words. It has been estimated by Miller and Gildea (1987) that by the age of 17, most English speakers will know approximately 80,000 word tokens. However, within this number there is a much smaller subset of words, referred to as a working vocabulary, that is used on an everyday basis.

Although words form the fundamental units of natural languages, many can be broken down into smaller components. Some words, such as *book*, are already in their simplest form. However the plural, *books*, contains two parts; the word *book* plus the suffix */s/*. These parts are called morphemes, and they represent the simplest meaningful components that can be used to build a word.

The words in a natural language can be grouped into various categories known as parts of speech, such as nouns, verbs, adjectives, adverbs and prepositions. Nouns, for example, are words for physical objects, names and abstract concepts, whereas verbs are words that represent actions and relations. The part of speech that a word belongs to influences language structure in two ways. Firstly, certain affixes can apply only to words of a particular category, while other affixes may affect words from different categories in different ways. For example, in English, the suffix */s/* on a verb is a present tense suffix used when the subject is third person singular (for example, *He sings quite well*). The suffix */s/* on a noun, however, refers to the plural case (although there are exceptions such as *feet* and *oxen*). Secondly, when forming a sentence, words of a particular category may only be combined with words in another category in specific ways. For example, an article may precede a noun, however it should never precede a verb. The rules that govern the valid arrangements of the different parts of speech are known as *syntax*.

Syntax defines how words can be grouped together into meaningful units such as phrases and sentences. For natural languages, it is possible to construct an infinite number of lawful sentences (Gleason & Ratner, 1993), yet to a native speaker, novel sentences that have never been heard before can be understood without difficulty. This is facilitated by the use of a finite number of syntactic rules which allow more complex constituents to be derived from simpler units. The rules that define the valid arrangements of words in a language are part of the *grammar* of that language. Using these rules, it is possible not just to understand valid sentences, but also to make judgements to determine if a sentence is grammatically invalid. Chomsky (1965) considered these two abilities to be distinct, referring to the linguistic knowledge used to make grammaticality judgements as *competence*, and the actual comprehension and production of speech as *performance*. It should be noted that much of our current understanding of linguistics has been influenced heavily by the work of Chomsky.

Research aimed at understanding the structure of sentences of natural language has led to the development of transformational grammars, which consist of phrase structure rules and transformational rules (Chomsky, 1957, 1965). The following are examples of some of the phrase structure rules that can be used to define English sentences. These rules contain elements that represent parts of speech (N – noun, V – verb, Aux – auxiliary, Art – article, Adj – adjective), phrases (NP – noun phrase, VP – verb phrase) and of course a symbol to represent sentences, S.

$$\begin{array}{l} S \quad \rightarrow \quad NP \text{ Aux VP} \\ NP \quad \rightarrow \quad \text{Art N} \\ NP \quad \rightarrow \quad \text{Art Adj N} \\ VP \quad \rightarrow \quad V \text{ NP} \end{array}$$

Note that there can be more than one way to define a constituent type (the example rules include two that define a noun phrase). Using these rules it is possible to decompose a sentence such as *The man will wash the red car* into its various constituents. *Parse trees* provide a useful way to illustrate the hierarchical relationships among constituents as

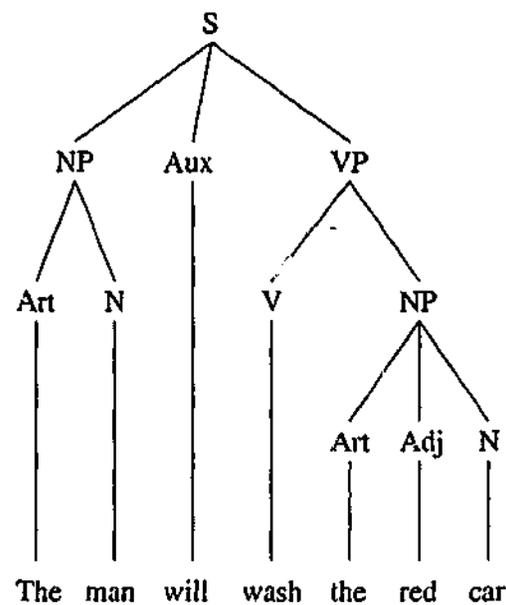


Figure 1.1: Parse tree for the sentence *The man will wash the red car*.

defined by the phrase structure rules. Figure 1.1 shows an example.

Transformational rules are needed to deal with sentence structures that cannot be captured by the phrase structure rules. While the sentence *The man knocked out the burglar* can be represented using just phrase structure rules, it is also possible to convey the same information by saying *The man knocked the burglar out*. In the first version of the sentence, the verb constituent describing what the man did is clearly *knocked out*. However, in the second version there is a discontinuity, with the words *knocked* and *out* separated by the noun phrase, *the burglar*. This can be dealt with using a transformational rule such as the following.

$$X \text{ Verb Particle NP } Y \implies X \text{ Verb NP Particle } Y$$

This rule can be applied to any sentence containing the constituent sequence specified on the left side of the transformational rule between the symbols X and Y (where X represents the constituents in the sentence that appear before the verb, and Y represents the constituents in the sentence that appear after the noun phrase). *The man knocked out the burglar* is such a sentence, where the verb is the word *knocked*, the particle is the word *out* and the noun phrase is *the burglar*. Using this rule, the order of the verb, particle and noun phrase may be transformed in the manner specified, allowing the sequence *knocked out the burglar*

to become *knocked the burglar out*.

The word order of a sentence is not only important in determining syntactic correctness, it is also important for semantic processing. Consider the two sentences, *The dog bit the boy* and *The boy bit the dog*. Both contain the exact same words and both are syntactically correct, yet they convey different meanings. This is because the functions of the constituents in the sentence are determined by their position. In a standard active sentence of English, listeners typically expect the first noun of a sentence to be the subject (the doer of a certain action), followed by a verb describing what the subject is doing (the action involved), and then another noun representing the object (the recipient of the action). English is therefore considered an SVO (Subject - Verb - Object) language, since this is the primary (although not the only) ordering of the constituents of a sentence. Most of the world's languages are SVO (Gleason & Ratner, 1993), although there are many languages with a different order (for example, Arabic is a VSO language and Korean is an SOV language). Some languages also use affixes to explicitly mark the functions of words, allowing for a more variable ordering (for example, Latin).

Despite the fact that word order provides a strong guide to the function of various sentence constituents, it is still possible to produce syntactically correct sentences whose meanings are ambiguous. Structural ambiguities occur when a sentence has more than one valid constituent structure. This can occur as a result of words that can act as different parts of speech (such as *fly*, which can be a noun or a verb), and also when the phrases of a sentence can be related to each other in several different ways. Consider the following sentence (from Fromkin & Rodman, 1993); *The boy saw the man with the telescope*. This sentence has two valid interpretations; first, that the boy saw the man who had a telescope, and second, that the boy used a telescope to see the man. For each of these interpretations, the sentence has a different valid syntactic structure. These two syntactic structures are illustrated using parse trees in Figure 1.2 (where PP represents a prepositional phrase). A result of structural ambiguity is that the intended meaning of a sentence, such as the

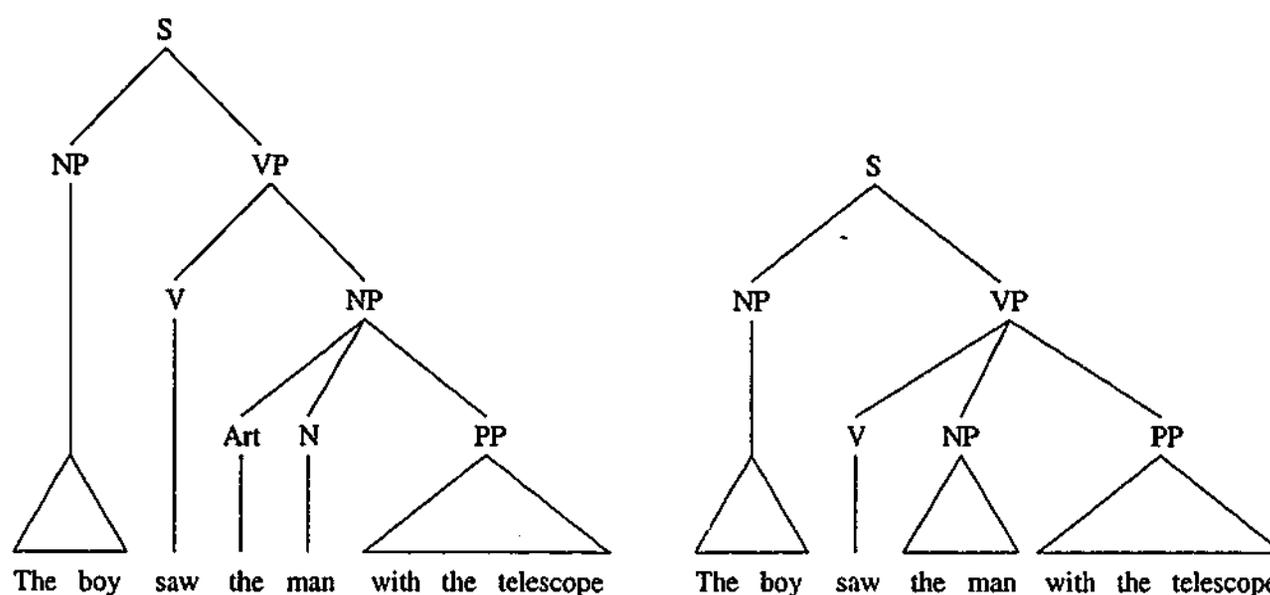


Figure 1.2: Two valid parse trees for the sentence *The boy saw the man with the telescope*.

example, cannot be determined by listening to the sentence in isolation. Rather, context is often required in order for the meaning of the sentence to be correctly interpreted, and even then it may remain ambiguous.

The previous example reveals that the meaning of a sentence is strongly bound to its syntactic structure. Syntax and semantics can still be distinguished when understanding a sentence; Chomsky gives as an example the sentence, *Colourless green ideas sleep furiously*, which is clearly syntactically valid but semantically nonsensical. However, it has been argued that in order to determine how the constituents of a sentence contribute to its overall meaning, it is appropriate to include semantics as part of the grammar (Katz & Fodor, 1963; Fodor & Pylyshyn, 1988). In a sentence, the phrases play specific roles in relation to other constituents, allowing natural languages to convey far more complex information than can be expressed by using words in isolation. Thus, it is the way in which the meaning of a sentence is dependent not only on the meaning of individual words, but also the syntactic rules that govern how they are combined, that gives natural languages their expressive power. This is not to say that meaning is exclusively tied to syntax. For example, the expression *kicked the bucket* can be used to refer to someone who has died, an

interpretation that is independent of its syntactic and literal semantic structure. However, such idioms form only a tiny part of most languages and importantly, it is still possible to use such expressions literally. Also, clearly there are semantic rules that govern the allowable sequences of semantic properties within a phrase or sentence.

1.1.2 Writing Systems

Although the use of spoken languages allows for efficient communication between people, the sounds are deployed over time, and so are only briefly available. Thus, a natural extension to spoken language use was to develop a more permanent method of linguistic communication; writing systems. Over the centuries, many different writing systems have been developed.

The earliest forms of writing were *pictographic systems* and *ideographic systems* (Gelb, 1963). Pictographic systems allow objects to be represented by symbols (graphemes) that pictorially resemble them in some way (for example, drawing a disc to represent the sun). Ideographic systems use arbitrary symbols to represent ideas by convention, without the need for any pictorial resemblance (of course, for many abstract concepts such as freedom, there is no obvious way to represent them pictorially). However, while these early forms of writing provided a rudimentary way for representing information, they did not represent individual words, or the sounds that make up the words (Akmajian, Demers, & Harnish, 1984).

The first modern writing systems were *logographic systems*. In logographic systems, the symbols do not directly represent concepts or objects, but rather they represent linguistic units (words and morphemes) (Crystal, 1987). This makes them more complete linguistic representations than either pictographic or ideographic systems. However, logographic systems have an important drawback; their use requires a large number of symbols to be memorized, thus impeding the acquisition of reading. This can be further complicated when words can be represented by multiple symbols in different ways. However, such systems also have certain benefits; they are not based on sounds, and thus people speaking different

dialects can still read the same logographic text. The Chinese writing system is an example of a writing system in use today that is logographically based.

A different way to construct a writing system that avoids many of the problems and ambiguities that can arise in logographic systems, is to use symbols to represent particular sounds. This is the idea behind *syllabic systems*, in which different symbols represent the different syllables that are needed to pronounce words. This introduces a very strong relationship between the writing system and spoken language, although it also sacrifices the feature of allowing the writing system to be read by people speaking different dialects. Despite the benefit of a close mapping between syllabic writing systems and spoken language, a key drawback is that for most languages, such as English, the number of syllables needed is very large (in English, there are around a thousand syllables, Rayner & Pollatsek, 1989). Mapping one symbol to each syllable would produce a writing system that is too complicated, whereas any attempt to reduce the number of symbols would allow ambiguities into the writing system. English is also ambisyllabic, presenting further complications (that is, some syllable boundaries can be difficult to define). For some languages, such as Japanese, the number of syllables is small (approximately 70, Just & Carpenter, 1987) making the syllabic system applicable. However, for many other languages a different system is required.

Alphabetic writing systems are also based on using symbols to represent sounds, but avoid the primary problem of syllabic systems. In alphabetic systems, symbols represent phonemes, which are the smallest sound units in speech. There are less than 100 phonemes used in all human languages, and a language such as English makes use of approximately 40 of them (Rayner & Pollatsek, 1989). Thus, alphabetic systems require a much smaller set of symbols, while still maintaining the strong link between written and spoken language. Note that in English, some phonemes are represented by combinations of symbols (for example, the sound represented by *th*), while some symbols represent multiple phonemes (for example, *a* in *father*, *bat* and *bake*). It is also not uncommon in alphabetic systems for

some phonemes to be represented by more than one symbol.

1.1.3 Visual Languages

Despite the benefits of writing systems, there are still many types and uses of information for which natural language representations are cumbersome. For example, while the layout and size of the rooms in a building can be described using written text, an architectural floor-plan usually provides a representation that is quicker and easier to interpret. For this reason, the use of diagrams is still important in how humans record and understand information. An important class of diagrams are those which make use of language-like rules. These diagrams are known as visual languages.

Chang (1990) defines visual languages as the systematic use of visual expressions to convey meaning, usually with the understanding that the spatial layout of the graphical components must be in two or three dimensions (Tortora, 1990). As such, written texts in languages such as English are not considered to be examples of visual languages since they are primarily sequential in nature. (Also, written text is primarily a manifestation of spoken language, rather than a language in its own right.) Diagrams that are direct depictions of objects are also not considered instances of a visual language, since they do not use visual elements in a systematic way. Rather, a feature of visual languages is that they incorporate rules that govern the spatial layout of the visual elements, allowing a distinction to be made between valid and invalid representations; that is, they have a syntax. Examples of visual languages include electronic circuit diagrams, music notation, and chemical structural formulae (Figure 1.3 illustrates valid instances of the latter two,



Figure 1.3: Valid examples of two visual languages: music notation and chemical structural formulae.



Figure 1.4: Invalid examples of two visual languages: music notation and chemical structural formulae.

while Figure 1.4 illustrates invalid instances).

Both writing systems and visual languages make use of symbolic representations to convey information. However, the use of multiple dimensions gives visual languages a representational advantage over sequential languages by allowing spatial layout to also play an important role. This extra flexibility provides many more representational possibilities when conveying information, however, the suitability of a given representation may vary depending on the type of information to be conveyed. Certain representations provide an advantage when important concepts can be directly mapped to specific visual features, making the information easier to interpret (Gurr, 1998). For example, in music notation, the vertical position of a note reflects its pitch, with notes of a higher pitch appearing higher than notes of a lower pitch (stave lines are also used to make this relationship more explicit). As a result, visual languages tend to be very content specific (for example, music notation is not used to represent mathematical concepts).

An important consequence of using multiple dimensions, is that the grammars used to describe the syntax of natural languages are not adequate for describing the syntax of visual languages. For example, the phrase structure rules used in transformational grammars rely on linear adjacency to compose constituents from simpler units, restricting their applicability to sequential languages only. To describe the syntax of visual languages, more complex formalisms are needed.

Of the many different approaches that have been proposed for describing visual languages, most have been grammar (syntax) based (although approaches have also been

G contains a subgraph that matches the L graph of that rule. When a production rule is applied, the subgraph in G that matches the L graph of that rule is replaced by the R graph of that rule, resulting in a new graph G' . The language that is described by such a grammar consists of all the graphs that can be generated by repeatedly applying the production rules, when starting with the axiom graph. Figure 1.5 gives an example of a graph grammar for describing the structural formulae of single chain hydrocarbons, with the grammar specified on the left, and an example derivation on the right.

The other main approach is also grammar based, and relies on attributes to encode the spatial (and semantic) information associated with each symbol. The values of these attributes are then taken into consideration in determining which rules of the grammar can be applied. When a rule of the grammar is applied to form a new constituent, the attributes of the new constituent are determined using the attributes of the components that compose it. In this way, the attributes of individual symbols can combine to influence the interpretation of the entire visual expression. Since there is no single way to sequence a multi-dimensional visual representation, the rules of the grammar apply to multisets (that is, unordered collections of symbols) rather than ordered sequences of symbols as is the case with phrase structure rules. Therefore, these grammars are referred to as attributed multiset grammars.

Figure 1.6 gives an example of a production rule for an attributed multiset grammar.

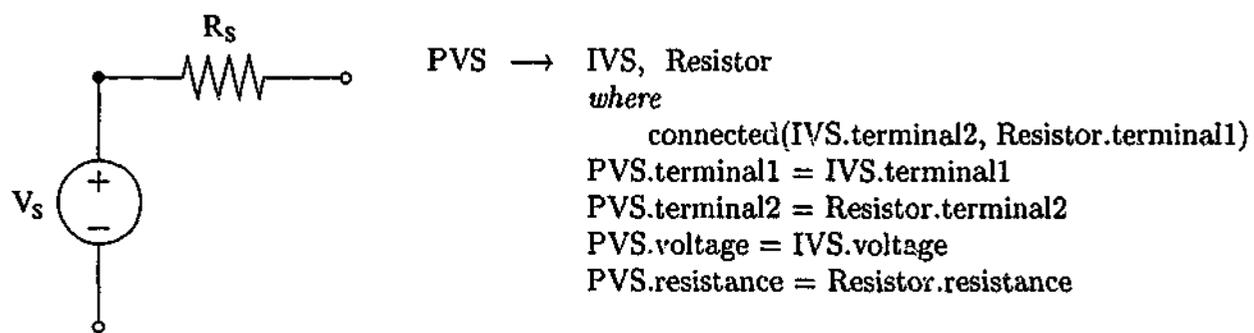


Figure 1.6: Example of an electronic circuit diagram of a Practical Voltage Source (left) and its corresponding production rule from an attributed multiset grammar (right).

On the left side of Figure 1.6 is part of an electronic circuit diagram that contains two circuit elements. The vertically aligned circuit element is an Ideal Voltage Source (IVS) and the horizontally aligned circuit element is a Resistor. When placed in series (as they are), they can represent a Practical Voltage Source (PVS). Next to the electronic circuit diagram is the rule that governs the production of a PVS. The right side of the rule can be split into three parts. The first part lists the components that are required to form the new constituent. In the example, there are two components: an IVS and a Resistor. The second part of the rule is a list of constraints on the values of the component attributes. These must be met for the rule to apply. There is only one constraint in the example (on the third line), and it specifies that the second terminal of the IVS must be connected to the first terminal of the Resistor (thus placing them in series). Finally, if the requirements of the first two parts are met, the new constituent can be formed with its own attributes, whose values are specified by the third part of the rule (the last four lines in the example).

1.1.4 Mathematical Notations

The visual languages that are the focus of this thesis are mathematical notations. The use of numbers is prevalent in modern human society, and thus the ability to manipulate numbers has become an important skill (Butterworth, 1999). Basic arithmetic skills are necessary to use a calendar, conduct financial transactions, or even follow a game of cricket. Other endeavours, such as forecasting the weather or sending satellites to the outer reaches of the solar system, require far more complex mathematical concepts to be understood. For the thousands of years that humans have been trying to master such concepts, they have been relying on various different notation systems to assist them. These systems have evolved and borrowed from each other to become the mathematical notations that are considered standard today.

A fundamental part of mathematical notation is how numbers are represented. All number systems incorporate symbols which represent specific values. However, they vary in how these symbols can be grouped together to form values which are not represented by

a specific symbol. One of the most common methods to represent such a value, is to use a group of symbols whose specific values add up to the desired value. Many early number systems were based on this method, perhaps the most famous being the Roman system. For example, the value of 128 in Roman numerals is represented by CXXVIII, where C represents 100, X represents 10, V represents 5 and I represents 1. Note that the symbols in this example are written from left to right in descending order with respect to their numerical value. As the Roman system developed, it also incorporated extra conventions; numbers should be represented by the minimum number of characters necessary (for example, 11 should be represented by XI rather than VIIIIII), and a symbol can also represent a value to be subtracted from the total if it appears to the left of a symbol of a higher value (allowing IX to more succinctly represent 9, as opposed to VIII).

Number systems based on adding symbol values were naturally suited to problems involving addition. However, for other manipulations, such as multiplying large values, they were quite cumbersome. This led to the development of the place value system, which forms the basis of the number system used in modern mathematical notations. In this system, the value of a symbol is influenced by its position. For example, in the number 652, the symbol 5 represents the value 50, whereas in the number 538 it represents the value 500. It should also be noted that place value systems require a symbol to represent zero, a feature that is not present in most other number systems, including Roman numerals. The numerals used in modern mathematical notations are referred to as Hindu-Arabic numerals, as they were originally developed in India and the Arab world. Early examples of these numerals appear in manuscripts from these regions which were written in the late ninth century (Cajori, 1928).

Mathematical problems were originally written as statements in natural language. (In Europe, the language typically used was Latin.) Such statements were usually unwieldy, both in regard to the amount of text needed to represent a problem, and how readily apparent the nature of the problem was to the reader. Thus, it was inevitable that specific

symbols would take the place of natural language text in describing various mathematical operations. However, the origins of the various symbols that are in common use today is quite varied. One of the most complete sources on this topic is Cajori (1928). Consider one of the most fundamental symbols in modern mathematical notations, the symbol for addition (+). The concept of addition was often expressed using the Latin word *et*, which means *and* (for example, using 4 *et* 5 to represent $4 + 5$). It is likely that the symbol + is based on one of the florid forms of this word. The symbol + (along with the symbol - to represent subtraction), first appeared in German manuscripts in the late fifteenth century. However, at a similar time in Italy, the symbols \bar{p} and \bar{m} were adopted (standing for *plus* and *minus* respectively), and the two sets of symbols competed with each other for over a century, with the symbols + and - finally winning out in the early seventeenth century. The symbol + (known as the Greek cross) not only had competition from other symbols, but also had to compete against variations. These include the Latin cross, \dagger (which was often placed horizontally to make typesetting easier, $\text{---}\dagger$), as well as fancier types of crosses such as ✠ . In fact, it was not uncommon for more than one symbol to represent a particular mathematical operation in the same manuscript. Even today, there are many ways to represent certain mathematical operations. For example, the division of three by four can be represented as $3 \div 4$ (common in English speaking countries), $3 : 4$ (common in Continental Europe), $3/4$ or $\frac{3}{4}$.

While the symbol + appears to have originated as an abbreviation (for the Latin word *et*), many other mathematical symbols are simply arbitrary ideographs. It was not uncommon for individual authors to come up with their own symbols, especially before the invention of the printing press when all copying had to be done by hand. Typically, however, these were rejected by other mathematicians. When a new symbol was adopted, it was usually by groups of mathematicians working independently from one another. Often, the adoption of a symbol had nothing to do with its suitability, but rather was due to other factors such as the popularity of a book in which it appeared (Cajori, 1928).

However, as one might expect, the overall trend appears to have been towards notations that are more useful. Consider the representation of unknown values (variables). In earlier symbolic representations of unknown values, a different symbol was often used when the unknown was raised to a different power, and thus only one unknown value could be represented in any given expression. For example, in early German manuscripts the unknown value, x , was represented by the symbol \mathcal{K} whereas the cube of the unknown value, x^3 , was represented by the symbol \mathcal{C} (Cajori, 1928). Over the intervening centuries, these representations have changed and evolved to form the modern notation of using a superscript to represent the power. The modern notation has many advantages; it allows the value of the power to be more easily determined and it also allows different unknown variables to use the same power notation (as in, $x^2 + y^2$). The modern notation also accommodates mathematical relationships that probably were not even considered when the fifteenth century notation was being developed, such as allowing an unknown value to be raised to the power of another unknown value (for example, y^x).

Mathematical notations are still being expanded and modified as new mathematical concepts are developed. However, the focus of this thesis is on a specific, well established subset of mathematical notations: modern algebraic notation. Since even today there is still some variation in the notational conventions used for algebra, it is prudent to specify more precisely the algebraic symbols and symbolic relationships that are considered standard in the context of this thesis. The symbols $+$ and $-$ represent addition and subtraction respectively. Multiplication is indicated by the concatenation of terms, and where two numbers are to be multiplied, the symbol \times is used. Division is represented by fractions, with the denominator and numerator centred with respect to the fraction line. Powers are represented using superscripts to the right of the base term, and the symbol $=$ is used to represent equality. Brackets are used when the order of precedence needs to be explicitly specified and variable naming will follow the standard convention (introduced by Descartes, 1637) of using the letters x , y and z . The following is an example of an algebraic expression

that uses these notational conventions.

$$y = \frac{\frac{1}{2}(x-3)^2}{7+9x} - x^3$$

While there are further mathematical concepts that can be classified as being algebraic, the examination of their use is not the primary focus of this thesis. It should be noted that with the advent of the computer, other algebraic notations have also come into popular use, such as the linearised versions of modern algebraic notation used in computer programming languages, and reverse polish notation (which is used by many calculators). Despite the increasing use of these other notations, algebraic problems in most fields, ranging from Biology to Economics, are still predominantly presented using modern algebraic notation, and therefore other algebraic notations will not be considered further.

1.2 Language Processing

One issue of particular interest in this thesis is whether people process mathematical expressions in a manner similar to the way in which they process sentences of natural language. Mathematical notation and natural language are clearly different in many ways, yet they both share common features that suggest that perhaps certain aspects of their processing could be similar. This section explores what is known about how both natural language and mathematical notations are processed.

1.2.1 Processing Natural Language

The processing of sentences of natural language requires the use of both long term and working memory. Long term memory is used to store lexical items and the grammatical rules, while the role of working memory is to temporarily store the working products of language processes during comprehension and production. However, an important issue with working memory is that it has a limited capacity and short retention duration (Baddeley, 1990; Miller, 1956; Peterson & Peterson, 1959). Sentences may contain many words, and thus they must be encoded in some way if information is not to be lost during comprehension. To overcome the capacity limits, working memory is able to deal with large

amounts of information by recoding it into successively larger meaningful chunks (Miller, 1956). Johnson (1968, 1970) has shown that the chunking of sentences of natural language is guided by syntax, with individual chunks conforming to grammatically defined units. However, the task of finding the correct syntactic structure of a sentence involves some important challenges.

The words at the beginning of a sentence can often be matched to more than one possible valid sentence structure. Processing further words in the sentence will typically reveal the correct syntactic structure (although as was seen earlier, there are complete sentences that can remain structurally ambiguous). For example, consider a sentence that begins with the words *John put the mustard in ...*. The word *in* usually signals the beginning of a prepositional phrase, however it is not clear what that prepositional phrase is modifying. If the complete sentence were *John put the mustard in the fridge*, then the prepositional phrase *in the fridge* should be attached to the verb phrase, describing where the mustard was put. However, if the complete sentence were *John put the mustard in the red jar away*, then the prepositional phrase would be *in the red jar*, which should be attached to the noun phrase as it relates to the mustard.

There are two main model types that can potentially be used to deal with such ambiguities when processing a sentence. The first type, known as parallel parsing models, involves keeping in memory all possible valid syntactic structures. When a structure is found to be incompatible with the next word in the sentence, it is eliminated, until eventually only one valid syntactic structure is left. Some structures can also be eliminated if they do not make sense semantically. However, one problem with this method is that it would place a heavy demand on memory resources, as many valid structures may be possible at the beginning of a sentence.

The second type, referred to as serial parsing models, involves choosing one particular structure, and assuming that it is the correct one. If, as further words are processed, it is found to be incorrect, an earlier point in the sentence is returned to and an alternative

structure is tried. This type of model appears to most closely reflect the way in which people process natural language, as it has been shown the people processing sentences with ambiguous structures typically prefer one alternative structure over others (for example, see Frazier & Rayner, 1982; Ferriera & Clifton, 1986; Trueswell, Tanenhaus, & Garnsey, 1994).

For a serial parser, there are two critical issues; firstly, how is the initial candidate structure chosen, and secondly, if the initial structure is wrong, how is the restructuring managed. Frazier and Fodor (1978) proposed an influential two-stage serial parsing model, where the first stage assigns lexical and phrasal information to short substrings, with the second stage then using this information to produce complete phrase structures for sentences. In this model, two important principles were developed that describe how the phrasal structure of a sentence is determined. They are the principles of *minimal attachment* and *late closure*.

The minimal attachment principle states that, as each word of a sentence is processed, the phrase structure of the sentence should remain as simple as possible (that is, the parse tree should contain the fewest nodes possible), while still remaining grammatically correct. For example, consider the following sentence (from Pinker, 1994, p. 212); *The cotton clothing is usually made of grows in Mississippi*. This is an example of a garden path sentence, where typically the initial processing leads to a point that reveals that the current phrasal structure is incorrect, and a new phrasal structure then needs to be constructed. When the word *clothing* is reached while processing the sentence, there are two options. Either *clothing* can become the subject noun of the sentence (with *cotton* as an adjective), or *cotton* can remain the subject noun, with the word *clothing* being the first word of a prepositional phrase used to modify the noun *cotton*. Although the later scenario is the correct one, it is also the more complex one in terms of phrasal structure, and thus typically the first option is chosen when the sentence is initially read.

The principle of late closure states that wherever possible, words are attached to the phrase that is currently being constructed. An example of this can be seen in the sentence

Peter said that Henry left yesterday. There are two valid ways to interpret this sentence. The most obvious one is that Peter made a statement along the lines of "Henry left yesterday". However, an equally valid interpretation is that yesterday, Peter said that Henry left. The latter interpretation is slightly less obvious because when the word *yesterday* is reached in the sentence, the current phrase being processed involves Henry leaving, while Peter speaking is represented by a phrase higher up in the parse tree structure. Thus, the sentence is typically processed by relating the word *yesterday*, to when Henry left.

It should be noted that much of the understanding of how humans process natural language has come not from studies on people listening to spoken language, but instead from studies of people reading written text. These studies typically involve response time measurements, or tracking eye movements in order to determine where attention is being directed in a sentence, and for how long. For example, evidence from the eye movements of readers has provided support for the principles of minimal attachment and late closure, in the form of regressions back to earlier parts of garden path sentences at the points that these principles predict (Frazier & Rayner, 1982).

During reading, the eyes do not move smoothly across the text, but rather jump (saccade) between fixations. When reading English text, fixations usually last for approximately 225 milliseconds (Rayner, 1998), although fixation durations can vary according to the nature of the reading material (for example, reading fiction usually requires shorter fixation durations than reading a text on physics). The saccades themselves usually take only 30 to 40 milliseconds, and the distance the eyes move during a saccade is typically around 8 character spaces (Just & Carpenter, 1987). The perceptual span when reading is approximately 14 characters in width, although this span is not equal around the centre of the visual field, as more of the letters to the right of the fixation point are perceived than to the left (Rayner & Pollatsek, 1989). The perceptual span can also be influenced by the location of word boundaries.

1.2.2 Processing Mathematical Notations

Although there are still gaps in our understanding of how natural language is processed, a substantial amount is known about how people parse and comprehend sentences. In contrast, little is known about how humans comprehend mathematical expressions despite centuries of use.

Research into how mathematical notations are processed has been fragmentary and has examined the problem from many different perspectives. Hayes (1973), for example, examined the role of visual imagery in solving mathematical problems. He presented participants with cards containing simple mathematical problems, such as those shown in Figure 1.7. The participants were required to solve the problems while at the same time reporting on their imagery during the solution process. It was found that a great deal of notation-related imagery was used in solving elementary mathematical problems. For example, in solving simple algebraic equations (such as the stimuli on the left of Figure 1.7), many participants reported movement of the symbol images. The nature of the imagery reported was found to be diverse both among participants, and also in the type and amount reported for particular problems.

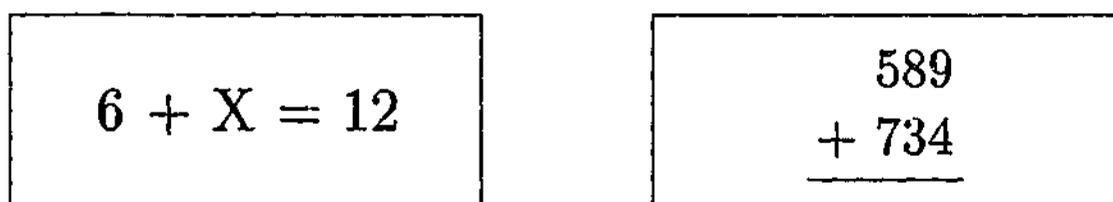

$$6 + X = 12$$
$$\begin{array}{r} 589 \\ + 734 \\ \hline \end{array}$$

Figure 1.7: Examples of stimuli used by Hayes (1973).

Kirshner (1989) examined the role of spatial information, using tasks in which algebraic notation was replaced by two different notations, one of which maintained the spacing found in algebraic notation, and one in which spacing was eliminated. Participants performed better using the notation with spacing, suggesting that spatial information plays a role in processing mathematical notations.

Ranney (1987) has conducted research that examines how the structure of mathematical

expressions is perceived. An apparent similarity to natural language led him to investigate whether similar effects to viewing words occurred when algebraic expressions were viewed. Recognition of letters in a word has been found to be superior to recognition of letters in a non-word, a phenomenon known as the word superiority effect (Reicher, 1969; Wheeler, 1970). To test for a corresponding effect with algebraic expressions, Ranney designed a task to determine if participants could more readily recognize a symbol as belonging to a string of symbols that formed a syntactically valid algebraic expression, over a string that contained the same symbols but did not form a syntactically valid algebraic expression. No corresponding algebra superiority effect was discovered. However, participants could more readily identify the category of a symbol (whether it was a variable or a numeral), if that symbol belonged to a string that formed a syntactically valid algebraic expression, as opposed to a string that did not form a syntactically valid algebraic expression. This suggests that the recognition of such expressions is based on structural content.

McCloskey and Caramazza (1987) have examined how number notations are processed by both normal participants and those with cognitive impairments. Results from this research indicate that there are dissociations between lexical and syntactic aspects of number processing, and that the processing of Hindu-Arabic numerals is distinct from verbal descriptions of numbers. However, as numbers form only one of many components that are used to compose algebraic expressions, it is not possible to draw any solid conclusions about the encoding of algebraic expressions from this work.

Although interesting, these results collectively do not provide a complete description of how humans encode and parse mathematical expressions. For example, they highlight the significance of specific visual features (for example, spacing) and structural content, but provide no indication of how these two factors fit together. Other gaps also exist in the research. For example, while it has been suggested that the processing of mathematical expressions is task dependent in nature (Sfard & Linchevski, 1994), there has been no work that follows up on this idea by demonstrating in what way different tasks may affect how

mathematical expressions are processed.

There has also been very little work that takes a grammatical approach towards understanding how humans process mathematical expressions. There has been some work with a focus on syntax; for example, Ernest (1987) developed an information processing model to examine how the mental representations used in a mathematical task relate to the syntactic structure of the expression involved. However, there is one thing in particular that is missing from the examination of the way in which mathematical expressions are processed; the systematic examination of the role of syntax of the form that has proven so valuable in developing an understanding of how humans comprehend sentences of natural language.

Ironically, much of the research that involves applying grammars to mathematical notations has stemmed from attempts to get computers to recognize and successfully parse mathematical expressions. This has led to many different approaches being used, including the use of graph grammars and attributed multiset grammars that are used to describe other visual languages (Anderson, 1968, 1977; Chang, 1970; Chan & Yeung, 2000). However, while useful in demonstrating how grammatical structure can be assigned to mathematical expressions, the development of these approaches has been largely independent of knowledge of how humans process mathematical notations.

1.3 Motivation for the Thesis

Despite the fact that natural language and mathematical notations share many common features, a key difference between them is that mathematical notations are visual languages with a two-dimensional structure, as opposed to the sequential structure of natural languages. This raises the question of just how do humans process mathematical notations.

While it has been shown in this chapter that the processing of mathematical notations has been examined from several different directions, the disconnected nature of the previous research reveals that the current level of understanding is far from complete. There appear to have been no attempts to compare the processing of mathematical expressions with

the processing of sentences of natural language, and there has also been little work that examines how mathematical expressions are used in the context of the human visual system (for example, there are very few eye-tracking experiments involving equations). This is surprising given the important role that mathematical notations play in so many human endeavours.

It is also surprising given that many students consider mathematics a difficult subject to learn. It may be possible that through a better understanding of how mathematical expressions are processed by experienced users of mathematics, educational benefits will result, allowing students to learn to use mathematical notation in a more efficient and productive way. Also, discovering how much difference or similarity there is between the comprehension of mathematical expressions and sentences of natural language, may lead to a better understanding of how the human mind processes language in general. For example, from existing research it is not clear if the processing of mathematical notations is independent of the way in which other languages (such as natural languages) are processed, or if the processes involved are related. It is this issue that provides the motivation for the research described in the following chapters.

Chapter 2

Encoding of Algebraic Expressions

As seen in Chapter 1, both natural language and mathematical notations share many common features. Both are symbolic representations which have a well defined syntax and semantics, and both make use of recursive rules which allow an infinite number of lawful expressions to be composed. It has been well established that syntax plays an important role in how humans process sentences of natural language (see Section 1.2.1). The similarities between natural language and mathematical notations suggest that mathematical expressions might be processed in a similar way. However, as discussed in Section 1.1.3, due to their two-dimensional nature, visual languages such as mathematical notations require considerably different grammars to those used to describe natural languages (such as graph grammars and attributed multiset grammars). Thus, it does not necessarily follow that experienced users of mathematics use internal representations that are dependent on syntax to encode expressions.

The main aim of this chapter is to determine how experienced users of mathematics mentally encode algebraic expressions, and what sort of internal representations support the encoding process. Although the role of syntactic processes is of particular interest, the function of processes that occur at the visual and mathematical "lexical" level will also be examined. Five experiments were constructed to explore the encoding process, three of which involved memory recognition tasks while the other two made use of the priming paradigm.

2.1 Syntactic Encoding of Algebraic Expressions

Experienced users of mathematics are capable of memorizing and working with large expressions that they have never seen before. Processing such expressions requires the use of both long term and working memory. However, working memory has a limited capacity and short retention duration (Baddeley, 1990; Miller, 1956; Peterson & Peterson, 1959), affecting the way in which large amounts of information are processed. To overcome the capacity limits, working memory is able to deal with large amounts of information by recoding it into meaningful chunks.

The structural principles that guide the encoding of chunks can be developed through knowledge and experience. Expertise has been shown to be correlated with the ability to efficiently chunk information (Groot, 1965), and examples of this have been demonstrated for visual languages such as circuit diagrams (Egan & Schwartz, 1979), chemical formulae (Johnstone & Kellett, 1980) and geometry (Koedinger & Anderson, 1990). More importantly for the purposes of this thesis, it has been shown that in the context of natural language, chunking of sentences is guided by syntax (see Section 1.2.1). It seems reasonable to hypothesize, then, that the encoding of mathematical expressions by experienced users of mathematics is based on mathematical syntax.

To test this hypothesis, a recognition task was constructed to determine which components of a previously presented mathematical expression participants can more readily recognize: a syntactically well-formed sub-expression or a non-well-formed sub-expression. A syntactically well-formed sub-expression refers to a sub-expression that is a valid mathematical expression on its own; for example, given the expression $8x^2 + 3(x - 2y)$ the sub-expression $8x^2$ is well-formed whereas $3(x -$ is not well-formed. If the encoding of equations is guided by syntactic structure, one would expect to see a significant performance advantage in the recognition of syntactically well-formed sub-expressions over non-well-formed sub-expressions.

Experiment 1

Method

Participants For all of the experiments discussed in this chapter, the participants were staff members, graduate or undergraduate students from the School of Computer Science and Software Engineering at Monash University. Mathematics formed a substantial component of their undergraduate training, and all were competent mathematicians who dealt with algebra frequently in their work. All participants were volunteers between the ages of 18 and 35 years, with normal or corrected-to-normal vision. There was some overlap in the participants involved in each experiment. Approximately one third of the participants were common to any given pair of experiments.

Twenty-four participants successfully completed this experiment. Data from an additional thirteen participants were not included due to excessive error rates.¹

Materials and Design One-hundred-and-twenty algebraic expressions were constructed, all consisting of between twelve and fourteen characters. The expressions contained at most one fraction and the variable names were x and y , since these are most commonly used. For each expression, sub-expressions of three types were constructed:

- a) A *well-formed sub-expression*, which is a component of the expression from which it is drawn, and has a valid syntactic structure on its own.
- b) A *non-well-formed sub-expression*, which is also a component of the expression from which it is drawn, but does not convey any coherent mathematical meaning on its own.
- c) An *incorrect sub-expression*, which is not part of the original expression. It can be either well-formed or non-well-formed. These act as fillers.

¹In order to check if the high number of exclusions affected the results of this experiment, the data analysis was also conducted on the complete data set (with no participants' data excluded). The results of this analysis indicated that the exclusion of the data from the thirteen participants with excessive error rates did not affect the experimental outcomes.

Table 2.1: Example expressions and sub-expressions used in examining syntactic well-formedness.

Expression	Sub-Expression		
	Well-Formed	Non-Well-Formed	Incorrect
$\frac{(y - 3y^2 + 1)^5}{4x}$	$y - 3y^2$	$y^2 + 1)$	$x^2 - 1$
$\frac{9}{x(2y - 5)} - 4x^3$	$(2y - 5)$	$\frac{\quad}{x(2$	$x(4y +$
$x = 6yx - \frac{2x + 2}{x}$	$2x + 2$	$= 6yx -$	$\frac{6x + 2}{y}$
$y = (x - 4)^3(x + 1)$	$(x + 1)$	$4)^3($	$5)^3($

Each of the sub-expressions contained between four and six characters. The average for well-formed sub-expressions was 4.89; for non-well-formed, 4.49; for incorrect, 4.72. The difference between these averages is not statistically significant. See Table 2.1 for examples of expressions and sub-expressions used. As the examples show, a variety of sub-expressions were used, some of which were bracketed, but most of which were not.²

In order to present each expression with all three of its sub-expression types, while ensuring that participants were presented with each expression only once to avoid practice effects, three counterbalanced versions of the experiment were constructed. For each version, there were forty instances of each type of sub-expression. Two additional expressions were constructed as practice items. The same practice items were used in each version. The items of each version were presented in a different pseudo-random order for each participant.

Procedure Participants were seated comfortably in an isolated booth. Items were displayed as black text on a white background on a 17" monitor at a resolution of 1024 × 768, controlled by an IBM compatible computer running a purpose designed computer program.

²Data analysis in Experiment 2 indicates that the presence or absence of brackets in a sub-expression has no influence on recognition performance.

The viewing distance from the monitor was approximately 50cm. The average width of the expressions in pixels was 177 (range 99-220) with an average height of 47 (range 26-61). The average width of the sub-expressions in pixels was 73 (range 39-192) with an average height of 26 (range 16-54).

Participants were given a brief statement of instructions before the experiment began. Practice items preceded the experimental items, and the participants took approximately fifteen minutes to complete the task. Progress was self-paced, with participants pressing the space bar to initiate the presentation of each trial.

Each item was presented in the centre of the monitor in the following sequence. First, an algebraic expression was shown to the participant for 2500ms. The expression then disappeared and the screen remained blank for 1000ms. Then the sub-expression was shown. The participant was required to decide whether the sub-expression was part of the previous expression, responding via a timed selective button press. They pressed the green button, (the '/' key on the right side of the keyboard), to indicate that the sub-expression was part of the original expression, or the red button, (the 'Z' key on the left of the keyboard), to indicate that the sub-expression was not part of the original expression. Participants were instructed to respond as quickly as possible, while taking care not to make too many errors. The sub-expression remained on the screen until a response was made.

The response time recorded was the time between the sub-expression first appearing and the participant's response. After the response, the participant was given feedback. If the response was correct then the word "CORRECT" appeared on the screen. Otherwise, the word "INCORRECT" appeared on the screen. In both cases, the participant's response time in milliseconds also appeared on the screen.

Data Treatment Data from participants with an overall error rate of more than 30%, or making in excess of 50% errors for any given sub-expression type, were excluded from the final analysis. Two measures were employed to reduce the unwanted effects of outlying data

points. Absolute upper and lower cut-offs were applied to response latencies, such that any response longer than 2500ms or shorter than 500ms was excluded from the response time data analysis and designated as an error (these cutoffs being selected as it is unlikely that an appropriate response can be made outside this range³). Secondly, standard deviation cut-offs were applied, so that any response time lying more than two standard deviations above or below a participant's overall mean response time was truncated to the value of the cut-off point (a technique known as winsorization, see Winer, 1962).

Three items, one from each condition of the experiment, were excluded from the analysis because at least 75% of participants failed to respond correctly to them. As a result, the final analyses were carried out over thirty-nine items per condition, not the original forty. Response time and error data were analysed by a series of analyses of variance (ANOVAs) over both participant (F_1) and item (F_2) data (allowing simultaneous generalization of the outcomes over the populations of both participants and algebraic expressions from which the samples are drawn). The probability level, $p < .05$, has been used as the criterion for statistical significance for all results discussed in this thesis.

Results and Discussion

The mean correct response time and error rate for the three sub-expression types are summarised in Table 2.2, along with the corresponding standard deviations (in parentheses). Planned comparisons of the data were conducted using two-way ANOVAs (versions \times sub-expression).⁴

The sub-expressions whose content was drawn from their corresponding expressions (i.e., both well-formed and non-well-formed sub-expressions), were responded to more rapidly than incorrect sub-expressions (well-formed: $F_1(1,21) = 80.62$, $p < .05$, $F_2(1,114) = 198.58$, $p < .05$; non-well-formed: $F_1(1,21) = 4.33$, $p < .05$, $F_2(1,114) = 15.73$, $p < .05$). This outcome is reflected in the error rate data also. Fewer errors were made, relative to incorrect

³The appropriateness of the cutoff values selected were confirmed by an analysis of the response time distribution.

⁴The first factor in the analyses is a 'dummy' factor created by the arbitrary assignment of target items to three separate lists for the purpose of counterbalancing.

Table 2.2: Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 1.

Sub-Expression	RT(ms)	%Error
Well-Formed	1147 (147)	13.1 (6.2)
Non-Well-Formed	1352 (228)	23.5 (8.8)
Incorrect	1429 (213)	32.8 (8.9)

sub-expressions, on responses to both well-formed ($F_1(1,21) = 129.93, p < .05, F_2(1,114) = 69.21, p < .05$) and non-well-formed sub-expressions ($F_1(1,21) = 9.92, p < .05, F_2(1,114) = 12.84, p < .05$).

More importantly, there is also a 205ms recognition advantage for sub-expressions that are well-formed components of their corresponding expression, over their non-well-formed counterparts ($F_1(1,21) = 46.67, p < .05, F_2(1,114) = 89.68, p < .05$). This recognition advantage holds for error rates also, with participants making significantly fewer errors on well-formed than non-well-formed sub-expressions ($F_1(1,21) = 23.50, p < .05, F_2(1,114) = 25.90, p < .05$). Clearly, the participants perceive the original expressions in a way that allows faster and more accurate recognition of well-formed sub-expressions than non-well-formed sub-expressions.

The results of the experiment support the hypothesis stated earlier, that experienced users of mathematics use an internal representation based on mathematical syntax to encode algebraic expressions. Syntactically well-formed sub-expressions are more readily recognized as a component of a previously seen expression, suggesting that the algebraic expressions are encoded into chunks that correspond either directly to these sub-expressions, or into chunks that can be rapidly combined to reconstruct these sub-expressions. In contrast, the lack of structure in non-well-formed sub-expressions means that mathematical syntax cannot guide the encoding of such sub-expressions when they appear on their own. If the encoded

representation of the original expression reflects the expression's syntactic structure, it is unlikely to contain any chunks that correspond either directly to the non-well-formed sub-expressions, or chunks that can be rapidly combined to reconstruct them. Therefore, additional processing will be required to find a level at which a comparison can be made, resulting in longer recognition times and making the procedure more prone to errors.

However, before concluding that syntactic processes are responsible for the results of Experiment 1, other potential accounts that rely on lower levels of processing need to be examined. This includes processing by the visual system, as well as "lexical" processing of the various symbols.

One alternative explanation of the results is that the encoding of algebraic expressions is guided primarily by visual features; that is, the symbols of an expression are not directly encoded according to their syntactic structure, but rather are grouped into chunks according to visual attributes such as size, spacing and layout. This explanation is not unreasonable given that standard mathematical notation is a visual language with a two-dimensional structure, and that visual features play a role in the processing of expressions (as shown by Kirshner, 1989). Thus, this possibility is examined in Experiment 2.

2.2 The Role of Visual Processes

Perception of complex visual stimuli requires processes that break the stimuli up into manageable units. Of particular relevance to the chunking of mathematical expressions, is the issue of how stimulus elements are grouped into larger units; that is, the issue of perceptual organization. In order to understand how a mathematical expression composed of various symbols is processed, it is necessary to understand which symbols are grouped together based purely on their visual attributes and layout.

Research into the problem of perceptual organization led Wertheimer (1923) to develop several laws about how stimulus elements are grouped together. The main principle underlying these laws is that perceptual objects will be organized so as to allow the resulting

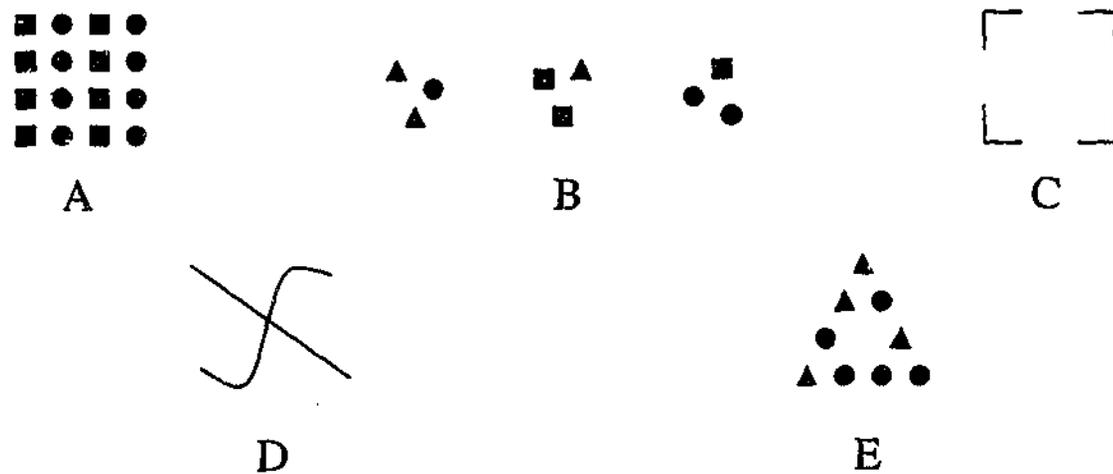


Figure 2.1: Examples of the laws of perceptual organization: (A) the law of similarity, (B) the law of proximity, (C) the law of closure, (D) the law of good continuation, and (E) the law of familiarity.

structure to be as simple as possible. A deficiency of this principle is the fact that 'simple' is a subjective notion, and it is possible to create example stimuli that can have several equally 'simple' organizations (Goldstein, 1996). Yet despite this, the laws do encapsulate some fundamental properties of perceptual organization. Examples of these laws include the law of similarity, which states that objects that are similar to each other tend to be grouped together (Bell & Bevan, 1968). The basis for such similarity could be features such as shape, size, or colour (Treisman, 1987), although as with the notion of 'simple', 'similar' can also be subjective. The law of proximity states that objects close together tend to be grouped together (Bower, 1967; Bell & Bevan, 1968), while according to the law of closure, a space enclosed by a contour is likely to be perceived as a single figure (Holmes, 1968). The law of good continuation states that elements close to each other tend to be grouped if they can be connected by a smooth line (Tuijl, 1980; Wouterlood & Boselie, 1992), and the law of familiarity states that elements will tend to form groups that appear familiar (Wertheimer, 1923). There are also several laws that apply specifically to motion.

Examples of the laws of perceptual organization appear in Figure 2.1. The law of similarity is illustrated in Figure 2.1 (A) where vertical columns are seen rather than horizontal rows because the columns consist of elements of the same shape. In Figure 2.1 (B) however,

differences in shape do not prevent three localized clusters from being seen, illustrating the law of proximity. Figure 2.1 (C) gives the impression of representing a square despite the gaps in the lines, illustrating the law of closure. A straight line is seen intersecting a curved line in Figure 2.1 (D) even though there are also other possibilities (such as four lines meeting at a central point, or two lines that are connected where they suddenly change direction). This illustrates the law of good continuation. Finally, the elements in Figure 2.1 (E) are seen to form a triangle, as per the law of familiarity. It should be noted that these laws of perceptual organization are descriptive rather than explanatory, but this is sufficient within the scope of this thesis.

When applied to algebraic expressions, these laws suggest certain chunking patterns based on purely visual features. For example, consider the following expression.

$$\frac{8x - 3y^2}{(y + 7x)^2}$$

The law of good continuation would suggest that the denominator and the numerator of the fraction would become separate chunks, a split enhanced by the fraction line separating the two. Also, the spacing surrounding the plus and minus symbols and the law of proximity suggest that groups such as $8x$ and $3y^2$ would form visual chunks. Even the curvature of the brackets tends to enclose the symbols within them into a single unit, as per the law of closure. According to these laws of perceptual organization then, the chunking that would be expected as a result of the general processes of visual perception should be similar to the chunks derived from breaking up an expression according to the rules of mathematical syntax. It is possible that this similarity has emerged as mathematical notations have evolved over the centuries, to assist the processing of mathematical expressions (Cajori, 1928). However, this similarity makes it difficult to definitively explain the results of Experiment 1, since it is not clear whether visual features or syntactic structure form the primary basis for the internal representations used to encode mathematical expressions.

To resolve this issue, a recognition task similar to that used in Experiment 1 was constructed for the purpose of determining which components of an algebraic expression ex-

perienced users of mathematics can more readily recognize: sub-expressions that form a "visual chunk" but are not syntactically well-formed, or sub-expressions that are syntactically well-formed but do not form a visual chunk. The immediate problem in designing such a task, is that the observation motivating this experiment (the fact the visual chunks frequently coincide with syntactically well-formed sub-expressions), also rules out normal algebraic notation as an appropriate stimulus. Thus, some sort of modification needs to be made to the notation in order to produce expressions with different chunking patterns, depending on whether syntactic structure or visual properties form the basis of the chunking. However, caution must be exercised, as possible modifications that are intended to subtly modify visual appearance, may result in a notation that experienced users of mathematics interpret either with difficulty, or incorrectly. For example, consider the idea of reversing relative spacing, such that the wide spacing surrounding symbols such as plus and minus signs are replaced with narrow spacings, and the narrow spacings that are found between concatenated product terms are replaced with wide spacings. The following is an example of this idea, with the original expression on the left, and the modified version on the right.

$$3x + 2y - 1 \qquad 3 \ x+2 \ y-1$$

The problem with the modified expression is that it appears like the contents of a matrix, consisting of three small, syntactically correct terms that are evenly spaced in a row. Therefore, the reaction of an experienced user of mathematics might be to interpret the modified expression as a collection of matrix elements, rather than an algebraic expression with modified spacings. Such a modification therefore, is not appropriate for this task. Instead, a modification is needed that will change the visual chunks of a mathematical expression without corrupting the syntactic structure.

The solution chosen was to use colour. This allowed us to take advantage of the important role colour perception plays in the grouping of objects. The algebraic expressions used were all composed of symbols of two colours, red and black, with small groups of adjacent symbols being assigned one of these colours. Thus, groups of symbols in the expressions

differed from their neighbouring groups by their colour. The following is an example of one of the expressions used.

$$y = x^2 + \frac{9}{3(4-6x)}$$

Red and black were chosen because they provide a clear contrast, with red having a greater visual impact than black (Alpern, Lawrence, & Wolsk, 1967).

Since colour is one of the visual properties that the law of similarity operates on (Treisman, 1987), it is reasonable to expect these groups of symbols to be perceptually organized into visual chunks. Also, since the uniformly coloured symbols within a visual chunk are adjacent, the law of proximity is being exploited, further enhancing the grouping effect generated by the different colours in the stimulus.

The symbols of the uniformly coloured visual chunks that were present in the expressions, represented syntactically non-well-formed sub-expressions (for example, $= x^2 +$ in the previous expression). Syntactically well-formed sub-expressions were also constructed for the recognition task, but these crossed the visual chunk borders and thus, were composed of symbols of more than one colour (such as $4 - 6x$ in the previous expression). If experienced users of mathematics use visual features as the primary basis of their internal representations, then there ought to be a performance advantage in the recognition of sub-expressions that form a visual chunk but are not syntactically well-formed, rather than sub-expressions that are syntactically well-formed but do not form a visual chunk. If, however, syntactic structure is more important when encoding algebraic expressions, one would expect a recognition advantage in the opposite direction.

Experiment 2

Method

Participants Twenty-four participants, all experienced users of mathematics, successfully completed the experiment. All participants had normal or corrected to normal vision, and could distinguish between the colours red and black. Excessive error rates resulted in

the exclusion of the data from an additional seven participants (as per the criteria given in Experiment 1).

Materials and Design Ninety algebraic expressions were constructed, all consisting of between twelve and fourteen characters. As in Experiment 1, the expressions contained at most one fraction and the variable names were x and y . The symbols in the expressions were either red or black, these colours being chosen because they provide a clear contrast on a white background. For each expression, sub-expressions of four types were constructed:

- a) A *syntactic sub-expression*, which is a component of the expression from which it is drawn that contains both red and black symbols, and thus, does not form a visual chunk. It is, however, syntactically well-formed.
- b) A *visual sub-expression*, which is a component of the expression from which it is drawn, in which all symbols are uniformly coloured, making it a single visual chunk. However, it is not syntactically well-formed, and thus, does not convey any coherent mathematical meaning on its own.
- c) An *ill-formed sub-expression*, which is a component of the expression from which it is drawn that contains both red and black symbols, and which is not syntactically well-formed. Thus, these are neither syntactic chunks nor visual chunks.
- d) An *incorrect sub-expression*, which is not part of the original expression. The symbols can be all of one colour, or of mixed colours, and the sub-expression can be either syntactically well-formed or non-well-formed. These act as fillers.

Each of the sub-expressions contained between four and six characters. The average for syntactic sub-expressions was 4.56; for visual, 4.51; for ill-formed, 4.53; for incorrect, 4.60. The difference between these averages is not significant. See Table 2.3 for examples of expressions and sub-expressions used. Again, some sub-expressions were bracketed and some were not.

Table 2.3: Example expressions and sub-expressions used in examining visual versus syntactic properties.

Expression	Sub-Expression			
	Syntactic	Visual	Ill-Formed	Incorrect
$y = x^2 + \frac{9}{3(4-6x)}$	$4 - 6x$	$= x^2 +$	$3(4-$	$= y^3 +$
$y(8 + x^3) - 8x^2y$	$8x^2y$	$x^3)-$	$)-8x$	$7y^2x$
$y = x^5 + \frac{4yx-4}{5y}$	$4yx-4$	$y = x^5 +$	$4yx-$	$x = y^2$
$\frac{9}{9y(5y+7)} - 4$	$(5y+7)$	$\frac{9}{9y(5$	$\frac{9}{y+7)}$	$x(5x-7$

For this experiment, only the first three types of sub-expression are of interest, with the incorrect sub-expressions only used as foils. The expressions were divided into two groups of forty-five. Three counterbalanced versions of the experiment were constructed for the first group of expressions, so that for each expression in that group, it was possible to present all of the first three sub-expression types, while ensuring that participants were presented with each expression only once in order to avoid practice effects. Each of the expressions in the second group were assigned the incorrect sub-expression type, and the use of the second group of expressions was consistent across all versions of the experiment. All ninety expressions were presented in each version of the experiment. For each version, this resulted in fifteen instances of each of the first three sub-expression types, with forty-five instances of the incorrect sub-expressions. Two additional expressions were provided as practice items. The same practice items were used in each version. The items of each version were presented in a different pseudo-random order for each participant.

Procedure The procedure was identical to that in Experiment 1. The environment, the sequence and timing of stimuli presentation, participant response mechanism and feedback

were the same. The average width of the expressions in pixels was 181 (range 123–237) with an average height of 46 (range 26–61). The average width of the sub-expressions in pixels was 70 (range 27–156) with an average height of 27 (range 16–61). Participants took approximately twelve minutes to complete the task.

Data Treatment The same measures used in Experiment 1 to reduce the unwanted effects of outlying data points, were applied to the data. No items had an error rate in excess of 75%, and thus, no items were excluded from the final analysis. The mean response time and error rate for incorrect sub-expressions are reported, but not considered in the analysis. For the other sub-expression types, response time and error data were analysed by a series of analyses of variance (ANOVAs).

Results and Discussion

The mean correct response time and error rate for the four sub-expression types are summarised in Table 2.4, along with the corresponding standard deviations (in parentheses). Two-way ANOVAs (versions \times sub-expression) of the data were conducted, carried out separately over participant and item data.

Table 2.4: Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 2.

Sub-Expression	RT(ms)	%Error
Syntactic	1167 (245)	17.5 (9.0)
Visual	1298 (266)	23.9 (11.1)
Ill-Formed	1309 (253)	21.7 (11.2)
Incorrect	1334 (230)	23.6 (6.0)

The response time results show that syntactic sub-expressions are recognized more readily than visual sub-expressions, with a significant performance advantage of 131ms

($F_1(1,21) = 10.14, p < .05, F_2(1,42) = 23.97, p < .05$). As would be expected from Experiment 1, performance with syntactic sub-expressions is also significantly better than for ill-formed sub-expressions, with a significant 142ms advantage ($F_1(1,21) = 23.43, p < .05, F_2(1,42) = 18.75, p < .05$). These trends also hold for error rates, although not all of these results are statistically significant. There is a 6.4% accuracy advantage for syntactic over visual sub-expressions ($F_1(1, 21) = 4.40, p < .05, F_2(1, 42) = 4.63, p < .05$), however, while performance on syntactic sub-expressions was also 4.2% more accurate than for the ill-formed sub-expressions, this result was only significant in the participant analysis ($F_1(1, 21) = 4.43, p < .05, F_2 = 1.86$).

These results provide no support for the hypothesis that visual features form the primary basis for the encoding of algebraic expressions by experienced users of mathematics. Despite the fact that colour provides an influential visual cue for grouping symbols together, the visual sub-expressions were identified more slowly, and with less accuracy than the syntactic sub-expressions. Furthermore, there was no significant response time recognition advantage for visual over ill-formed sub-expressions (11ms difference; $F_1 < 1, F_2 < 1$). This was also true for error rate data (2.2% difference; $F_1 < 1, F_2 < 1$). Both of these sub-expression types are the same syntactically (being non-well-formed), but they differ in the colour composition of their symbols. The absence of any recognition advantage indicates that colour had no influence on performance, despite the fact that it highlighted an alternative means of grouping the symbols in the expressions.

To determine if brackets played a disproportionately influential role in the recognition of syntactic sub-expressions, a further sub-analysis was also carried out. Brackets not only involve the visual feature of enclosing symbols, but are also likely to be more familiar to the participants due to their use in natural language text. Many of the syntactic sub-expressions used in the experiment were based on brackets. For example, consider the expression, $4x + 7(2y - 1)$. A sub-expression based on brackets is either a bracketed term, such as $(2y - 1)$ in the previous expression, or the contents of a bracketed term, such as, $2y - 1$. Of

the forty-five syntactic sub-expressions constructed for the experiment, twenty-three were based on brackets while the remaining twenty-two were not. Due to uneven item numbers in each version of the experiment, only a participant-based analysis could be conducted. Both response time and error rate analyses showed no significant recognition advantage between sub-expressions that were based on brackets (1174ms response time, 18.9% error rate), and those that were not (1166ms response time, 16.2% error rate), with only an 8ms response time difference, ($F_1 < 1$) and a 2.7% error rate difference ($F_1 < 1$). This indicates that brackets do not disproportionately affect the encoding of algebraic expressions.

The results of Experiment 2 indicate that visual features do not play the dominant role in the encoding of algebraic expressions. This is not surprising, given that the constituent structure of sentences of natural language is also based primarily on syntactic structure. However, before any definitive claim can be made that syntactic structure might be the primary basis of these internal representations, another potential account of the outcome of Experiment 1 must also be considered. It is possible that groups of symbols that are familiar to experienced users of mathematics might eventually become stored as tokens in long term memory; that is, common sub-expressions of equations could establish their own internal representations, such that the encoding of subsequent presentations of these sub-expressions would only require recognition, and not syntactic processing. If many such tokens were stored in long term memory, then they could be used to encode large expressions. The stored tokens would in effect be processed in much the same way that words are processed in the encoding of natural language text. Hence, this idea suggests that there might be some sort of long term memory store for mathematical notations that is equivalent to a natural language lexicon.

A simple example of a possible token is the term x^2 , which is commonly found in many mathematical formulas. If one concedes that an experienced user of mathematics might identify this term as a lexical-like token, then in principle it is also possible that more complex combinations of symbols, such as the sub-expressions used in the first two

experiments, could also be stored as lexical-like tokens in long term memory. Therefore, the idea of a "lexicon" of mathematical sub-expressions is examined in Experiment 3.

2.3 Masked and Visible Priming

For the past 30 years the priming paradigm has been used extensively to explore how various stimuli are recognized, in particular, words and objects (for example, Meyer & Schvaneveldt, 1971; Becker, 1980; Heij, Dirks, & Kramer, 1990). In any priming paradigm, the participant responds to a target stimulus that is presented on the computer screen following the presentation of a prime stimulus; with the prime either related or unrelated to the target in some way. The logic behind the paradigm is that response times for targets will be facilitated by a related prime, relative to an unrelated one, if the nature of the prime-target relationship is relevant to the processes of "lexical access". In the standard form of the priming task, referred to as *visible priming*, both primes and targets are presented so as to be clearly visible to the participants. However, this allows participants to discover the prime-target relationship, make use of episodic memory (Forster & Davis, 1984) and adopt expectancy-based strategies to aid responses (for example, Heyer, Briand, & Dannenbring, 1983). Forster and Davis's (1984) *masked priming* paradigm overcomes this problem by presenting the prime in such a way that participants are not consciously aware of its presence, and thus, cannot be aware of the prime-target relationship. In word recognition, typically the prime is presented very briefly (50-60ms), and is placed temporally between a forward mask consisting of a row of hash marks presented for 500ms (for example, #####), and a backward mask, which is the target, presented for 500ms. All are presented in the same spatial location at the centre of the computer screen with the prime and target presented in a different case or font. Half the targets are words and half are pronounceable nonwords, with a speeded lexical decision (word/nonword classification judgment) being made on the target letter string. This technique is now well-established and recognized within the literature as a reliable means of investigating lexical processing

free from extra-linguistic influences.

While masked priming stimuli are unavailable for conscious report, they have been shown to produce a variety of reliable priming effects, including effects based on form similarity in both alphabetic and logographic languages (for example, Forster, 1998; Forster & Davis, 1984; Forster, Davis, Schoknecht, & Carter, 1987; Hong & Yelland, 1997) and morphological relatedness (for example, Forster, 1998; Forster et al., 1987). Importantly, since primes and targets are visually dissimilar, being presented in a different case or font, these priming effects cannot be occurring at a (pre-lexical) visual processing level. This claim is further supported by the observation that priming effects are found for word targets only with no evidence for priming of nonword targets, which have no entries in lexical memory. This, coupled with the absence of effects of both expectancy (Forster, 1998) and episodic memory (Forster & Davis, 1984), places masked priming effects at the level of lexical processing. Thus, the masked priming effect is simply the consequence of the processing of the prime running 60ms in advance of that for the target. The masking of the prime means that insufficient information is available to the processing system to enable full recognition. However, if the prime and target are converging on the same lexical candidate(s), then the prime has already done much of the access processing for the target. This facilitates lexical access to, and recognition of, the related target.

Importantly, Johnston, Hayward, and Sujica (1999) have recently extended masked priming beyond the language domain to the recognition of more complex visual forms, three-dimensional objects. They constructed a task in which participants were required to determine if a target stimulus was a common object or a nonsense object. They found that common objects were identified more rapidly if they were preceded by an identity prime (a 60% scaled version of the target stimulus), than if they were preceded by a control prime (a 60% scaled version of a different object), suggesting that there are internal representations for common three-dimensional objects stored in long term memory. If this is the case, it is possible that similar internal representations might also exist for common combinations of mathematical symbols.

2.4 The Role of Lexical Processes

The aim of Experiment 3 is to determine if there is some sort of memory store equivalent to a lexicon, in which common sub-expressions are stored as lexical-like tokens. Of particular interest is whether the results of Experiment 1 were a function of "lexical" level representations rather than syntactic representations.

In order to test this idea, a task was constructed that makes use of the masked priming paradigm. Participants were presented with expressions that were either syntactically correct or contained a single syntax error (for example, an unmatched bracket or a misplaced equals sign). Their task was simply to determine if these expressions were syntactically valid or not. A masked prime preceded the presentation of each expression, with the prime being a sub-expression of the full expression. The sub-expressions were similar to those used in Experiment 1, being either syntactically well-formed or non-well-formed. If the well-formed sub-expressions that were responded to more rapidly in Experiment 1 are stored as lexical-like tokens in long term memory, while their non-well-formed counterparts are not, then one would again expect the well-formed sub-expressions to provide a performance advantage as masked primes.

Experiment 3

Method

Participants Twenty-four experienced users of mathematics successfully completed the experiment. None of the participants reported being aware of the masked primes. Data from one additional participant was excluded due to an error rate in excess of 50% for one of the conditions.

Materials and Design Sixty algebraic expressions were constructed, all containing between twelve and fourteen characters. The expressions contained at most one fraction, and the variable names were x and y . Thirty of the expressions were then modified so that they were not syntactically valid. These invalid expressions thus, appeared superficially to

Table 2.5: Example expressions and primes used in Experiments 3 and 4.

Type	Expression	Prime Type	
		Well-Formed	Non-Well-Formed
Valid	$(7 + 8y)(x + 8y)$	$7 + 8y$	$8y)(x +$
	$2x^2 + \frac{9(xy + 4)}{4y}$	$(xy + 4)$	$9(xy +$
Invalid	$\frac{(9 = 2y)(3x + 2)}{y}$	$3x + 2$	$y)(3x +$
	$\frac{+}{x^4(7y + 6)} - yx$	$7y + 6$	$^4(7y +$

be correct, but closer inspection would reveal that they were not. Examples of the syntax errors are non-matching parentheses or the incorrect placement of an operator.

For each expression, both valid and invalid, two types of sub-expression were created. These sub-expressions were used in the experiment as primes. The sub-expressions types were potential lexical units (equivalent to the syntactically well-formed sub-expressions defined in Experiment 1), and non-well-formed sub-expressions (also defined in Experiment 1). Examples of the expressions and sub-expressions used can be seen in Table 2.5. Each of the sub-expressions contained between four and six characters. The average for well-formed sub-expressions was 4.82; for non-well-formed, 4.72. The difference between these averages is not statistically significant.

Two counterbalanced versions of the experiment were constructed in order to present both sub-expression types for each expression, but ensuring that participants were presented with each expression only once to avoid practice effects. Thus, for each version, there were fifteen instances of each sub-expression type for both valid and invalid expressions. Five additional expressions were constructed as practice items, with the same practice items being used across both versions. The items of each version were presented in a different

pseudo-random order for each participant.

Procedure Participants were seated comfortably in an isolated booth. Items were displayed as black text on a white background on a 15" monitor at a resolution of 640×480 , controlled by an IBM compatible computer running the DMASTR programs.⁵ The average width of the expressions in pixels was 142 (range 81-187) with an average height of 42 (range 21-50). The average width of the sub-expressions in pixels was 56 (range 32-89) with an average height of 20 (range 9-48).

Participants were given a brief statement of instructions before the experiment began. Practice items preceded the experimental items, and the participants took approximately ten minutes to complete the experiment. Progress was self paced, with participants pressing a foot pedal to initiate the presentation of each trial.

Each item was presented in the centre of the screen in the following sequence. First, a feature mask constructed from straight and curved lines appeared for 500ms. The size of the mask image was 250×65 pixels, making it large enough to completely obscure any of the expressions. One of the sub-expressions appeared next for 56ms, acting as a prime. As it was centred, it was not necessarily in the same position as it appeared in the expression. Then the expression (either valid or invalid) was shown, remaining on the screen until the participant responded. The participants were required to determine whether or not the expression was syntactically valid, responding via a timed selective button press. A blue button was pressed with their preferred hand to indicate that they thought the expression was valid, and a red button was pressed with their non-preferred hand to indicate that they thought the expression was syntactically invalid. Participants were instructed to respond as quickly as possible, while taking care not to make too many errors. If the participant took longer than 4000ms to respond, the trial ended with the message "No response" and the response was designated as an error.

⁵DMASTR (DisplayMaster) is a suite of DOS programs written for IBM compatible PCs by K.I. Forster and J.C. Forster at the University of Arizona. The software is designed for the measurement and analysis of reaction times.

The response time recorded was the interval between the onset of the target expression and the participant's response. After the response, the participant was given feedback. If the response was correct then the word "Correct" appeared on the screen along with the response time in milliseconds. If the response was incorrect, "... Wrong ..." appeared on the screen.

Data Treatment For data to be included in the final analysis, participants were required to have an overall error rate of no greater than 25%, and no more than 50% errors for any condition. The same measures used to reduce the unwanted effects of outlying data points in the previous experiments, were applied to the data. Response time and error data were analysed by a series of analyses of variance (ANOVAs), over both participant and item data. Since the aim is to determine how mathematicians encode valid expressions, data from the invalid expressions were not included in the analysis, as they were simply foils for the decision task.

Results and Discussion

The mean correct response time and error rate for the different prime types in both parts of the experiment, are summarized in Table 2.6 along with the corresponding standard deviations (in parentheses). Planned comparisons of the data were conducted using two-way ANOVAs (versions \times sub-expression).

Table 2.6: Mean correct response times (ms) and error rates (%) as a function of prime type for Experiment 3

Prime Type	RT(ms)	%Error
Well-Formed	1151 (432)	1.1 (3.2)
Non-Well-Formed	1156 (393)	1.1 (2.6)
Δ	5	0.0

The data indicates that masked priming with syntactically well-formed sub-expressions provides no significant advantage over masked priming with non-well-formed sub-expressions in either response time ($F_1 < 1$, $F_2 < 1$) or accuracy ($F_1 < 1$, $F_2 < 1$). The absence of a priming effect suggests that experienced users of mathematics do not have lexical-like internal representations of the well-formed sub-expressions used in Experiment 1, indicating that the results of that experiment were not a function of "lexical" level representations. This, of course, does not exclude the possibility that some collections of symbols are represented as lexical-like tokens. For example, this may well be true for common sub-expressions such as x^2 .

This result is not surprising, since if the sub-expressions in the first experiment were stored as lexical-like tokens, there would need to be a vast number of them in long term memory. Although long term memory is capable of storing large numbers of tokens (for example, Chinese text is composed of thousands of characters), the algebraic sub-expressions appear to be more like rule-governed collections of tokens, where the tokens are the individual symbols. It appears therefore, that the internal representations used by experienced users of mathematics to encode algebraic expressions must depend on processing that takes place at a level beyond "lexical" processing.

2.5 The Role of Post-Lexical Processes

Experiment 4 draws on the more traditional, visible priming paradigm, in which the prime is presented unmasked for a duration typically in the order of 500ms, before the target stimulus is shown. Thus, participants are consciously aware of the prime before they see the target stimulus. This difference means that visible priming can be used to determine the influence of different prime types on processes that occur beyond the level of "lexical" processing.

Experiment 4 uses visible priming to test the hypothesis that the encoding of algebraic expressions involves processes that occur beyond the level of "lexical" processing.

Experiment 4 differs from Experiment 3 in one aspect only; the primes are visible to the participants. If the encoding of algebraic expressions does involve processes that occur beyond "lexical" processing, then one would expect to see a task performance advantage for expressions that are primed with syntactically well-formed sub-expressions, over those that are primed with non-well-formed sub-expressions.

Experiment 4

Method

Participants Twenty-eight experienced users of mathematics successfully completed this experiment. No data from any participant was excluded. The criteria for inclusion were that same as for Experiment 3.

Materials and Design The expressions and sub-expressions used in this experiment are identical to those used in Experiment 3 (see Table 2.5 for examples). The expressions were either syntactically valid or invalid, and the sub-expressions were used as primes.

The design of the experiment is the same as for Experiment 3, with two counterbalanced versions being constructed. Again, the items of each version were presented in a different pseudo-random order for each participant. Some, but not all of the participants in this experiment were also participants in Experiment 3. Due to this, Experiment 4 was conducted at least six weeks after Experiment 3, to minimize any possible recollection of the items used.

Procedure The environment for this experiment was identical to that in Experiment 3, with both experiments also having a very similar procedure. The only difference is that in this experiment, visible priming rather than masked priming is used. Items were presented in the centre of the screen in the following sequence. The prime was presented first for a duration of 500ms. Then the target expression was presented. The participants were instructed to pay attention to the prime, but also made aware that their aim in the task

was to determine if the expression presented after the prime was syntactically valid, and that they were not required to make judgements about the prime itself.

The response mechanism and feedback were the same as for Experiment 3 and again participants were asked to try and respond as quickly as possible while trying not to make any errors. Participants took about ten minutes to complete the experiment.

Data Treatment Data treatment is the same for this experiment as it was for Experiment 3.

Results and Discussion

The mean correct response time and error rate for the different prime types in the experiment, are summarized in Table 2.7 along with the corresponding standard deviations (in parentheses). Planned comparisons of the data were conducted using two-way ANOVAs (versions \times sub-expression), carried out separately over participant and item data.

Table 2.7: Mean correct response times (ms) and error rates (%) as a function of prime type for Experiment 4

Prime Type	RT(ms)	%Error
Well-Formed	1086 (400)	1.0 (2.4)
Non-Well-Formed	1126 (406)	1.2 (2.6)
Δ	40	0.2

There is a significant 40ms response time advantage in favour of syntactically well-formed sub-expressions over non-well-formed sub-expressions ($F_1(1, 26) = 4.70, p < .05$, $F_2(1, 28) = 4.22, p < .05$). The analysis for error rates however, shows no significant advantage for well-formed primes ($F_1 < 1, F_2 < 1$). The response time results show that visible priming with syntactically well-formed sub-expressions provides a significant advantage over visible priming with non-well-formed sub-expressions, in the task of determining the

syntactic validity of algebraic expressions. This priming effect indicates that well-formed sub-expressions play a role in aiding responses to the expressions presented in the task. This corresponds with the results of Experiment 1, in which well-formed sub-expressions were also found to play a role in the encoding of algebraic expressions.

The results of Experiment 3 indicated that internal representations consisting of lexical-like tokens stored in long term memory, are not the primary basis by which experienced users of mathematics encode algebraic expressions. This combines naturally with the result of Experiment 4 to indicate that the encoding of algebraic expressions is based primarily on processes that occur beyond the level of "lexical" processing. These results then, are consistent with the idea that it is syntactic processing that plays the primary role in the encoding of algebraic expressions by experienced users of mathematics.

However, while these results suggest that mathematical syntax is important in the encoding of algebraic expressions, they do not reveal the level of syntactic structure that underlies the internal representations used; in particular it does not reveal whether or not the encoding of algebraic expressions reflects the syntactic structure of the entire expression. For example, it may be that the internal representation relies upon a set of syntactic templates which are used to break an expression into a collection of well-formed sub-expressions. This would result in an internal representation that could produce the results seen in the first four experiments, but such an encoding would not necessarily capture the correct constituent structure of a mathematical expression. Thus, this possibility seems to be unlikely. Instead, a more likely possibility is that the internal representations used are based on a hierarchical structure, similar to that used to describe the syntactic structure of sentences of natural language. Not only does this possibility better explain the ability of experienced users of mathematics to successfully manipulate memorized expressions, but since the results of the first four experiments indicate that there are some similarities between how sentences of natural language and mathematical expressions are encoded, it seems reasonable to consider that these processes might also have other features in common.

2.6 Syntactic Encoding with Phrasal Structure

As discussed in Section 1.1.1, the transformational grammars used to understand how humans parse natural language consist of phrase structure rules and transformational rules. Parse trees illustrate the hierarchical relationships among constituents as defined by the phrase structure rules, showing how lexical units are grouped together into larger constituents. These constituents are then also grouped into larger constituents, until they eventually combine to form a sentence.

Despite the fact that transformational grammars are not sufficient to describe standard mathematical notations, the more powerful grammars that can describe such notations (such as the attributed multiset grammars discussed in Section 1.1.3) still allow the symbols in an expression to be grouped into constituents that are analogous to phrases. Therefore, just as sentences contain constituent phrases such as noun and verb phrases, mathematical expressions can be split up into constituents that are based on mathematical operations, such as addition or multiplication, allowing their phrasal structure to be determined. Consider as an example, the following grammar which describes a subset of algebraic notation.

$$\begin{array}{l}
 E \rightarrow T \\
 E \rightarrow E + T \\
 E \rightarrow E - T \\
 T \rightarrow F \\
 T \rightarrow FT \\
 F \rightarrow (E) \\
 F \rightarrow \textit{number} \\
 F \rightarrow \textit{variable}
 \end{array}$$

In this grammar, an expression is defined by E , with the tokens being the symbols $+$, $-$, $($, $)$ and *numbers* and *variables*. The constituent types that make up the rest of the phrasal structure are a term (T), and a factor (F). Note that there is more than one way to define each constituent type. This example restricts itself to the subset of algebraic notation that contains only a one-dimensional structure (it does not define fractions or exponentials), so that attributes are not required in the grammar. However, the idea of

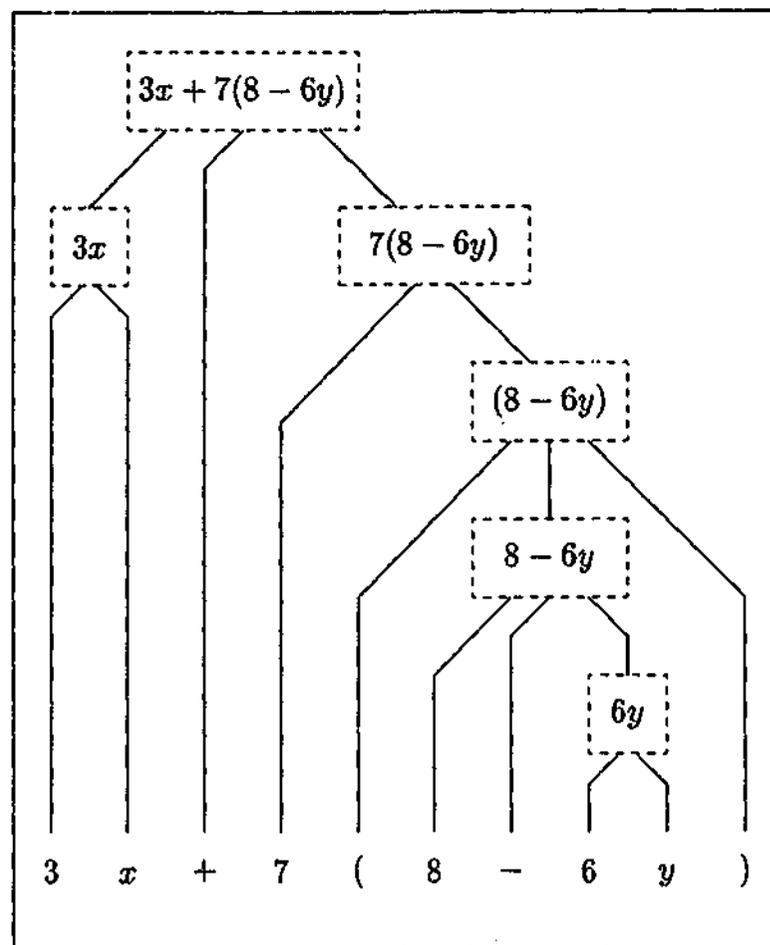


Figure 2.2: Parse tree for $3x + 7(8 - 6y)$

decomposing mathematical expressions into phrasal constituents applies just as readily to two-dimensional mathematical notations.

Consider the expression, $3x + 7(8 - 6y)$. A parse tree which illustrates the constituent structure of this expression, as defined by the simple grammar given above, is shown in Figure 2.2. Note that it is possible to define many different grammars for describing the syntax of mathematical expressions, all of which will result in the same constituent structure.

In the parse tree in Figure 2.2, each node represents a phrasal constituent, which is a syntactically well-formed sub-expression of the expression given. However, not all syntactically well-formed sub-expressions that appear in the expression are phrasal nodes occurring in the parse tree. For example, both $8 - 6y$ and $3x + 7$ are well-formed sub-expressions, but while the former is a phrasal node in the parse tree (which could be considered a subtraction phrase), the latter is not, despite being a valid mathematical expression in its own

right.

To test if experienced users of mathematics encode algebraic expressions in a way that reflects the phrasal structure in the parse tree, a recognition task similar to that used in Experiment 1 was constructed. The task was designed to determine if participants can more readily recognize sub-expressions of an algebraic expression that form a phrasal node in the parse tree (for example, $8 - 6y$ from the expression in Figure 2.2), as opposed to sub-expressions that are also syntactically well-formed, but do not form a phrasal node in the parse tree (such as $3x + 7$ from the expression in Figure 2.2). Note that all of the well-formed sub-expressions presented in Experiments 1 and 2 were phrasal nodes in this sense. If experienced users of mathematics encode algebraic expressions in a way that takes phrasal structure into account, a significant performance advantage in recognizing sub-expressions that form a phrasal node in the parse tree, over those that do not, would be expected.

Experiment 5

Method

Participants Twenty-four participants, all experienced users of mathematics, successfully completed the experiment. Data from an additional eight participants were excluded due to excessive error rates, with the criteria for exclusion in this experiment being the same as for Experiment 1.⁶

Materials and Design Seventy-five algebraic expressions were constructed, all consisting of between twelve and fourteen characters. As with the previous experiments in this chapter, the expressions contained at most one fraction and the variable names were x and y . For each expression, sub-expressions of three types were constructed:

- a) A *phrasal sub-expression*, which is a well-formed component of the expression from which it is drawn, and is a phrasal node in this expression's parse tree. It conveys the

⁶As with Experiment 1, it was found that the high number of exclusions in this experiment did not affect the experimental outcomes.

Table 2.8: Example expressions and sub-expressions used in examining phrasal properties.

Expression	Sub-Expression		
	Phrasal	Non-Phrasal	Incorrect
$(8y - 8x)(4y + 1)$	$4y + 1$	$y - 8x$	$4y - 9$
$y = 8 + \frac{8y - 9x^2}{7y^6}$	$8y - 9x^2$	$8y - 9$	$8x + 9$
$\frac{7x}{8(2y - y^3x^4)}$	y^3x^4	$y - y^3x$	$y - x^3y$
$3 - \frac{5}{7 - x^2(8 - y)}$	$(8 - y)$	$7 - x^2$	$2 - x^4$

same meaning on its own that it conveys in the expression.

b) A *non-phrasal sub-expression*, which is also a well-formed component of the expression from which it is drawn, but does not convey the same meaning on its own that it conveys in the expression. It is not a phrasal node in the expression's parse tree.

c) An *incorrect sub-expression*, which is not part of the original expression. It is also a well-formed expression. These act as fillers.

Each of the sub-expressions contained between four and six characters. The average for phrasal sub-expressions was 4.78; for non-phrasal, 4.54; for incorrect, 4.60. Again, the difference between these averages is not statistically significant. See Table 2.8 for examples of expressions and sub-expressions used.

As in Experiment 1, three counterbalanced versions were constructed to avoid practice effects, with the items of each version being presented in a different pseudo-random order for each participant. For each version, there were twenty-five instances of each type of sub-expression. Two additional expressions were constructed as practice items.

Procedure The procedure followed that of Experiment 1, with the environment, display timing of the stimuli, participant response mechanism and feedback being the same. The average width of the expressions in pixels was 187 (range 91–244) with an average height of 45 (range 26–59). The average width of the sub-expressions in pixels was 74 (range 25–111) with an average height of 23 (range 16–52). Participants took approximately ten minutes to complete the task.

Data Treatment The same measures used in Experiment 1 to reduce the unwanted effects of outlying data points, were applied to the data. It was necessary to exclude two items from the final analysis due to error rates in excess of 75%. One further item also had to be removed in order to balance the number of items in each version of the experiment. As a result, the final analyses were over twenty-four items per condition, not the original twenty-five. Response time and error data were analysed by a series of analyses of variance (ANOVAs) over both participant and item data.

Results and Discussion

The mean correct response time and error rate for the three sub-expression types are summarised in Table 2.9, along with the corresponding standard deviations (in parentheses). Analysis of the data was conducted using two-way ANOVAs (versions \times sub-expression).

There was a significant response time advantage for phrasal sub-expressions over non-

Table 2.9: Mean correct response times (ms) and error rates (%) as a function of sub-expression type for Experiment 5.

Sub-Expression	RT(ms)	%Error
Phrasal	1153 (178)	14.8 (8.7)
Non-Phrasal	1349 (205)	25.2 (11.2)
Incorrect	1382 (246)	20.3 (9.5)

phrasal sub-expressions, despite the fact that the latter were well-formed and appeared in the full expression (196ms difference; $F_1(1,21) = 68.06, p < .05, F_2(1,69) = 64.29, p < .05$). Phrasal sub-expressions were also responded to significantly faster than were incorrect sub-expressions (229ms difference; $F_1(1,21) = 64.95, p < .05, F_2(1,69) = 80.60, p < .05$). Error rate analysis also reveals a significant advantage for phrasal sub-expressions over non-phrasal sub-expressions ($F_1(1,21) = 18.83, p < .05, F_2(1,69) = 16.61, p < .05$), while there was also a strong trend favouring phrasal sub-expressions over incorrect sub-expressions, although the participant-based analysis did not quite reach significance ($F_1(1, 21) = 4.21, p = .053, F_2(1, 69) = 5.46, p < .05$). There was no significant difference between non-phrasal and incorrect sub-expressions, for either response times or error rates.

These outcomes are consistent with the claim that algebraic expressions are perceived in a way that allows for faster and more accurate recognition of phrasal sub-expressions than non-phrasal sub-expressions.

The results suggest that the internal representations used by experienced users of mathematics to encode algebraic expressions, are based on the phrasal structure of the expressions. The chunks that an expression is encoded into are not rapidly combined to reconstruct just any syntactically well-formed sub-expression. Rather, the chunks favour reconstruction of those sub-expressions that form phrasal constituents of the original expression.

2.7 Summary

The results presented in this chapter show that the internal representations used by experienced users of mathematics to encode algebraic expressions strongly rely on a knowledge of mathematical syntax that reflects phrasal structure. Evidence has been presented that directly supports this conclusion, as well as evidence that accounts for competing explanations of the data.

The first experiment began with the observation that standard mathematical notation and natural language text share many common features. It was natural therefore, to con-

sider the possibility that experienced users of mathematics might encode mathematical expressions in a manner similar to the way in which people encode sentences. A memory recognition task was constructed to examine the role of mathematical syntax in encoding algebraic expressions. The results showed that experienced users of mathematics can more readily identify those parts of a previously seen expression that are syntactically well-formed, than those that are non-well-formed, indicating that syntax plays an important role in the internal representations used to encode algebraic expressions. However, other competing ideas also needed to be considered.

One of these competing ideas was that the encoding of algebraic expressions by experienced users of mathematics may be a consequence of visual properties rather than syntactic structure. This notion is based on the observation that the constituents that would be expected if algebraic expressions were encoded according to syntactic structure, are similar to the constituents that would be expected if algebraic expressions were chunked based on the visual processes of perceptual organization. Experiment 2 examined this possibility, again using a memory recognition task. However, the results indicated that syntactic structure plays a more important role in the encoding of algebraic expressions by experienced users of mathematics, than do visual features. Nonetheless, it seems reasonable to suggest that the syntax of mathematical notations may have evolved such that the processing of mathematical expressions can benefit from the visual processes of perceptual organization.

One other possibility that could have accounted for the results seen in the first two experiments is that experienced users of mathematics may have internal representations that consist of lexical-like tokens stored in long term memory. These tokens could be common combinations of symbols, similar to the syntactically well-formed sub-expressions used in Experiment 1. If enough of these tokens are present in long term memory, then algebraic expressions could be encoded into groups of these lexical-like tokens in much the same way that the characters in a natural language sentence are grouped into words. This would suggest that the processing that occurs at a "lexical" level might play a more important

role than syntactic processing in the encoding of algebraic expressions by experienced users of mathematics. Experiment 3 made use of the masked priming paradigm to examine this idea. The data from this experiment suggests that the results of Experiment 1 cannot be explained by the idea that the internal representations used by experienced users of mathematics consist of lexical-like tokens of well-formed sub-expressions.

The results of Experiments 2 and 3 indicate that visual features and "lexical" processes do not play a more important role than syntactic structure in the encoding of algebraic expressions. However, to confirm that syntactic processing is the primary mechanism used to encode algebraic expressions, we needed to show that the encoding process occurs at a level beyond "lexical" processing. This was done using visible priming in Experiment 4.

While the results of the first four experiments suggest that mathematical syntax forms the primary basis for the encoding of algebraic expressions, they did not reveal the level of syntax involved. To examine this, a further recognition task was constructed to examine the phrasal structure of algebraic expressions. The results indicate that there is a recognition advantage for sub-expressions that form a phrasal constituent of the original expression, over other well-formed sub-expressions that do not represent phrasal constituents. This result not only highlights that syntax plays an important role in the encoding of algebraic expressions, but also indicates that the correct constituent structure of the expression is captured in the encoding process. Thus, the results of all five experiments in this chapter combine to support the conclusion that experienced users of mathematics use internal representations based on mathematical syntax and the phrasal structure of a parse tree, to encode algebraic expressions.

The results discussed in this chapter are consistent with what other researchers have discovered, and add to our understanding of the internal representations used by experienced users of mathematics to encode algebraic expressions. Kirshner (1989), for example, examined the role of spatial information in processing mathematical notations, using a task in which algebraic notation was replaced by two different notations, one with spacing simi-

lar to algebraic notation, and one with no spacing. He found that spatial information plays a significant role in the processing of mathematical notations. The results in this thesis do not contradict this idea, but rather put it into context with regard to the role of mathematical syntax; spatial information may be important, but syntax forms the primary basis of the internal representations used by experienced users of mathematics to encode algebraic expressions. The results also concur with the findings of Ranney (1987), which suggest that the recognition of algebraic expressions is based on structural content. Ranney also found that when viewing an algebraic expression, no effect similar to the word superiority effect occurs. This is in agreement with the results of Experiment 3, which indicated that algebraic sub-expressions are not processed in the same way as words (that is, as lexical-like tokens).

It has also been noted by many researchers (for example, Sfard & Linchevski, 1994) that the information that is extracted from mathematical expressions can be task dependent in nature. However, three of the experiments in this chapter did not place the expressions presented in any form of mathematical task. They were recognition tasks, and as such did not require the encoding of the expressions in any way that directly draws on mathematical knowledge. Nevertheless, the outcomes of each of these experiments indicated that the encoding of algebraic expressions is guided primarily by mathematical syntax. This outcome is consistent with the conclusion that such knowledge is fundamental in processing algebraic expressions.

Given that experienced users of mathematics use internal representations that are based primarily on syntax in the encoding of algebraic expressions, the next step is to consider how such expressions are parsed. This is the focus of the following chapters.

Chapter 3

Restricted Focus Viewer

The results of the experiments in Chapter 2 showed that the encoding of algebraic expressions by experienced users of mathematics is based primarily on phrasal structure. This leads to the question of how such expressions are parsed. However, before this issue can be addressed, it is first necessary to examine which methodological techniques are appropriate for such an investigation. This is the focus of this chapter.

Eye-tracking equipment is an important tool that has been used in understanding how natural language text is parsed. Eye-tracking equipment records which region of a visual stimulus is being looked at by a participant at any moment, allowing the focus of visual attention to be tracked. Since complex visual representations can rarely be taken in at a single glance, following the focus of visual attention can provide important insight into the strategies used to comprehend such representations. For this reason, eye-tracking equipment has been of great benefit to researchers in examining the processes involved in the comprehension of visual stimuli, such as those used in reading (Rayner, 1998), diagrammatic reasoning (Hegarty, 1992), cartography (Steinke, 1987), scene perception (Rayner & Pollatsek, 1992) and cognitive processes in general (Just & Carpenter, 1976; Yarbush, 1967).

However, eye-tracking equipment varies greatly in both resolution accuracy and sampling rate, and in the context of algebraic expressions, the need for high quality eye-tracking equipment is very important. This is largely due to the fact that algebraic expressions require exceptionally accurate calibration in order to obtain useful eye movement data, unless

$$\begin{array}{r} 301 \\ 589 \\ + 734 \\ \hline \end{array}$$

Figure 3.1: Example of the type of problem used by Suppes et al. (1983).

the expressions are made unrealistically large. Factors that influence the need for accurate calibration include the two-dimensional layout of algebraic expressions, and the variation in symbol sizes through the use of subscripts and superscripts (for example, consider the size difference in the expression $(2x - 7)^{2x-7}$ between the base and exponent components). More importantly, the lexical units of an algebraic expression are typically single symbols, rather than collections of symbols (as seen in the results of Experiment 3, which showed that typical sub-expressions consisting of 4 to 6 characters are not stored as lexical-like units in long term memory). Also, unlike natural language text, the lexical units of algebraic expressions are not always separated by a consistent space.

Given these considerations, it is perhaps not surprising that there appears to have been very little eye-tracking research involving mathematical notations. Suppes, Cohen, Laddaga, Anliker, and Floyd (1983) for example, examine the eye movements of participants while they are doing simple arithmetic problems. However this research only looks at a single very specific layout of addition and subtraction problems, such as the one shown in Figure 3.1. Salvucci and Anderson (1998) examined the cognitive processes involved in solving simple algebraic equations, however, in this work only one-dimensional notations were used (for example, using a forward slash, '/', for division instead of fractions, and not using exponents), and the symbols in the expressions were very widely spaced (to make up for limitations on the accuracy of eye-tracking equipment) resulting in an unnatural appearance.

Since eye-tracking equipment was not readily available for the research conducted in

this thesis, a new computer based tool for tracking visual attention was developed: the Restricted Focus Viewer (RFV). The RFV allows the tracking of visual attention directed towards an image presented on a computer monitor. This tool is cheap, easy to set up, and the settings can be tailored to allow the stimulus elements of interest to be investigated. It is non-intrusive and requires no calibration. The RFV may also be a useful tool for other researchers who do not have ready access to eye-tracking equipment with a sufficient resolution accuracy or sampling rate for a particular task.

The RFV uses image blurring to restrict how much of the image can be clearly seen, with only a small region in focus. The region of focus can be moved around using a computer mouse. The idea behind using restrictions in the visual field is not new. In research on reading for example, the number of characters that can be processed in one fixation has been examined by using visual restrictions (Rayner & Pollatsek, 1989; Osaka & Oda, 1994). Image blurring has been used to understand how people take in information from software manuals (Ummelen, 1997). Studies have also been conducted which involve the use of artificial scotomas (blind spots) to disrupt visual processing (Henderson, McClure, Pierce, & Schrock, 1997), and the notion of a movable window has been used before in examining visual search (Stark et al., 1992). However, the difference is that in this case these features are combined in a generic configurable tool.

While other simple tools have previously been created that use a computer mouse to track where attention is being directed, they do not approach the level of sophistication of the RFV. 'Mouselab' for example (Payne, Bettman, & Johnson, 1993), involves presenting the user with a stimulus consisting of several opaque 'boxes' (rectangular regions on the screen) which conceal stimulus components. By moving the mouse pointer over a box, the content of that box is revealed. When the mouse is moved off that box, the content is concealed again. The program collects data about the time at which various boxes are revealed by the user, and for how long.

However, such tools have several important limitations. The nature of the opaque boxes

destroys many visual cues about the stimulus components that they conceal, making it impossible for a user to take in a broad overview of the stimulus in question. Labels are often added to guide the user in exploring the stimulus, but as these are not concealed, the program cannot determine when the labels are being looked at. The sudden revealing or concealing of box content is unnatural and can be distracting. Also, it has been recognized that, despite mouse-based data from tools such as 'Mouselab' being less noisy and variable than eye-tracking data, the nature of the tool can sometimes alter the user's strategy in a particular task (Lohse & Johnson, 1996). For certain types of stimuli and tasks, these limitations may not present a significant problem. However, for many tasks, including examining how algebraic expressions are parsed, the use of such a tool is clearly inappropriate.

While the RFV is certainly related to tools such as 'Mouselab', it is significantly further developed, and has been designed to overcome many of the limitations seen in previous tools. In the next section of this chapter, the implementation of the RFV is described in detail. It is a generic computer based tool, explicitly designed for research into how humans reason with and comprehend visual stimuli such as algebraic expressions. Features include graded blurring, motion blur and a data replay tool. Subsequent sections describe both a qualitative and a quantitative analysis of the RFV. This includes an empirical validation of the RFV, in which results obtained from a mental animation experiment using eye-tracking equipment are compared with results obtained using the RFV instead.

3.1 Restricted Focus Viewer Tool

The Restricted Focus Viewer (RFV) is a computer program which takes a visual stimulus, blurs it and displays it on a computer monitor, allowing the participant to see only a small region of the stimulus in focus at any time. The region in focus can be moved using the computer mouse. The RFV records what the participant is focusing on at any point in time, and the data can be played back using a replayer. In this section the RFV and the

replayer are described in more detail.

3.1.1 Description of the RFV

The human visual system can only focus on objects at the centre of the visual field. The region surrounding this area of sharp focus is still perceived, but the further from the centre of the visual field an object is, the more coarse is the perception of it (Tovée, 1996). Most of the time, visual attention is directed to the centre of the visual field, although this is not always the case as it is possible to covertly attend to other locations (Coren, Ward, & Enns, 1994).

The RFV has been designed to reflect these aspects of the human visual system. The key idea is that only the part of the image under the focus window is shown clearly, with the remainder of the image out of focus. To keep most of the visual stimulus out of focus, the RFV uses image blurring. Figure 3.2 gives an example of this with an algebraic expression as the stimulus. The original expression is shown on the left, with a blurred representation of it on the right. The blurred image still allows the general form of the expression to be perceived, thus allowing the user to move directly from one region of the image to another. However, it does not reveal the finer details of expression, with the individual symbols being indiscernible unless the user of the RFV specifically focuses on them with the focus window. (Note that some of the blurred images in this chapter have been made slightly darker so that they are clearer when printed. Some detail has been lost due to a reduced greyscale when printing.)

It is clear from the example in Figure 3.2 that large structural features of the expression (such as the fraction) are suggested in the blurred image. The degree of blurring required for a specific type of visual stimulus depends on the size and visual characteristics of the

$$4 + \frac{x + 7}{9(4 + 9y)}$$



Figure 3.2: Example of a visual stimulus and its corresponding blurred image.

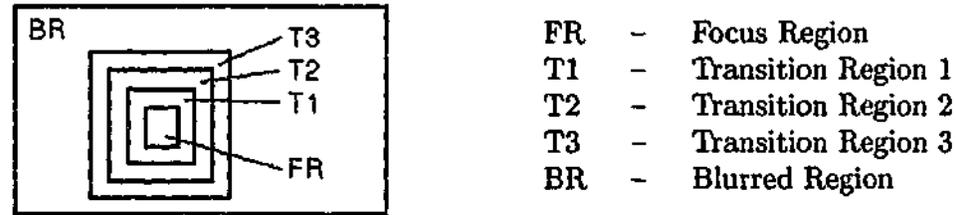


Figure 3.3: Regions of the stimulus used to achieve the graded blurring effect.

stimulus elements of interest. This is discussed in more detail later in this section.

The *focus window* of the RFV is the region in which the stimulus is visible in full detail. Two important issues concerning the focus window for the RFV are firstly, how "natural" it will look, and secondly, how big it should be.

Initial experience with the focus window during the development of the RFV suggested that it is not sufficient to simply have a box on the screen in which the stimulus is in focus, while the rest of the image is blurred. The boundary between the two regions is too distinct, leading to a very unnatural effect that users found distracting. Also, a sharp cut-off will enable the participant to guess about neighbouring parts of the image from features near the edge of a reasonably sized focus box, whereas a smaller focus box eliminates cues that are needed for successful navigation to neighbouring elements. A graded blurring effect, such that the transition from blurred to focus appears smooth and seamless, is needed to prevent this.

A graded blurring effect is achieved by the technique illustrated in Figure 3.3. The outer rectangle defines the stimulus area which is fully blurred. The innermost box is the region of focus. Surrounding this focus region are three transition regions. Each transition region is slightly more blurred than the last, so that there is only a subtle difference between neighbouring regions. The overall result is the appearance of a smooth transition from the region of the image in focus, to the region which is fully blurred. Using the mouse to move the focus window therefore moves not only the focus region, but also the transition regions.

Figure 3.4 gives two examples of the focus window positioned over different regions of the expression shown in Figure 3.2. The size of the focus window is determined not only

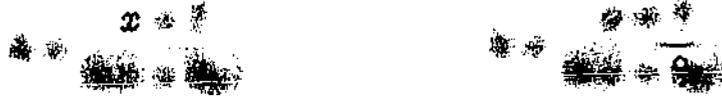


Figure 3.4: Two examples of the focus window on different regions of the stimulus.

by the dimensions of the focus region, but also by those of the three transition regions. However, the two most important sets of values to be considered are the size of the focus region box, and the size of outermost transition region box. For the experiments discussed in this thesis, the focus region and transition region boxes are all square, and centred around the current mouse co-ordinates, although this need not be the case. At the centre of the focus window there is also a small grey dot, to allow users to keep track of the focus window location when it is placed on an empty region of the image.

The degree of blurring and the size of the focus region need to be adjusted in accordance with the size of syntactic elements in the stimulus (for algebraic expressions, the syntactic elements are the symbols). Experience with the RFV to date has led to the development of the guidelines described in Table 3.1, which can be applied to determine these parameters. Note that although these parameters can be adjusted easily for a given experiment, the RFV tool does not allow them to be changed during an individual trial. This is to prevent inconsistencies in the way that the participant views the stimulus.

Another feature that was implemented so that the RFV would more accurately mimic the way humans perceive visual stimuli is *motion blur*. This is based on the fact that during saccadic eye movements, visual information is not processed (Rayner & Pollatsek, 1989). Thus, if the user of the RFV moves the mouse at high speed (that is, over a large distance on the screen in a small amount of time), the focus window will not achieve full focus. Once the user reduces the speed of the mouse motion back to below a certain threshold, or stops moving the mouse completely, full focus in the focus window will return. This feature helps in defining the temporal boundary between fixations and movements.

When the focus window is stationary or moving slowly, all of the regions listed in

Table 3.1: Guidelines for setting RFV parameters.

RFV Parameter	Setting Guidelines
<i>Focus Region Size</i>	
Goal	Should be slightly smaller than the bounding box of a typical stimulus element
Lower Limit	Must allow recognition of any one element of the stimulus when region is centred over the element
Upper Limit	Should prevent simultaneous recognition of two neighbouring elements when placed between them
<i>Transition Region Size</i>	
Goal	Should indicate the direction of neighbouring connected elements
<i>Level of Blurring</i>	
Lower Limit	Should be sufficient that any two elements are indistinguishable and that overall connectivity cannot be established
Upper Limit	Should allow identification of stimulus boundaries (at least the overall shape)
<i>Motion Blur Onset</i>	
Goal	Should allow separation between fixation and movement
Lower Threshold	Should not allow "brass rubbing" strategy, that is, identification of stimulus by waving window rapidly over it
Upper Threshold	Should allow slow navigation with continuous focus over a connected stimulus when the task requires it

Figure 3.3 are present. During motion blur however, only the outermost transition region is present. Because this region has less blurring than the rest of the image, the user is still able to track the location of the focus window on the stimulus. However, it is not possible to determine the finer details of that location without slowing or stopping the mouse. Only then will full focus be available. Table 3.1 also describes guidelines for appropriate motion

blur settings; that is, how fast the mouse needs to be moving before motion blur occurs.

The RFV outputs a data file that records the motion of the focus window. For each trial, information about the trial is specified, followed by lines that contain details about each updated mouse movement. The lines are composed of five data items. First is the stimulus number (as more than one stimulus may be presented in a single trial). Second is the time elapsed (in milliseconds) since the RFV was initiated for that particular trial. The next two values are the x and y co-ordinates of the centre of the focus window with respect to the top left corner of the window containing the stimulus image. The final piece of information is a flag to indicate whether or not the focus window was motion blurred or not. This data allows the experimenter to exactly replicate the state of the RFV while the participant was performing the task. For a more complete discussion of user input and the output produced by the RFV, see Appendix C.

3.1.2 Data Replayer

The data replayer is a companion tool to the RFV that can read in a data file generated by the RFV, and replay the way that the RFV participant moved the focus window over the stimulus. This provides a useful experimental analysis tool, allowing the experimenter to review the participants actions. It can play back the focus trace at faster than real-time, and it can also draw a scan-path line over the original stimulus based on the locations of the centre of the focus box. Figure 3.5 gives an example of this for the algebraic expression shown in Figure 3.2. It can be seen that in this instance, the participant started at the left of the algebraic expression and began scanning to the right. The numerator of the fraction was scanned over from left to right once, with the participant then returning to

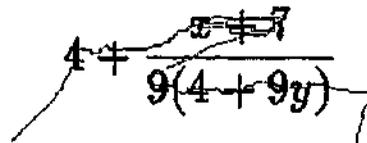


Figure 3.5: Example of a data replayer scan-path.

the beginning of the numerator and scanning it again. The participant then continued on to the denominator of the fraction, which was also scanned in a left-to-right manner.

3.2 A Qualitative Comparison with Eye-Tracking

In Experiment 6, the performance of the RFV in contrast to a specific (mid-range) eye-tracking system is examined in a qualitative manner, by using both techniques to explore how participants scan over simple algebraic expressions. The aim of this experiment is to compare how well the RFV and eye-tracking equipment can indicate where in an algebraic expression attention is being directed, and what problems can arise when obtaining this data. Being a qualitative comparison, the focus of the analysis is on a descriptive comparison of the two techniques, rather than a statistical comparison. The experiment has been designed such that a single group of participants uses both techniques, thus reducing any possible influences that could arise from individual differences. Note that for this particular experiment, we were not yet interested in explaining how participants scan over the algebraic expressions, just the relative performance of the RFV and eye-tracking equipment.

Experiment 6

Method

Participants Eight participants successfully completed the experiment. All were graduate or undergraduate students from the Psychology Department at the University of California in Santa Barbara. All participants were volunteers with normal or corrected-to-normal vision, and were familiar with algebra. Note that the participants in this experiment were not as experienced with algebra as the participants used in the other experiments described in this thesis. However, as the purpose of this experiment is simply to provide a comparison between the RFV and eye-tracking equipment, this is not a concern. Data from one participant was excluded due to an excessive error rate.

Table 3.2: Examples of the expressions used in Experiment 6, along with their corresponding statements and whether those statements were correct.

Expression	Statement	Correct
$(7 + 2y)^3$	The digit 4 appears in the expression	False
$\frac{x^2 - 3x}{4}$	When $x = 3$ the expression solves to equal 0	True
$4x^2 - \frac{x}{2}$	The expression contains the sub-expression $4y^3$	False

Materials and Design Twenty-four algebraic expressions were constructed, all consisting of exactly seven characters. The expressions contained at most one fraction and the variable names were x and y , since these are most commonly used. For each expression, a statement about the expression was constructed that was either true or false. Half of the expressions had true statements, while the other half had false statements. The statements varied in how difficult it was to judge their correctness. Some were simple, requiring the participant only to determine if a particular symbol was present in the expression. Others were more difficult, requiring the expression to be used in a calculation. Table 3.2 gives examples of the expressions that were created and their corresponding statements.

In order to monitor how participants view each expression using both the RFV and eye-tracking equipment, while ensuring that participants were presented with each expression only once to avoid practice effects, two counterbalanced versions of the experiment were constructed. For the first version, a group of twelve expressions were assigned to the RFV and a second group of twelve expressions were assigned to the eye-tracking equipment. For the second version, the same two groups of expressions were used, but assigned to the opposite techniques. Two additional expressions and statements were constructed for each group of twelve expressions to be used as practice items. The items for each group of expressions were presented in a different pseudo-random order for each participant.

Procedure Participants were initially presented with written and verbal instructions, giving examples of the sort of expressions and statements involved in the experiment. Half of the participants viewed the eye-tracking expressions before the RFV expressions, with the other half doing the experiment in the reverse order. For both the RFV and eye-tracking components of the experiment, items were displayed in black on a white background on a 20" monitor.

For the RFV component, the monitor was running at a resolution of 1024×768 , with the presentation of items controlled by an IBM compatible computer running a version of the RFV tool whose settings had been tailored to this experiment. Participants were seated comfortably at a viewing distance from the monitor of approximately 50cm. The average width of the algebraic expressions in pixels was 92 (range 61-120) with an average height of 38 (range 24-58). The RFV focus box had an edge length of 16 pixels, while the outermost transition box had an edge length of 30 pixels. This allowed one symbol to be viewed in focus at a time. The motion blur was set to a low tolerance, requiring the mouse to be stopped or moving very slowly in order for full focus to be maintained.

The only interface mechanism used by the participants was a standard computer mouse. Participants were initially given two practice items, allowing them to become familiar with the RFV tool, followed by the twelve experimental items. Progress was self-paced, with each trial initiated by a single click with the mouse on a button at the bottom of a blank screen containing the prompt "Click button to continue". This action started the timer (to provide a reference time for the rest of that trial), and a blurred image of the expression appeared in the centre of the screen. The participant could move the window of focus over different symbols in the expression by moving the mouse, allowing the entire expression to be read.

Once the participants had read the expression, they clicked a button at the bottom of the screen using the mouse. This stopped the timer and made the blurred expression disappear, replaced by the statement about the expression (which was not blurred). The

participants were required to determine if the statement was true or false with respect to the expression that immediately preceded it. At the bottom of the screen were two buttons, one labelled "TRUE" and the other "FALSE". When the participant had decided on the validity of the statement, they single clicked on the appropriate button. The RFV tool recorded the response given and the time taken. No feedback was given to the participant. Participants were instructed to try and read the algebraic expressions as quickly as possible, while still ensuring that they did not make too many errors when determining the validity of the statements.

The eye-tracking component of the experiment was conducted using an Iscan corneal-reflectance and pupil-center eye-tracker (Model RK-426). It has a resolution of less than one degree of visual angle, and samples the participants' gaze every 16 milliseconds. Participants were seated approximately 1m from the monitor, and a headrest was fitted comfortably to restrict head movements. The subject was asked to move as little as possible during the experiment. In order to better determine which part of each expression was being fixated on at any given time, the size of the algebraic expressions for the eye-tracking component of the experiment were larger, with an average width in pixels of 368 (range 246-478) and an average height of 149 (range 92-228). The screen resolution was also lowered to 800×600 . As a result, each expression subtended a larger visual angle than for the RFV component of the experiment (an average of 10.5° for the eye-tracking component, compared with 4.1° for the RFV component).

After the eye-tracking equipment had been calibrated (which is necessary for the successful tracking of eye movements), the participant was asked to fixate on an asterisk that appeared in the centre of the screen and to push a button to begin each trial. The algebraic expression then appeared in the centre of the screen, and the participant read the expression. Once the expression was read the button was again pushed, and the expression was replaced with a statement about the expression. After determining if the statement was true or false, the participant responded verbally and the experimenter recorded their

answer. As with the RFV component, two practice items preceded the twelve experimental items.

The experiment took approximately 25 minutes to complete.

Data Treatment Only one participant had an error rate in excess of 25%, and thus was excluded from further analysis. While error rates are considered, the time taken to determine if a statement is correct is not a primary interest, and thus is not analysed. The main focus instead is on how the algebraic expressions were scanned by the participants.

Results and Discussion

The primary purpose of the statements was to force the participants to carefully read the expressions. There was no significant difference in the error rates when using the RFV (10.5%, $sd = 7.4$) as compared to using the eye-tracking equipment (10.4%, $sd = 5.9$). However, participants took approximately 22% longer to view each expression when using the RFV (7.81 seconds, $sd = 2.86$), as opposed to when the eye-tracking equipment was used (6.40 seconds, $sd = 2.23$). This difference was analysed using a two-way ANOVA (versions \times interface type) over participant data, and found to be statistically significant ($F(1, 7) = 14.84, p < .05$). The slower performance is likely to result in part from the difference in the ballistic speed of human motor control of the arm and the eye. Also, the blurring of the stimulus with only a small region of focus means that one would expect participants to take longer in moving from one area of the expression to another area, as there is a loss of parafoveal vision. As the same participants completed both the eye-tracking and RFV components of the experiment, this difference is not due to individual differences. The slower performance indicates that for tasks that require fast responses, or are dependent on absolute speed measurements, the RFV would not be appropriate.

The main aim of this experiment is to compare how well both the RFV and eye-tracking equipment perform in tracking visual attention. Thus, in this experiment we do not analyse fixation or gaze durations, but instead focus on the locations in the stimulus where visual



Figure 3.6: Examples of eye-tracking scan-path data which shows good calibration of the eye-tracking equipment.

attention was directed.

The calibration procedure varied in difficulty between participants, and how well the calibration was maintained across the experimental items also varied between participants. Given that calibration of the equipment occurred immediately before the first items were presented, generally the first few items seen by a participant produced good scan-path data. Examples of these good scan-paths are shown in Figure 3.6. Note that even these good scan-paths would need some minor adjustments before further analysis. For example, much of the scanning around the x^2 in the left expression appears lower than it probably should be.

Even when good scan-path data is obtained, other factors need to be taken into consideration. A key issue is that at any point in the eye-tracking data, it is impossible to determine if the participant was focusing on a single symbol or a collection of symbols. For example, the exponent of the bracketed term in the right expression (the digit 3) is hardly scanned directly at all, yet it is an important part of the expression. It is therefore not clear from this example how much processing this symbol is receiving, since it appears that it is always being viewed in conjunction with neighbouring symbols.

This problem can be overcome, however, by supplementing the eye-tracking technique with contingent change techniques. Contingent change techniques were developed by McConkie and Rayner (1975) and Rayner (1975) in eye-tracking studies that investigated perceptual span during the reading of English text. These techniques involve restricting

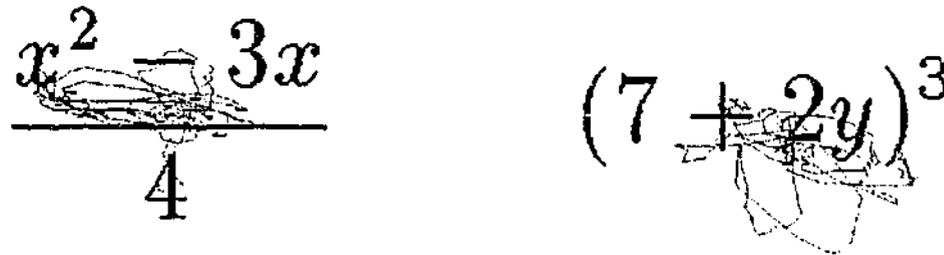


Figure 3.7: Examples of eye-tracking scan-path data which shows bad calibration of the eye-tracking equipment.

the amount of the stimulus that can be seen around the current eye fixation location. Each time the eyes move to view a new area of the stimulus, the area of restriction also moves. This technique is similar to that employed by the RFV.

A major concern with eye-tracking equipment is that calibration does not last. Even though the number of expressions in each version of this experiment was quite low, the calibration for all participants noticeably deteriorated during the task. Frequent re-calibration can be used to reduce this problem, although consideration must be given as to how the interruptions from re-calibrating the eye-tracking equipment might affect participant performance. Also, the data from several participants demonstrated that apart from gradual deterioration, the calibration can be suddenly lost at almost any moment (sometimes the reason is clear, such as if the participant moves too much, but often it is not). Examples of eye-tracking data when the calibration has effectively been lost are shown in Figure 3.7. These scan-paths are no longer useful as data and must be discarded.

In contrast, the RFV does not have any problems relating to calibration. No initial calibration is required, and by the nature of the RFV's design, it can never get out of alignment. Figure 3.8 gives examples of scan-paths obtained with the RFV on the same expressions shown in Figures 3.6 and 3.7. These scan-paths are much more directed, as they result from conscious movements of a computer mouse. Also, since the size of the focus window corresponds to the size of a single symbol, it is always easy to determine exactly what is being focused on. A further benefit of the RFV is that the stimuli can be



Figure 3.8: Examples of RFV scan-path data on the same algebraic expressions shown in Figures 3.6 and 3.7.

presented in a normal size, whereas for the eye-tracking system used in this experiment, the stimuli had to be made unnaturally large in order to obtain accurate scan-paths. For more accurate eye-tracking systems, however, this issue may not present a problem.

Thus, these results indicate that, for stimuli such as algebraic expressions, the RFV performs favourably as a tool for tracking visual attention when compared to the eye-tracking system used in this experiment.

3.3 Validation of the RFV

The results of Experiment 6 indicate that, for certain stimuli, the RFV is a suitable tool for tracking visual attention. However, before using the RFV in experimental work, an important issue is to determine if it interferes with the way in which humans comprehend visual stimuli. This issue has been previously raised by other researchers investigating the data from simpler mouse-based tools (such as 'Mouselab', Lohse & Johnson, 1996). Clearly, there is some difference since the participant must use a computer mouse rather than simple eye and head movements to change their focus. However, it is important to the validity of the RFV technique that this should not fundamentally affect the way in which visual stimuli are processed.

Research to quantitatively examine the validity of the RFV has been conducted by Blackwell, Jansen, and Marriott (2000).¹ This work replicated a mental animation experiment involving pulley system diagrams (Hegarty, 1992). The original experiment was carried out using eye-tracking equipment, with the replicated experiment using the RFV

¹This research was a collaborative effort conducted during the course of the PhD.

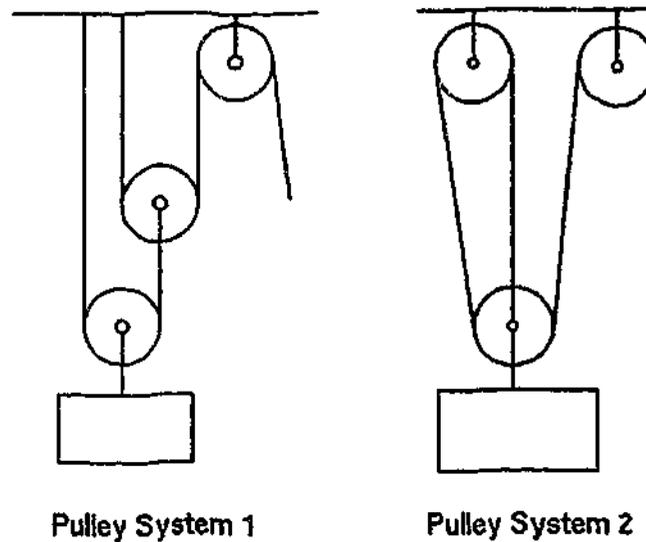


Figure 3.9: The two pulley systems used.

instead. The aim of the experiment was to determine if the RFV could produce results comparable to those produced using eye-tracking equipment.

For the replicated experiment (that is, the RFV experiment), two diagrams of pulley systems were constructed which were identical to those used in Experiment 1 in Hegarty (1992) (see Figure 3.9). They each consisted of three pulleys, a weight, braces from the ceiling to support some pulleys and sections of rope that were attached to the ceiling or weight, and went over or under the pulleys. In each system there was also a free end to the rope, and participants were required to infer the motion of the system components when the free end of the rope was pulled. The mirror images of these pulley systems were also used giving a total of four pulley system images.

For each pulley image there were twelve statements, each about the motion of one of the system components. Six of these statements were true and six were false.

When the free end of the rope is pulled in any pulley system, a causal chain of inferences can be made about the motion of the components. For example, in pulley system 1, pulling the rope causes the rope to move right over the upper pulley, turning it clockwise. From this knowledge it is possible to infer the motion of the middle pulley and so on. In this way, each pulley in the pulley systems can be defined as being at the beginning, middle or end of the causal chain of events. The statements about the motion of the pulley system

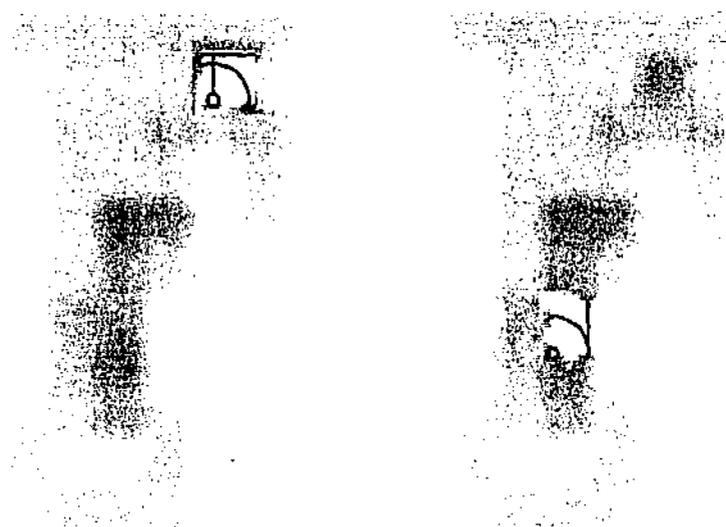


Figure 3.10: Two examples of the focus window on different regions of pulley system 1.

components are equally divided among the pulleys at each of these three locations in the causal chain. In pulley system 2, “the rope moves to the right under the lower pulley” is an example of a true statement about an event at the middle of the causal chain, and “the upper right pulley turns counterclockwise” is an example of a false statement about an event at the beginning of the causal chain.

Each statement was presented as a single line of text. A stimulus was composed of a statement appearing on the left, with the diagram of a pulley system on the right. This gave a total of 48 stimuli. Each participant was shown all 48 stimuli twice, with a rest between the two blocks.

The size of the RFV focus window was set to be slightly smaller than a single pulley. Examples of the focus window on different regions of pulley system 1 are given in Figure 3.10.

The participants were required to determine if the statement was true or false with respect to the pulley system presented. When the stimuli were presented, at the bottom of the screen were two buttons, one labelled “TRUE” and the other “FALSE”. When the participant had decided on the validity of the statement, he or she selected the appropriate button. The time taken to decide was recorded by the RFV, along with the response given. Participants were instructed to try and respond as quickly as possible, while still trying not to make too many errors.

Results and Discussion

The aim of the replicated experiment was to compare the results obtained using the RFV with eye-tracking results, to determine if the RFV affects the way in which humans comprehend visual stimuli, in a different manner to eye-tracking equipment. The main focus of the analysis was therefore to examine key data trends and significant results obtained in the original eye-tracking experiment, and see if the RFV results concurred.

Errors The overall error rate for the replicated experiment was considerably lower than that for the original experiment, probably due to participants in the replicated experiment having considerably more experience in dealing with technical diagrams. Despite this fact, the data trends were very similar.

The data comparisons were conducted using two-way ANOVAs (causal chain position \times repetition), carried out over participant data. In the original eye-tracking experiment, the position in the causal chain of the component referred to in the statement had a significant effect on error rates. This effect was also present in the RFV data ($F(2, 20) = 5.33$, $p < .05$). This can be clearly seen in Figure 3.11, where the RFV data from the replicated experiment on the right is compared with the data from the original eye-tracking experiment on the left. (Note that as the participants in the replicated experiment were more familiar

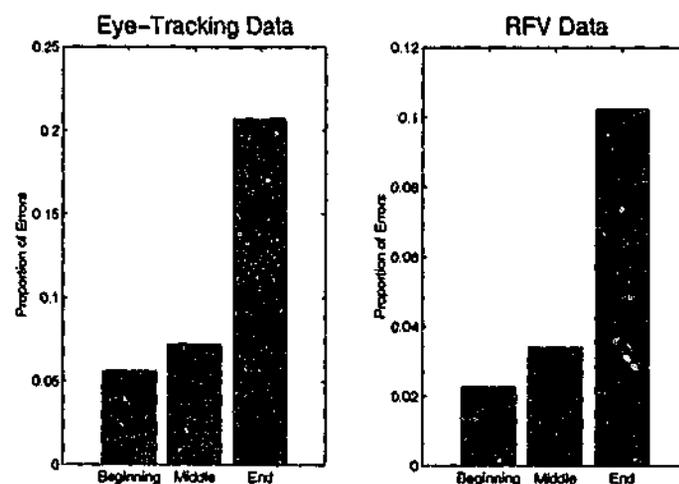


Figure 3.11: Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the proportion of errors at different causal chain positions.

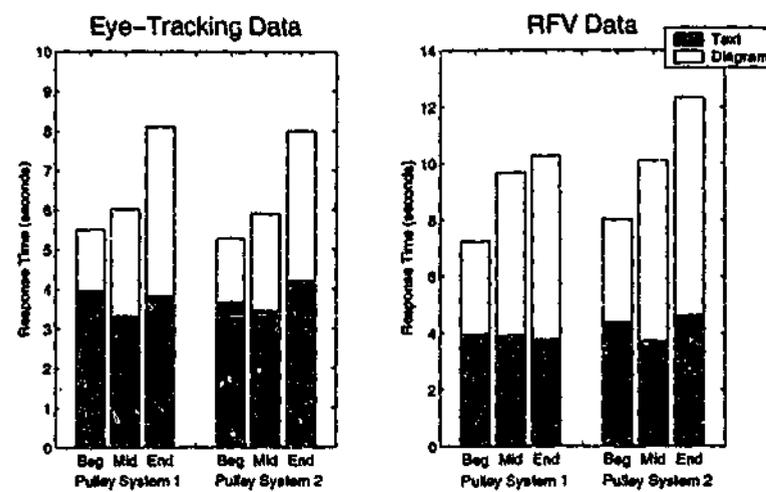


Figure 3.12: Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the mean response times for different trial types.

with technical diagrams than those in the original experiment, the overall error rate for the replicated experiment is lower.) There was also a trend for participants to make fewer errors on the second repetition of the stimuli (3.0%, $sd = 5.4$) than on the first repetition (7.6%, $sd = 12.5$), however this trend was not quite statistically significant, though approached it ($F(1, 10) = 4.52, p = .059$). This trend was also apparent in the eye-tracking experiment.

Response Times Figure 3.12 shows the mean reaction times (overall height of the bars) for each pulley system, for statements referring to components at different positions in the causal chain. As with the error graphs, the RFV data from the replicated experiment is shown on the right, with the eye-tracking data on the left for comparison. The times have been divided into the time spent reading the statement, and the time spent inspecting the diagram.

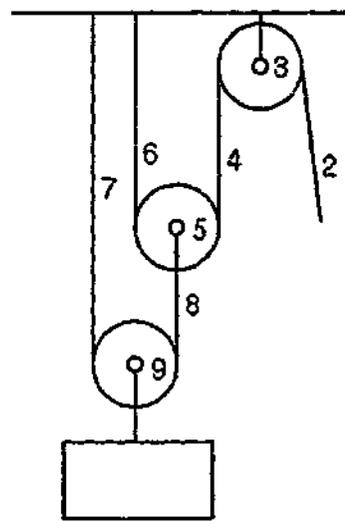
Response times for the two pulley systems were analysed separately, to allow for differences in the configurations. In the original eye-tracking experiment, the repetition caused a practice effect which resulted in participants responding significantly faster on the second repetition of the stimuli than on the first. The same effect was seen in the RFV data. The response advantage was 3.65 seconds for pulley system 1 ($F(1, 10) = 44.40, p < .05$), and 4.05 seconds for pulley system 2 ($F(1, 10) = 38.07, p < .05$). The original experiment also

showed that the position in the causal chain of the component referred to in the statement had a significant effect on response times. Again, the RFV data corresponded to the data from the original experiment, with position in the causal chain significantly influencing response times ($F(2, 20) = 18.87, p < .05$ for pulley system 1; $F(2, 20) = 24.15, p < .05$ for pulley system 2). This effect can be seen in Figure 3.12.

The results of the replicated experiment are clearly in agreement with those of the eye-tracking experiment. However, the participants using the RFV take approximately 50% longer to respond. As was discussed in Experiment 6, the increased response time is likely to result in part from the fact that the participants scan the stimuli using a computer mouse. Despite the extra time taken, the overall trends in the data are similar.

Further data analysis that was carried out in the eye-tracking experiment involved examining how long participants were inspecting different components of the pulley systems. In particular, for each statement, the components in the diagram were divided into those whose motion occurred before the component referred to in the statement, the referent itself, and those components whose motion occurred after the referent. This was also done for the replicated experiment, allowing for a further breakdown of response time spent viewing the diagram. Rectangular bounding boxes were used to enclose regions of the diagrams containing pulleys, rope strands, the ceiling and the weight, just as in the original eye-tracking experiment. This allowed the order of fixations on the components of the diagrams to be determined, along with how long those fixations were. Figure 3.13 gives an example of a fixation protocol taken from the RFV data.

The gaze duration was defined as the total time spent fixating on components, in certain locations in the causal chain with respect to the referent in the statement. The graphs of the gaze duration data are shown in Figure 3.14. Again, the RFV data from the replicated experiment is on the right, with the eye-tracking data on the left. The original eye-tracking experiment showed that when looking at the pulley system, participants spend most of their time inspecting the referent and the components whose motion precedes that of the



Gaze	Object Fixated	Gaze Duration
1	Statement	1884 ms
2	Pull rope	500
3	Upper pulley	595
4	Right upper rope	345
5	Middle pulley	1160
6	Left upper rope	180
7	Left lower rope	150
8	Right lower rope	61
9	Lower pulley	5917

Figure 3.13: Example of an RFV fixation protocol for the statement *The lower pulley turns counterclockwise*

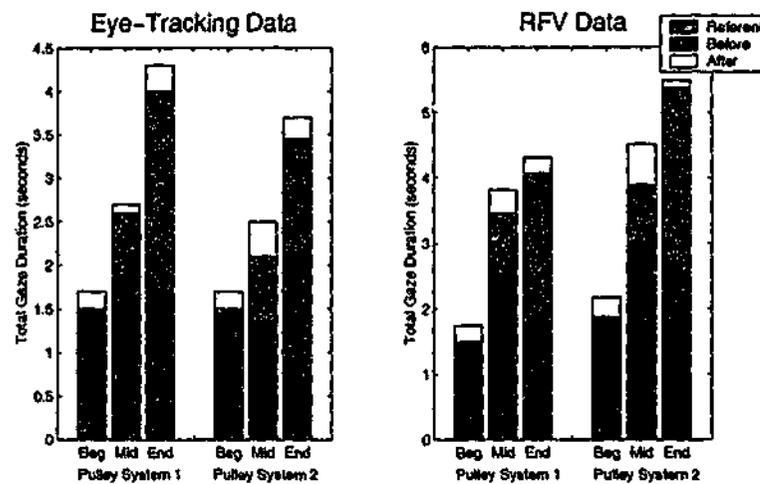


Figure 3.14: Comparison of eye-tracking data from Hegarty's original experiment and RFV data from the replicated experiment, examining the breakdown of gaze duration on different components of the pulley systems.

referent in the causal chain of events. The RFV data shows the same result for both pulley systems.

Due to the fact that many participants in the replicated experiment had more experience with technical diagrams than the participants in the original eye-tracking experiment, some minor differences in strategy would be expected. However, the results of the replicated experiment indicate that the response time, accuracy and gaze duration trends obtained from the original experiment using eye-tracking techniques, are the same as those obtained using

the RFV. There were no key significant results from the original eye-tracking experiment that the RFV experiment failed to obtain.

3.4 Summary

This chapter has examined a new tool for tracking visual attention: the Restricted Focus Viewer (RFV). This tool is a generic computer based tool, explicitly designed for research into how humans reason with and comprehend visual stimuli such as algebraic expressions. The motivation for creating the RFV was to provide an alternative technique to eye-tracking for researchers who might not have access to eye-tracking equipment with a sufficient resolution accuracy or sampling rate for studying stimuli such as algebraic expressions.

The results of Experiment 6 show that the RFV performs favourably compared to eye-tracking when used with algebraic expressions. The RFV has also been shown to perform well with other stimuli such as pulley system diagrams (Blackwell et al., 2000), with results indicating that using the RFV does not significantly affect the manner in which visual stimuli are processed. As a result, other researchers have also begun using the RFV with different stimuli (for example, tracking how attention shifts between text and graphics in documents that contain both, Futrelle & Rumshisky, 2001).

For the purpose of this thesis, the RFV provides a robust technique for tracking visual attention that is suitable for use with algebraic expressions. Thus, the RFV is an appropriate tool to be used in examining how experienced users of mathematics parse such expressions. The parsing of algebraic expressions is the topic of the next chapter.

Chapter 4

Parsing of Algebraic Expressions

The findings of Chapter 2 indicate that the encoding of algebraic expressions by experienced users of mathematics is guided by mathematical syntax, with individual chunks conforming to syntactically defined units. This naturally raises the question of how algebraic expressions are parsed; that is, how the syntactic structure of an expression is obtained from its visual representation.

The written forms of natural languages, being sequential in nature, are read in specific directions, with occasional back-tracks to regions of the text that have already been scanned (see Section 1.2.1). This well defined order in which words are processed is due to the written text being based on spoken language, where sounds are deployed in a specific order over time. Mathematical notation, however, is a visual language with a two-dimensional layout. As such, there is no single way to sequence the symbols in a mathematical expression, with many different symbol orders being possible (as discussed in Section 1.1.3). Thus, in order to understand how experienced users of mathematics parse algebraic expressions, it is necessary to first understand the order in which the symbols in an expression are processed, by examining how they are visually scanned. This is the primary focus of this chapter.

Also of interest in this chapter are possible similarities and differences between the parsing of algebraic expressions and the processing of sentences of natural language. Since natural language comprehension is better understood than the comprehension of any other language, it provides a logical basis for comparison. Therefore, information which has

proven useful in understanding how natural languages are processed, such as where people pause when parsing and how much time is needed for processing different types of "lexical" units, is also examined for algebraic expressions.

Three experiments were constructed to investigate how experienced users of mathematics scan algebraic expressions. The first experiment involved an expression construction task, in which participants were required to read in algebraic expressions a piece at a time. This task examines the order in which experienced users of mathematics prefer to read in expressions, by varying the order in which the expression components are revealed.

The second and third experiments involve tasks that use the Restricted Focus Viewer (which was specifically designed for this purpose), to track the order in which the symbols of an expression are scanned, and measure how long the symbols are focused upon. The RFV scanpath data from these experiments is then analysed using a technique that is not commonly used in this area: Markov Chain analysis. Using this analysis technique, it is possible to evaluate various models of how experienced users of mathematics scan algebraic expressions. The technique allows for both simple and complex models to be evaluated and compared. Thus, using the RFV scanpath data, it is possible to conduct a detailed examination of the major factors involved in the scanning of algebraic expressions, which will in turn enhance our understanding of how experienced users of mathematics parse such expressions. The Markov Chain technique is discussed in detail before being applied to experimental data.

4.1 Processing Expressions a Piece at a Time

In Chapter 2 it was seen that algebraic expressions are encoded into phrasal constituents by experienced users of mathematics. This finding suggests that visual scanning of an algebraic expression should be influenced by that expression's syntactic structure. For example, consider the simple algebraic expression, $3y^2 + 7x$. The parse tree for this expression is shown on the left of Figure 4.1, with the phrasal constituents appearing in different coloured

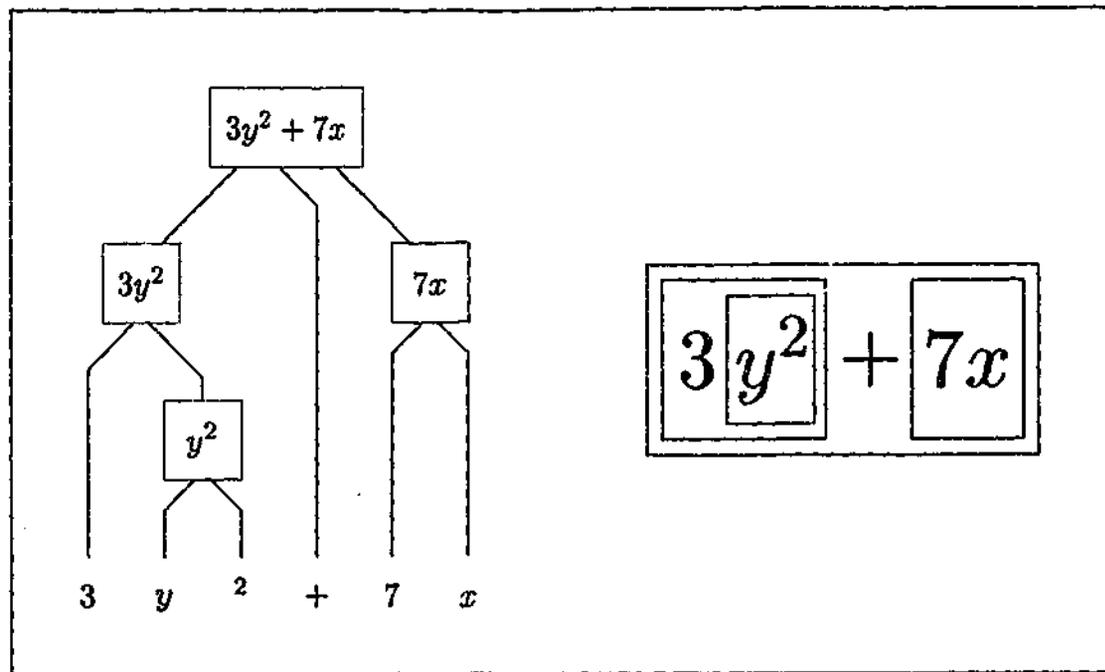


Figure 4.1: Parse tree for the expression $3y^2 + 7x$

boxes. On the right of Figure 4.1, the expression appears in its normal layout, with the phrasal constituents again indicated by coloured boxes.

In general, algebraic expressions do not contain any redundant symbols (that is, symbols that can be omitted from the expression without any loss of information). As a result, when scanning an algebraic expression, each symbol needs to be processed. There are 720 different possible symbol orders for processing the six symbols of the example expression (assuming there is no back-tracking).¹

However, if the order in which the symbols are processed is dependent on the syntactic structure of the expression, the number of possible sequences is greatly reduced. Using the expression on the right of Figure 4.1, with different coloured boxes surrounding the phrasal constituents, a *syntax-based scanning order* can be defined as follows. Upon entering a box to process a specific symbol, the parser cannot leave that box until all of the symbols in that box have been processed. The first symbol to be processed may be any symbol in the expression, and if several unprocessed symbols remain in a box that has been entered, one of them is chosen as the next symbol to be processed.

¹In reality of course, back-tracks will occur when an expression is read, however this issue does not need to be dealt with until empirical scanning models are developed in Experiment 8.

The following is an example of a syntax-based scanning order, using the expression in Figure 4.1. Consider that the first symbol fixated on is the digit 3. This symbol is inside two boxes; the green box for the phrasal constituent $3y^2$, which itself is inside the purple box that contains the entire expression. The syntax-based scanning order requires all of the symbols in the green box to be processed before any other symbols in the expression. This allows the next symbol to be either the symbol y , or the 2 symbol. Note that from the standpoint of syntax, neither of these two symbols is preferred over the other as the next symbol to be processed. If, for example, the 2 symbol is selected, then the processing has entered the red box (while still remaining in the green and purple boxes). Since there is one other symbol in the red box that has not been processed yet (the y symbol), this symbol must be processed next. At this point, the phrasal constituents in the red and green boxes have been fully processed. The purple box however still contains three more symbols, any of which may be the next one processed. If, for example, the next one chosen is the symbol 7 (inside the blue box), then it must be followed by the symbol x (the only other symbol in the blue box), with the $+$ symbol being processed last. The sequence just described is as follows.

$$\rightarrow 3 \rightarrow ^2 \rightarrow y \rightarrow 7 \rightarrow x \rightarrow +$$

In fact, for the example expression, there are 48 valid syntax-based sequences for processing the six symbols. Examples of some of these are:

$$\rightarrow y \rightarrow ^2 \rightarrow 3 \rightarrow x \rightarrow 7 \rightarrow +$$

$$\rightarrow + \rightarrow x \rightarrow 7 \rightarrow y \rightarrow ^2 \rightarrow 3$$

$$\rightarrow x \rightarrow 7 \rightarrow + \rightarrow ^2 \rightarrow y \rightarrow 3$$

$$\rightarrow ^2 \rightarrow y \rightarrow 3 \rightarrow + \rightarrow 7 \rightarrow x$$

Note that the symbols that form a phrasal constituent of the expression must form a sub-sequence (although in any order) of the overall syntax-based sequence. For example,

since $7x$ is a phrasal constituent, each syntax-based sequence must contain either the sub-sequence $7 \rightarrow x$ or $x \rightarrow 7$. The symbol 7 and the symbol x are not allowed to be separated by any other symbols in the sequence.

While syntax-based scanning greatly reduces the number of valid symbol sequences, there still remains a large number of valid syntax-based sequences. Also, as the number of symbols in an expression increases, the number of syntactically valid symbol sequences also increases. It is reasonable to hypothesize that experienced users of mathematics scan the symbols of an algebraic expression in a consistent manner. This would mean that for each type of phrasal constituent, the symbols of that constituent are processed in a specific order.

As was seen in Section 1.1.4, much of the work during the development of mathematical notations (and particularly in Europe) was in Latin. As a result, the layout of algebraic notations is closely related to the layout of written Latin text. It should also be noted that the experienced users of mathematics considered in this thesis are proficient users of English. Both Latin and English text are written and read in a left-to-right (and top-to-bottom) direction. Thus, it is reasonable to hypothesize that the syntactic elements of algebraic expressions are also primarily processed in a left-to-right (and top-to-bottom) manner, by experienced users of mathematics who are also proficient users of English.

The algebraic notation considered in this thesis (as defined in Section 1.1.4), consists primarily of seven types of grammatical units. These grammatical units are illustrated in Figure 4.2, where the red squares can be replaced by certain syntactically well-formed sub-expressions, such as a number or variable, or more complex elements such as a fraction or product. For example, $\square + \square$ can become $8x^3 + \frac{1}{x}$ where the first red box corresponds to the sub-expression $8x^3$, and the second red box corresponds to $\frac{1}{x}$. The right side of Figure 4.2 contains the predicted order (given by blue arrows) in which the components of a grammatical unit are processed, based on a left-to-right (and top-to-bottom) processing direction. (Note that the first component to be examined, indicated by the longer arrows,

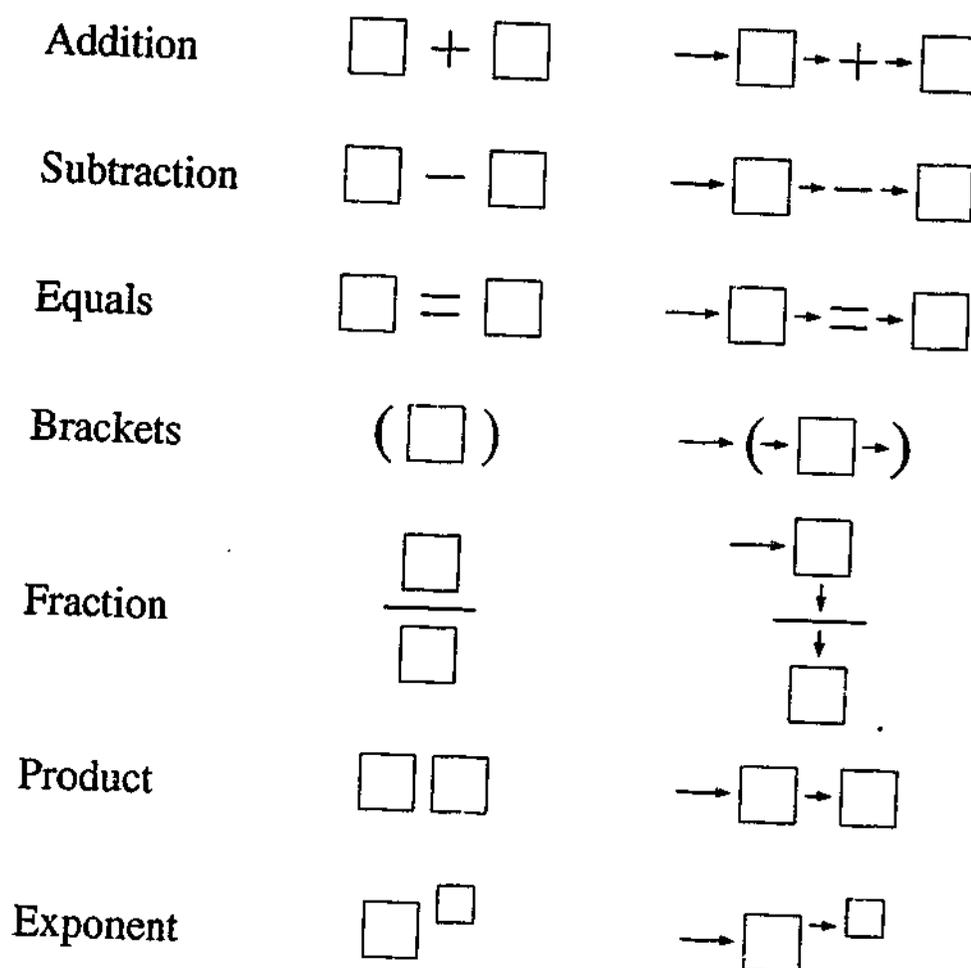


Figure 4.2: Grammatical units for the algebraic notation considered in this thesis, and the predicted order in which the elements of the units are scanned.

can be approached from any direction.)

This combination of using syntactic structure, and a left-to-right (and top-to-bottom) scanning order within the grammatical units, results in a unique expected symbol sequence for processing any given algebraic expression. For the example expression in Figure 4.1, this symbol sequence is the following.

$$\rightarrow 3 \rightarrow y \rightarrow 2 \rightarrow + \rightarrow 7 \rightarrow x$$

Such an ordering of the symbols in an algebraic expression shall from here on be referred to as the *syntax-lrtb ordering* (where lrtb stands for left-to-right, top-to-bottom).

To test the hypothesis that experienced users of mathematics scan the symbols of an algebraic expression in a syntax-lrtb ordering, an expression construction task was developed. Algebraic expressions were partitioned into three sub-expressions, which were then

shown to the participants one at a time, in their correct spatial location. From viewing the three sub-expressions, participants were required to mentally reconstruct the overall expression. If scanning was based on a syntax-lrtb ordering, a memorization advantage would be expected when the sub-expressions were presented in this order, as opposed to a different order.

As well as varying the order of presentation, two different types of sub-expression were used in the experiment to partition the expressions. One type was syntax-based and one was not. For example, consider the two sub-expressions below, both of which are part of the above example expression.

$$3y^2$$

$$y^2+$$

Both of these sub-expressions contain symbols that represent a sub-sequence of the syntax-lrtb order of the original expression (that is, the syntax-lrtb order of the original expression contains the sub-sequence $3 \rightarrow y \rightarrow ^2$ and the sub-sequence $y \rightarrow ^2 \rightarrow +$). However, the sub-expression on the left is based on syntax, being a phrasal constituent of the example expression, while the sub-expression on the right is not. Based on the results thus far in this thesis, one would expect to see a memorization advantage in favour of the syntax-based sub-expressions.

Experiment 7

Method

Participants The participants used in this and all other experiments discussed in this chapter were either students (graduate or undergraduate) or staff members from the School of Computer Science and Software Engineering at Monash University. All were competent mathematicians for whom mathematics formed a substantial component of their undergraduate training, and they all frequently worked with algebra. All participants were volunteers who were proficient users of English, aged between 18 and 35 years, with normal or corrected-to-normal vision.

Thirty participants successfully completed this experiment. Data from an additional six participants were excluded due to excessive error rates.

Materials and Design Seventy-two algebraic expressions were constructed, all consisting of exactly seven characters. The expressions contained at most one fraction, and the variable names were x and y . Due to the nature of the experiment, none of the expressions contained brackets.² The expressions were made deliberately small to ensure that participants had a reasonable chance of memorizing them.

Each expression was then subjected to two types of partitioning. The first partitioning was into syntax-based sub-expressions and the second partitioning was into non-syntax-based sub-expressions. In both cases, the expression was partitioned into three sub-expressions of between one and four characters in length. Due to the short length of the expressions, there were restrictions on how a given expression could be partitioned into three sub-expressions of a particular type. The following definitions are given to further clarify the two types, using the expression $7x^2 + 4y^3$ as an example.

a) *Syntax-based partitioning*: each sub-expression is either a phrasal constituent of the original expression (such as $4y^3$), or an operator with one complete operand (such as $+4y^3$, but not $+4y$).

b) *Non-syntax-based partitioning*: at least one (usually two) of the three sub-expressions will not be the same as those defined for the syntax-based partitioning. The non-syntax-based sub-expressions are instead designed to be as inconsistent with the phrasal constituents of the original expression as possible (for example, $^2 + 4y$).

The symbols of both types of sub-expression however, represent a sub-sequence of the syntax-irrb order of the symbols in the original expression. Examples of the expressions and sub-expressions used can be seen in Table 4.1. The sub-expressions are labelled A, B

²Bracketed terms would make the partitioning used in the experiment difficult, especially since each expression consisted of only seven characters. The processing of brackets is thoroughly examined in Experiment 8.

Table 4.1: Example expressions and sub-expressions used in Experiment 7.

Expression	Sub-Expressions					
	Syntax-Based			Non-Syntax-Based		
	A	B	C	A	B	C
$y = \frac{6}{x} + 2$	$y =$	$\frac{6}{x}$	$+ 2$	$y = \frac{6}{x} +$	$\frac{6}{x}$	2
$xy + \frac{3x}{8}$	$xy +$	$3x$	$\frac{3x}{8}$	x	$y + \frac{3x}{8}$	$\frac{3x}{8}$
$xy + y^2 + 4$	xy	$+ y^2$	$+ 4$	x	$y + y^2$	$+ 4$

or C depending on their position in the syntax-lrtb ordering of the symbols in the original expression.

The sub-expressions were presented to participants in one of three orders. In the *forward* order the sub-expressions were presented in their syntax-lrtb order, as the sequence ABC. The *backward* order was the reverse of this, using the sequence CBA. Finally, the *jumbled* order was designed so that the first two parts of the expression that were presented could not be put together until the final sub-expression had been revealed, and there were no left-to-right sequences. The sequence for this order was ACB, with the middle sub-expression in the syntax-lrtb ordering (sub-expression B), presented last.

A yes or no question was also designed for each expression, to determine whether the expression has been accurately constructed by the participant. The questions ranged in difficulty from asking whether a particular symbol was present in the expression, to requiring the participant to replace a variable with a certain value, and then simplify the expression. Table 4.2 gives examples of the questions created and their corresponding expressions. For any given expression, participants would not know in advance how difficult the question

Table 4.2: Examples of the expressions and corresponding questions used in Experiment 7, and the correct answers to the questions.

Expression	Question	Answer
$x(8x + y)$	When $x = 1$ does the expression reduce to $8 + y$?	Yes
$\frac{5y}{9x} + y$	Does the digit 4 appear in the expression?	No
$\frac{2 + x^2}{7x}$	Is the sub-expression $+ x^2$ part of the expression?	Yes

would be, and thus only accurate construction could guarantee good performance in the task.

For each expression, the two sub-expression types were presented for each of the three presentation orders described above, resulting in six conditions. To ensure that no participant were presented with any expression more than once, thus avoiding practice effects, six counterbalanced versions of the experiment were constructed. For each version, there were twelve instances of each condition. Six additional expressions (one exemplar of each type) were constructed as practice items. The same practice items were used in each version. Five participants completed each version, each receiving the items in a different pseudo-random order.

Procedure Participants were seated comfortably in an isolated booth. Items were displayed as black text on a white background on a 17" monitor at a resolution of 1024×768 , controlled by an IBM compatible computer running a purpose designed computer program. The average width of the expressions in pixels was 86 (range 57-123) with an average height of 47 (range 22-59). The average width of the sub-expressions in pixels was 37 (range 6-98) with an average height of 24 (range 1-56).

Participants were given a brief statement of instructions before the experiment began. For each algebraic expression, they were required to view the three sub-expressions one at

a time, and from these construct a mental representation of the complete expression. They would not know in advance which type of sub-expressions would be used for a particular expression (syntax-based or non-syntax-based), or the presentation order (forward, backward or jumbled). Practice items preceded the experimental items, and the participants took approximately twenty minutes to complete the task. Progress was self-paced, with participants pressing the space bar to initiate the presentation of each trial.

Each sub-expression was presented in the correct spatial location based around the centre of the screen. (Note that if the sub-expressions were not in their correct spatial locations, then sub-expression sequences such as $7y^2$ followed by $+$ followed by $2x^3$ would be ambiguous, as they could represent $7y^2 + 2x^3$ presented in a forward order, or $2x^3 + 7y^2$ presented in a backward order.)

The initial press of the space bar saw the first sub-expression appear. This would be sub-expression A if a forward or jumbled order were being presented, or sub-expression C if a backward order was being presented. This would remain on the screen until the participant pressed the space bar again to reveal the second sub-expression. This remained on the screen until the participant pressed the space bar for a third time. The final sub-expression was then shown. Finally, a fourth press of the space bar would reveal a question about the expression. The question appeared in the centre of the screen. Participants were required to decide if the answer to the question was yes or no, responding via a timed selective button press. They pressed the green button, (the '/' key on the right side of the keyboard), to indicate a "Yes" response, and the red button, (the 'Z' key on the left of the keyboard), to indicate a "No" response. The question remained on the screen until a response was made.

Participants were instructed to try and memorize each sub-expression as quickly as possible, while still ensuring that they would later be able to accurately answer the question. Similar instructions were also given with regard to answering the question, with participants told to answer quickly while trying to remain accurate. The program running the experi-

ment recorded the amount of time each participant spent viewing each sub-expression. The question response time, between the onset of the question and the participants' response, was also recorded. Feedback was given after each trial. The feedback was based around the response to the question, with the question response time in milliseconds reported, along with the word "CORRECT" or "INCORRECT" appearing on the screen, indicating if their response was the correct one.

Data Treatment The responses to the questions were used to determine which participants' data were included in the final analysis. For inclusion, participants were required to make no more than 25% errors overall, and no more than 50% errors for any given condition (combination of sub-expression type and order of presentation). Response times for the questions were not analysed. The data of interest is the time taken to memorize the three sub-expressions of each expression for each condition. To reduce the unwanted effect of outliers, standard deviation cut-offs were applied. Any response time lying three or more standard deviations above or below a participants' overall mean for that presentation position (first, second or third sub-expression presented), was truncated to the value of the cut-off point (Winer, 1962). All items were included for each participant, regardless of whether the question for a particular item was answered correctly. Response time and error rate data were analysed by a series of analyses of variance (ANOVAs), over both participant (F_1) and item (F_2) data. Again, the probability level, $p < .05$, has been used as the criterion for statistical significance.

Results and Discussion

The mean total time taken to memorize the entire expression (that is, the sum of the times taken to memorize the first, second and third sub-expressions presented) for each combination of sub-expression type and order of presentation, is shown in Table 4.3. The mean error rates in responding to the questions are also shown. In each case, the corresponding standard deviation is given in parentheses. Comparisons of this data were conducted using

Table 4.3: Mean total memorization times (ms) and question error rates (%) as a function of sub-expression type and presentation order for Experiment 7.

Sub-Expression Type	Presentation Order		Time(ms)		%Error	
Syntax-Based	Forward	(ABC)	4967	(2502)	12.8	(11.1)
	Backward	(CBA)	5270	(2544)	12.8	(9.5)
	Jumbled	(ACB)	5364	(2751)	13.6	(11.3)
Non-Syntax-Based	Forward	(ABC)	5684	(2877)	15.6	(9.7)
	Backward	(CBA)	6197	(2856)	17.2	(9.8)
	Jumbled	(ACB)	6337	(2939)	13.3	(9.7)

three-way ANOVAs (versions \times sub-expression type \times presentation order).³

As can be seen from the total memorization times, syntax-based partitioning provides a clear performance advantage when memorizing an expression. For the forward presentation order, there is a significant advantage of 717ms for syntax-based sub-expressions over non-syntax-based sub-expressions ($F_1(1, 24) = 17.99, p < .05, F_2(1, 66) = 22.74, p < .05$). This significant advantage was also seen for the backward presentation order (927ms difference; $F_1(1, 24) = 63.81, p < .05, F_2(1, 66) = 47.19, p < .05$), and the jumbled presentation order (973ms difference; $F_1(1, 24) = 39.17, p < .05, F_2(1, 66) = 36.76, p < .05$). For the error rates however, while there appeared to be a slight trend in favour of the syntax-based sub-expressions, this was not statistically significant for any of the presentation orders in either a by participant or by item analysis ($F_1 = 1.74, F_2 < 1$, for the forward order; $F_1 = 4.12, F_2 = 1.64$, for the backward order; $F_1 < 1, F_2 < 1$, for the jumbled order).

The primary aim of this experiment was to determine if mathematical expressions are processed with a preference to a certain scanning order. For the syntax-based sub-expressions, there is a significant performance advantage for processing in the forward

³As with previous experiments, the first factor in the analyses is a 'dummy' factor created by the arbitrary assignment of target items to six separate lists for the purpose of counterbalancing.

presentation order over the backward order (303ms difference; $F_1(1, 24) = 6.68, p < .05$, $F_2(1, 66) = 4.85, p < .05$), and also for the forward order over the jumbled order (397ms difference; $F_1(1, 24) = 7.48, p < .05$, $F_2(1, 66) = 6.53, p < .05$). This gives support for the hypothesis that experienced users of mathematics have a preference for scanning algebraic expressions in a syntax-lrtb order. The backward order also provided a 94ms advantage over the jumbled order, however this was not statistically significant ($F_1 < 1, F_2 < 1$). The overall trend was also seen for the non-syntax-based sub-expressions, with the forward presentation order resulting in faster processing than the backward order (513ms difference; $F_1(1, 24) = 15.99, p < .05$, $F_2(1, 66) = 9.49, p < .05$), and also faster processing than the jumbled order (653ms difference; $F_1(1, 24) = 21.12, p < .05$, $F_2(1, 66) = 18.73, p < .05$). Again there was a 140ms advantage for the backward order over the jumbled order, but this was not significant ($F_1 = 1.77, F_2 < 1$). This further implies that processing of the symbols in an algebraic expression is based on a syntax-lrtb ordering.

Interestingly, the error rate analysis indicates that there is no significant difference between the error rates for the three presentation orders, regardless of sub-expression type ($F_1 < 1, F_2 < 1$ for syntax-based sub-expressions; $F_1 = 1.66, F_2 < 1$ for non-syntax-based). This is surprising, given that the forward presentation order allowed for much faster processing of the expressions than the backward or jumbled orders. It appears that although experienced users of mathematics take longer to encode an expression when it is presented in a non-preferred order, such an order is not detrimental to the quality of the internal representation that results from the encoding process.

Further exploration of the data of Experiment 7 was conducted, examining the mean time spent viewing each sub-expression during the memorization phase. Since the mean number of characters per sub-expression varied (for the syntax-based partitioning, the average number of characters in sub-expression A was 2.40; in sub-expression B, 2.11; in sub-expression C, 2.49; for the non-syntax-based partitioning, the average number of characters in sub-expression A was 2.22; in sub-expression B, 2.90; in sub-expression C, 1.88),

Mean Viewing Time per Character

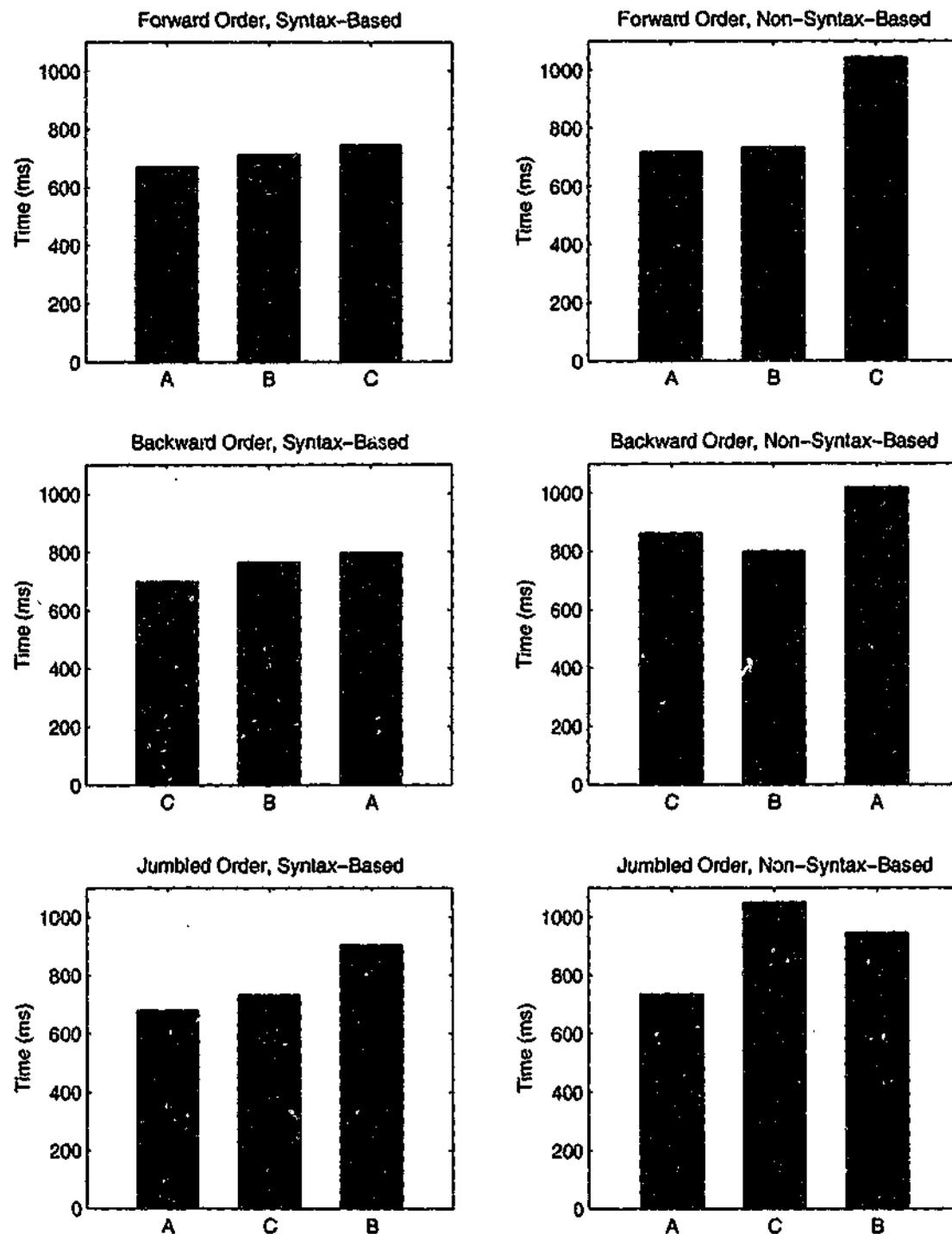


Figure 4.3: Mean viewing time per character for each sub-expression in Experiment 7, as a function of sub-expression type and presentation order.

the mean viewing time per character is examined for each of the six conditions of the experiment. These times are illustrated in the six graphs in Figure 4.3. The three graphs on the left of Figure 4.3 are for the syntax-based sub-expressions, while the three graphs on the right are for the non-syntax-based sub-expressions. The top two graphs contain data

for the forward presentation order, the middle two for the backward presentation order, and the bottom two for the jumbled presentation order. In each graph, the left bar represents the sub-expression that appeared first, the middle bar represents the sub-expression that appeared second, and the right bar represents the sub-expression that appeared last (for each graph, the three bars are labelled with their corresponding sub-expression, A, B or C).

All of the syntax-based graphs in Figure 4.3 suggest a general trend that the more symbols of an expression that have already been processed, the more time the symbols in the next sub-expression take to process. For the forward order, this trend is barely noticeable, which suggests that experienced users of mathematics are efficient at scanning in this direction. For the backward order and jumbled order, the trend becomes more apparent, which suggests that experienced users of mathematics are not as adept at scanning in these directions. Also noteworthy, is that symbols in the third sub-expression for the jumbled order (sub-expression B) take considerably longer to process. This is likely to be because the first and second sub-expressions are not directly connected, and thus the final sub-expression needs to bind together the existing but separate syntactic structures of the first and second sub-expressions. The non-syntax-based graphs in Figure 4.3 reveal a more disjoint picture, which suggests that the processing of such sub-expressions is an unfamiliar task for the participants.

Overall, therefore, the results of this experiment reinforce the importance of a forward scanning order and syntactic structure in the processing of algebraic expressions. However, further supporting evidence is needed before we can be confident in suggesting that a syntax-based scanning order is the one used by experienced users of mathematics. In Experiment 8, therefore, this issue is assessed at a higher level of resolution by using the RFV to examine how experienced users of mathematics scan complete expressions. In order for the RFV scanpath data to be useful to this end, a robust data analysis technique is required that will allow for the development and evaluation of various scanning models. The technique

used in this thesis is Markov Chain analysis.

4.2 Markov Chains and Model Analysis

People process complex visual stimuli such as algebraic expressions by continually shifting their visual attention from one region of the stimulus to another. Such shifts in the direction of visual attention can be tracked using eye-tracking equipment or the Restricted Focus Viewer (RFV), and the resulting scanpath data captures the sequence in which the stimulus elements were examined. This knowledge can give important insights into the cognitive processes used in the comprehension of a particular stimulus type, and allow for models that explain the scanning pattern to be developed. However, in order to determine how accurately such scanning models can predict experimental data, and to usefully compare competing models, rigorous data analysis techniques are needed.

For visual attention to be tracked, individual fixations and saccades need to be identified from the scanpath data. The task of fixation identification has been a major focus of research involving the analysis of eye-tracking data, and many techniques and algorithms for converting eye-tracking data into sequences of fixations and saccades have been developed (for example, Scinto & Barnette, 1986; Latimer, 1988; Salvucci & Anderson, 1998). In many cases it is not known in advance which regions of a stimulus will be of interest to a participant, so a second phase of analysis is sometimes required to determine what these are. This is often true in experiments involving scene perception, where it can be difficult to predetermine what constitutes a 'stimulus element' (for example, see Yarbus, 1967).

In the context of this thesis, fixation identification does not present a significant challenge. Using the RFV, identifying fixations is a relatively straightforward task, with the motion blur feature providing a convenient way of distinguishing between fixations and saccades. Also, the stimuli being examined (algebraic expressions) consist of well defined visual elements in the form of individual symbols (in contrast to many of the stimuli used in studies of scene perception).

Fixation identification allows certain information, such as mean fixation durations and saccade lengths, to be determined for specific stimuli types. As a result, much of the research involving eye-tracking does not go beyond fixation identification in the analysis of scanpath data. However, of more interest in this thesis is how the order in which stimulus elements are fixated can be analysed to reveal details of the cognitive processes being used by the viewer of the stimulus. There has been far less research into this area and the nature of the work varies considerably, with the methods used ranging from comparisons using string edit distance measures (Brandt & Stark, 1997), to the use of probabilistic grammars (Salvucci, 1999). For the purposes of this thesis, however, an analysis technique is required that allows for the development of detailed models that describe how the elements of a stimulus are scanned. One analysis technique that is particularly well suited to the development of models for describing sequential data, is Markov Chain analysis.

Markov Chains belong to a more general class of methods known as Markov Models (more specifically, 'Markov Chain' is another name for a first-order Markov Model). Markov Models have been widely used in fields ranging from Computer Science (for example, Rabiner, 1989; Arnold & Guessarian, 1996; Saul & Jordan, 1999) to Biology (for example, Baldi, Chauvin, Hunkapiller, & McClure, 1994; Krogh, Brown, Mian, Sjölander, & Haussler, 1994; Durbin, Eddy, Krogh, & Mitchison, 1998). However, their use is not common in the analysis of scanpath data. Notable exceptions include the work of Stark and Ellis (1981), Rosbergen, Wedel, and Pieters (1997) and Salvucci and Anderson (1998). Of these, the work of Rosbergen et al. (1997) is most similar to the approach that is described later in this section. Rosbergen et al. (1997) use Markov Chains to compare competing models which describe how viewers direct their attention towards print advertisements. Although similar, the analysis used in this thesis is more fine grained, examining the processes involved in scanning the elements of a visual language, rather than considering only four large regions of a print advertisement.

In the following sub-sections, Markov Chains are examined in the context of scanpath

data from algebraic expressions. The use of Markov Chains to represent scanning models is explored, and model selection criterion, which allows statistical comparisons to be made between competing scanning models, is discussed.

4.2.1 Transition Data and Markov Chains

An important aspect of parsing is the order in which the symbols of an expression are processed. Note that even though expressions are encoded into constituents which may contain several symbols, the results of Chapter 2 indicate that these constituents are syntactic rather than lexical in nature, meaning that the symbols need to be examined individually before the constituents can be composed.

Consider, for example, the simple expression $3x+7$, and the following three hypothetical sequences which describe the order in which three mathematicians might scan the symbols of the expression.

Sequence 1: $\rightarrow 3 \rightarrow x \rightarrow 3 \rightarrow x \rightarrow + \rightarrow 7$
 Sequence 2: $\rightarrow 3 \rightarrow x \rightarrow 7 \rightarrow + \rightarrow 7$
 Sequence 3: $\rightarrow 3 \rightarrow x \rightarrow + \rightarrow 7 \rightarrow 3 \rightarrow x \rightarrow 3$

It can be seen that for each symbol in these sequences, certain symbols are more likely to occur immediately after it than others. For example, the symbol 3 is frequently followed by the symbol x , but never by the $+$ symbol. Each pair of symbols in a sequence represents a *transition*; for example, when the symbol 3 is followed by the symbol x , it is referred to as a transition from 3 to x . From the sequence data, it is possible to derive a *transition table*, that lists how often each symbol is followed by each other symbol. Table 4.4 gives the transition table for the example sequence data given above.⁴

The transition table in this example is the 4×4 set of values that gives the number of transitions from any symbol to any other symbol. Note that for the analyses conducted in this thesis, only transitions from one symbol to a different symbol are considered, with transitions from a symbol to itself ignored (resulting in the main diagonal of the transition

⁴Note that if a sequence contains N symbols, then it contains $N - 1$ transitions as there is no transition from the last symbol in the sequence.

Table 4.4: Transition table based on the example data using the expression $3x + 7$.

From	To			
	3	x	+	7
3	0	5	0	0
x	2	0	2	1
+	0	0	0	3
7	1	0	1	0

table containing only zeros). This is because such transitions are primarily due to noise in the data, and do not actually represent shifts in attention (as after the transition, attention remains directed at the same symbol as before the transition).

The total number of transitions from each symbol can be calculated by adding the number of transitions in each column of a particular row. Thus, it can be seen that in the example sequence data, there were five transitions from the symbol 3, all of which were to the symbol x , while there were only two transitions from the symbol 7, one of which went to the symbol 3 with the other going to the $+$ symbol. Using this information, it is possible to work out the probability of each transition based on the data contained in the transition table. If each value in the transition table is divided by the total number of transitions for that row, then the result is a table of probabilities for each transition. Such a table specifies a *Markov Chain*. Table 4.5 gives an example of a Markov Chain based on the transition frequencies given in Table 4.4. Note that by definition, for all Markov Chains the probabilities in each row must sum to exactly 1.

4.2.2 Scanning Models

The Markov Chain in Table 4.5 is derived directly from the transition frequencies given in Table 4.4. However, Markov Chains can also be based on hypothetical scanning models.

Table 4.5: Markov Chain based on the transition frequencies from the example data using the expression $3x + 7$.

From	To			
	3	x	+	7
3	0.0	1.0	0.0	0.0
x	0.4	0.0	0.4	0.2
+	0.0	0.0	0.0	1.0
7	0.5	0.0	0.5	0.0

For example, consider the following hypothetical model: when an experienced user of mathematics is viewing a particular symbol in an expression, they are twice as likely to shift their attention to the symbol immediately to the right of the current symbol, than to any other symbol. This would mean that, for the example expression $3x + 7$, one would expect twice as many transitions from 3 to x than from 3 to + or from 3 to 7 (remembering that transitions from 3 back to 3 are not considered). Table 4.6 gives an example of a Markov Chain reflecting this model. Note that in the last row of this Markov Chain, each of the

Table 4.6: Markov Chain based on a scanning model favouring left-to-right transitions.

From	To			
	3	x	+	7
3	0	0.5	0.25	0.25
x	0.25	0	0.5	0.25
+	0.25	0.25	0	0.5
7	0.33	0.33	0.33	0

three possible transitions all have equal probability, as there are no symbols to the right of the 7.

Using a Markov Chain model, it is possible determine how probable a given symbol sequence is. For example, consider again Sequence 1 from the previous examples for the expression $3x + 7$.

$$\rightarrow 3 \rightarrow x \rightarrow 3 \rightarrow x \rightarrow + \rightarrow 7$$

The probability of this sequence is simply the product of the probabilities of each individual transition in this sequence (of which there are five).

$$Pr(\text{Seq 1}) = Pr(3 \rightarrow x) \times Pr(x \rightarrow 3) \times Pr(3 \rightarrow x) \times Pr(x \rightarrow +) \times Pr(+ \rightarrow 7)$$

Note that because the probabilities in this expression are being multiplied, the order in which they appear in the expression makes no difference to the final result. The expression can also be simplified, as one of the transitions, $Pr(3 \rightarrow x)$, occurs more than once. This is done by using the exponent notation for terms that are multiplied by themselves (where $a \times a = a^2$, $a \times a \times a = a^3$ and so on).

$$Pr(\text{Seq 1}) = Pr(3 \rightarrow x)^2 \times Pr(x \rightarrow 3) \times Pr(x \rightarrow +) \times Pr(+ \rightarrow 7)$$

To calculate the value of this expression, the value for each transition probability in the expression is taken from the Markov Chain model being considered. Different Markov Chain models will produce different probabilities for a given sequence. If the Markov Chain being considered is denoted by M , then the probability of Sequence 1, given the Markov Chain, M , is denoted⁵ by $Pr(\text{Seq 1} | M)$. In this example, if the Markov Chain, M , is the one given in Table 4.6, the resulting probability will be the following.

$$Pr(\text{Seq 1} | M) = 0.5^2 \times 0.25 \times 0.5 \times 0.5 = 0.015625$$

It is also straightforward to extend this process from using a single sequence to using multiple sequences. In such a situation, the probabilities calculated for the individual sequences

⁵This notation is known as a conditional probability, where $Pr(A|B)$ can be read as the probability of A occurring given that B is true.

(for a specific Markov Chain), are simply multiplied together to give the probability of a group of sequences.

As seen in the previous section, for any group of sequences the number of occurrences of each possible transition can be stored in a transition table. Using a transition table, the probability of a group of sequences being produced by a specific model can be more easily computed. Let the transition table data be denoted by D . Table 4.4 gives an example of a transition table, where D represents the 4×4 matrix that contains the frequencies of each possible transition. The model is denoted by M (as it was in the above example), and is a matrix of the same size as D , containing the Markov Chain probabilities for each of the possible transitions (such as the Markov Chain in Table 4.6). Then, the probability of the transition data, D , given a particular model, M , is denoted by $Pr(D | M)$. This value is referred to as the *likelihood* of the model M , given the data D .

For both matrices, D and M , individual matrix elements can be referred to by specifying their row and column as a subscript. For example, if M represents the Markov Chain in Table 4.6 which has 4 rows and 4 columns, then $M_{1,3}$ refers to the value 0.25 (row 1, column 3), and $M_{4,2}$ represents the value 0.33 (row 4, column 2).

Every element in the transition data matrix, $D_{i,j}$, represents the number of times a particular transition occurred (note that i and j represent any of the rows and columns respectively of the transition table). The corresponding element in the Markov Chain, $M_{i,j}$, represents the probability of that transition occurring. Therefore, the probability that a specific transition (specified by the values i and j) occurred a specific number of times (as given by $D_{i,j}$), for a Markov Chain in which that transition has a specific probability (given by $M_{i,j}$), is the following.

$$Pr(D_{i,j} | M_{i,j}) = M_{i,j}^{D_{i,j}}$$

For the entire transition data matrix, D , and a given model, M , the probability of the data given the model is the product of the probabilities calculated for each possible transition defined by the transition table (that is, all valid combinations for i and j). This can be

expressed as:

$$Pr(D|M) = \prod_{i,j} M_{i,j}^{D_{i,j}}$$

This expression is known as a *likelihood function*.⁶ (Note that any value raised to the power of 0 is equal to 1.)

4.2.3 Model Parameters and Maximum Likelihood

The model represented by the Markov Chain given in Table 4.6 is based on the idea that transitions are twice as likely to go to the symbol to the immediate right of a given symbol, than to any other symbol. While it seems reasonable to consider a model that favours a left-to-right scanning direction, the idea of transitions to the immediate right being exactly twice as probable as others is clearly arbitrary. It may be more appropriate to make transitions to the immediate right three or four times as probable. Thus, a problem with this scanning model is that the relative importance of different classes of transitions is not obvious.

One way to get around this problem is to use models that contain *parameters*, and use some of the data to determine the best parameter values for the models. For example, consider the Markov Chain given in Table 4.7. In this model, the probability of a transition from a given symbol to the symbol on its immediate right is not explicitly given. Instead, it is represented by a parameter, α . Other possible transition probabilities (represented by dots) are also not explicitly given, as their values will depend on the value of α . The parameter α can take any value between 0 and 1, and each time a different value of α is used, the other transition values are adjusted so that the probabilities in each row sum to 1. This means that for the first three rows, the probabilities represented by dots are all equal to $\frac{1-\alpha}{2}$. For example, if α is set to 0.6, then in the first row, the transition from 3 to x would have a probability of 0.6, whereas the transition from 3 to + and the transition from 3 to 7 would each have a probability of 0.2. Note that as the last row does not contain

⁶In the likelihood function, the symbol \prod represents a product of multiple terms in the same way that \sum represents the sum of multiple terms.

Table 4.7: Markov Chain based on a left-to-right scanning model that uses parameters.

From	To			
	3	x	+	7
3	0	α	.	.
x	.	0	α	.
+	.	.	0	α
7	.	.	.	0

the parameter α (since there are no symbols to the right of the 7), each transition will be equally probable with a value of 0.33 (as was the case in Table 4.6).

By using α , the model can be fine tuned through parameter adjustment. However, this raises a question: given that the α parameter may be set to different values, which value of α is best? For any given transition data, D , and a model with parameters, M , varying the values of the parameters changes how well the model is able to predict the data; that is, the likelihood of the model, $Pr(D|M)$, depends on the parameter values. In determining the best values for the parameters of the models used in this thesis, the principle of *maximum likelihood* is applied. The principle of maximum likelihood states that the best model is the one which best predicts the observed data. In terms of model parameters, this means that the best parameter settings are those that maximize the value of $Pr(D|M)$. Consider the case where the transition data, D , is that seen in Table 4.4, and the model is the one represented by the Markov Chain in Table 4.7. Table 4.8 shows how well the model predicts the transition data for various values of the parameter α . Note that when α is set to 0.5, the model is the same as the model seen in Table 4.6.

Following the principle of maximum likelihood, it is clear that of the α values given in Table 4.8, the value of 0.8 gives the model the best predictive performance for the specified

Table 4.8: Predictive ability of the model, M , shown in Table 4.7, given the transition data, D , shown in Table 4.4, for various parameter settings.

Parameter α	$Pr(D M)$
0.3	0.000000028
0.4	0.000000315
0.5	0.000001695
0.6	0.000005375
0.7	0.000010593
0.8	0.000011930
0.9	0.000004843

transition data. It is possible to make a more accurate estimation of the optimal value for α (to any desired decimal place). However, it should be noted that increased accuracy in the analysis comes at the cost of increased computation time.

Before continuing, it is first necessary to understand why the likelihood values appear so small. The more transitions that occur in the data, the smaller the likelihood value of the models will be. This is because each transition probability in the model must have a value less than 1, and the product of two such values will always be smaller than the values used to produce the product. For example, consider a fair coin, which can be accurately modelled as follows: $Pr(\text{Heads}) = 0.5$, $Pr(\text{Tails}) = 0.5$. For a sequence of three coin tosses, the likelihood of any outcome will be 0.125, whereas for a sequence of ten coin tosses, any outcome will have a likelihood of 0.0009765. The latter value does not reflect a weaker model (as the model is the same in both cases), but rather reflects the fact that as the length of the sequence increases, the number of different possible sequences also increases. Thus, relative performance between models is what is important, not absolute performance.

It should be noted that typically, before parameter estimation is done, the available

data is split into two subsets: training and test data. Estimation of the parameters of a model is conducted using only the training data, with the parameters inferred from the training data then used with the test data to examine how well that model performs with respect to other competing models. This technique is known as cross-validation (Stone, 1974). Using separate sets of data for parameter estimation and the assessment of model performance prevents the accuracy of a model from being overestimated.

4.2.4 Comparing Models

Through the use of Markov Chains it is possible to develop various scanning models, and to determine how well they can predict transition data. However, in comparing competing models, it is not sufficient only to look at model performance. Consideration must also be given to the principle of *parsimony*, which states that, given two models that both perform equally well, the model that provides the simplest explanation of the data should be chosen.

While these two factors, model performance and model complexity, provide a logical criterion for model selection, balancing them is a difficult task. For example, should a complex model with good predictive performance be preferred over a simpler model that gives only reasonable performance?

Many different methods of model selection have been developed, some of which are specific to certain model types or problem domains, while others are more generally applicable (for an overview of model selection criteria, see Ghosh & Samanta, 2001). This chapter will make use of two of the most well-known and commonly used statistical rules for model selection: Akaike's Information Criterion (AIC) (Akaike, 1973), and Schwarz's Bayesian Information Criterion (BIC) (Schwarz, 1978). These methods provide a score for each model, indicating how suitable the model is given the sample data. They are used in fields ranging from Genetics (Abney, McPeck, & Ober, 2000) to Econometrics (Judge, Griffiths, Hill, Lütkepohl, & Lee, 1985).

When calculating a score for a given model, both of these methods consider model

performance (in the form of the natural logarithm of that model's likelihood⁷), and both have a penalty for model complexity. It is in the penalty component that the two methods differ. AIC only considers the number of free parameters (degrees of freedom) of a model when determining the penalty for model complexity. BIC however, considers both the number of free parameters of a model, and the number of observations that make up the data. If a specific model, M_i , has p_i free parameters, and the data, D , is composed of n observations, then the scores for the two methods are calculated as follows.

$$AIC = -2 \log Pr(D | M_i) + 2p_i$$

$$BIC = -2 \log Pr(D | M_i) + p_i \log n$$

For both of these methods, the lower the score the better the model for a given data sample. Also, for either criterion, two models can be compared by calculating the score difference between the models. This reveals not only which model is more appropriate (the model with the lowest score), but the difference also indicates how much better suited a model is than the competing model. A model is considered to be statistically significantly better than a competing model if the score difference is greater than 2 (Sakamoto, Ishiguro, & Kitagawa, 1986). It is important to note that the AIC and BIC model selection criteria are independent of each other; that is, AIC scores can only be compared with other AIC scores, and BIC scores can only be compared with other BIC scores.

In general, for large samples, BIC will select for simpler models, with AIC being biased towards more complex models because it does not take into account the number of observations in the data (Ghosh & Samanta, 2001). In order to ensure robust results, both AIC and BIC will be used in this chapter in the comparison of models. A model will only be considered superior to a competing model if both methods indicate that it is significantly better.

⁷Logarithms are used to make the data analysis more tractable.

4.3 Developing Scanning Models

In Experiment 8, the RFV is used to investigate how experienced users of mathematics parse algebraic expressions. The main aim of this experiment is to track the order in which the symbols of an algebraic expression are scanned by experienced users of mathematics, and to use this data to develop and assess various scanning models, using the Markov Chain techniques described in Section 4.2. Other aspects of the parsing of algebraic expressions that allow for a comparison with what is known about natural language processing, are also of interest. Thus data about how long symbols are fixated on, and what factors influence fixation durations, are also examined.

Experiment 8

Method

Participants Twenty-four participants successfully completed this experiment. Data from one additional participant was excluded due to an excessive error rate.

Materials and Design Forty algebraic expressions were constructed, all consisting of exactly eleven characters. The expressions were divided into eight groups, each containing five expressions. Although each of the expressions within a group was unique, they all shared a similar form; that is, one expression in the group could be changed to become another expression in that group by replacing any variable with another variable, any digit with another digit, or by replacing a plus operator with a minus operator or vice versa. Fraction lines and brackets remained constant among each of the expressions within a group. Table 4.9 illustrates this by specifying the form of the expressions in each of the eight groups, as well as giving example expressions. To represent the form of an expression, the letter *v* indicates where any variable (*x*, *y* or *z*) can be used, the letter *d* indicates where a digit (1 to 9; zero not included) can be used, and the letter *o* indicates a plus or minus operator (+ or -).

Table 4.9: The expression form used for each group in Experiment 8, along with example expressions.

Group	Expression Form	Example Expression
1	$\frac{(v^d \circ d)^{v \circ d}}{d}$	$\frac{(x^2 + 1)^{y-7}}{4}$
2	$\frac{d}{(v^d \circ d)^{v \circ d}}$	$\frac{4}{(y^2 + 9)^{x-8}}$
3	$v \circ \frac{dv \circ d}{d \circ v^d}$	$x + \frac{4y - 1}{9 - x^2}$
4	$\frac{dv \circ d}{d \circ v^d} \circ v$	$\frac{4x - 8}{1 + z^2} + x$
5	$d(v^d \circ dv)^d \circ d$	$7(x^4 + 2z)^2 - 3$
6	$d \circ d(v^d \circ dv)^d$	$9 - 5(z^2 + 8y)^3$
7	$v^{d \circ v} \circ dv^d \circ dv$	$x^{7-y} - 5y^8 + 4z$
8	$(v \circ d)^d(v \circ d)$	$(y + 7)^3(y - 3)$

The expressions were presented one at a time for participants to read, using the RFV tool. To determine whether the expressions were accurately processed by the participants, a statement that could be either true or false was designed for each expression. Half of the expressions had true statements, with false statements being developed for the other half. The statements varied in how difficult they were to verify, being similar in nature to those used in Experiment 6. Some were simple, requiring the participant to determine only if a particular symbol was present in the expression. Others were more difficult, requiring participants to perform a mathematical operation on the expression. Thus, only accurate comprehension of the expression could guarantee good performance in the task. Table 4.10 gives examples of statements created and their corresponding expressions.

To avoid biasing the results by having the mouse cursor initially positioned at only

Table 4.10: Examples of the expressions and corresponding statements used in Experiment 8, and whether those statements were correct.

Expression	Statement	Correct
$\frac{(x^2 + 1)^{y-7}}{x}$	There were exactly two different variables in the expression	True
$y - \frac{8x + 3}{1 + y^3}$	The expression contained the sub-expression $8 - y^3$	False
$(y - 1)^2(z - 6)$	The expression expands to $(y - 1)(y - 1)(z - 6)$	True

one location by the RFV tool, two versions of the experiment were developed. In the first version, the initial mouse cursor position was on the left side of the screen, and in the second version the mouse cursor was initially positioned on the right side of the screen. The expression always appeared in the centre of the screen. All forty expressions were presented for each version of the experiment. Five additional expressions and statements were constructed as practice items. Half the participants did the first version of the experiment, and half did the second version, with the expressions in each version being presented in a different pseudo-random order for each participant.

Procedure Participants were initially presented with written and verbal instructions, which gave examples of the sort of expressions and statements involved in the experiment. Items were displayed in black on a white background on a 20" monitor at a resolution of 1024×768 , controlled by an IBM compatible computer running Version 2.1 of the RFV program.⁸ Participants were seated comfortably at a viewing distance from the monitor of approximately 50cm. The average width of the algebraic expressions in pixels was 138 (range 111-167) with an average height of 41 (range 22-58). The RFV focus box had an edge length of 16 pixels, while the outermost transition box had an edge length of 30

⁸This version of the RFV program can be found at <http://www.csse.monash.edu.au/~tonyj/RFV/> along with the user's manual and tutorial (Jansen, 2001).

$$\frac{4x - 8}{1 + z^2} + x$$

Figure 4.4: Examples of the focus window over two different regions of the expression shown on the left.

pixels. This allowed one symbol to be viewed in full focus at a time, as can be seen in the examples in Figure 4.4. The onset of motion blur occurred at a mouse speed of 150 pixels per second, requiring the mouse to be stopped or moving very slowly in order for full focus to be maintained.

The only interface mechanism used by the participants was a standard computer mouse. Progress was self-paced, with each trial initiated by a single click with the mouse on a button which contained the prompt "Click the button to continue". The button was located either at the far left or far right of the screen, depending on the version of the experiment. This action started the timer (to provide a reference time for the rest of that trial), and a blurred image of the expression appeared in the centre of the screen. The participant could move the window of focus over different symbols in the expression by moving the mouse, allowing the expression to be read. They were initially given five practice items to allow them to become familiar with the RFV program, followed by the forty experimental items.

Once the participants had read an expression, they single clicked the mouse button. This stopped the timer and made the blurred expression disappear, replaced by the statement about the expression (which was not blurred). The participants were required to determine if the statement was true or false with respect to the expression that immediately preceded it. Beneath the statement were two buttons, a green one labelled "TRUE" and a red one labelled "FALSE". When the participant had decided on the validity of the statement, they single clicked on the appropriate button. The program recorded the response given, and the participant was then given feedback. If the response was correct, then the word "Correct" appeared on the screen, along with the statement response time. Otherwise, the

word "Incorrect" appeared on the screen by itself.

Participants were instructed to try and read the algebraic expressions as quickly as possible, while still ensuring that they did not to make too many errors when determining the validity of the statements. The experiment took approximately 15 minutes to complete.

Data Treatment The purpose of the statements was to force the participants to carefully read the expressions. Therefore the time taken to determine if a statement was correct is not analysed. Rather, the data of interest is the RFV data that reveals how the expressions were scanned. One participant failed to correctly judge more than 70% of the statements, and thus was excluded from further analysis.

The RFV data was converted into fixation protocols for each trial. Rectangular bounding boxes were used to enclose each symbol, with each fixation representing the duration of time that the centre of the focus window was inside the bounding box of a symbol, with the window in full focus (that is, without motion blur). If the same symbol was fixated on twice in succession, the two fixations were collapsed into a single fixation whose duration is the sum of the two original fixation times. The transition data was then obtained from the fixation sequences.

Results and Discussion

Figure 4.5 gives examples of two typical scanpaths taken from the RFV data, and the fixation protocols derived from these scanpaths are shown in Figure 4.6. Each fixation in the protocols is represented by a pair of numbers; the first number, which is in italics, allows the order of fixations to be determined, and the second number represents the amount of time that particular symbol was fixated on (in milliseconds). For example, for the algebraic expression on the left of Figure 4.6, the symbol y is labelled with the numbers *4*:234. This indicates that the symbol y was the fourth symbol fixated on, and the fixation duration was 234ms. For the $+$ symbol in the same expression there are two labels; *2*:79 and *14*:77. This indicates that the symbol was fixated on twice, those fixations being the second and



Figure 4.5: Example RFV scanpaths for two of the algebraic expressions used in Experiment 8.

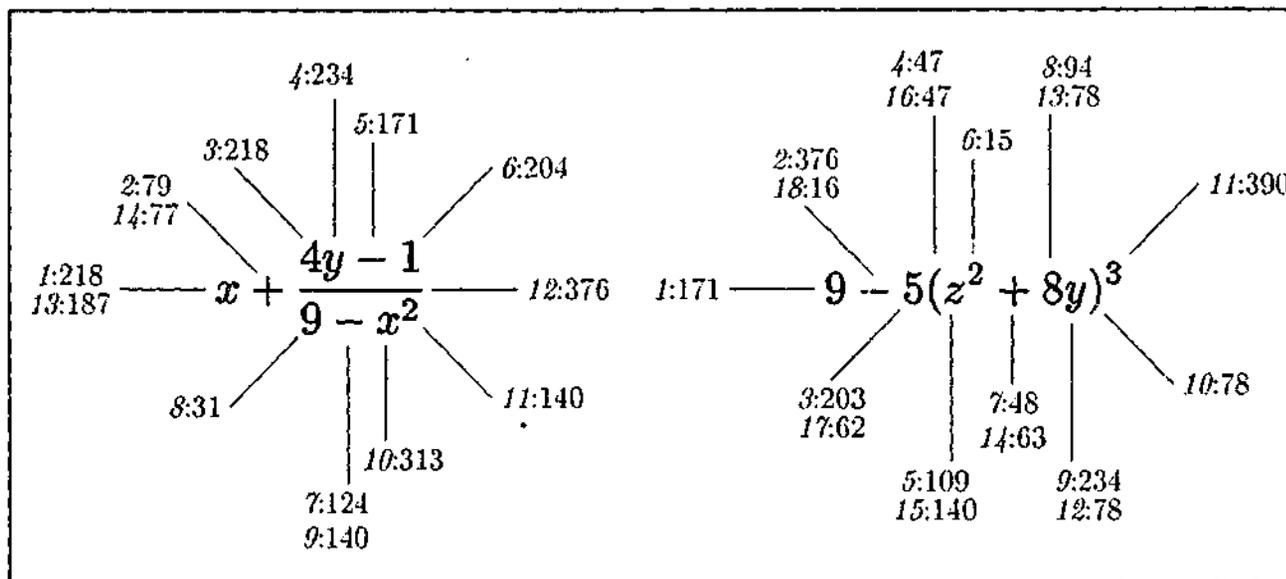


Figure 4.6: Example RFV fixation protocols for the two scanpaths shown in Figure 4.5.

the fourteenth, and lasting for 79ms and 77ms respectively.

As can be seen in the example protocols in Figure 4.6, the fixation durations for the various symbols in the expressions can vary considerably. Table 4.11 gives a break down of the mean fixation times, along with the corresponding standard deviations (in parentheses), for the five different types of symbols present in the expressions. Two-way ANOVAs (versions \times symbol type) of the data were conducted, carried out over participant data only (item-based analyses are not possible as there are unequal numbers for each symbol type).

The two groups of symbols that had the longest fixation durations were digits and variables, although there was no significant difference in fixation durations between these groups (3ms difference; $F < 1$). The symbols with the next longest mean fixation duration were plus and minus operators, which were fixated on for significantly less time than digits

Table 4.11: Mean symbol fixation times (ms) as a function of symbol type for Experiment 8.

Symbol Type	Examples	Fixation Time (ms)
Brackets	(,)	160 (36)
Digits	1, 2, 3, ... 9	285 (108)
Variables	x, y, z	288 (82)
Fraction Lines	—	89 (37)
Plus and Minus	+, -	230 (57)

(55ms difference; $F(1, 22) = 12.63, p < .05$) and variables (58ms difference; $F(1, 22) = 35.64, p < .05$). Also, plus and minus operators were on average fixated on for longer than brackets (70ms difference; $F(1, 22) = 102.04, p < .05$), and brackets were fixated on for longer than fraction lines (71ms difference, $F(1, 22) = 46.97, p < .05$).

These results are consistent with the findings of Ranney (1987), which indicated that the recognition of symbols in an algebraic expression is guided by symbol type (see Section 1.2.2). They suggest that the different types of symbols used in algebraic expressions have different levels of importance, and that the difficulty of integrating the information conveyed by a symbol with other parts of an expression that have already been processed, varies depending on the role played by that symbol. Brackets, for example, are used to explicitly specify the order of precedence in which certain operations must take place. However, brackets primarily play a syntactic role, and as they do not represent either a value or an operation, they are semantically less important than symbols that do represent values and operations. Thus it seems reasonable that brackets would need to be fixated on for less time. In contrast, the longest mean fixation durations occur for digits and variables, both of which represent values (either known or unknown) in algebraic expressions. This result has similarities with how natural language text is read, where content words such as

nouns and verbs are typically fixated on for longer than function words such as articles and prepositions (Just & Carpenter, 1987).

However, a key difference between algebraic notation and natural language text is that the former is a visual language with a two-dimensional structure. As a result, spatial layout adds an extra cue that can be used to identify an operation before the symbol for that operation has even been recognized. (Indeed, many mathematical operations are specified not by an explicit symbol at all, but instead are specified purely by spatial layout; for example, multiplication, as in the expression $6yx$.) This may explain why fraction lines require such short fixations. Being the only operations in the expressions used in this experiment whose components are arranged with a vertical layout, the presence of a fraction in an expression is usually easy to detect. Visual cues can also assist in the recognition of plus and minus symbols, which have more space surrounding them than other horizontally adjacent symbols, making them visually more distinct (as can be seen for example, with the plus symbol in the expression $9x + 2y^4$). Thus, visual cues appear to assist the processing of algebraic expressions.

Note that the idea that visual cues play an important role in the processing of algebraic expressions does not conflict with the findings of Experiment 2, where it was shown that the encoding of algebraic expressions is not primarily based on visual features. Although the primary basis for encoding is syntax, visual features still support the encoding process by suggesting which syntactic rules are relevant to various parts of an expression. For example, seeing parts of an expression arranged with a vertical layout may suggest that a fraction is present. If a fraction is actually present, then the visual layout will have acted as a prime, allowing for faster syntactic processing of the expression. Indeed, it seems quite reasonable to suggest that the assistance provided by visual features was an influential factor in mathematical notation developing as a two-dimensional visual language rather than a sequential language.

Turning to the issue of scanning, an important initial question to ask about how expe-

rienced users of mathematics process algebraic expressions is: where in an expression does the scanning begin? Given that the results of Experiment 7 suggest that expressions are scanned in a syntax-lrtb order, one would expect scanning to begin at the initial symbol of an expression, as determined by the syntax-lrtb order. For the expressions used in this experiment, this is the leftmost symbol, or if the expression begins with a fraction, the leftmost symbol in the numerator of that fraction. Thus, the next part of the data analysis examines if the symbols initially fixated on by the participants are the expected ones.

Since each expression is blurred when presented by the RFV, participants often need one or two navigational fixations to correctly establish the bounds of the expression. Allowance is made for this by considering the first three fixations, rather than just the first fixation. Table 4.12 shows the proportion of trials for which the initial symbol of an expression (as determined by the syntax-lrtb order) is one of the first three symbols fixated on, for each expression group. Overall, the initial symbol is one of the first three fixations in 0.86 of the trials, suggesting that experienced users of mathematics generally begin at the initial symbol of an expression (as determined by the syntax-lrtb order). However, for two of the expression groups (1 and 8), the proportion is much lower than for the others.

Table 4.12: The proportion of trials for which the initial symbol (as determined by the syntactic structure) was one of the first three symbol fixations, as a function of expression group for Experiment 8.

Expression Group	1	2	3	4	5	6	7	8
Proportion of Trials where the Initial Symbol is one of the First Three Fixations	0.62	0.85	0.93	0.89	0.96	0.94	0.97	0.75

The initial symbol in expression groups 1 and 8 are open brackets, whereas the initial symbol for the other expression groups are digits or variables. A comparison of the data using a two-way ANOVA (version \times initial symbol type) over both participant and item data, shows that when the initial symbol is a bracket, then it is one of the first three fixations

in a significantly lower proportion of trials than if the initial symbol is another symbol type (a .24 difference; $F_1(1, 22) = 43.23, p < .05, F_2(1, 8) = 104.73, p < .05$). However, if for expression groups 1 and 8 the initial bracket is ignored, and the initial symbol is instead defined as being the symbol to the immediate right of the initial bracket, then the proportion of trials for which the initial symbol was one of the first three symbol fixations becomes 0.97 and 0.99 for expression groups 1 and 8 respectively. This difference indicates that experienced users of mathematics tend to initially fixate on the first symbol in an algebraic expression that conveys semantic information (as determined by the syntax-lrtb order). It appears that brackets, playing primarily a syntactic role, are often not explicitly fixated.

Given the new definition of a initial symbol (being the first symbol that conveys semantic information, as determined by the syntax-lrtb order), a further comparison of the data was conducted using a two-way ANOVA (versions \times expression group). The analysis showed that the experiment version (that is, whether the mouse cursor was originally placed to the left or to the right of the expression), did not significantly influence the proportion of trials for which the initial symbol was one of the first three symbols fixated (a .01 difference; $F_1 < 1, F_2 < 1$). This result suggests that experienced users of mathematics scan in a manner that is independent of the side of the expression to which their visual attention is initially directed.

However, the proportion of trials for which the initial symbol was one of the first three symbols fixated was influenced by the different expression groups ($F_1(7, 154) = 3.97, p < .05, F_2(7, 56) = 5.32, p < .05$). A further breakdown of these results indicated that the initial symbol is one of the first three fixations in a significantly lower proportion of trials if the expression begins with a fraction (groups 1, 2 and 4), than if it does not (a .06 difference; $F_1(1, 22) = 9.73, p < .05, F_2(1, 8) = 12.74, p < .05$). Thus, while experienced users of mathematics tend to initially fixate on the first symbol in an algebraic expression that conveys semantic information (as determined by the syntax-lrtb order), certain features, such as fractions, also have some influence on how an expression is scanned.

Understanding which symbol experienced users of mathematics prefer to start at when scanning an algebraic expression, while important, is only a small part of understanding how such expressions are scanned. The primary aim of this experiment is to develop an overall model of scanning, which is based around the shifts in attention from one symbol to another that occur when an expression is being processed. Of particular interest is how these shifts in attention tie in with the results of Chapter 2, in which it was shown that the encoding of algebraic expressions by experienced users of mathematics is based primarily on syntax.

The RFV fixation protocols for the 24 participants over the 40 expressions in this experiment provide a rich pool of data that consists of a total of 15,294 transitions. Appendix A gives the transition table data for each of the eight expression groups. The data was split into two subsets: training and test data. The training data consists of the transition data for 12 of the participants, which contains 7,433 transitions, and is used for parameter estimation for the scanning models that contain parameters. The test data consists of the transition data for the remaining 12 participants, which contains 7,861 transitions, and is used for assessing the performance of the models being considered.

Given that each expression in this experiment contains exactly 11 symbols, and that transitions from a particular symbol back to itself are not considered, a transition from any symbol can be to any one of 10 other symbols. Thus, with each transition having a random probability of 0.1, the probability of the test data being produced with this random model is 10^{-7861} . However, models can be developed which more accurately predict the transition data.

To begin with, a naive scanning model is considered which is based on the fact that the symbols in an expression vary in how distant they are from each of the other symbols. It is reasonable to consider that transitions between neighbouring symbols are more likely to occur than transitions between more distant symbols. This idea can be represented as a model, by defining the probability of each transition as being inversely proportional to the

Euclidean distance⁹ between the symbols. This model is denoted by M_{ID} (where ID stands for Inverse Distance). For this analysis, the distance between any two symbols is defined as the distance between the centre of the bounding boxes of the two symbols, and is measured in pixels. The only exception to this distance measure is in the case of fraction lines. Since fraction lines can be quite wide, the distance between a fraction line and a symbol in either the denominator or numerator of that fraction is simply the vertical distance from the centre of that symbol to the fraction line. Consider the following expression which was used in this experiment.

$$\frac{4}{(y^2 + 9)x^{-8}}$$

For the version of this expression viewed by the participants, the distance between the symbol 4 and the fraction line was 19 pixels, the distance between the 4 and the + symbol was 38 pixels, while the distance between the 4 and the y was 57 pixels. Thus, according to the Inverse Distance Model, a transition from the 4 to the fraction line would be twice as likely as a transition from the 4 to the + symbol, and three times as likely as a transition from the 4 to the y . Note that as the distance between symbols for any expression is fixed, this model contains no free parameters.

The probability of the test data, D_{test} , given the Inverse Distance Model, M_{ID} , is $Pr(D_{test} | M_{ID}) = 10^{-6010}$ (to the nearest order of magnitude). This model is over 1800 orders of magnitude better at predicting the test data than using random chance which, as expected, is an enormous improvement. However, the Inverse Distance Model is not a very sophisticated model, and is unlikely to closely approximate the way in which experienced users of mathematics scan algebraic expressions. (Note that as the model analysis is being conducted with a very large number of transitions, the likelihood values for the various models are bound to be very low. As discussed in Section 4.2.3, it is the relative performance of the models that is important.)

The experimental results in this thesis thus far have clearly indicated that syntax plays

⁹If one symbol is located at (x_1, y_1) and another is located at (x_2, y_2) , then the Euclidean distance between them is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

an important role in the processing of algebraic expressions by experienced users of mathematics. Thus it is reasonable to hypothesize that the order in which the symbols of each expression are scanned corresponds to the ordering of the symbols as determined by syntactic structure (as discussed in Section 4.1). The transition data given in Appendix A is presented using a syntax-lrtb ordering of the symbols. For each of the eight expression groups, these transition tables clearly show that the majority of transitions from each symbol are to the symbol that occurs next in that expression's syntax-lrtb ordering. Thus, syntax forms the basis of the models that will now be considered.

In these syntax based models, the probability of a transition from one symbol to the next in a syntax-lrtb order is denoted by the parameter α . An example of such a transition, for the simple expression $8x^2 + 5$, is from the symbol $+$ to the 5. The transition data also suggests that participants often back-tracked, and that when they did so they most frequently went back by only one symbol in the order (such as a transition from the symbol x to the 8 in the above example). The probability for such a transition is denoted by the parameter β in the syntax based models. Furthermore, the transition data indicates that when scanning in a forward direction, frequently a symbol was skipped over (such as a jump from the symbol 2 to the 5). These transition probabilities are denoted by the parameter γ . Table 4.13 gives an example of which transitions these three parameters apply to in a Markov Chain for the example expression $8x^2 + 5$.

While the transition data suggests that these three parameters, α , β and γ , represent important aspects about how experienced users of mathematics scan algebraic expressions, it is not clear whether the best scanning model should include all or only some of these parameters. While including more parameters would most likely improve the predictive performance of a model, it would also add to that model's complexity. To deal with this trade-off between model performance and model complexity, the AIC and BIC model selection methods are used, both of which take into account these two important factors when selecting the most appropriate model.

Table 4.13: Markov Chain for the example expression $8x^2 + 5$, showing the role of the three parameters, α , β and γ , in the syntax based models.

From	To				
	8	x	2	+	5
8	0	α	γ	.	.
x	β	0	α	γ	.
2	.	β	0	α	γ
+	.	.	β	0	α
5	.	.	.	β	0

Four syntax based models were initially created. All of the models incorporated the α parameter, since the transition data indicated that most of the scanning was done in a syntax-lrtb order. They varied however, in which of the other two parameters they included. Syntax Model I incorporated the α parameter only. As back-tracking was the next most common feature of the transition data, Syntax Model II involved both the α and β parameters. Syntax Model III involved only the parameters α and γ , while Syntax Model IV involved all three parameters, α , β and γ .

The RFV fixation protocols indicate that there was occasionally some random scanning of the expressions. This is not unexpected, with possible causes including navigation difficulties (due to inexperience with the RFV perhaps), or simply indecision by the participant. When such scanning occurred, the resulting transitions were biased in favour of transitions to neighbouring symbols rather than to more distant symbols (an observation which formed the basis of the Inverse Distance Model). Thus, for each syntax based model, the probabilities of the transitions that are not represented by a parameter will be weighted according to the inverse distance between the symbols, such that transitions between spatially close symbols will be more probable than transitions between more distant symbols (as was the

case for the Inverse Distance Model). For example, consider the last row of the Markov Chain for the expression $8x^2 + 5$, shown in Table 4.13. The last row represents transitions from the symbol 5. The probability of the transition from the 5 to the + symbol will be set to the value of β , while based on the relative distances between the symbols, the transition from the 5 to the 2 will be more probable than the transition from the 5 to the x , which in turn will be more probable than the transition from the 5 to the 8 (since the symbol 2 is closer to the 5 than the symbol x , which in turn is closer than the symbol 8). If the β parameter is set to 0.4, then the other three possible transitions must have probabilities that sum to 0.6 (since all the probabilities in a Markov Chain row must sum to exactly 1).

For Syntax Models I, II, III and IV, the optimal parameter settings inferred using the training data, $D_{training}$, are shown in Table 4.14.

Table 4.14: Parameter values for Syntax Models I to IV, inferred using the training data, $D_{training}$, for Experiment 8.

Model	Parameter		
	α	β	γ
Syntax I	0.73	-	-
Syntax II	0.72	0.11	-
Syntax III	0.73	-	0.08
Syntax IV	0.73	0.10	0.08

It can be seen from these parameter values that the α parameter has far more importance than the others, as it represents transitions that are far more probable than any others. This raises the issue of whether or not the other parameters play an important enough role in a syntax based model to justify their presence. To determine this, the performance of the four models was examined using the test data, D_{test} , and the model scores for the two model selection criteria, AIC and BIC, were calculated. The results of these calculations

Table 4.15: Number of parameters, predictive accuracy and model selection scores for Syntax Models I to IV, for Experiment 8.

Model (M_i)	p_i	$Pr(D_{test} M_i)$	AIC Score	BIC Score
Inverse Distance	0	10^{-6010}	27676.5	27676.5
Syntax I	1	10^{-4227}	19467.5	19474.5
Syntax II	2	10^{-4227}	19470.7	19484.6
Syntax III	2	10^{-4184}	19273.8	19287.7
Syntax IV	3	10^{-4170}	19211.5	19232.4

can be seen in Table 4.15.

Table 4.15 shows that the syntax based models clearly outperform the Inverse Distance Model, and that the inclusion of each extra parameter in the syntax based models typically results in an improvement in predictive accuracy (shown as the $Pr(D_{test} | M_i)$ value) that is several orders of magnitude in size. As a result, both of the method selection criteria (AIC and BIC) indicate that the increased model complexity that arises from the inclusion of more free parameters is justified given the substantial improvement in predictive performance that results (the only exception being Syntax Model II). Syntax Model IV performs the best, with the lowest AIC and BIC scores. This indicates that the syntax-lrtb order, back-tracks and jumps over symbols are all important factors in how experienced users of mathematics scan algebraic expressions.

Further observation of the transition data, however, suggested that the forward jumps over symbols (represented previously by the parameter γ), did not occur uniformly. Specifically, the symbols that were more likely to be skipped over appeared to be exponents, operators and brackets. Thus, further models were developed in which the γ parameter was abandoned, and instead two new parameters were included: δ to represent skips over brackets and operators (plus and minus symbols as well as fraction lines), and ϵ to represent

Table 4.16: Markov Chain for the example expression $9(y^3 + 1)$, showing the role of the four parameters, α , β , δ and ϵ , in the syntax based models.

From	To						
	9	(y	^3	+	1)
9	0	α	δ
(β	0	α
y	.	β	0	α	ϵ	.	.
^3	.	.	β	0	α	δ	.
+	.	.	.	β	0	α	.
1	β	0	α
)	β	0

skips over exponents. Skips over other symbols were ignored, while the parameters α and β were retained. Table 4.16 gives an example of which transitions these new parameters apply to in a Markov Chain for the expression $9(y^3 + 1)$.

While the transition data suggests that the two new parameters, δ and ϵ , are important, it is not clear if the best scanning model should include both or only one of them. Thus, three new models were created. Syntax Model V contained the parameter δ , Syntax Model VI contained the parameter ϵ , while Syntax Model VII incorporated both δ and ϵ . All three models included the parameters α and β . The optimal parameter settings inferred for Syntax Models V, VI and VII, using the training data, $D_{training}$, are shown in Table 4.17.

The performance of the three models, using the test data, D_{test} , and scores for the AIC and BIC model selection methods are given in Table 4.18. Both Syntax Model V and Syntax Model VII outperformed Syntax Model IV. Syntax Model VII clearly represents the best scanning model of all the models considered in this experiment, with the lowest AIC score and also the lowest BIC score.

Table 4.17: Parameter values for Syntax Models V to VII, inferred using the training data, $D_{training}$, for Experiment 8.

Model	Parameter			
	α	β	δ	ϵ
Syntax V	0.72	0.11	0.08	-
Syntax VI	0.72	0.11	-	0.09
Syntax VII	0.72	0.11	0.08	0.09

Table 4.18: Number of parameters, predictive accuracy and model selection scores for Syntax Models V to VII, for Experiment 8.

Model (M_i)	p_i	$Pr(D_{test} M_i)$	AIC Score	BIC Score
Syntax V	3	10^{-4170}	19207.6	19228.5
Syntax VI	3	10^{-4213}	19409.5	19430.4
Syntax VII	4	10^{-4156}	19146.4	19174.3

The results of this model analysis indicate that syntactic structure forms an important part of how experienced users of mathematics scan algebraic expressions. This is consistent with what has been discovered about how experienced users of mathematics encode algebraic expressions (see Chapter 2). Both forward scanning and back-tracks are primarily based on the syntax-lrtb order of the symbols in the expression being scanned, as are skips over symbols.

For the skips over exponent symbols (represented in Syntax Model VII by the parameter ϵ), it is likely that the cause is in part due to the smaller size of these symbols. The size of the RFV focus window is based primarily around the standard sized symbols, which most of the symbols in the algebraic expressions are. As the bounding boxes surrounding exponent symbols are typically smaller than the size of the focus window, it is possible for exponent

symbols to be at least partially viewed in the focus window, and thus identified, without the centre of the focus window being over the symbol. Thus, it is likely that skips over exponent symbols are not actually part of the way in which experienced users of mathematics scan algebraic expressions, but rather reflect limitations of the RFV.

For brackets and operators however, the problem of small symbol size does not occur as these are standard sized symbols. This suggests therefore, that skips over these symbols (represented in Syntax Model VII by the parameter δ) occur intentionally when algebraic expressions are scanned. For operators, this is most likely due to the fact that there are strong visual cues that hint at the identity of the symbol (such as the vertical layout associated with fractions, or the extra spacing on either side of plus and minus symbols). Thus an explicit fixation may often not be necessary.

Brackets however, differ from the other symbols in another way. Unlike digits and variables which convey semantic information, brackets primarily play a syntactic role in an expression, being used to explicitly indicate the order of precedence of certain operations. Thus, as long as a reader is aware of them, they need not be explicitly fixated. This cannot be done with digits or variables, because being aware that a symbol is a digit or a variable is not enough; the value of the digit, or the name of the variable must be determined if the expression is to be used successfully. This result therefore, is analogous to what is seen in how readers parse natural language text, in which content words are more likely to be explicitly fixated than function words (Just & Carpenter, 1987). It is also consistent with the earlier analysis of where scanning begins, which showed that brackets are often not explicitly fixated when they are the first symbol of an algebraic expression.

The final part of the data analysis for this experiment further examines the role of syntax in the scanning of algebraic expressions by experienced user of mathematics. As seen in Section 1.2.1, the chunking of sentences of natural language is guided by syntax, with individual chunks conforming to grammatically defined units. Associated with this form of chunking is the observation that, when reading a passage of text, readers pause significantly

longer at the end of clauses and sentences (Mitchell & Green, 1978; Just & Carpenter, 1980). The purpose of this longer fixation is to allow the components of a grammatically defined unit to be encoded together as a chunk in working memory. It has already been shown in Chapter 2 that experienced users of mathematics also encode algebraic expressions into chunks that conform to grammatically defined units, and Experiment 8 has shown that the scanning of an algebraic expression is based primarily on the syntax-lrtb ordering of the symbols. Given the similarities between the encoding of sentences of natural language and mathematical expressions, it is reasonable to consider that experienced users of mathematics might also pause longer at the end of the syntactic constituents that make up an expression.

To determine if this is true, the symbols in the expressions used in this experiment were divided into two categories; those that represented the last symbol of a syntactic constituent, and those symbols that did not. For example, consider the following expression which was used in the experiment.

$$7(x^i + 2z)^2 - 3$$

In this example, the five symbols in red each represent the last symbol of a syntactic constituent, while the black symbols do not (based on a syntax-lrtb ordering of the symbols). The role of these symbols in the syntactic structure of the expression can be more clearly seen by examining the expression's parse tree, shown in Figure 4.7. The red symbols each correspond to the last symbol in a syntactic constituent of the parse tree (shown in the dashed boxes). Note that it is possible for a symbol (such as the z symbol in the example) to be the last symbol of more than one syntactic constituent.

If the parsing of algebraic expressions occurs using processes similar to those used in the parsing of natural language text, then one would expect the last symbol in each syntactic constituent to be fixated on for longer than other symbols. Table 4.19 gives the mean fixation times, along with the standard deviations (in parentheses), for both symbols that are the last element of a syntactic constituent (end symbols), and for all those that are not (non-end symbols), for each of the eight groups of expressions. Three-way ANOVAs

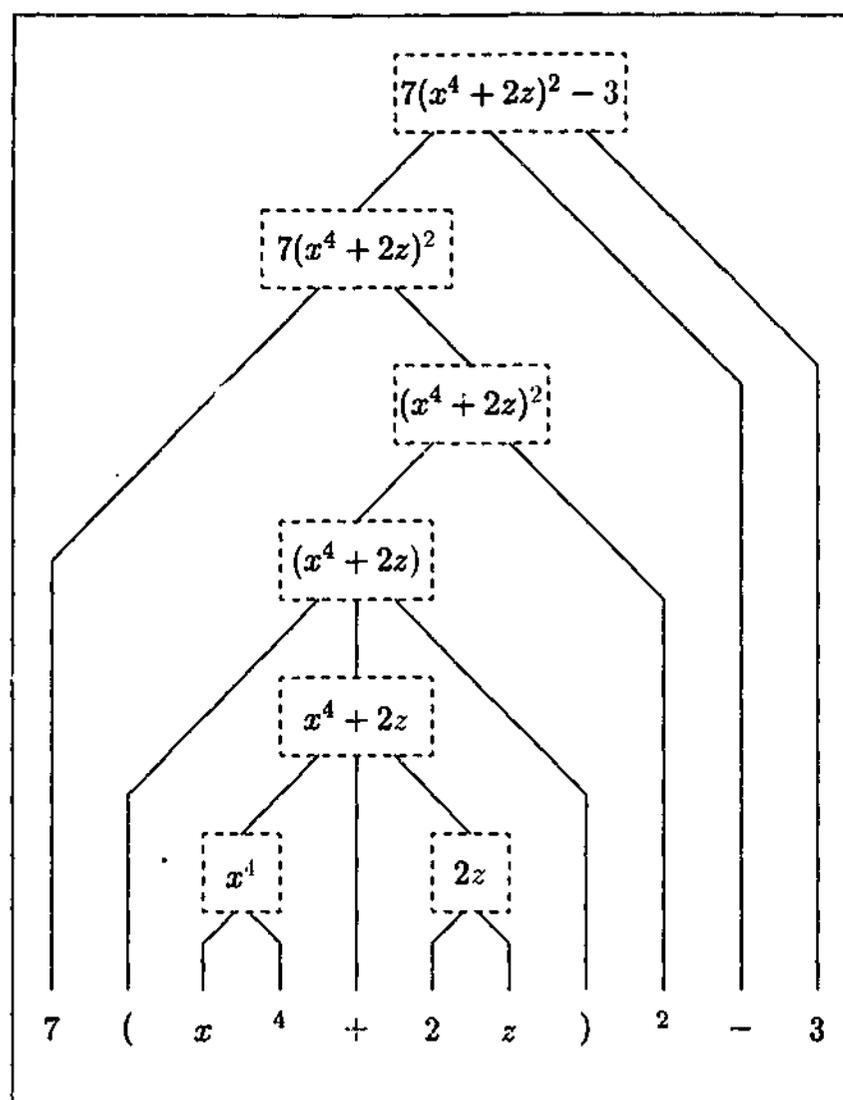


Figure 4.7: Parse tree for the expression $7(x^4 + 2z)^2 - 3$

(versions \times expression group \times symbol type) of the data were conducted, carried out separately over participant and item data.

The mean fixation times show that end symbols are fixated on for longer than non-end symbols (including operators), with a significant overall difference of 53ms ($F_1(1, 22) = 11.50$, $p < .05$, $F_2(1, 32) = 112.10$, $p < .05$). They also showed that there is a significant difference in the mean fixation times for symbols in the different expression groups ($F_1(7, 154) = 7.95$, $p < .05$, $F_2(7, 32) = 7.80$, $p < .05$), and that there is a significant interaction between the different expression groups, and whether a symbol was an end symbol or not ($F_1(7, 154) = 5.08$, $p < .05$, $F_2(7, 32) = 5.57$, $p < .05$). It is interesting to note that this last interaction appears to be largely due to a difference in the processing of items in

Table 4.19: Mean fixation times (ms) for symbols that are the last element of a syntactic constituent (end symbols) and those that are not (non-end symbols), as a function of expression group for Experiment 8.

Expression Group	Fixation Time (ms)	
	End Symbols	Non-End Symbols
1	247 (97)	205 (52)
2	241 (110)	234 (56)
3	273 (119)	220 (60)
4	301 (128)	212 (44)
5	315 (156)	227 (59)
6	284 (110)	246 (80)
7	318 (155)	256 (61)
8	264 (125)	214 (71)

expression group 2 (see Table 4.19), although it is not clear what features of this expression group have caused this difference. This is something that could be investigated further in the future.

These last results suggests that not only do end symbols take longer to process, but also that how much longer is influenced by the actual expression. It would appear that in some instances, the processes of parsing and encoding an algebraic expression into syntactic chunks is easier than in other instances, resulting in both shorter fixations overall, and less difference between the fixation times for end and non-end symbols. Of course, in light of previous results, one would expect that expressions containing a greater proportion of brackets or fraction lines would have smaller mean fixation times per symbol (for example, expression group 7 is the only one containing no brackets or fraction lines, and has the longest mean fixation times for each symbol category in Table 4.19).

One cause for concern in this analysis is that it was seen in Table 4.11 that operators (both fraction lines and plus and minus symbols) are fixated on for significantly less time

Table 4.20: Mean fixation times (ms) for symbols that are the last element of a syntactic constituent (end symbols) and the non-operator symbols that are not (non-end symbols, excluding operators), as a function of expression group for Experiment 8.

Expression Group	Fixation Time (ms)	
	End Symbols	Non-End Symbols (excluding operators)
1	247 (97)	223 (70)
2	241 (110)	259 (79)
3	273 (119)	257 (66)
4	301 (128)	243 (60)
5	315 (156)	224 (57)
6	284 (110)	257 (83)
7	318 (155)	260 (65)
8	264 (125)	210 (88)

than symbols such as digits and variables. This is important, because the operators that are used in this experiment never appear as the last symbol in a syntactic constituent. It is not clear therefore, whether the inclusion of operators in the analysis of end symbols versus non-end symbols may have resulted in a biased outcome. To avoid this possibility the analysis is again considered, this time with the non-end symbols restricted to only include those symbol types that could also act as end symbols (that is, excluding operators). The mean fixation times for this restricted class of non-end symbols can be seen in Table 4.20 (with the fixation times for the end symbols again listed for comparison).

A comparison of the mean fixation times for the end symbols with the restricted class of non-end symbols (excluding operators), shows that end symbols are still fixated on for longer, with a significant overall difference of 38ms ($F_1(1, 22) = 5.94, p < .05, F_2(1, 32) = 49.23, p < .05$). The other comparisons were also found to produce significant results.

Mean fixation times varied significantly with the different expression groups ($F_1(7, 154) = 5.94, p < .05, F_2(7, 32) = 5.74, p < .05$), and there was a significant interaction between the expression groups and whether a symbol was an end symbol or not ($F_1(7, 154) = 6.35, p < .05, F_2(7, 32) = 9.10, p < .05$). For these analyses, the only symbols used were digits, variables and brackets, all of which can appear in algebraic expressions as either end symbols or non-end symbols. Thus, these results indicate that, as with the parsing of natural language text, the parsing of algebraic expressions is indeed influenced by syntactic boundaries, with experienced users of mathematics pausing at the last symbol of each syntactic constituent (according to a syntax-lrtb ordering) in order to integrate the preceding symbols into a phrasal unit.

The scanning models developed in Experiment 8 have provided strong support for the idea that experienced users of mathematics scan the symbols of algebraic expressions in a syntax-lrtb order. Experiment 9 is designed to verify this result, by also examining a scanning order that has not yet been considered: a simple left-to-right (and top-to-bottom) scanning pattern, that does not take syntax into account. While the results of Experiment 8 suggest such a pattern is not used, they do not include an explicit comparison of a simple left-to-right (and top-to-bottom) scanning model with a syntax based scanning model. Therefore, these two alternatives are examined in Experiment 9, in which Markov Chains are again used to compare these scanning models.

4.4 Verifying the Scanning Models

Despite its two-dimensional nature, mathematical notation has evolved with a bias towards a left-to-right layout. In many respects, this is hardly surprising considering that much of the work during the development of mathematical notations (particularly in Europe) was in Latin. Since written Latin is a left-to-right language, mathematical notations also began to adopt this direction as mathematical problems evolved from being written as natural language statements, to being represented symbolically (see Section 1.1.4). For example,

consider the following equation.

$$y = 5 + 6x^2$$

The symbols of this expression are all horizontally adjacent, with all of the operations (such as equality, addition and multiplication) based on a left-to-right arrangement of symbols. Even the exponent, though written as a superscript symbol, is still placed to the right of its base symbol. The most common exception to such horizontal symbol arrangements is the division operation, which has a vertical layout that includes a fraction line (although even in this case, there are less common alternatives such as the \div symbol). There are other mathematic concepts that also use a vertical layout, such as the summation in the following expression.

$$\sum_{i=0}^M \frac{1}{i^2 + 3}$$

However, of the most commonly used mathematical operations, most are represented by symbols arranged in a left-to-right manner.

One side effect of this fact is that, for many algebraic expressions, a syntax-lrtb scanning order is identical to a simple left-to-right (and where appropriate, top-to-bottom) scanning order. Indeed, this was the case for the expressions used in Experiment 8, thus making it difficult to compare a model based on a syntax-lrtb ordering with a model based on the simpler left-to-right (and top-to-bottom) symbol order.

To compare two such scanning models, a task similar to the one used in Experiment 8 was constructed for Experiment 9. The main difference for this experiment is that the expressions of interest were designed such that the syntax-lrtb ordering of the symbols is different from a purely left-to-right (and top-to-bottom) ordering of the symbols (from here on referred to as an *lrb ordering*). These expressions contain two or three fractions, separated only by minus operations. Since the minus symbols between fractions are level with the fraction lines, all the symbols in the numerators of the fractions form an uninterrupted horizontal sequence, as do all the symbols in the denominators of the fractions. In between these two sequences is a row of horizontal lines consisting of fractions lines and

minus symbols, which acts as a strong visual boundary. Thus, if the scanning of algebraic expressions was biased towards an lrtb symbol order, then it would be expected that all of the numerator symbols would be processed first, followed by the row of horizontal lines (made up of fraction lines and minus symbols), and finally all of the denominator symbols. For example, consider the following expression.

$$\frac{x^2 + 1}{7} - \frac{4}{3y}$$

The syntax-lrtb ordering of the symbols in this expression is as follows.

→ x → 2 → + → 1 → ——— → 7 → - → 4 → — → 3 → y

In contrast, the lrtb ordering of the symbols is the following.

→ x → 2 → + → 1 → 4 → ——— → - → — → 7 → 3 → y

Again, the Restricted Focus Viewer (RFV) is used to track the order in which the symbols of each expression are scanned, and Markov Chains are used to compare the competing scanning models. The main aim is to determine if the best performed model in Experiment 8 (Syntax Model VII) is indeed a syntax based model, or instead is based purely on a left-to-right (and top-to-bottom) scanning model. Based on the results of Experiments 7 and 8, the syntax based model is expected to perform significantly better.

Experiment 9

Method

Participants Twelve participants successfully completed this experiment. No data from any participant was excluded.

Materials and Design Eighteen algebraic expressions were constructed for this experiment. Six of the expressions were designed such that the syntax-lrtb ordering of the symbols is distinct from a simple lrtb ordering of the symbols. These expressions contained two or three fractions, separated only by minus operations as described above. The six expressions of this form contained either eleven or fourteen characters, and are the stimuli used for testing the performance of the competing scanning models.

Table 4.21: Examples of the expressions used in Experiment 9.

Model Testing Expressions	Filler Expressions
$\frac{6+x}{7} - \frac{3}{x} - \frac{1}{y^2}$	$\sum_{i=2}^{N-1} (3x+7)^i$
$\frac{1}{y^3-6} - \frac{5}{2x}$	$\frac{4}{y^3} + \frac{z}{2} + \frac{z-1}{9}$
$\frac{x}{2} - \frac{y^2}{3} - \frac{1}{x-1}$	$\int_{-1}^1 \int_{2z}^{4z} z^2 - 2x \, dx \, dz$

The twelve remaining expressions contained between eleven and nineteen characters, and were used to add variety to the expression forms presented to participants, thus acting as fillers. Some of the fillers included more complex mathematical notations (such as summations and integrals), to help prevent any scanning direction biases occurring. Examples of the algebraic expressions used in this experiment can be seen in Table 4.21.

As with Experiment 8, statements that could be either true or false were constructed for each expression, in order to determine if the expression had been accurately processed by the participants. The nature of the statements was the same as for those in Experiment 8. Two versions of the experiment were also created, with the RFV focus window being initially placed to the right of each expression for half of the participants, and placed initially on the left of the expression for the other half. All of the expressions were shown in each version of the experiment. The RFV settings for this experiment were identical to those used in Experiment 8, and three additional expressions and statements were constructed as practice items.

Procedure The procedure for this experiment was identical to that for Experiment 8. The average width of the algebraic expressions in pixels was 156 (range 97-239) with an average height of 58 (range 51-67). The experiment took approximately 10 minutes to complete.

Data Treatment The data treatment for this experiment was identical to that for Experiment 8, using the same exclusion criterion, again ignoring the time taken to determine if a statement was correct, and converting the RFV data into fixation protocols and then transition data in the same manner.

Results and Discussion

The main aim of this experiment is to determine if the best performed model in Experiment 8 is indeed a syntax based model, or instead is based on a left-to-right (and top-to-bottom) scanning model. In Experiment 8, the best performed model was Syntax Model VII, which is based on a syntax-lrtb ordering of the symbols in each expression, and has four parameters, α , β , δ and ϵ . The α parameter represented transitions to the next symbol in the specified order, with the β parameter representing transitions to the previous symbol in the specified order. The δ parameter represented transitions that skipped forward over brackets or operators, while the ϵ parameter represented forward skips over exponents. All transitions not represented by a parameter are weighted according to the inverse of the distance between the symbols.

Syntax Model VII is one of the two models that will be used to examine the transition data from this experiment. The other model will be referred to as Left-to-Right Model VII. This model uses the same four parameters as Syntax Model VII, as well as weighting other transitions according to the inverse distance between symbols. However, it is based around an lrtb ordering of the symbols in each expression, rather than a syntax-lrtb ordering. For example, consider again the following expression (for which the syntax-lrtb and lrtb symbol orders are described in Section 4.4).

$$\frac{x^2 + 1}{7} - \frac{4}{3y}$$

In Syntax Model VII, the α transition (to the next symbol in the specified order) from the symbol 7 in this expression, is to the $-$ symbol. However, in Left-to-Right Model VII, the α transition from the symbol 7 is to the symbol 3. Note that in Experiment 8, since there

was no difference between the syntax-lrtb ordering and the lrtb ordering of the symbols in the expressions, all of the syntax based models in effect also represented a left-to-right based scanning model. Thus, for the expressions used in Experiment 8, there would have been no difference between Left-to-Right Model VII and Syntax Model VII.

The RFV fixation protocols for the 12 participants over the six model testing expressions, provide a total of 1,276 transitions. Appendix B gives the transition table data for each of the six model testing expressions. As with Experiment 8, the data was split into two subsets: training and test data. The training data consists of the transition data for six of the participants, which contains 581 transitions, and is used for parameter estimation for the scanning models. The test data consists of the transition data for the remaining six participants, which contains 695 transitions, and is used for assessing the performance of the models being considered.

For Syntax Model VII and Left-to-Right Model VII, the optimal parameter settings inferred using the training data, $D_{training}$, are shown in Table 4.22. The parameter settings for Syntax Model VII from Experiment 8 are also included for comparison.

From the parameter settings, it can be seen that Syntax Model VII is far more consis-

Table 4.22: Parameter values for the Experiment 9 models, inferred using the training data, $D_{training}$, along with the values inferred for the corresponding model in Experiment 8.

Model	Parameter			
	α	β	δ	ϵ
<i>Experiment 9</i>				
Syntax VII	0.63	0.10	0.13	0.15
Left-to-Right VII	0.23	0.08	0.01	0.11
<i>Experiment 8</i>				
Syntax VII	0.72	0.11	0.08	0.09

tent with the settings of the corresponding model in Experiment 8, than is Left-to-Right Model VII. Some differences between the settings for the two experiments were expected, due to differences between the expressions used in this experiment and those used in Experiment 8. For example, the expressions in this experiment contained more fractions but less brackets than those in Experiment 8, and some of the expressions in this experiment also consisted of more characters. However, from the parameter settings in Table 4.22, it is clear that Syntax Model VII follows a similar trend to the settings seen in Experiment 8, whereas Left-to-Right Model VII varies considerably.

To see if this difference in parameter settings translates into a significant advantage for Syntax Model VII, the performance of the two models was examined using the test data, D_{test} , and the model scores for the two model selection criterion, AIC and BIC (see Section 4.2.4), were calculated. Also, to serve as a naive comparison, the performance of the Inverse Distance Model (which was described in Experiment 8) was also examined using the test data from this experiment. The results of these analyses can be seen in Table 4.23.

Table 4.23: Number of parameters, predictive accuracy and model selection scores as a function of model type for Experiment 9.

Model (M_i)	p_i	$Pr(D_{test} M_i)$	AIC Score	BIC Score
Inverse Distance	0	10^{-587}	2703.3	2703.3
Syntax VII	4	10^{-468}	2164.2	2182.3
Left-to-Right VII	4	10^{-618}	2855.8	2874.0

Table 4.23 shows that Syntax Model VII clearly outperforms both the Inverse Distance Model and Left-to-Right Model VII, with a predictive performance that is over 100 orders of magnitude better than the other two models, and the lowest AIC and BIC scores. Surprisingly, the performance of Left-to-Right Model VII is significantly worse than that of the Inverse Distance Model. This indicates that not only is the scanning of algebraic expressions by experienced users of mathematics based primarily on syntax, but also that a left-to-right

(and top-to-bottom) scanning model gives inferior prediction compared to a model whose transition probabilities are based only on how close or distant neighbouring symbols are. This results also suggests that any of the seven syntax based scanning models used in Experiment 8 would outperform Left-to-Right Model VII, as in Experiment 8 the seven syntax based scanning models all performed significantly better than the Inverse Distance Model, whereas in this experiment Left-to-Right Model VII has performed significantly worse than the Inverse Distance Model.

This result also makes sense considering that more complex mathematical expressions can contain elements which would make an lrtb ordering of the symbols difficult to determine. For example, consider the following expression.

$$\sum_{i=0}^{M-1} \frac{(i-3)^2}{i+1}$$

While an lrtb symbol order would start with the three symbols that appear above the summation symbol, $M-1$, it is not clear which symbol would be next in such an order. The open bracket in the fraction numerator is to the right of the first three symbols, but it is not at the same horizontal level, and thus perhaps the summation symbol is more appropriate as the next symbol in the order. The summation symbol also presents other problems. Being larger than the other symbols in the expression, there are three symbols that appear to its immediate right, but at varying heights (the open bracket, fraction line, and the i symbol in the fraction denominator). This further obfuscates the appropriate symbol order.

Overall then, it is clear from this experiment that experienced users of mathematics prefer to scan the symbols of an algebraic expression in a syntax-lrtb order. However, this result has implications in an area of parsing that is important in natural language processing: syntactic ambiguity.

It appears that scanning algebraic expressions in a syntax-lrtb order rarely leads to any syntactic ambiguities during parsing.¹⁰ This means that, in general, as each new symbol of

¹⁰In fact, this appears to be the case for any syntax based scanning order.

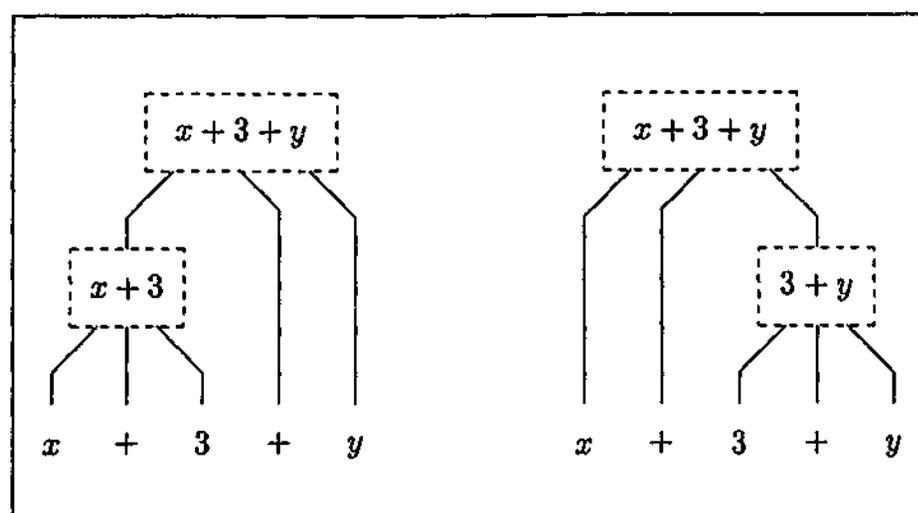


Figure 4.8: Parse trees for $x + 3 + y$

an expression is processed, there is only a single way in which it can be incorporated into the existing constituent structure of the expression, or there is only a single way in which the constituent structure can be modified to accommodate the new symbol. Moreover, when algebraic expressions that contain syntactic ambiguities are encountered, it often does not matter which syntactic structure is chosen, as all possibilities will still lead to a valid overall structure.

One type of algebraic expression that contains syntactic ambiguities consists of those that obey the law of associativity. When adding the terms in the expression $x + 3 + y$ for example, the result is not affected by the order in which the additions occur. The two possible orders in which addition can occur are represented by the two syntactic structures in Figure 4.8. For this expression, it does not matter if the syntactic structure on the left or on the right of Figure 4.8 is chosen, as both are equally valid. This is true even if the expression is part of a larger expression, such as $(x + 3 + y)^2$ or $\frac{8z}{x + 3 + y}$. Thus, even when a syntactic ambiguity is encountered, there will not be a "garden path" effect such as can occur with the parsing of natural language sentences. In this respect, algebraic notation is quite different from natural language.

It should also be noted that algebraic notation includes conventions that help to prevent syntactic ambiguities. For example, consider the following two expressions which contain

fractions within fractions.

$$\frac{3}{\frac{x}{2}} \qquad \frac{\frac{3}{x}}{2}$$

In both of these expressions, a smaller font size is used for the fractions that lie within another fraction, a common convention in the typesetting of algebraic expressions. This convention clarifies the constituent structure of the expressions, eliminating any potential ambiguities, and thus allowing them to be successfully parsed. (Note that brackets could also be used to clarify the correct structure of the expressions.)

As discussed in Section 1.2.1, syntactic ambiguity is an important issue in parsing sentences of natural language, as the first few words of a sentence can sometimes be matched to more than one possible valid sentence structure. The processing of further words in the sentence is then required to reveal the correct structure (although some complete sentences will remain syntactically ambiguous). Syntactic ambiguity makes the process of parsing sentences more complex, and attempts to explain how such ambiguities are dealt with have led to the development of both serial and parallel models of sentence parsing.

By avoiding syntactic ambiguities in the processing of algebraic expressions, or by only having ambiguities for which all possibilities are equally valid (so that no back-tracking is required), the parsing of algebraic expressions becomes much simpler, and requires less memory resources. This is obviously a useful feature, as using less memory in the processing of algebraic expressions allows more memory to be used in the mathematical manipulation of such expressions. It may well be that the syntax and conventions of algebraic notation evolved to eliminate syntactic ambiguities, thus making parsing more efficient. The absence of syntactic ambiguities would mean that modelling how algebraic expressions are parsed by experienced users of mathematics would be considerably simpler than modelling the parsing of natural language sentences. There would be no need for choices between serial and parallel processing, or the use of principles such as minimal attachment or late closure. However, the experiments in this chapter do not provide enough evidence to determine whether or not syntactic ambiguity is an issue in the parsing of algebraic expressions,

leaving this possibility as an interesting avenue for further research.

While the focus of this thesis has been limited to the most commonly used subset of algebraic notation (see Section 1.1.4), some of the filler expressions used in this experiment did contain examples of notations that represent more complex mathematical concepts (specifically, summations and definite integrals). While these expressions are not the focus of any data analysis in this thesis, they do represent areas in which the research described in this thesis could be extended. Thus, they are considered briefly here.

Both summations and integrals represent more advanced mathematical concepts than any of the other operations considered in this thesis, and this is reflected in their notational representations. For example, the representation of a definite integral has five components; the integral symbol, \int , the lower limit (subscript of the integral symbol), the upper limit (superscript of the integral symbol), the expression to be integrated, and a term to indicate the variable of integration (such as dx). These notations also use a two-dimensional layout for a single operation. The summation for example, is represented by a large summation symbol, \sum , with the range of the summation defined above and below this symbol (vertical adjacency), while the expression that the summation applies to is placed to the right of the summation symbol (horizontal adjacency).

Figure 4.9 gives example RFV scanpaths from two participants over an expression that contains a double summation, while Figure 4.10 gives example RFV scanpaths from two participants over an expression that contains a double integral.

Several features of how experienced users of mathematics parse expressions such as those in Figures 4.9 and 4.10 are suggested by the RFV scanpaths. For example, they

$$\sum_{i=0}^{2M} \sum_{j=0}^{P+1} a_i - b_j$$

Figure 4.9: Example RFV scanpaths from two participants over an expression with a double summation used in Experiment 9.

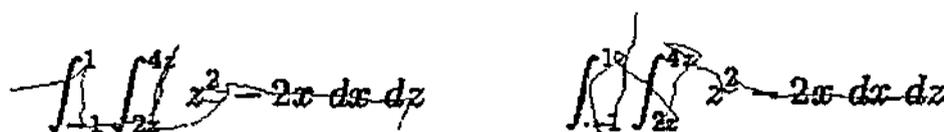


Figure 4.10: Example RFV scanpaths from two participants over an expression with a double integral used in Experiment 9.

show considerable variation among different individuals, especially when compared with the simpler algebraic expressions that have been the focus of this thesis. It is likely that this is in part due to a greater variance in the amount of experience that the different participants had with the notations used in these expressions. The scanpaths also suggest that certain participants were flexible in the way that they process such expressions. For example, the scanpath on the left of Figure 4.9 shows that the participant scanned the first summation from top to bottom, before moving to the second summation which was then scanned in a bottom to top direction. Another common feature of the scanpaths is that many regions are fixated on multiple times. This is not entirely surprising given that the expressions involve more complex mathematical concepts.

The parsing of these expressions by experienced users of mathematics appears to involve even more sophisticated processes than what has been seen for the simpler algebraic expressions that have been the focus of this thesis. Thus, this leaves plenty of scope for further research into understanding how mathematical notations are processed and understood by experienced users of mathematics. It would also be interesting to examine if there is a difference in the way in which participants scan an algebraic expression, depending on whether they subsequently answer a question about that expression correctly or incorrectly. Such research, however, is beyond the scope of this thesis.

4.5 Summary

The results presented in this chapter show that the scanning of algebraic expressions by experienced users of mathematics is primarily based on mathematical syntax. Evidence has been presented that directly supports this conclusion, as well as evidence that accounts for

competing scanning models.

The results of Chapter 2 indicate that the encoding of algebraic expressions by experienced users of mathematics strongly relies on a knowledge of mathematical syntax. Thus, in Experiment 7, the idea that the symbols of an algebraic expression were scanned in a syntax-lrtb order (as determined by the expression's syntactic structure) was considered. An expression construction task that involved the piecemeal presentation of an expression was developed to examine if experienced users of mathematics prefer a syntax-lrtb order when reading an algebraic expression. This experiment also investigated whether any advantage was gained by dividing the expression into syntax-based components as opposed to non-syntax-based components.

The results of this experiment showed that a syntax-lrtb order was preferred over other presentation orders (backward and jumbled) when reading an algebraic expression. They also showed that experienced users of mathematics found it easier to process syntax-based components, and that the amount of time required to memorize and integrate a component of an expression was affected by how much of that expression had already been read and memorized.

To further assess the issue of scanning order at a higher level of resolution, Experiment 8 examined scanning models using complete expressions and the Restricted Focus Viewer (RFV). Using Markov Chain models to analyse the order in which the symbols of an expression were fixated, the data was found to be consistent with the findings of Experiment 7, showing that syntax formed the primary basis for the scanning of algebraic expressions by experienced users of mathematics. The results also suggested several similarities with how natural language text is read.

Finally, it was noted that, for the algebraic expressions used in Experiment 8, the syntax-lrtb ordering of the symbols was identical to a left-to-right (and top-to-bottom) ordering of the symbols. This led to the construction of Experiment 9, which used expressions for which the syntax-lrtb ordering of the symbols was distinct from an lrtb ordering of the symbols.

Again, the RFV was used to track the order in which the symbols of each expression were scanned, with the data being used to compare the performance of a syntax based scanning model with a simple left-to-right (and top-to-bottom) scanning model. The results showed that the way in which experienced users of mathematics scan algebraic expressions correlates much more strongly with a syntax based scanning model. Although a left-to-right (and top-to-bottom) ordering is utilized within the grammatical constituents of an expression, experienced users of mathematics scan complete expressions in a manner that is primarily based on syntax, and not a simple left-to-right (and top-to-bottom) processing order.

Chapter 5

General Discussion

The aim of this thesis was to explore how experienced users of mathematics encode and parse algebraic expressions, in particular examining whether algebraic expressions are processed in a manner similar to sentences of natural language.

The investigation began with an examination of the encoding of algebraic expressions. The results of the first five experiments indicate that the internal representations used by experienced users of mathematics to encode algebraic expressions strongly rely on a knowledge of mathematical syntax that reflects phrasal structure. Experiment 1 showed that the encoding of expressions is guided by some systematic structural principle. Experiments 2 and 3 eliminated major roles for visual features or "lexical" tokens in the outcomes of Experiment 1. Experiment 4 pointed at a post-lexical level cause, and Experiment 5 confirmed a form of phrasal syntax as the guiding force in the encoding of algebraic expressions seen in Experiment 1.

These results extend and expand upon the previous findings of other researchers. For example, although the importance of spatial information in the processing of mathematical notations was examined by Kirshner (1989), it was not investigated in the context of other factors that might also be influential. In this thesis, however, the role of visual features (Experiment 2) was considered in conjunction with the roles played by syntactic structure (Experiments 1 and 5) and "lexical" tokens (Experiments 3 and 4). Taking into account these other factors makes it possible to refine previous findings such as that of Kirshner;

spatial information may be important, but syntax forms the primary basis of the internal representations used by experienced users of mathematics to encode algebraic expressions.

Following on from the investigation of how algebraic expressions are encoded, the focus of the thesis shifted to examining how experienced users of mathematics parse algebraic expressions. This investigation was primarily based on an analysis of how such expressions are scanned. An expression construction task was developed for Experiment 7, while in Experiments 8 and 9, the Restricted Focus Viewer (RFV) was used to record the order in which the symbols of an algebraic expression are processed. Using bounding boxes to define the stimulus elements of interest in the expressions (the individual symbols), the RFV scanpath data was converted into fixation data which revealed the order in which the symbols of the expressions were fixated upon, and for how long. Transition data was then obtained from the fixation sequences, which revealed how often attention shifted between pairs of symbols during the processing of the expressions. Markov Chains were then used to represent different possible models of how experienced users of mathematics scan algebraic expressions. These models defined the probability of each possible transition that appeared in the RFV transition data, allowing the likelihood of the observed transition data to be calculated.

Markov Chains are frequently used in other fields that involve sequential data (for example, analysing DNA sequences in Genetics), however, their use appears to be relatively uncommon in the analysis of scanpath data. This is surprising given the potential of this technique; the use of Markov Chain modelling in concert with the RFV data in this thesis provided a unique avenue for obtaining a more detailed account of the important features for the parsing of algebraic expressions than has been possible before. After the competing Markov Chain models had been developed, the AIC and BIC model selection criterion were then used to provide a rigorous statistical basis for comparing these models, examining not only how well the models predicted the transition data, but also taking into account model complexity.

The results of Experiments 7, 8 and 9 indicate that, as with the encoding of algebraic expressions, the scanning of algebraic expressions is based primarily on mathematical syntax. It was shown that, although a left-to-right (and top-to-bottom) ordering is utilized within the grammatical constituents of an expression, experienced users of mathematics scan complete expressions in a manner that is primarily based on syntax, and not a simple left-to-right (and top-to-bottom) processing order. Experiment 8 also produced several other interesting parsing related results. For example, fixation times varied for different symbol types, with symbols that convey semantic information (such as variables and digits) being fixated on for longer than symbols that play a more functional role (such as brackets). Symbols that conveyed semantic information were also less likely to be skipped over than other symbols. The results of Experiment 8 also indicated that when reading algebraic expressions, experienced users of mathematics also pause for longer at the end of syntactic constituents, further reinforcing the important role played by syntax in the processing of such expressions.

An important outcome of these results is that they reveal many marked similarities between the way in which experienced users of mathematics encode and parse algebraic expressions, and the way in which sentences of natural language are processed. For example, Johnson (1968, 1970) has shown that, in the case of natural language sentences, chunking is guided by syntax, with individual chunks conforming to grammatically defined units. This is analogous to the findings of Experiments 1 to 5, in which algebraic expressions were also shown to be encoded into grammatically defined units. When processing a sentence, content words are typically fixated on more often and for longer than function words (Just & Carpenter, 1987). This finding is similar to the results of Experiment 8 in which it was shown that symbols that convey semantic information are fixated on more often and for longer than symbols that play a more functional role. Also, it has been shown that when reading natural language text, people pause for longer at the ends of sentences and clauses, indicating that the scanning of words is being influenced by the syntactic structure of the

sentence (Mitchell & Green, 1978; Just & Carpenter, 1980). The results of Experiment 8 indicate that longer pauses at the end of phrasal constituents occurs for algebraic expressions too, with the results of Experiments 7 to 9 revealing that the scanning order used by experienced users of mathematics is also based on syntactic structure.

This outcome is somewhat surprising. While some similarities were expected between the processing of natural language and the processing of algebraic notations, the two-dimensional structure of algebraic expressions suggested that there might also be substantial differences. Certainly there are some important differences, such as the ability of experienced users of mathematics to accurately (although more slowly) process expressions presented in an unusual symbol order, and the apparent lack of syntactic ambiguities involved in the parsing of algebraic expressions. Overall, however, the results of the experiments in this thesis reveal a remarkable similarity between the processing of sentences of natural language, and the encoding and parsing of algebraic expressions by experienced users of mathematics.

5.1 Implications

The results of the experiments discussed in this thesis have implications in several areas. First of all, the similarity between the mechanisms used to process algebraic expressions and sentences of natural languages suggest that there may be common grammatical processes that apply generally to the processing of not just natural languages and mathematical notations, but possibly to many other forms of language also. If this is the case, then investigations into the processing of algebraic expressions and possibly other languages with a well defined syntax (such as music notation), may profit from drawing on the well established theories of natural language processing. It may be that all language processing shares certain core processes.

However, while the findings of this thesis are consistent with such a conjecture, they are far from proving it. For example, in the experiments discussed in this thesis, all of the

participants were native readers of English, which is processed in a left-to-right direction. Thus, the finding that the syntax-lrtb symbol order provides the best model for describing how algebraic expressions are scanned may be related to the fact that the participants normally read text in a left-to-right direction. It is not clear whether native readers of Hebrew or Chinese, for example, would also process algebraic expressions in manner akin to the syntax-lrtb model, or whether they would be influenced by the direction in which language text is read in those languages. Thus, this is one possible direction for future research.

A second implication is that the similarities between the processing of natural language text and the processing of algebraic notation by experienced users of mathematics could be taken into account in education when teaching students algebraic notation. If there are common grammatical processes that are used to process both algebraic notation and natural language text, then the difference between experienced users of mathematics and students may simply be experience. Direct comparisons between the structure of algebraic expressions and natural language sentences are unlikely to be of benefit, as people are not consciously aware how they parse sentences. However, tuition that focuses on the constituent structures of algebraic expressions, and how these structures can be combined to form larger expressions, may be of benefit.

A further implication is that understanding the way in which experienced users process algebraic expressions could be beneficial in creating more effective software for reading out mathematical expressions to blind people (for example, see Stevens, Brewster, Wright, & Edwards, 1994). By grouping the symbols in an expression in a manner consistent with the preferred encoding of the listener, information can be conveyed much more efficiently, allowing the expression to be understood faster and more easily.

5.2 Future Work

There are several areas in which the findings of this thesis can be expanded. The first is to go beyond the basic algebraic notations which have been the focus of this thesis, and explore the notations that are used to represent more complex mathematics concepts. These notations also play a vital role in many human endeavours; for example, in disciplines such as Engineering, calculus is very important, with differential equations and integrals used frequently. As was briefly discussed in Experiment 9, the inclusion of such notations appears to require the use of more sophisticated parsing processes. Thus, this area provides plenty of scope to add to our understanding of how humans process mathematical notations.

Another possibility for future work is to examine how algebraic expressions are processed by less skilled users of mathematics. This involves not just how expressions are encoded and parsed by people at specific skill levels, but also encompasses the learning process involved as a person goes from being a beginner to being highly competent. There has been some initial work done in this area which indicates that some inexperienced students encode algebraic expressions in a manner similar to experienced users of mathematics (Jansen, Marriott, & Yelland, 2000). However, this work is preliminary in nature, and there is considerable room for expansion.

Finally, similar investigations to those in this thesis can also be carried out on other visual languages, such as music notation or circuit diagrams. This would not only contribute to our understanding of how these particular languages are processed, but also to our understanding of how languages in general are processed.

5.3 Conclusion

Overall, the results of this thesis have indicated that the processing of algebraic notation has much in common with the processing of natural language. In particular, it has been shown that, like sentences of natural language, the encoding and parsing of algebraic expressions is based primarily on syntax, and reflects the constituent structure of the expressions being

processed. These results suggest therefore, that there may be certain core language processes that are common to the processing of both algebraic notation and natural language. If this is the case, there may well be common grammatical processes that are universally used in the processing of all forms of language. Thus, it may be that Chomsky's universal grammar is more universal than even he thought.

Appendix A

Transition Data for Experiment 8

This appendix contains the transition data derived from the RFV fixation protocols obtained for the 24 participants in Experiment 8 (see Section 4.3). For each of the eight expression groups, the symbols in the expressions are mapped to the first 11 alphabetic characters (*a* through *k*) in order to allow the transition tables to be presented in a clear and concise manner. For each expression group, the order of the alphabetic characters that result from the symbol mapping corresponds to the syntax-lrtb ordering of the symbols, which is based on syntactic structure (as defined in Section 4.1). For example, consider the following mapping from the simple expression on the left, to the characters on the right.

$$(x + 3)^2 \qquad abcde^f$$

Each of the symbols in the expression on the left corresponds to the symbol in the same position in the expression on the right. Thus, this symbol mapping consists of the symbol pairs shown in Table A.1.

Table A.1: Symbol mapping example for the expression $(x + 3)^2$.

Original Expression Symbol ...	is Mapped to Character
(a
<i>x</i>	b
+	c
3	d
)	e
²	f

Expression Group 1

The symbol mapping for this expression group is defined as follows.

$$\frac{(v^d o d)^{v o d}}{d} \qquad \frac{a t^c d e f^g h i}{k}$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, v represents a variable, d represents a digit and o represents a plus or minus operator. Note that for this expression group, the fraction line is mapped to the symbol j .

Table A.2: Transition Table for Expression Group 1.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	82	14	1	1	0	0	0	0	0	0
b	13	0	147	24	8	0	1	2	0	5	3
c	0	32	0	129	23	5	0	2	1	0	0
d	5	14	10	0	151	9	3	1	0	1	1
e	2	3	3	14	0	174	19	6	1	6	6
f	1	3	2	5	31	0	149	16	2	8	5
g	0	0	0	2	6	18	0	175	2	3	2
h	1	3	2	4	3	8	30	0	149	26	14
i	3	3	0	2	4	6	5	43	0	61	19
j	14	32	12	9	4	1	2	2	1	0	95
k	5	8	2	4	3	1	1	1	0	50	0

Expression Group 2

The symbol mapping for this expression group is defined as follows.

$$\frac{d}{(v^d o d)^{v o d}} \qquad \frac{a}{cd^e f gh^{i j k}}$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, v represents a variable, d represents a digit and o represents a plus or minus operator. Note that for this expression group, the fraction line is mapped to the symbol b .

Table A.3: Transition Table for Expression Group 2.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	82	24	13	13	0	0	0	3	0	2
b	56	0	36	30	20	1	4	3	1	4	2
c	8	5	0	83	4	0	2	2	0	0	0
d	4	5	10	0	146	23	4	0	0	1	1
e	3	9	1	43	0	132	9	1	1	1	0
f	3	2	2	8	8	0	149	8	6	1	0
g	3	3	2	3	3	18	0	149	10	4	2
h	0	6	0	1	1	4	26	0	131	11	2
i	5	8	2	2	1	4	3	16	0	147	0
j	5	8	2	1	3	5	2	1	28	0	132
k	8	9	5	3	1	3	4	3	9	40	0

Expression Group 3

The symbol mapping for this expression group is defined as follows.

$$v o \frac{d v o d}{d o v^d} \qquad a b \frac{c d e f}{h i j^k}$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, v represents a variable, d represents a digit and o represents a plus or minus operator. Note that for this expression group, the fraction line is mapped to the symbol g .

Table A.4: Transition Table for Expression Group 3.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	138	10	6	2	0	10	6	4	0	0
b	32	0	110	6	1	0	25	6	0	0	1
c	8	3	0	161	10	3	11	2	1	1	0
d	5	2	34	0	169	10	11	3	3	1	0
e	3	2	6	25	0	159	14	4	6	1	1
f	4	9	9	9	26	0	63	34	20	2	2
g	12	9	24	28	7	7	0	52	30	5	5
h	7	4	0	2	0	0	3	0	137	9	3
i	6	5	0	1	0	0	2	48	0	154	7
j	9	5	4	1	2	1	22	8	18	0	107
k	10	3	4	2	6	1	23	3	9	22	0

Expression Group 4

The symbol mapping for this expression group is defined as follows.

$$\frac{d v o d}{d o v^d} o v \qquad \frac{a b c d}{f g h^i} j k$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, *v* represents a variable, *d* represents a digit and *o* represents a plus or minus operator. Note that for this expression group, the fraction line is mapped to the symbol *e*.

Table A.5: Transition Table for Expression Group 4.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	145	18	3	5	1	1	2	0	0	1
b	48	0	159	10	11	4	4	2	0	2	0
c	7	34	0	168	8	8	3	1	0	2	0
d	5	6	31	0	73	46	23	5	0	12	0
e	27	29	9	8	0	55	24	8	3	12	3
f	4	2	0	0	11	0	144	13	1	3	1
g	2	1	1	0	6	50	0	161	11	1	0
h	1	0	1	0	10	4	24	0	142	25	9
i	5	4	2	2	31	1	5	11	0	85	7
j	5	3	1	3	8	2	1	5	1	0	133
k	9	8	8	9	11	1	1	9	2	23	0

Expression Group 5

The symbol mapping for this expression group is defined as follows.

$$d(v^d o dv)^d o d \qquad abc^d e fgh^i j k$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, v represents a variable, d represents a digit and o represents a plus or minus operator.

Table A.6: Transition Table for Expression Group 5.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	144	27	2	1	0	1	0	0	1	0
b	43	0	150	8	2	0	0	0	0	0	0
c	13	31	0	156	30	6	4	0	1	0	1
d	5	3	24	0	129	17	3	2	0	1	0
e	4	1	7	11	0	153	9	3	2	2	0
f	4	1	0	4	13	0	169	13	0	0	0
g	1	3	7	0	6	19	0	160	11	3	1
h	5	1	3	0	2	1	23	0	138	16	2
i	4	2	0	1	3	2	6	9	0	111	14
j	1	0	3	0	4	2	1	3	2	0	119
k	18	4	5	2	3	6	1	2	2	13	0

Expression Group 6

The symbol mapping for this expression group is defined as follows.

$$d o d(v^d o d v)^d \qquad a b c d e^f g h i j^k$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, *v* represents a variable, *d* represents a digit and *o* represents a plus or minus operator.

Table A.7: Transition Table for Expression Group 6.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	134	13	1	2	0	1	0	0	0	0
b	25	0	145	10	4	0	0	0	0	0	1
c	11	17	0	153	12	2	2	2	0	1	0
d	4	6	30	0	143	14	3	2	0	1	0
e	3	4	3	25	0	139	38	3	2	3	0
f	0	2	2	3	28	0	120	9	2	2	1
g	2	0	0	2	21	8	0	149	9	2	0
h	0	0	2	1	5	3	11	0	164	6	2
i	0	0	4	2	2	0	8	18	0	166	6
j	5	4	0	2	0	2	6	5	29	0	133
k	14	3	1	6	6	2	9	8	9	25	0

Expression Group 7

The symbol mapping for this expression group is defined as follows.

$$v^{d \circ v} \circ d v^d \circ d v \qquad a^{b c d} e f g^h i j k$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, v represents a variable, d represents a digit and \circ represents a plus or minus operator.

Table A.8: Transition Table for Expression Group 7.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	148	28	5	2	0	1	0	1	0	0
b	15	0	162	4	3	0	0	0	0	0	0
c	18	19	0	174	11	3	1	0	1	0	0
d	8	4	25	0	134	8	3	3	0	0	0
e	9	1	4	4	0	146	10	0	1	0	0
f	5	2	1	2	11	0	163	7	4	2	0
g	9	2	0	0	7	35	0	153	14	3	0
h	5	0	2	1	2	4	18	0	115	21	2
i	1	0	0	0	4	0	17	9	0	121	9
j	6	0	0	0	0	0	7	0	18	0	136
k	9	3	3	2	2	4	9	0	10	37	0

Expression Group 8

The symbol mapping for this expression group is defined as follows.

$$(v o d)^d(v o d) \qquad abcde^fghijk$$

The expression form is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. In the expression form, *v* represents a variable, *d* represents a digit and *o* represents a plus or minus operator.

Table A.9: Transition Table for Expression Group 8.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	101	7	1	1	0	0	0	0	0	0
b	19	0	140	6	1	1	0	1	0	0	0
c	6	18	0	145	7	3	2	1	1	1	0
d	2	5	12	0	145	12	5	1	1	0	0
e	2	2	7	18	0	124	11	4	0	1	0
f	0	1	4	5	8	0	105	22	4	1	0
g	0	1	1	0	4	10	0	109	8	1	2
h	0	1	1	2	1	1	12	0	124	9	0
i	2	1	1	1	1	0	1	14	0	124	1
j	4	5	4	1	4	0	1	3	12	0	74
k	1	1	1	3	1	2	0	2	2	13	0

Appendix B

Transition Data for Experiment 9

This appendix contains the transition data derived from the RFV fixation protocols obtained for the 12 participants in Experiment 9 (see Section 4.4). For each of the six model testing expressions, the symbols in the expressions are mapped to the first 11 or 14 alphabetic characters (depending on the number of characters in the expression) in their syntax-lrtb order (as done in Appendix A).

Note that the transitions for a simple left-to-right (and top-to-bottom) ordering of the symbols in each expression are the same as those contained within the transition tables in this appendix. The only difference is that with an alternative symbol ordering, each possible transition will be in a different location in the table. For example, consider the two transition tables shown in Table B.1. Both of these tables represent the exact same transition data, with the only difference being the ordering of the symbols that index the table.

Table B.1: Two example transition tables which contain the exact same data, but are presented using different symbol orders.

From	To			
	a	b	c	d
a	11	12	13	14
b	21	22	23	24
c	31	32	33	34
d	41	42	43	44

From	To			
	b	d	c	a
b	22	24	23	21
d	42	44	43	41
c	32	34	33	31
a	12	14	13	11

Expression 1

The symbol mapping for this expression is defined as follows.

$$\frac{x^2 + 1}{7} - \frac{4}{3y} \qquad \frac{a^b c d}{f} g \frac{h}{jk}$$

The expression is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. Note that for this expression, the fraction lines (from left to right) are mapped to the symbols *e* and *i*.

Table B.2: Transition Table for Expression 1.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	8	1	1	0	0	0	0	0	0	0
b	1	0	12	1	0	0	0	0	0	0	0
c	1	2	0	15	0	0	0	0	0	0	0
d	0	0	2	0	9	5	0	1	0	0	0
e	2	0	1	0	0	9	1	0	0	0	0
f	0	0	0	0	2	0	8	3	2	0	0
g	0	0	1	0	0	0	0	6	4	1	0
h	0	0	0	0	0	0	0	0	11	0	3
i	0	0	0	0	0	0	1	4	0	6	6
j	0	1	0	0	0	0	1	0	0	0	8
k	0	1	1	0	1	1	0	0	0	4	0

Expression 2

The symbol mapping for this expression is defined as follows.

$$\frac{2+y^4}{8} - \frac{4x}{9} \qquad \frac{abc^d}{f} g \frac{hi}{k}$$

The expression is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. Note that for this expression, the fraction lines (from left to right) are mapped to the symbols *e* and *j*.

Table B.3: Transition Table for Expression 2.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	14	1	0	1	0	2	0	0	0	0
b	4	0	17	1	3	0	0	0	0	0	0
c	0	2	0	12	8	1	0	0	1	0	0
d	0	0	5	0	3	3	1	0	1	0	0
e	4	4	0	0	0	12	1	1	0	1	0
f	1	2	0	0	2	0	6	2	2	3	0
g	0	0	1	0	1	0	0	8	0	2	0
h	0	0	0	0	1	0	0	0	15	2	0
i	1	2	0	0	0	0	1	5	0	11	3
j	0	0	0	0	1	0	1	2	4	0	8
k	2	0	0	0	0	2	1	0	1	2	0

Expression 3

The symbol mapping for this expression is defined as follows.

$$\frac{1}{y^3 - 6} - \frac{5}{2x} \qquad \frac{a}{c^d e f} g \frac{h}{jk}$$

The expression is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. Note that for this expression, the fraction lines (from left to right) are mapped to the symbols *b* and *i*.

Table B.4: Transition Table for Expression 3.

From	To										
	a	b	c	d	e	f	g	h	i	j	k
a	0	12	0	4	0	0	0	0	0	0	0
b	2	0	7	6	1	0	3	0	0	0	0
c	1	1	0	9	6	1	0	0	0	0	0
d	1	1	7	0	11	3	0	0	0	0	0
e	0	0	0	1	0	18	1	0	1	0	0
f	2	3	1	0	3	0	10	2	0	0	0
g	3	2	0	0	0	0	0	8	4	0	1
h	0	0	0	0	0	0	1	0	15	0	0
i	0	0	0	0	0	0	1	5	0	10	6
j	1	0	0	0	0	0	0	1	0	0	11
k	0	0	2	1	0	0	1	0	2	4	0

Expression 4

The symbol mapping for this expression is defined as follows.

$$\frac{x}{2} - \frac{y^2}{3} - \frac{1}{x-1} \qquad \frac{a}{c} d \frac{e^f}{h} i \frac{j}{lmn}$$

The expression is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. Note that for this expression, the fraction lines (from left to right) are mapped to the symbols *b*, *g* and *k*.

Table B.5: Transition Table for Expression 4.

From	To													
	a	b	c	d	e	f	g	h	i	j	k	l	m	n
a	0	12	5	2	1	0	0	0	0	0	0	0	0	0
b	3	0	12	1	0	0	0	0	0	0	0	0	0	0
c	2	2	0	9	3	0	2	0	0	0	1	0	0	0
d	0	0	0	0	8	3	2	0	0	0	0	0	0	0
e	1	0	0	1	0	8	10	3	0	0	0	0	0	0
f	0	0	0	0	4	0	7	0	0	0	0	0	0	0
g	1	0	1	0	5	0	0	14	2	2	0	0	0	0
h	0	0	0	0	1	0	4	0	8	1	1	0	1	0
i	0	2	0	0	0	0	0	0	0	2	9	0	0	0
j	0	0	0	0	1	0	0	0	1	0	9	1	0	0
k	0	0	1	0	0	0	0	0	2	6	0	11	1	1
l	0	0	0	0	0	0	1	0	0	0	1	0	12	1
m	1	0	0	0	0	0	0	0	0	0	0	3	0	12
n	1	0	0	0	0	0	0	0	0	1	1	3	3	0

Expression 5

The symbol mapping for this expression is defined as follows.

$$\frac{3}{x^2} - \frac{y}{7} - \frac{y+2}{4} \qquad \frac{a}{c^d} e \frac{f}{h} i \frac{jkl}{n}$$

The expression is shown on the left, and the mapped symbols in the same relative positions in the expression on the right. Note that for this expression, the fraction lines (from left to right) are mapped to the symbols *b*, *g* and *m*.

Table B.6: Transition Table for Expression 5.

From	To													
	a	b	c	d	e	f	g	h	i	j	k	l	m	n
a	0	17	5	1	0	0	0	0	0	0	0	0	0	0
b	3	0	12	6	1	0	0	0	1	0	0	0	0	0
c	1	1	0	11	3	2	2	0	0	1	0	0	0	0
d	0	1	4	0	12	2	0	0	0	0	0	0	0	0
e	1	1	0	0	0	12	5	0	0	0	0	0	0	0
f	1	0	0	0	0	0	16	5	2	1	0	0	0	0
g	0	0	0	0	0	7	0	14	3	1	0	0	0	0
h	3	0	0	1	0	2	2	0	8	1	1	0	2	0
i	2	0	0	0	1	1	0	0	0	8	1	0	3	0
j	2	0	0	0	0	0	0	0	0	0	15	1	1	0
k	0	0	0	0	0	0	0	0	0	2	0	16	1	3
l	1	0	0	0	0	0	0	0	0	1	3	0	10	3
m	0	0	0	0	0	0	0	0	2	4	1	0	0	11
n	2	1	0	0	2	1	0	1	0	0	1	0	0	0

Appendix C

Restricted Focus Viewer Version 2.1 User's Manual and Tutorial

The Restricted Focus Viewer (RFV) is a computer based tool for tracking visual attention. It has been developed as an alternative to eye-tracking equipment, although it is not intended to be a replacement for such equipment. It is a cheap, non-intrusive and easy to set up system, that provides accurate data about which region of a stimulus is being looked at. This enables experimenters to determine which elements of a stimulus are being focused on at any time while a task is being performed. The RFV uses image blurring to restrict how much of a stimulus can be viewed in focus, while the computer mouse is used to shift the region of focus to different areas. Data from the RFV can also be replayed using the companion Replayer program.

This report describes Version 2.1 of the RFV. This is a totally new version of the RFV, and it is not compatible with any previous versions, including Version 2.0 (described in Jansen, 2000). Many new features have been added to this version, providing more flexibility for experimental designers. These include the experiment designer being able to control the layout of visual elements, and providing feedback to participants based on their responses. Enhancements have also been added to make the program more user friendly. For example, only a single input file is now needed for any given experiment, with the program able to modify the order of the experimental items for different participants.

Section C.1 will discuss the system requirements for running the RFV and Replayer programs, including the tools required and the files that need to be downloaded. In Section C.2, the RFV program is described in more detail, focusing specifically on how the program uses stimuli in tracking visual attention. This section will also include recommendations for setting RFV parameters that deal with stimulus presentation. Section C.3 will look at the Input File that the experimenter must create in order to run an experiment using the RFV. Then, Section C.4 describes the actually running the RFV program on a computer, and also discusses how the RFV user interacts with the program. Section C.5 provides a tutorial that illustrates the features available in Version 2.1 of the RFV. In Section C.6, the output data file produced by the RFV is described, and then in Section C.7, the Replayer program that uses these output data files is examined to illustrate how data from the RFV can be replayed. The Replayer is a very useful tool for both data analysis, and also for generating retrospective verbal protocols. Finally, Section C.8 contains the specifications for the input file parameters.

The RFV has only recently been created, and the development process is continuing. Therefore, any feedback on either of the RFV or Replayer programs (or this User's Manual) would be welcome. The testing done with the RFV to date has given promising results (Blackwell et al., 2000), and it is hoped that other researchers will also find the RFV to be a useful tool in understanding cognitive processes that involve visual attention, and examining how people reason with visual stimuli.

C.1 System Requirements

The RFV and Replayer programs have been written in Java, which is a platform independent language. As such, it can be run on many different architectures and with many different operating systems, including PC's running either Windows or Linux. However, Java is an interpreted language and as such does not run as fast as code that is compiled specifically for a particular system. Thus it is recommended that a reasonably fast machine be used, with a clock speed of at least 300 MHz.

C.1.1 Java 2

The RFV and Replayer programs were written using the Java 2 language. Both programs are standalone Java applications, not applets. The user needs to have tools for running Java 2 programs (Java software version 1.2 or higher) on their system in order to use them. Note that they will not work with tools for Java software versions 1.0 or 1.1. Tools needed to run the programs, such as either a Development Kit or a Runtime Environment for Java 2, can be downloaded from the following website:

<http://java.sun.com/j2se/1.3/>

This site contains downloads for various architectures and operating systems, as well as documentation for using the tools.

Some releases of Java 2 for specific platforms include a Just-In-Time (JIT) compiler. This allows Java code to be compiled in a system specific manner, which can result in the program running faster. The user may wish to take advantage of this if they have a system for which a JIT compiler is available. Note that if a JIT compiler is available, it is usually run by default by the Java application launcher.

C.1.2 Program and Source Files

To run the RFV and Replayer programs, the `RFV.jar` and the `Replayer.jar` files are needed. These files are available for free download and use. The source code (in `.java` files) is also available so that the programs may be modified. The website where the files can be obtained from is:

<http://www.csse.monash.edu.au/~tonyj/RFV/>

The programs and source code are available under the conditions of the GNU Public License, a copy of which is available at the above website.

C.2 Description of the RFV

The Restricted Focus Viewer (RFV) is a computer based tool for tracking visual attention. Its design is in part based on the human visual system, which can only focus on objects at the centre of the visual field. The region surrounding this area of sharp focus is still perceived, but the further from the centre of the visual field an object is, the more coarse is the perception of it (Tovée, 1996).

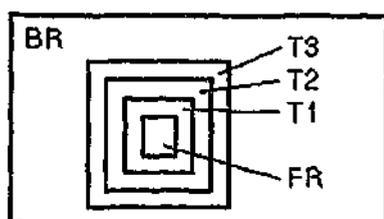
The design of the RFV attempts to reflect this idea through the use of image blurring. The RFV displays a blurred stimulus image on a computer monitor, allowing the participant to see only a small region of the stimulus in focus at any time. The region in focus, called the focus window, can be moved using the computer mouse. The RFV records what the participant is focusing on at any point in time, and the data can be played back using the Replayer program.

C.2.1 The Focus Window

The focus window of the RFV is the region in which the stimulus is visible in full detail. In order for the focus window to look 'natural', a graded blurring effect is needed, such that the transition from blurred to focus appears smooth and seamless.

A graded blurring effect is achieved by the technique illustrated in Figure C.1. The outer rectangle defines the stimulus area which is fully blurred. The innermost box is the region of focus.

Surrounding this focus region are three transition regions. Each transition region is slightly more blurred than the last, so that there is only a subtle difference between neighbouring regions. The overall result is the appearance of a smooth transition from the region of the image in focus, to the region which is fully blurred. Using the mouse to move the focus window therefore moves not only the focus region, but also the three transition regions.



- FR - Focus Region
- T1 - Innermost Transition Region
- T2 - Middle Transition Region
- T3 - Outermost Transition Region
- BR - Blurred Region

Figure C.1: Regions of the stimulus used to achieve the graded blurring effect.

It is clear then that apart from the stimulus image in full focus, four other images are needed for the different levels of image blurring in the blurred and transition regions. At the centre of the focus window there is also a small black dot, to allow users to keep track of the focus window location when it is centred on an empty region of the image.

C.2.2 Stimulus Images and Blurring

The aim of the blurred region is that it should still be possible to perceive the broad structure of the original stimulus image. The individual components and finer details in the stimulus should be indiscernible, requiring the user to move the focus window over that area of the stimulus in order to determine exactly what is there. However, it should not be so blurred that the user has difficulty in navigating from one stimulus component to another. Figure C.2 gives an example with an algebra expression as the stimulus, and the corresponding blurred image in which the symbols are indiscernible.

$$4 + \frac{x + 7}{9(4 + 9y)}$$



Figure C.2: Example of a visual stimulus and its corresponding blurred image.

Different kinds of images have different spatial properties. Thus, different techniques are required in order to produce a blurred image that obscures the finer details, while still allowing the general form of the image to be perceived. Since a human observer is needed to verify if an appropriate level of image blurring has been obtained, the RFV does not do any image blurring itself, but rather reads in image files that have already been blurred.

For each stimulus to be presented, five image files need to be provided. These are the image proper (that is, the image in full focus), the image with the appropriate level of blurring for the innermost transition region, the images for middle and outermost transition regions, and finally the fully blurred image. Figure C.3 gives an example of the five images used to present the stimulus shown in Figure C.2, with the fully focused image at the top and fully blurred image at the bottom. Each of the three transition region images in between are slightly more blurred than the image above it. The RFV program dynamically combines these images to produce a smooth transition from the blurred region to the region in full focus (using the method described in the previous section). Figure C.4 gives two examples of the focus window in different positions over the stimulus shown in Figure C.2.

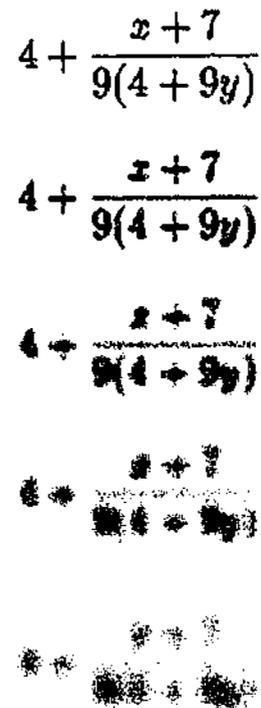


Figure C.3: An example of the five images needed to present a stimulus.



Figure C.4: Two examples of the focus window on different regions of the stimulus.

As was mentioned before, different techniques are required to successfully blur different types of images. Consider for example, mathematical equations and circuit diagrams as stimuli. Mathematical equations usually are composed of a closely packed group of symbols. A standard blurring algorithm, which can be found on most modern computer graphics programs, is generally sufficient to successfully blur such a stimulus.¹ Different levels of blurring can be achieved by varying the blur radius, or by running the blur algorithm on the image more than once.

Circuit diagrams on the other hand usually have large empty regions, and single lines representing wires that are hard to blur. One approach to effectively blur such images is to pixelize the image first (that is, decrease the image resolution without changing the image size, which again can be done by using a pixelize algorithm found on most computer graphics programs), and then run a standard blurring algorithm. By varying the size of pixelization, and the amount of blurring, and running these algorithms multiple times on a stimulus image, it is possible to obtain various levels of blurring that allow the general form of the diagram to still be perceived, but leave the individual components indiscernible.

¹For example, on systems running Windows, the program Paint Shop Pro has a filter which allows images to be blurred. On Unix systems, the program XV has a blur algorithm.

C.2.3 Motion Blur

Another feature that was implemented so that the RFV would more accurately mimic the way humans perceive visual stimuli is motion blur. If the user of the RFV moves the mouse at high speed (that is, over a large distance on the screen in a small amount of time), the focus window will not achieve full focus. Once the user reduces the speed of the mouse motion back to below a certain threshold, or stops moving the mouse completely, full focus in the focus window will return. This feature helps in defining the temporal boundary between fixations and movements.

When the focus window is stationary or moving slowly, all of the regions listed in Figure C.1 are present. During motion blur however, only the outermost transition region is added to the blurred stimulus. Because this region has less blurring than the rest of the image, the user is still able to track the location of the focus window on the stimulus. However, it is not possible to determine the finer details of that location without slowing or stopping the mouse. Only then will full focus be available.

C.2.4 Guidelines for Setting Parameters

When designing an experiment that will use the RFV, there are several important parameters that the experimenter must set. Below are some basic heuristic guidelines for setting these parameters. These are suggestions based on previous experience in using the RFV, but by no means are they intended to be strict rules for parameter settings.

Level of Blurring

The goal of blurring the stimulus is to generate an image where it is difficult to identify the finer details. The minimum level of image blurring should be sufficient that any two stimulus elements are indistinguishable, and that the connections between elements cannot be established. It should only be possible to accurately identify an element if that element is in the focus region. Of course, for different types of stimuli the size of the elements will vary, and thus different levels of blurring will be necessary. The maximum level of blurring should still allow identification of the stimulus boundaries (at least the convex hull). This is to ensure that participants can still navigate from one region of interest in the stimulus to another.

Focus Window Size

The central focus region of the focus window should allow identification of a single element of the stimulus. Thus it should not be so small that identification is difficult when the focus window is centred over an element. It also should not be so large that it allows the simultaneous identification of two or more neighbouring elements, since this will make it difficult for the experimenter to determine which element the user was focusing on. Generally, the focus region should be slightly smaller than the bounding box of a typical stimulus element. Each transition region should be only slightly larger than the region within it. The role of the transition region is to provide a smooth transition from focused to blurred, and to indicate the direction of neighbouring connected elements.

Motion Blur Speed

The motion blur feature allows for a distinction to be made between movements and fixations. The threshold speed should not be so high that users can use a 'brass rubbing' strategy. That is, identifying the stimulus by rapidly moving the focus window over it. However, it should also allow for slow navigation between connected stimulus elements. The mouse speed at which motion blur onset occurs will depend largely on the type of stimulus and the nature of the task. For example, if the stimulus is a circuit diagram, then the motion blur speed should be high to allow the user to trace the path of the wires without losing focus. If instead the stimulus is mathematical equations, then motion blur onset should occur for much slower mouse speeds.

C.3 RFV Input File

The input file is a text file containing the settings for the RFV, in the form of parameter declarations. To declare a parameter, the parameter name needs to be input, followed by an opening bracket. Then the required arguments must be included, followed by any optional arguments that are being used. The declaration is completed by a closing bracket. The arguments of a parameter may be labels, filenames, keywords, integers, or other parameters. Some parameters may require no arguments.

The parameter names are strings that may only contain alphabet characters and the underscore character. When specified in the input file, the alphabet characters may be in either lower or upper case. For example, to specify a line of text, the RFV does not care whether the corresponding parameter is input as `TEXT.LINE` or `text.line`. Keywords are treated in the same way. Keywords however take no arguments, and as such are not followed by a pair of parentheses.

Labels are input as strings that appear within double quotation marks (`"`). A label may not cross multiple lines, and must have both the opening and closing double quotation marks. Any characters may be used within a quotation, except of course the double quotation mark, since the RFV would interpret this as the end of the label. If the double quotation mark is needed as part of a string token, simply use two consecutive single quotation marks. Filenames are specified in the same way as labels, using double quotation marks. If a file is not in the directory from which the RFV will be run, the directory path to that file should also be included with the filename. Integers are simply specified as a string of digits that may or may not be preceded by a minus sign.

As has been mentioned, parentheses are used to enclose the arguments of a parameter. In this respect, the parameter declarations share some resemblance to function declarations in many programming languages. The arguments of a parameter in the input file should be separated by whitespace (that is, a single space, a tab, or a newline). The input file does not require that the arguments of a parameter be separated by a comma or a semicolon, as is used in many programming languages. However, since the input file shares similarities with programming languages, all commas and semicolons that are found in the input file will be ignored, so that the RFV can still process an input file in which these characters may have been inserted out of habit.

The RFV allows for comments to be inserted into the input file. Whenever two forward slashes (`//`) are found on a line of the input file, the remainder of that line is ignored, allowing comments to be included in the file.² Note that two forward slashes will not be interpreted as beginning a comment if they are contained within the double quotation marks of a label. Parameter names, keywords, integers, commas and semicolons, will also be interpreted part of a label string if enclosed within double quotation marks.

The first parameter that must be declared within an input file is the `DEFAULTS` parameter. After this has been fully defined, the input file must contain at least one `COMPONENT` parameter. These parameters of course take other parameters as arguments. For a complete list of the parameter specifications, and how each parameter can be used, see Section C.8. Section C.5 also contains a tutorial in which an example input file is generated that utilizes all of the features of the RFV.

C.4 Running the RFV

Being a Java 2 program, the RFV is run using a launcher for Java 2 applications. The program has been bundled into a Java JAR file called `RFV.jar`. If we want to run the program with the sample input file, `sample.rfv` (which is available from the RFV website), then the program can be run from the command line using the command:

```
java -jar RFV.jar sample.rfv
```

The `java` command at the beginning indicates that we wish to run a Java program (remember, it must be software version 1.2 or higher). The `-jar RFV.jar` means the program is to be found in the JAR file called `RFV.jar`, and `sample.rfv` is the name of the input file we wish to use.

This command assumes that `RFV.jar` is in the current directory, and that `sample.rfv` is also in the current directory. If these files are located elsewhere, the directory path should also be included

²The style of the comments are similar to those used in the C++ programming language.

with the filename. The `sample.rfv` file will also assume that all the related images are in the current directory (these images are also available from the RFV website). If the images to be used are in a different directory from the one which the RFV will be run, the input file should contain the full directory path with each image filename.

When the command line given above is entered, the RFV setup window should appear as is shown in Figure C.5. The setup window contains a region to enter a subject ID label at the top. This label should consist only of alphabetical characters, digits and the underscore character. No spaces are allowed. The subject ID will become the data output filename (with a `.rfvd` extension). It is important to note that if a file already exists in the current directory with the same filename, it will be overwritten. There are also two lists, one of the available components (which are defined with the `COMPONENT` parameter in the input file), and one for the presentation order. By selecting one or more components from the available list, it is possible to add them to the presentation order list. Components will be inserted into the presentation order list immediately above the current component that is highlighted in this presentation order list. If more than one component in the available list has been selected, then it is possible to insert the components in either the order they appear in the available list, the reverse of this order, or in a pseudo-random order. This depends on which of the three insert buttons are used. Note however, that each component can only appear in the presentation order list once.

For example, we can present all of the components in a pseudo-random order by pressing the 'Select All' button under the available components list, and then the 'Insert in Random Order' button. Before we can run the experiment, a valid subject ID must also be entered. If we used a subject ID called 'test', then the setup window should appear similar to Figure C.6 when we are ready to run the experiment. Once the 'Run Experiment' button is pressed, the setup window will disappear, and the RFV window will appear. The first block in the first component in the presentation order list will be loaded and displayed. Note that each block is not loaded until it needs to be displayed.

Beneath the 'Run Experiment' button, there is a status region. In Figure C.5 it contains the text 'RFV Version 2.1'. This region is used to report any messages to the user. For example, if the user tries to insert same component in the presentation order list twice, or run the experiment with an invalid subject ID, this will be reported in the status region.

C.4.1 RFV Errors

The RFV program does error checking when processing the input file, to make sure the correct syntax has been used when specifying the parameters. However, each block is not checked until it is loaded, and therefore the listing of all of the available components in the setup window does not mean that they have been checked. It is important that the input file be checked thoroughly, and that it is tested by the experimenter before letting experimental participants use the RFV.

Other errors may also be reported by the RFV, such as system problems or, for example, if the RFV window is set to be too small to fit in all the visual elements defined in a block. The error messages will provide some information, but if this is insufficient to allow the problem to be resolved, send email to the address listed on the RFV website with details of the error and the system set up. Any feedback or suggested improvements for the RFV are also welcome.

C.4.2 Guidelines for Running the RFV

The RFV accesses the system clock for all functionality that involves timing. However, Java is an interpreted language and as such the workload of the system can affect how accurately the program carries out and records timed events. It is thus important to ensure that the performance of the RFV is not affected by other processes on the system on which it is running.

There are two ways in particular to help achieve this. Firstly, it is recommended that computer does not remain connected to a network while the RFV program is being used. If it is, the system may have to deal with requests that can take up system resources. Secondly, there should not be any other processes running at the same time as the RFV. Again, this is simply to ensure that all system resources can be dedicated to running the RFV program.

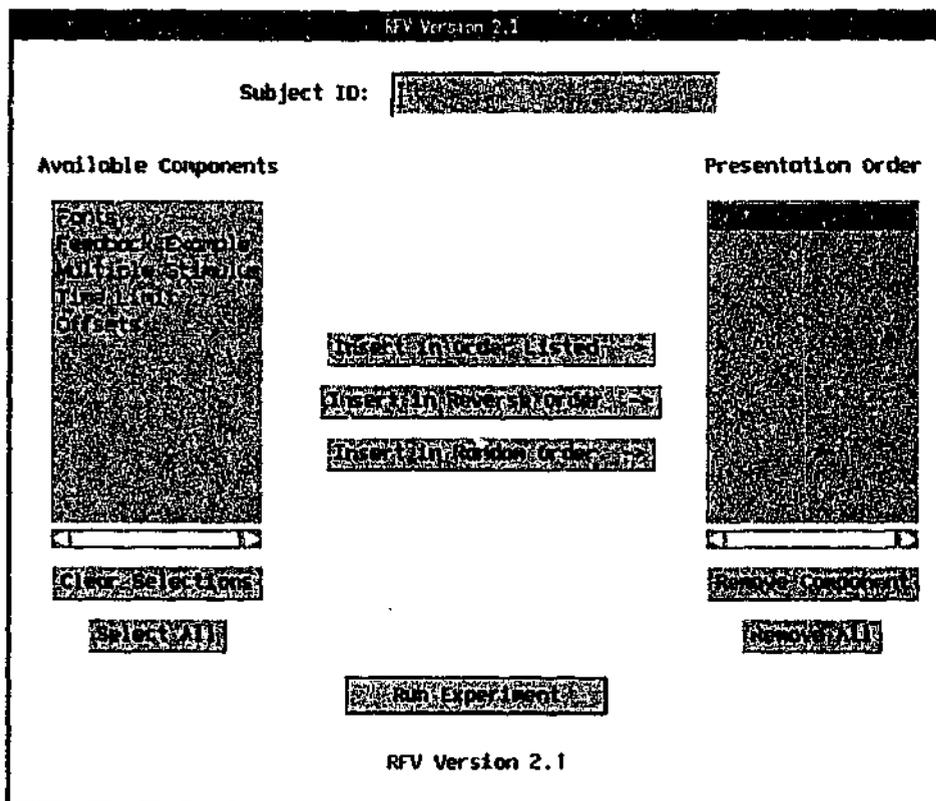


Figure C.5: The RFV setup window.

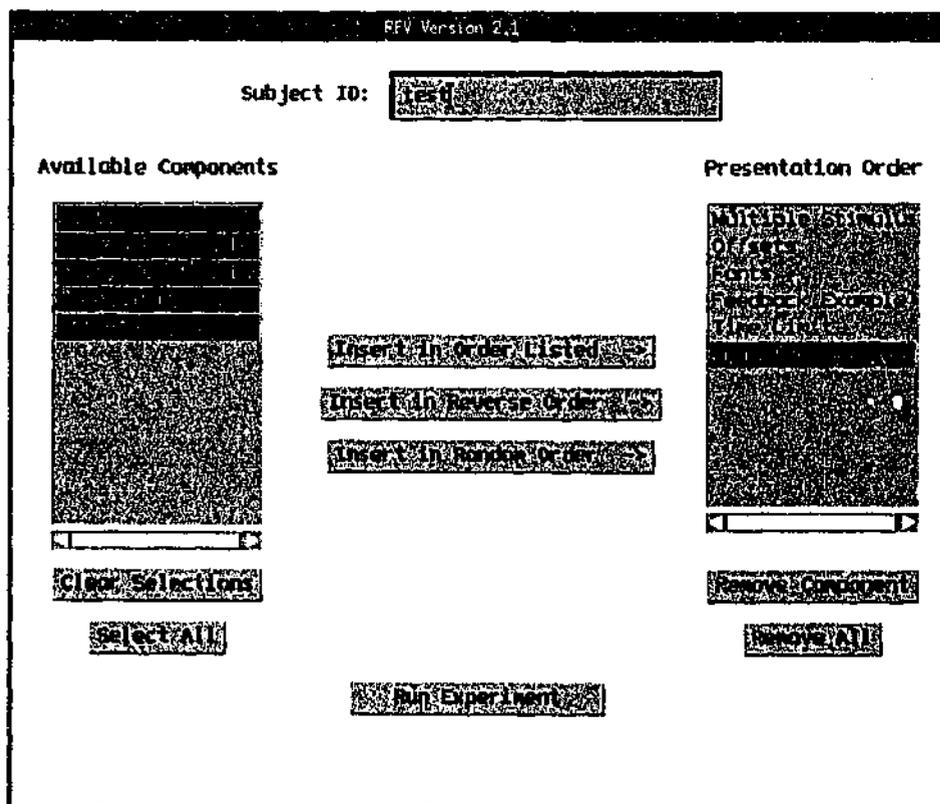


Figure C.6: Setting up an experiment.

C.4.3 Participant Interaction with the RFV

The participant interacts with the RFV using a standard computer mouse. Movements of the mouse over the stimulus correspond to movements of the focus window. Note that when not in a stimulus region, the mouse cursor is simply a default pointer. However, in a stimulus region it is a small black dot. This dot allows the participant can keep track of where they are when they move to an empty part of a stimulus region.

The only other mouse function is to allow the participant to select a button response. This requires the participant to move the mouse cursor into the area of the button they wish to select, and single click with the mouse button. If no buttons are present, a mouse click anywhere in the RFV window will allow the participant to continue on to the next block. (For multiple button mice, click with the main mouse button, which is usually the left mouse button).

The participant therefore does not need access to the keyboard when using the RFV program. There is in fact only a single keyboard command. This is pressing the 'Q' key in order to quit the program. Care must be taken, since this command will work at any time when the RFV is running. It is recommended that the keyboard be kept in a position such that only the experimenter has access to it during an experiment. Even when all blocks have been presented and the end of experiment message is being displayed, the RFV window will remain visible until the 'Q' key is pressed.

C.5 Tutorial

This tutorial gives an example of generating an input file for the RFV. The aim of this example is to demonstrate the features of the RFV. It will begin by showing the simplest possible input file that the RFV can use, which will then be extended to demonstrate the full range of features and options available when using the RFV. With a knowledge of these features, experiment designers can then use as many or as few as is necessary for their purpose.

The tutorial will show the example input files as they are being generated. So that the most recent additions to an input file can be clearly distinguished, they will appear in italics. Also, to prevent too much repetition, three vertical dots will also be used in places to indicate the presence of parts of an input file that have not been repeated. Consider the following example.

```

:
display(
    text_line("Restricted Focus Viewer")
)
```

In this example, the `TEXT_LINE` parameter is italicized, indicating that this is what has been added since the input file was last shown. Also, the three vertical dots indicate that there are parts of the input file before the `DISPLAY` parameter that have been shown in previous examples, but have not been repeated this time to save space. Note that comments will not appear in these examples (since the tutorial itself is supposed to provide a commentary), although they may be inserted anywhere in an input file with the appropriate prefix (use `//` to insert a comment). The example input file will use lowercase letters for the parameter names. This tends to make the file more readable, however, uppercase letters could also be used.

C.5.1 The Visual Elements

The initial aim of this example is to generate a simple and valid RFV input file. The input file must begin by specifying certain default values. These must be declared as arguments of the `DEFAULTS` parameter; that is, they must be specified within the brackets of `DEFAULTS`. There are three sets of values that must be specified, and six that are optional. We begin our input file then by declaring the `DEFAULTS` parameter.

```

defaults(
)
```

Next, we consider the three compulsory parameters that must be specified with `DEFAULTS`. The first of these is `FRAME_SIZE`. This parameter defines the size of the RFV window. It takes two integers as arguments, specifying the width and height of the window in pixels. For this example, we will generate a window that is 800 × 600 pixels.

```
defaults(
    frame_size(800 600)
)
```

The second compulsory parameter is `FOCUS_WINDOW`. This parameter specifies the default dimensions of the four regions used to specify the focus window. Each stimulus that is defined later in the input file will use the default focus window, unless a different sized focus window is explicitly set for it. The parameter accepts as arguments four pairs of integers, with each pair specifying the width and height (in pixels) of a region. The first pair specify the dimensions for the outermost transition region. The second and third pair specify the dimensions of the middle and innermost transition regions respectively, and the final pair specify the dimensions of the focus region (see regions shown in Figure C.1). The regions, therefore, are specified in descending order relative to their size. An appropriate size for the focus window regions will depend on the size of the elements in the stimulus images. We now add a default focus window to our input file. Each of the regions we define for this example are square, meaning the width and height value for each pair of integers is the same.³

```
defaults(
    frame_size(800 600)
    focus_window( 50 50
                 42 42
                 36 36
                 32 32)
)
```

The final compulsory default parameter is `MOTION_BLUR_SPEED`. This parameter takes a single integer as its argument, which is the speed at which motion blur onset occurs (in pixels per second). When the mouse moves over a stimulus at less than this speed (or the mouse is stationary), all regions of the focus window are present (see Figure C.1). However, when the mouse moves at greater than this speed over a stimulus, the focus window is motion blurred; that is, only the outermost transition region is added to the blurred region. As with the `FOCUS_WINDOW`, each stimulus will implicitly use the default motion blur speed unless a different speed is explicitly set for it. For our input file, we will set a motion blur speed of 120 pixels per second.

```
defaults(
    frame_size(800 600)
    focus_window( 50 50
                 42 42
                 36 36
                 32 32)
    motion_blur_speed(120)
)
```

Although there are also six optional parameters that can be included in `DEFAULTS`, we will not consider them just yet. Note that the order in which the three compulsory parameters were specified is not important, so long as they were specified exactly once within the brackets of `DEFAULTS`.

With the defaults defined, it is time to start adding `COMPONENTS` to the input file. Each component is composed of one or more `BLOCKS`, and before we continue it is important to understand exactly what a component and a block is, and how they are different.

Firstly, what appears in the RFV window at any time is defined by a `BLOCK`. Each block determines the layout of the elements of a single screen presentation. These elements can include lines

³The `FOCUS_WINDOW` parameter can also have `FOCUS_WINDOW_OFFSETS` defined, but this is rarely used.

of text, buttons, static images, and of course, blurred RFV stimuli. The duration of time that a block will appear for can be specified, and it is possible to provide feedback that is dependent on a participants response.

A COMPONENT is a convenient way for binding together a group of blocks in a set order. These components can then be arranged in any order using the setup window of the RFV. Consider a typical experimental design. It is likely that there will be instructions, and that they will require more than one block. All the blocks that are necessary for the instructions can be grouped within a single component. There may also be practice items, which are the same for each participant. These can also be grouped together in a single component. For the experimental items themselves, if the experiment is self-paced, each item block may be preceded by a block that prompts the participant to continue when ready. These pairs of blocks (one pair for each experimental item) can also form components. Using the setup window, it is then easy to specify that the instructions should appear first, followed by the practice items, which are then followed by the experimental items in a different pseudo-random order for each participant.

For our input file, we will begin with just a single COMPONENT. Each component has an associated label to identify it, which makes things easier when determining the ordering of the components to be used in an experiment. Each component label must be unique, with no two components sharing the same label. The label must be the first argument of the component, appearing after the opening bracket. We will give our component the very unimaginative name, "Component 1". Obviously, when creating an input file for an actual experiment, more meaningful labels such as "Instructions" or "Item 3" would be more useful.

```
defaults(
  :
)
component( "Component 1"
)
```

Within our COMPONENT, we will define a single BLOCK. Blocks do not have an associated label, since the blocks within a component will always appear in the order in which they are specified.

```
defaults(
  :
)
component( "Component 1"
  block(
  )
)
```

The BLOCK parameter has several optional arguments. One of those is the DISPLAY parameter, which is used for specifying the elements that will appear in the RFV window. We add this to our block now.

```
defaults(
  :
)
component( "Component 1"
  block(
    display(
  )
  )
)
```

There are several possible parameters that can be specified within DISPLAY. However, to begin with a simple example, we shall just insert a line of text using TEXT_LINE.

```

defaults(
  :
)
component( "Component 1"
  block(
    display(
      text_line("A simple line of text")
    )
  )
)

```

This is now a valid input file, and it is also one of the simplest input files that the RFV can be run with. In the DEFAULTS parameter we specified the size of the RFV window, the default focus window size and motion blur speed. Note that these last two parameters will not be used in the example as it stands, since there are no blurred RFV stimuli specified in the input file. Nonetheless, they must be specified since the RFV is expecting to have to manipulate stimuli at some point (after all, that is the main purpose of the program). After the DEFAULTS had been specified, we defined a single COMPONENT. This component will display one BLOCK that contains only a single line of text.

When we run the RFV with the example input file, the setup window should appear looking similar to the one in Figure C.7.

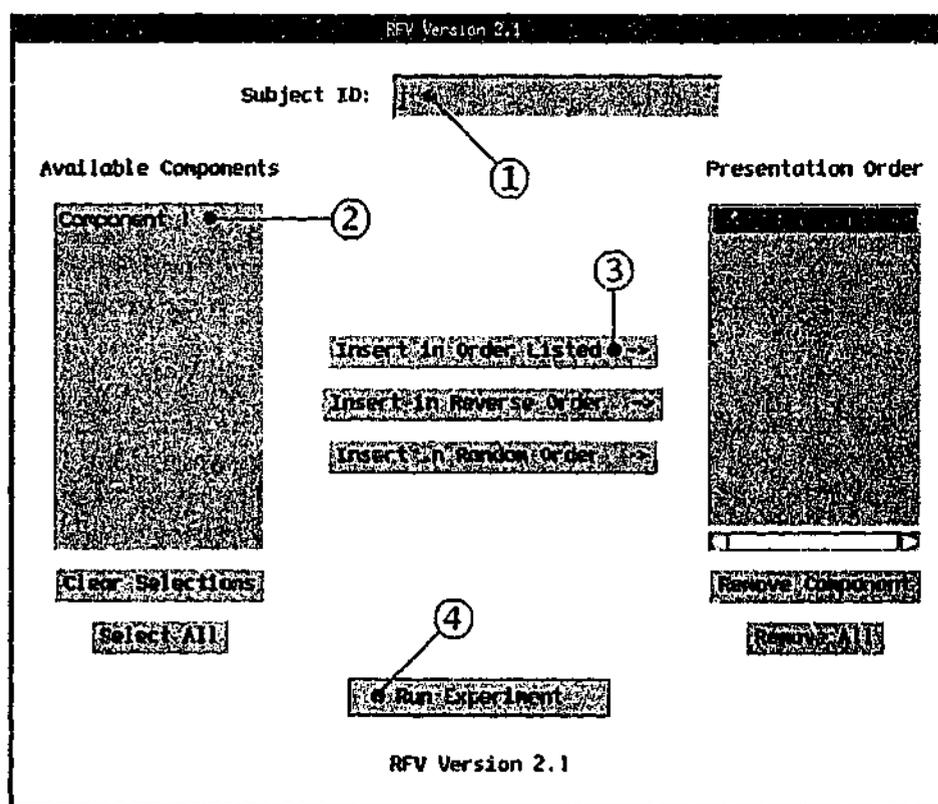


Figure C.7: The steps to take in the RFV setup window.

To run an experiment using this input file, the following steps must be taken.

1. Enter a valid subject ID. It may consist only of alphabetical characters, digits and the underscore character. No spaces are allowed. The subject ID will become the data output filename (with a .rfvd extension).
2. Select "Component 1" from the list of available components. This will be the only component available in this example.

3. Insert the component into the presentation order list. Since only one component is selected, the insert button that is used will not matter. The component will appear in the presentation order list immediately above the currently highlighted element (which is 'End of Experiment').
4. Select the 'Run Experiment' button.

Where in the setup window each of these steps should be taken is shown in Figure C.7.

When we run the experiment, the setup window will disappear, and the RFV window will appear with the dimensions given in the input file. The program will immediately load the block that was specified within the component, and display the content of the `TEXT_LINE` parameter in the middle of the screen, as shown in Figure C.8. Nothing else appears in the window.

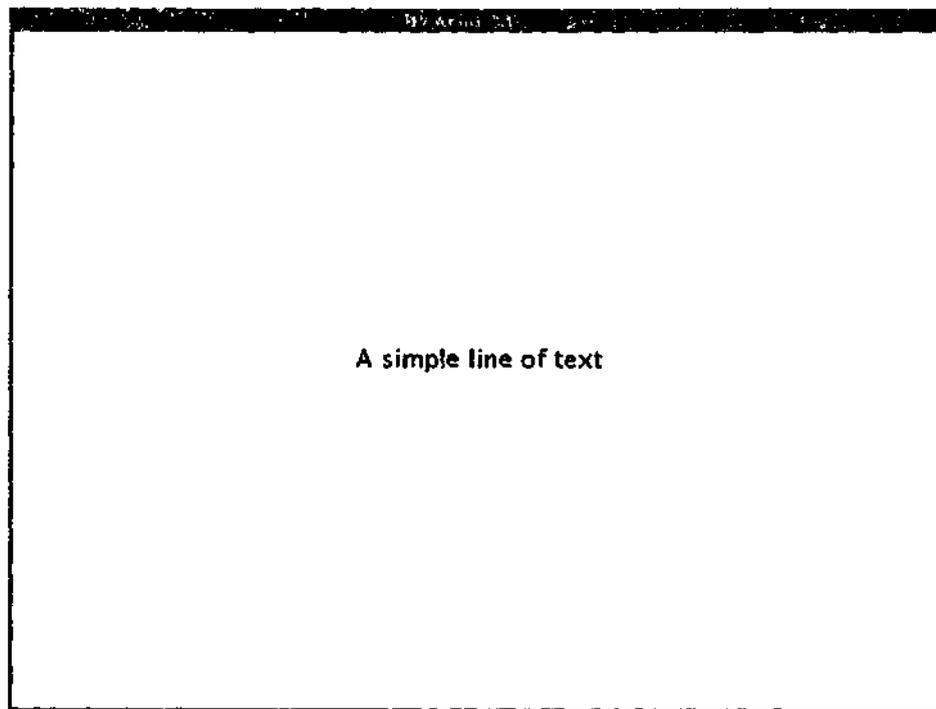


Figure C.8: The RFV window after loading the input file block.

Since no buttons or time limit was specified in the block, the contents of the RFV window will remain unchanged until the mouse button is clicked with the cursor inside the RFV window. Then, the end of experiment block will appear, as there are no other blocks in "Component 1", and no other components to be presented. The end of experiment block appears as in Figure C.9.

Once this block appears, no mouse actions will affect the RFV window, and it will remain idle on the screen until the program is quit by pressing the 'Q' key. Note that the 'Q' key will quit the RFV program at any time, from either the setup window or the RFV window. This ends our first run of the RFV.

The `TEXT_LINE` that we have specified in the input file came up as black text in a default font. However, we can specify the font and/or colour we want if we wish. The font type can be specified using the `FONT` parameter, which must appear as an argument of `TEXT_LINE` after the label has been specified. `FONT` takes three arguments. The first is the name of the font, which can be one of *Serif*, *SansSerif*, *Dialog*, *DialogInput* or *Monospaced*. Some of these fonts might not be available on certain systems, and some systems might also have other valid fonts. The font name is given as a label. The second argument is a keyword indicating the style of the font. This can be either `PLAIN`, `BOLD`, `ITALIC` or `BOLD.ITALIC`. Finally, the point size of the font must be specified. Just as some fonts are not available on all systems, not all fonts are always available in every size or style. If the RFV cannot load the font specified, it will revert to a default font.

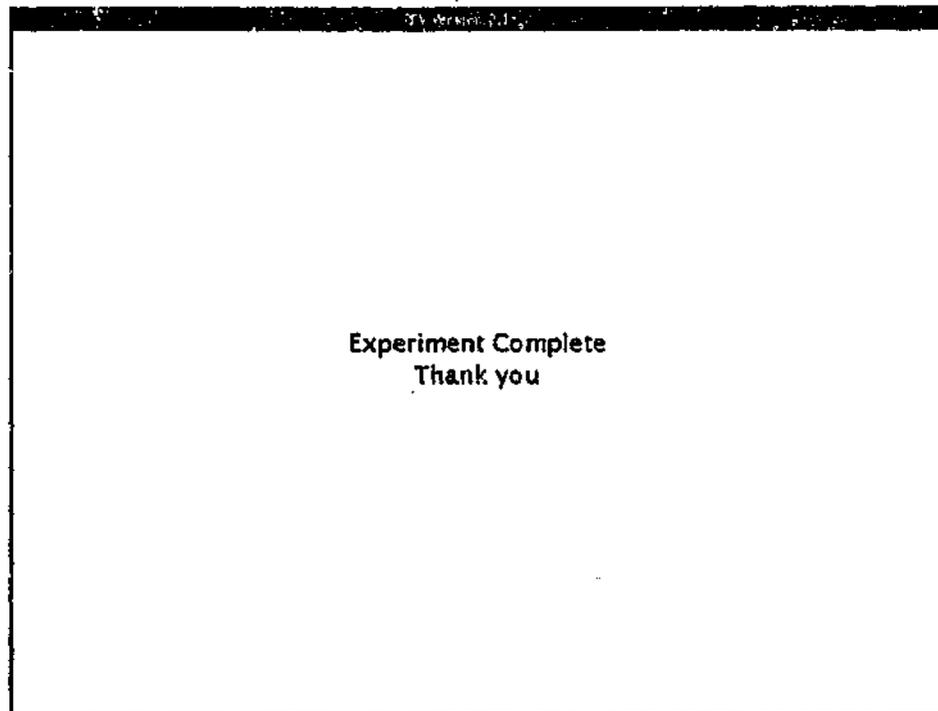


Figure C.9: The end of experiment block.

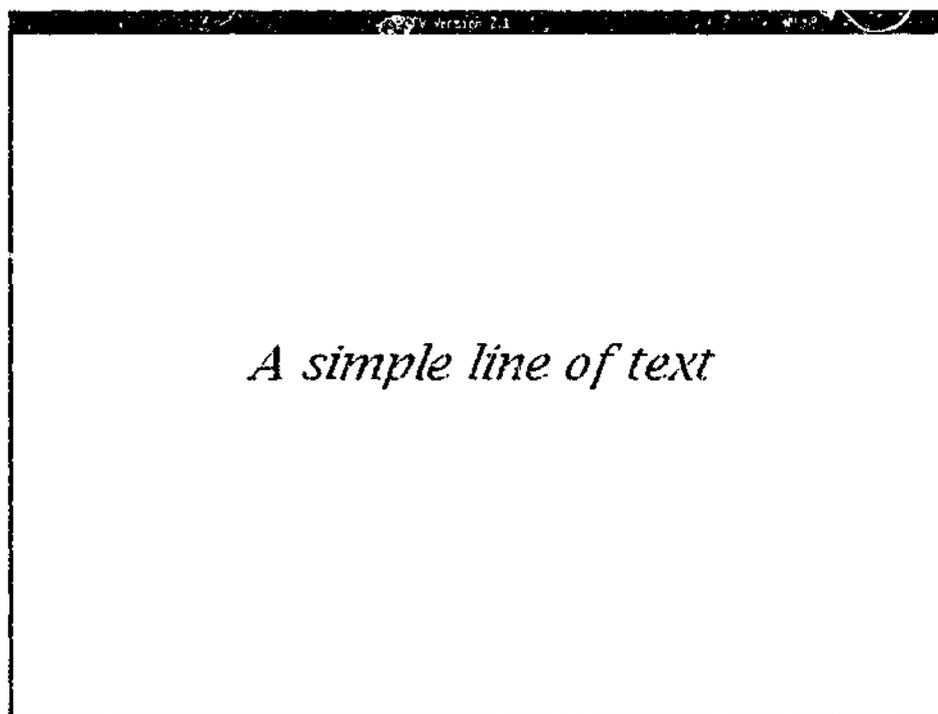


Figure C.10: A different font and colour for the line of text.

```

:
component( "Component 1"
  block(
    display(
      text_line("A simple line of text"
        font("SansSerif", ITALIC, 36) )
    )
  )
)

```

The colour of the text can be changed by including the FOREGROUND parameter. This takes three integers as its arguments, each with a value between 0 and 255 inclusive. The three values are the red, green and blue (RGB) components of the colour being specified (in that order). For example, the colour red is specified as (255 0 0) with a maximum contribution from the red component, and no contribution the green or blue components. Some common colours and their corresponding RGB values are listed here.

Red	(255 0 0)	Green	(0 255 0)
Blue	(0 0 255)	White	(255 255 255)
Black	(0 0 0)	Yellow	(255 255 0)
Cyan	(0 255 255)	Magenta	(255 0 255)
Orange	(255 128 0)	Purple	(128 0 255)

In our input file, we will make our line of text red.

```

:
component( "Component 1"
  block(
    display(
      text_line("A simple line of text"
        font("SansSerif", ITALIC, 36)
        foreground(255 0 0) )
    )
  )
)

```

If we run the RFV with the input file now, the block in "Component 1" should look similar to Figure C.10. Note however, that after clicking the mouse button, the end of experiment block looks the same as it did before. This is because the FONT and FOREGROUND parameters we specified apply only to the one line of text. Other lines of text are not affected, even if they were specified within the same block. Thus, if we wanted to change the font and colour of more than one line of text, we would have to use the FONT and FOREGROUND parameters on each TEXT_LINE that we wanted to change.

But what if we wanted most of our text to have a specific font and colour that was different from the default one? It would certainly be a hassle to have to specify the FONT and FOREGROUND parameters for each line of text in our input file. Fortunately, this is not required. Instead, it is possible to specify the TEXT_FONT and TEXT_FOREGROUND parameters that you wish to be the new defaults. Naturally, these are specified within DEFAULTS, and are two of the six optional default parameters that can be specified. We now add a different default text font and a default text colour of green to our input file.

```

defaults(
  frame_size(800 600)
  focus_window( 50 50
                42 42
                36 36
                32 32)
  motion_blur_speed(120)
  text_font("Monospaced", BOLD, 30)
  text_foreground(0 160 0)
)
component( "Component 1"
  :
)

```

If we again run the RFV with the latest additions to the input file, the first thing that you should notice is that the block defined in "Component 1" looks the same as it did last time. This is because the line of text in this block has a specific FONT and FOREGROUND specified for it, and this always takes precedence over the default settings. However, if no explicit font or colour is specified in the text line, then the default values are used. This can be seen by clicking the mouse button in the "Component 1" block to reveal the end of experiment block. Here, the two lines of text are generated using the new default font and colour we have specified, as is shown in Figure C.11.

This completes our examination of TEXT_LINES. Although all the options available to manipulate the font and colour have been looked at, in many cases the default values provided by the RFV will be sufficient, and the extra parameters will not need to be used. The next visual element that we will examine, is the BUTTON.

A BUTTON can be specified in a similar manner to a line of text. In its most basic form, it takes a single label as its argument. In the input file, we now defined a new block in "Component 1" after the block containing the TEXT_LINE. In this new block, we will insert a single button.

```

:
component( "Component 1"
  block(
    :
  )
  block(
    display(
      button("Button Label")
    )
  )
)

```

Now when we run the RFV with the input file, the block that appears after the first block in "Component 1" with the TEXT_LINE example, should appear similar to Figure C.12, containing a single gray button with a black label. The button will be large enough to accommodate the specified label.

With the first block, a mouse click anywhere in the RFV window would result in the next block being immediately loaded. However, this is not the case with our new block. In any block that contains one or more buttons, only a mouse click inside a button will be accepted.

As with TEXT_LINES, attributes of a BUTTON can also be specified. There are three attributes that can be set. Again, FONT and FOREGROUND can be defined to change the label, and BACKGROUND can be used to change the background colour of the button. Thus, if we want to make our button have blue text on a yellow background, we can add the following to the input file.

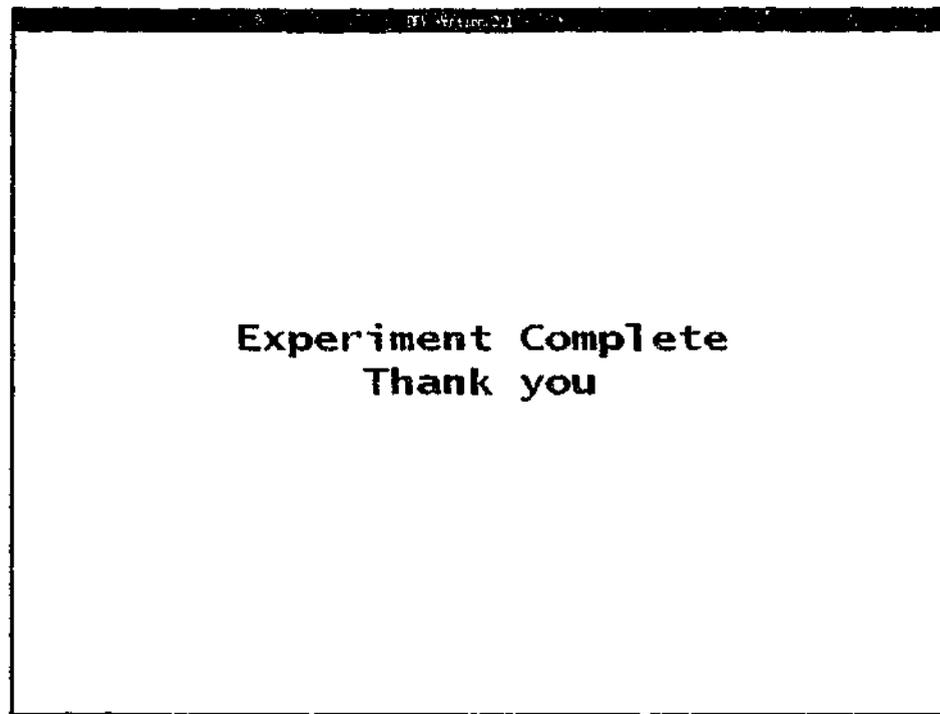


Figure C.11: The result of changing the default text font and colour.

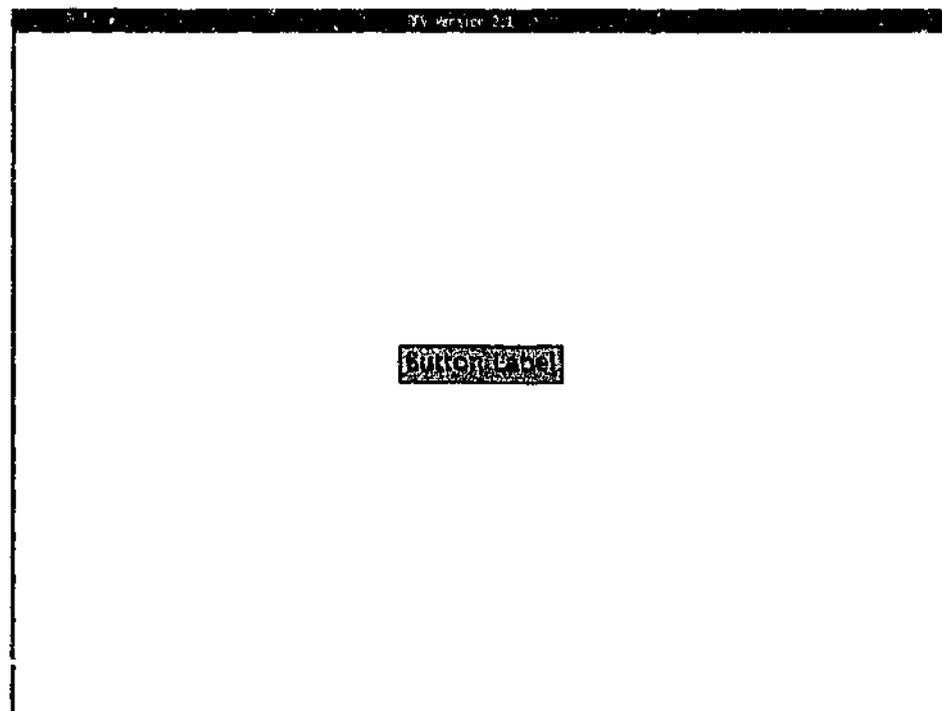


Figure C.12: An example of a button.

```

:
component( "Component 1"
  block(
    :
  )
  block(
    display(
      button("Button Label"
        foreground(0 0 255)
        background(255 255 0) )
    )
  )
)

```

The effects of this change are shown in Figure C.13. Note that the button border is the same colour as the button label. You may also have noticed that the default font of this button is different to the default `TEXT_FONT` that was specified earlier. This is because default text parameters are specified separately to default button parameters. Therefore, just as there were two default text attributes that could be specified within `DEFAULTS`, there are also three button attributes. These parameters are `BUTTON_FONT`, `BUTTON_FOREGROUND` and `BUTTON_BACKGROUND`, and they work as would be expected from the previous example with the text attributes. As with the default text parameters, the three button parameters are optional within the `DEFAULTS` parameter.

Since we have been discussing the colour attributes of visual elements, it is probably a good point to mention one more. The background colour of the RFV window for a given block can be specified using the `BACKGROUND` parameter inside a block. For example, to make the block containing the button have an orange background, we can do the following.

```

:
component( "Component 1"
  block(
    :
  )
  block(
    background(255 128 0)
    display(
      button("Button Label"
        foreground(0 0 255)
        background(255 255 0) )
    )
  )
)

```

This block will then appear as in Figure C.14 the next time the RFV is run with this input file. Note that `BACKGROUND` could have been placed either before or after the `DISPLAY` parameter. As with previous colour changing parameters, it is possible to change the background of all blocks in the input file by using the `DISPLAY_BACKGROUND` parameter as an argument of the `DEFAULTS` parameter. This parameter is optional, and will be overridden within a block if a `BACKGROUND` is explicitly defined. The default background colour is white.

The third visual element that the RFV is capable of displaying is a static image. The images may be in either `GIF` or `JPEG` format, and the image filename is the only argument for this parameter. There are no optional arguments. If the image specified is not in the directory that the RFV will be run from, the path to the image file must also be included with the filename. We now add a new block with a static image to the input file.

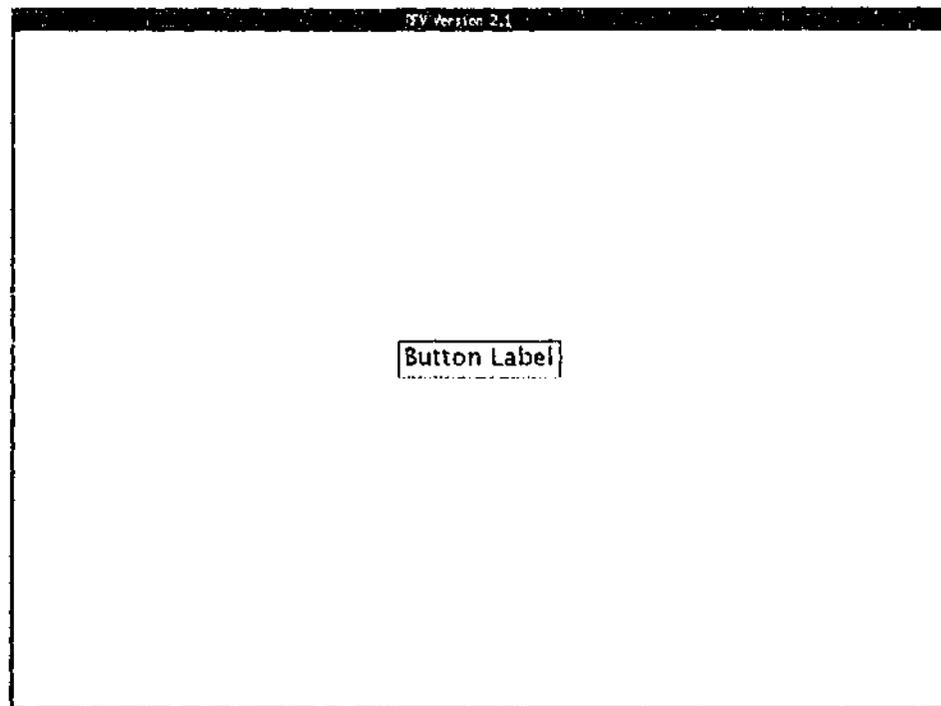


Figure C.13: Changing the button colours.

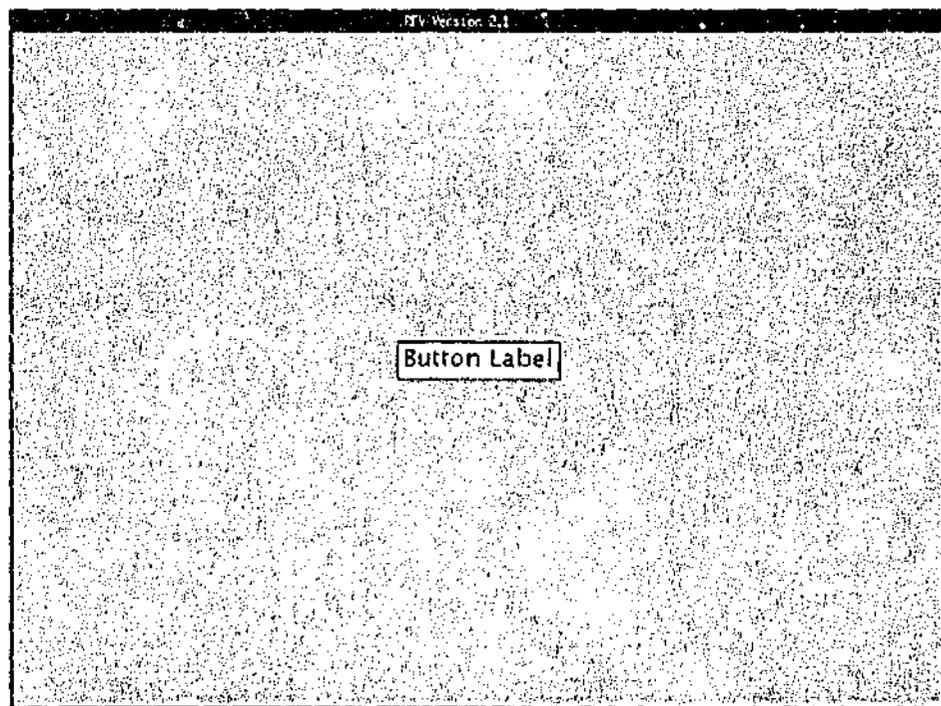


Figure C.14: Changing the RFV window background colour.

```

:
component( "Component 1"
:
  block(
    display(
      image("equation.gif")
    )
  )
)

```

When we run the RFV with the new block containing the image added, the block appears as shown in Figure C.15.

The final visual element that the RFV can display is a **STIMULUS**. The primary reason for designing the RFV was to be able to display this type of visual element. A stimulus consists of five images, one of these in full focus, and the other four with varying levels of blurring. The **STIMULUS** parameter takes the filenames of the five images as its arguments. As with the **IMAGE** parameter, all files must be in the GIF or JPEG format, and if they are not in the directory that the RFV will be run from, the path to the image files must also be included. All five images must of course be of the same dimensions, and the image files are listed in order from the fully blurred image through to the fully focused image.

```

:
component( "Component 1"
:
  block(
    display(
      stimulus("pulley_d.gif"
              "pulley_c.gif"
              "pulley_b.gif"
              "pulley_a.gif"
              "pulley_focus.gif")
    )
  )
)

```

Now if we run the RFV again with this addition to the input file, the last block in "Component 1" will appear as in Figure C.16. Note that in this example, the focus window is positioned near the middle of the stimulus area, revealing part of stimulus in full focus.

The **STIMULUS** uses the **FOCUS_WINDOW** and **MOTION_BLUR_SPEED** settings that were defined in the **DEFAULTS** parameter. However, as with the fonts and colours for **TEXT_LINES** and **BUTTONS**, it is possible to override the default settings for a given stimulus by specifying a **FOCUS_WINDOW** and/or **MOTION_BLUR_SPEED** parameter as an optional argument of the **STIMULUS** parameter (following the five filenames).

C.5.2 Layout of Visual Elements

We have now shown the four types of visual element that the RFV can display. However, so far we have seen only one element displayed per block, and in each case they have mysteriously appeared in the centre of the RFV window. We now turn our attention to composing blocks consisting of multiple elements, and specifying the layout of these elements. First however, we will specify a new **COMPONENT** called "Component 2", and specify a block within it.

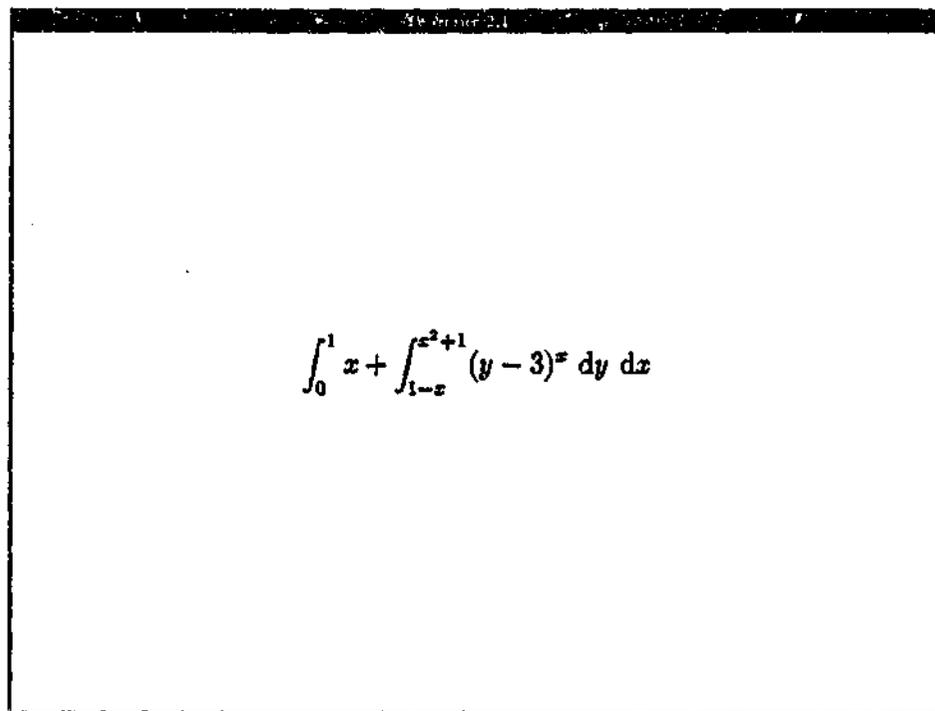

$$\int_0^1 x + \int_{1-x}^{x^2+1} (y-3)^x dy dx$$

Figure C.15: An example of a static image.

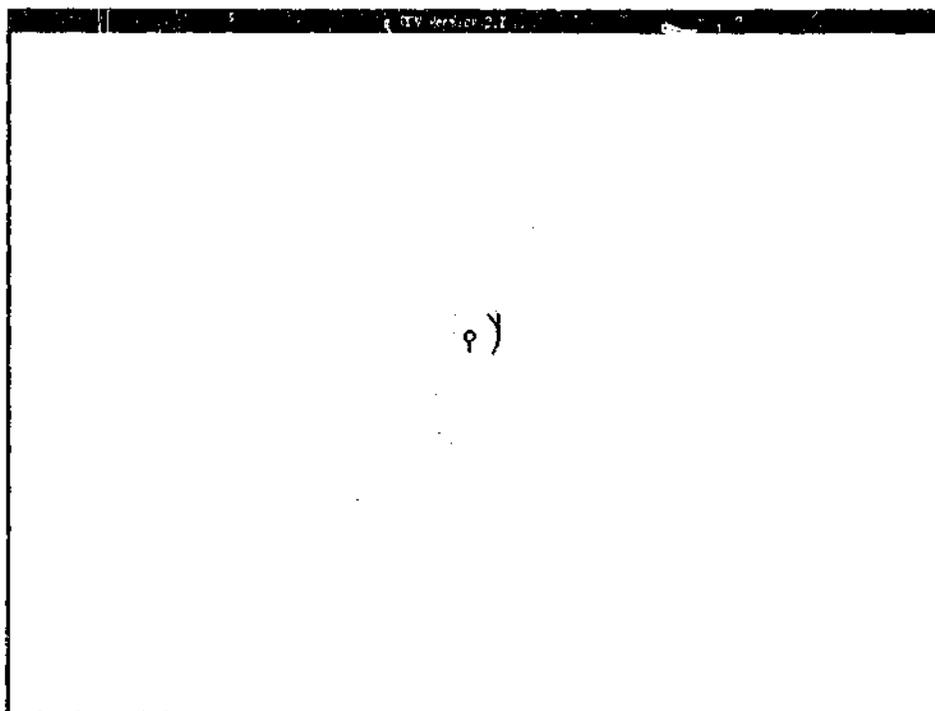


Figure C.16: An example of a stimulus with the focus window near the centre.

```

      :
      component( "Component 1"
                :
      )
      component( "Component 2"
                block(
                )
      )

```

We now add a DISPLAY parameter to the new block, and insert a TEXT_LINE and IMAGE within it.

```

      :
      component( "Component 2"
                block(
                  display(
                    text_line("Another line of text")
                    image("equation.gif")
                  )
                )
      )

```

When we run the RFV with the input file now (being sure to add "Component 2" to the presentation order list in the setup window), the RFV window will appear as is shown in Figure C.17 when we get to the newly specified block.

The RFV program treats the DISPLAY parameter as defining a column of visual elements. The order in which the elements are displayed corresponds to the order in which they were specified in the input file. In the new block we defined in "Component 2", the line of text was specified before the static image, and thus the text line appears above the image in Figure C.17. Each element is centred with respect to the width of the RFV window. However, although the pair of elements is centred vertically also, there is no space between the line of text and the image.

The height of the text plus the height of the image is less than the height of the RFV window. (If otherwise, the RFV program would stop with an appropriate error message.) This means that there is extra vertical space in the RFV window beyond what is needed to display the visual elements specified. By default, the program will evenly divide this extra vertical space to appear above and below the elements that are defined. However, it is possible to take control of the distribution of this vertical space by using the V_SPACE parameter.

Using the V_SPACE parameter inside the column defined by the DISPLAY parameter, allows the allocation of vertical space to be specified in the input file. For example, we could put all of the extra vertical space above the two visual elements.

```

      :
      component( "Component 2"
                block(
                  display(
                    v_space(
                    text_line("Another line of text")
                    image("equation.gif")
                    )
                  )
                )
      )

```

Now when we run the RFV on "Component 2", the block will appear as in Figure C.18. Here, there is no extra vertical space between or below the elements. All the extra vertical space is above them.

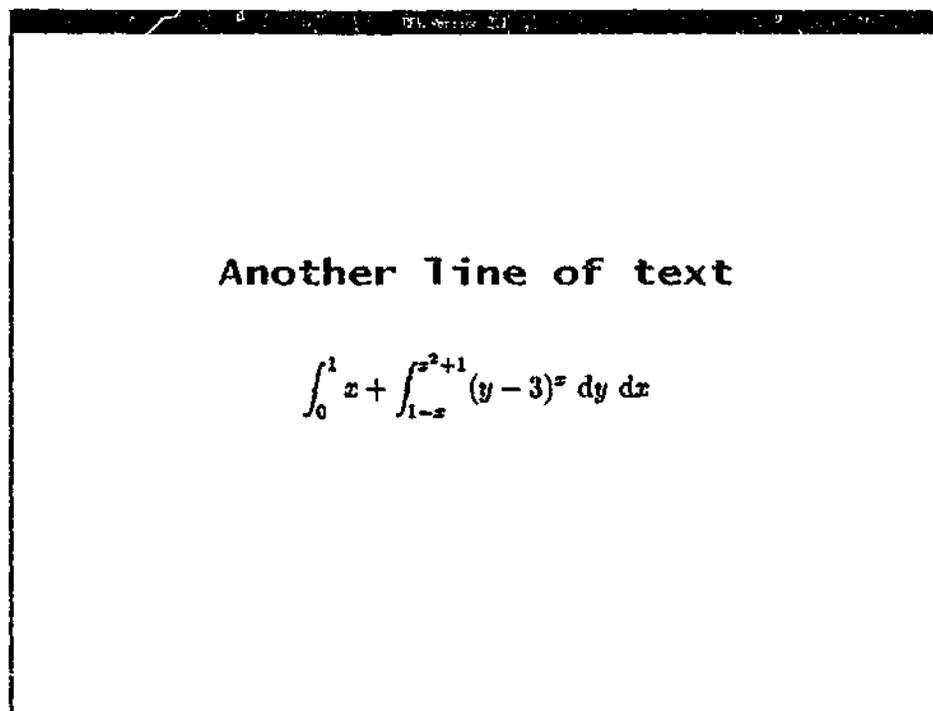


Figure C.17: A block containing a line of text and a static image.

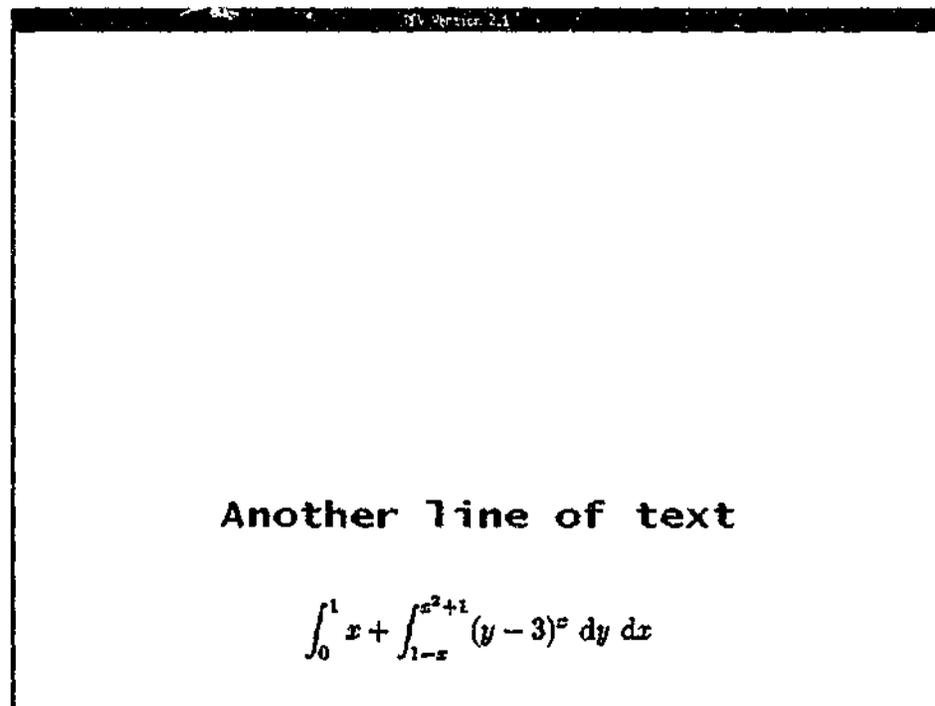


Figure C.18: Modifying the distribution of vertical space.

The `V_SPACE` parameter can be used any number of times inside the `DISPLAY` parameter. If it is used four times for example, then each instance of `V_SPACE` will represent one quarter of the extra vertical space available. It is also possible to specify multiple `V_SPACE` parameters in succession. Thus, we can specify that one quarter of the extra vertical space should appear above the text line, another quarter should be between the text line and the static image, and the other half should be below the the static image. This is done by adding another three instances of the `V_SPACE` parameter to our input file.

```

:
component( "Component 2"
  block(
    display(
      v_space()
      text_line("Another line of text")
      v_space()
      image("equation.gif")
      v_space()
      v_space()
    )
  )
)

```

The resulting layout of the visual elements is shown in Figure C.19. You will have noted that the `V_SPACE` parameter is followed by a pair of brackets. This is because it can take an optional argument. This argument is an integer to specify a specific height (in pixels). For example, we may not have wanted our text line and image pressed together as was the case when we first created them. On the other hand, the `V_SPACE` that has been added between them may also be too large. Sometimes (as we will do now), it is convenient to say that we want a specific vertical gap (for example, of 20 pixels) between elements.

```

:
component( "Component 2"
  block(
    display(
      v_space()
      text_line("Another line of text")
      v_space(20)
      image("equation.gif")
      v_space()
      v_space()
    )
  )
)

```

The block in "Component 2" will now appear as in Figure C.20. Note that the remaining three `V_SPACES` that do not have an argument now each represent one third of the extra vertical space, since the `V_SPACE` with the argument is now treated as a visual element. This means that the extra vertical space in the block is now calculated as the difference between the RFV window height and the heights of the text line, static image and the 20 pixel high `V_SPACE`. One third of the extra vertical space is above the line of text, with the remaining two thirds below the static image.

So far, all of the layout has been in a single column. No doubt however, it will be desirable to layout elements horizontally also. This can be done by using the `ROW` parameter. We now add to our input file a row containing two buttons. We will place this row between the last two `V_SPACE` parameters.

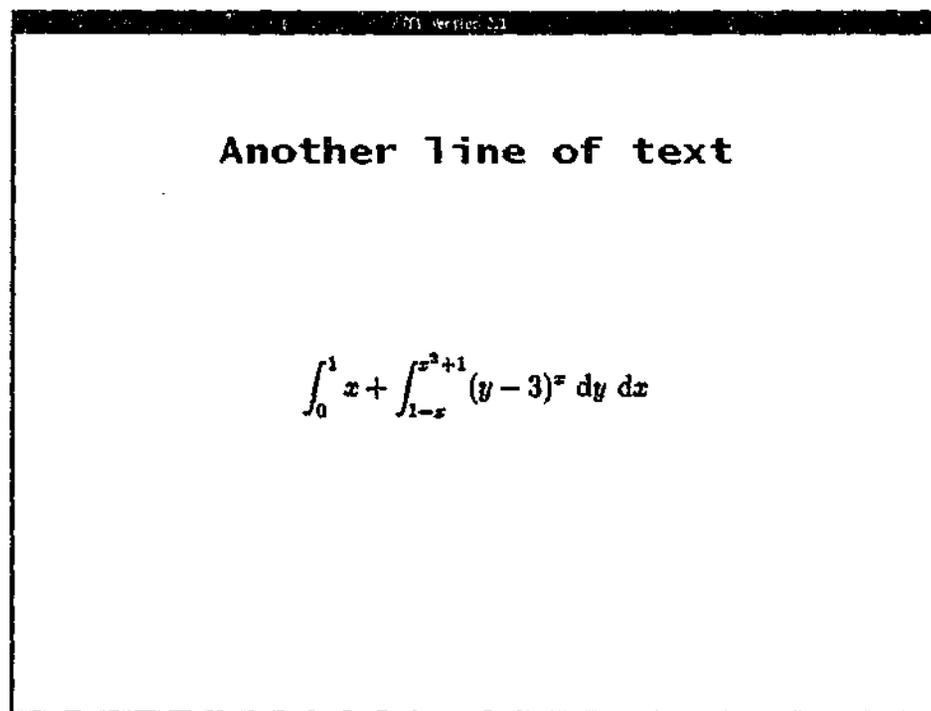


Figure C.19: Further modifying the distribution of vertical space.

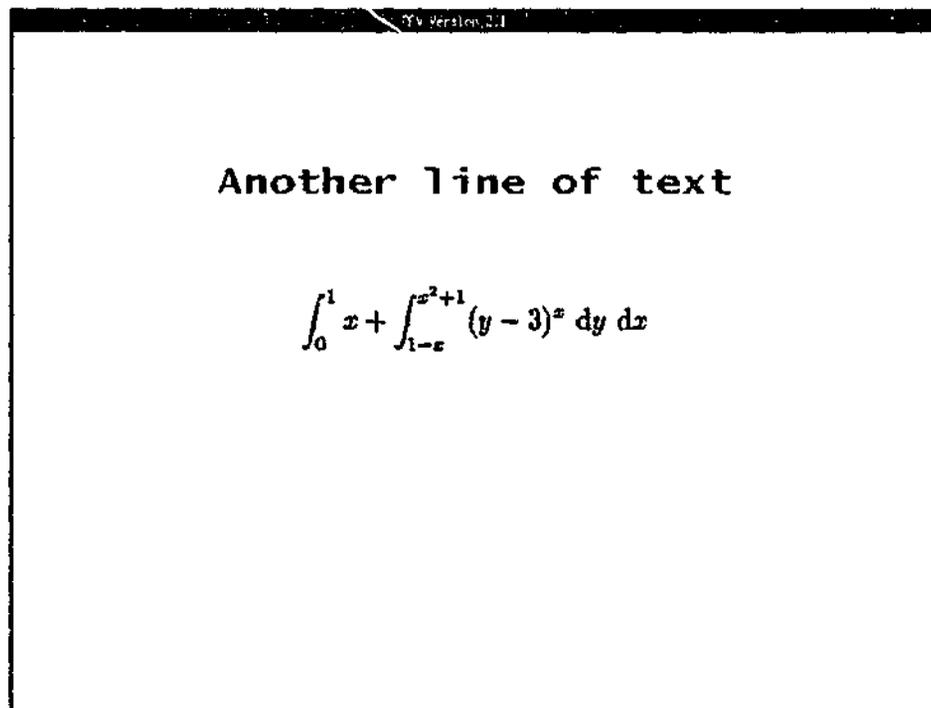


Figure C.20: Using a fixed vertical space in the layout of the elements.

```

      :
      component( "Component 2"
        block(
          display(
            v_space()
            text_line("Another line of text")
            v_space(20)
            image("equation.gif")
            v_space()
            row(
              button("First Button")
              button("Second Button")
            )
            v_space()
          )
        )
      )

```

This row is now treated the same as a visual element within the column defined by the DISPLAY parameter. The height of the row is the height of the tallest element in the row. When we now run the RFV, the block where we have just specified the ROW appears as in Figure C.21.

The buttons appear right next to each other, with the extra horizontal space appearing evenly distributed on either side of them. Just as we could control vertical space with the V_SPACE parameter in the column defined by DISPLAY, it is also possible to control the extra horizontal space in a row by using the H_SPACE parameter. For example we add a H_SPACE between the two buttons.

```

      :
      component( "Component 2"
        block(
          display(
            :
            row(
              button("First Button")
              h_space()
              button("Second Button")
            )
            v_space()
          )
        )
      )

```

This forces the two buttons to appear on either side of the RFV window, as in Figure C.22, with all of the extra horizontal space between them.

The H_SPACE parameter works similarly to the V_SPACE parameter, with extra horizontal space being divided equally among all of the H_SPACES specified within a row. As with V_SPACE, it is also possible to specify a fixed horizontal width (in pixels) by providing an integer as an argument to the H_SPACE parameter. Thus we can separate the buttons by 100 pixels. Note that when we do this, the row no longer contains any H_SPACES that operate on the extra horizontal space, since the H_SPACE parameter with an argument is treated like a visual element. Thus, the RFV again will automatically divide the extra horizontal space on either side of the buttons, with a fixed horizontal space of 100 pixels also appearing between them.

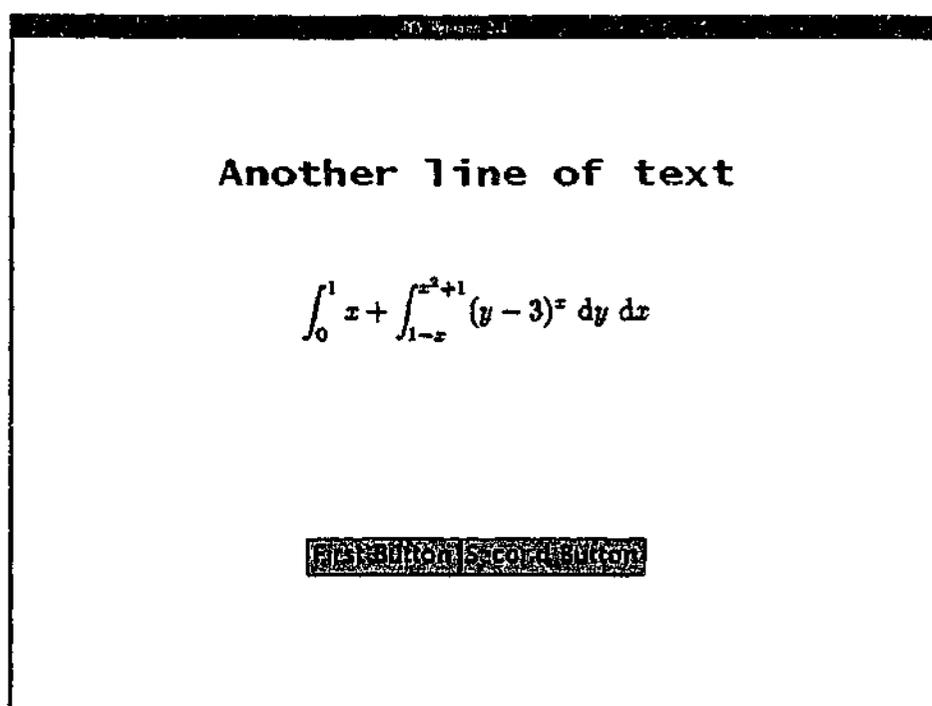


Figure C.21: Adding a row to a block.

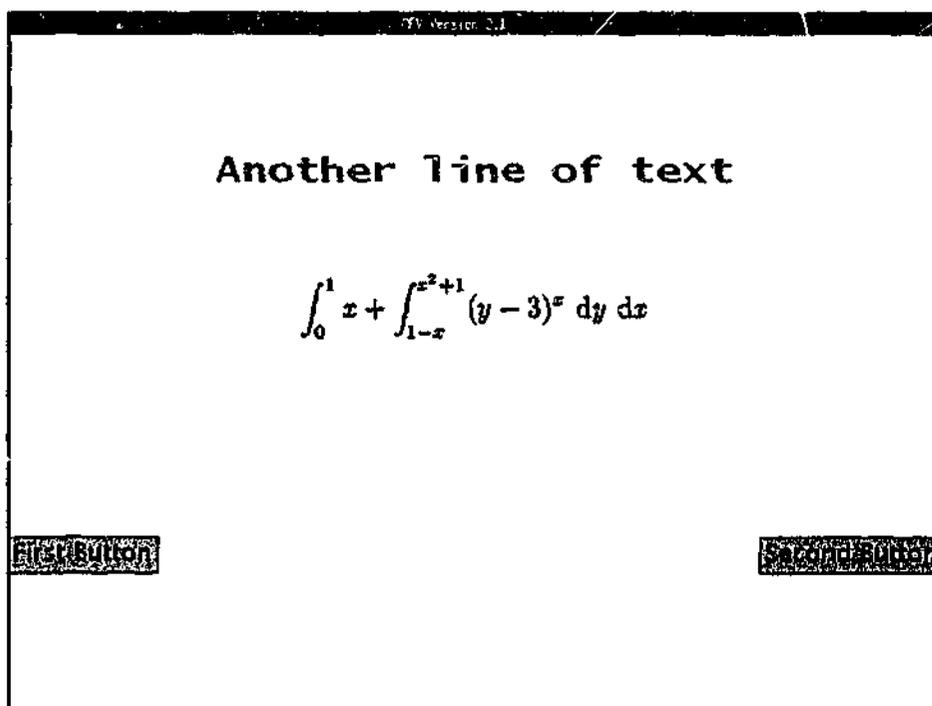


Figure C.22: Manipulating the extra horizontal space in a row.

```

:
component( "Component 2"
  block(
    display(
      :
      row(
        button("First Button")
        h_space(100)
        button("Second Button")
      )
      v_space()
    )
  )
)

```

The effect of this change can be shown in Figure C.23.

Of course it may sometimes be desirable to lay things out vertically within a row. This can be done with the COLUMN parameter. We will now define a new block that makes use of COLUMNS within ROWS.

```

:
component( "Component 2"
  block(
    :
  )
  block(
    display(
      v_space()
      row(
        text_line("Some text")
        h_space(50)
        column(
          button("Button 1")
          v_space()
          button("The Second Button")
        )
      )
      v_space()
      row(
        image("pulley_focus.gif")
        h_space(50)
        column(
          button("Button 3")
          v_space()
          button("Button Number 4")
          v_space()
        )
      )
      v_space()
    )
  )
)

```

When the RFV is run on the input file now, this block will appear as is shown in Figure C.24. This newly defined block may appear complicated. However, to make it clearer as to how the layout

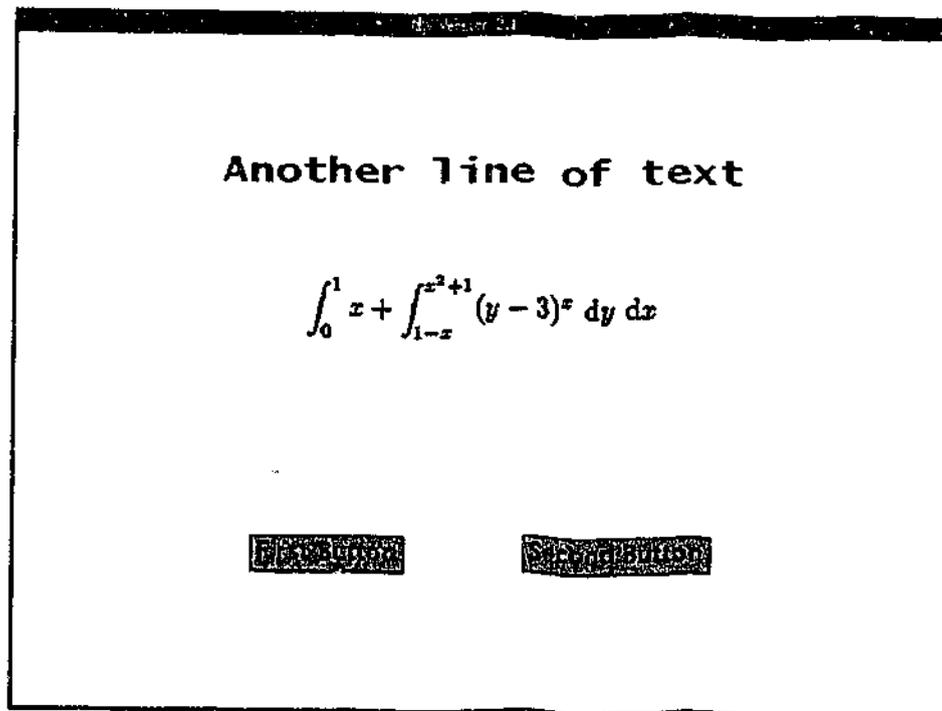


Figure C.23: Further manipulating the horizontal space in a row.

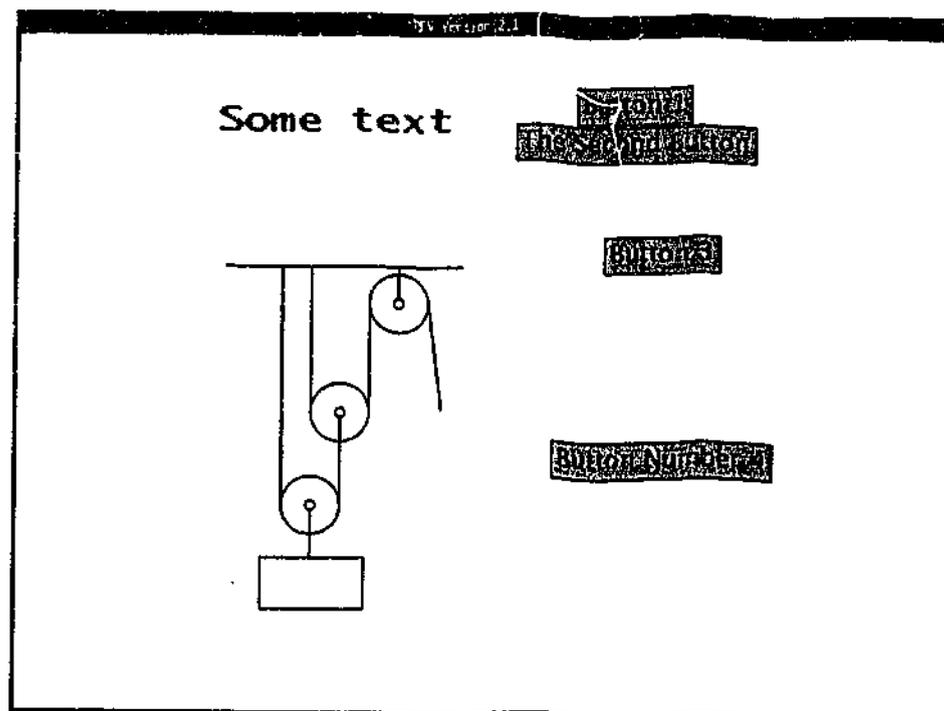


Figure C.24: Example of columns defined within rows.

of the elements is organized, Figure C.25 shows the same block marked with red lines to indicate the boundaries of the rows and columns. Keep in mind that the entire RFV window itself forms a column defined by the DISPLAY parameter. This column contains two ROWs, with the extra vertical space in the column divided equally between the space above the first row, the space between the two rows, and the space below the second row. In Figure C.25 the boundaries of these rows are indicated by the red lines that have been drawn across the entire width of the RFV window.

The first row contains a line of text, a horizontal gap of 50 pixels, and then a column which contains two buttons. The input file also specifies how the extra vertical space within the column should be distributed. However, in the first row in Figure C.25, there is no extra vertical space in the column containing the two buttons. What has happened? When the RFV determines the height of a row it simply uses the maximum height of all of the elements specified within that row. If a column is specified within a row, then the total height of all the elements in that column is used as the column height. Since in the first row the total height of the two buttons in the column is greater than the height of the line of text, the height of the row becomes the total height of the two buttons in the column. The column of course has the same height as the row it is specified within. Since the height of the column is only exactly enough to accommodate the two buttons specified within it, the extra vertical space within the column is zero. Thus, the V_SPACE parameter has no effect on the final layout. The line of text is vertically centred within the row.

The second row is similar to the first, consisting of a static image, a horizontal gap of 50 pixels, and then a column which contains two buttons. However, in this row, the static image has a greater height than the total height of the two buttons defined in the column. Thus the height of the row, and thus also of the column defined in this row, becomes the height of the static image. In this instance, since the height of the column is greater than the total height of the elements defined within it, there is extra vertical space to be allocated. This is then divided up evenly among the two V_SPACE parameters, resulting in the column being split up as is indicated in Figure C.25.

It is important to note that if a V_SPACE or a H_SPACE parameter takes an argument, it is then treated like a visual element in determining the size of a row or column. For example, in the first row we can force a gap to appear between the buttons "Button 1" and "The Second Button" by specifying how many pixels high the vertical space should be. This can be done by placing a value (in this case 20) as an argument of the V_SPACE parameter that is defined between the two buttons.

```

:
component( "Component 2"
:
  block(
    display(
      v_space()
      row(
        text_line("Some text")
        h_space(50)
        column(
          button("Button 1")
          v_space(20)
          button("The Second Button")
        )
      )
    )
  )
)

```

The block will now appear as in Figure C.26 when the RFV is run.

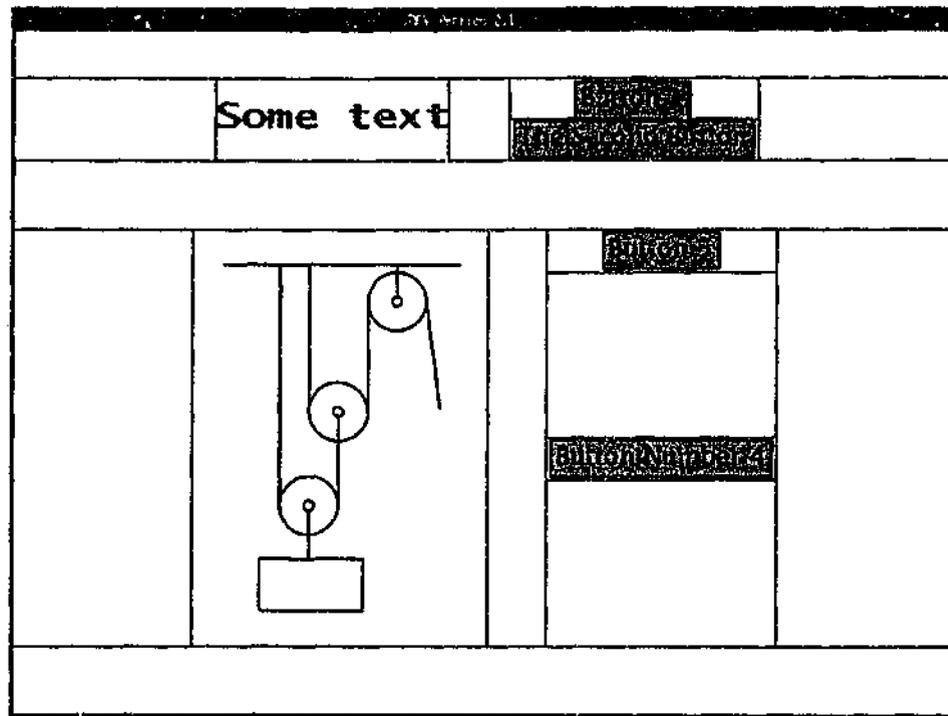


Figure C.25: Layout of various rows and columns within the RFV window.

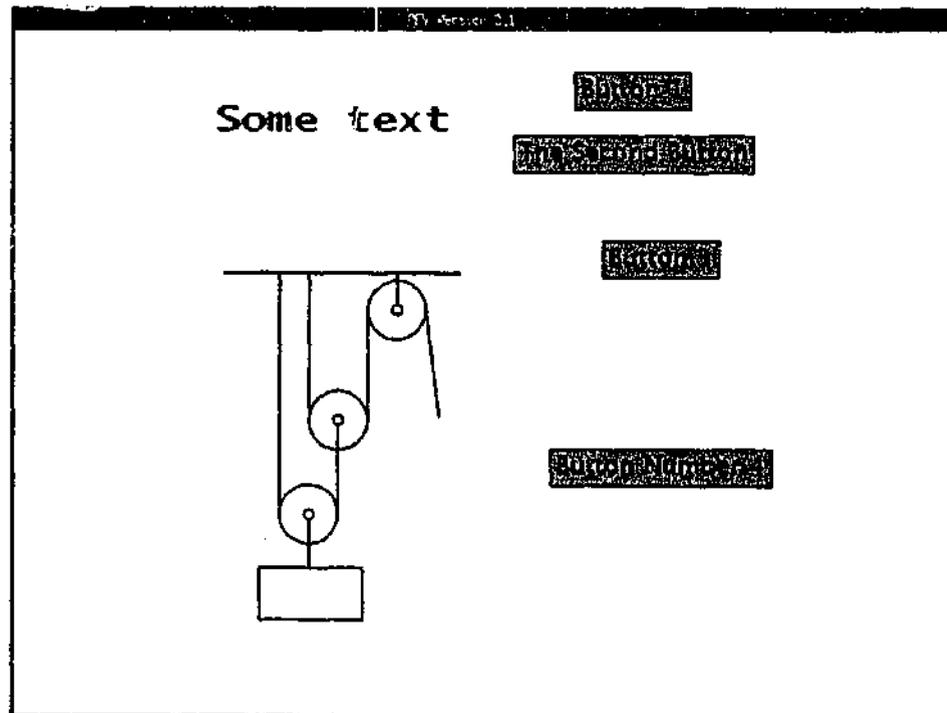


Figure C.26: Forcing a gap between the two buttons in the top row.

It is possible to continue defining ROWs within COLUMNs and COLUMNs within ROWs to any level desired. However, for most cases it likely that the column defined by the DISPLAY parameter and the occasional use of the ROW parameter will be sufficient.

C.5.3 Feedback and Time Limits

We now turn our attention to the topic of providing feedback. In many experimental designs, it is important that the participant receive feedback that lets them know things such as whether their last response was correct, or how long they took to respond. The RFV has a mechanism for providing such feedback. Before we continue however, we will first specify a new COMPONENT called "Component 3", and specify a block within it.

```

:
component( "Component 1"
:
)
component( "Component 2"
:
)
component( "Component 3"
  block(
)
)

```

Within this new block, we will specify a DISPLAY containing a statement and a row consisting of two buttons. The buttons display "TRUE" and "FALSE", allowing a participant to make a choice about the validity of the statement.

```

:
component( "Component 3"
  block(
    display(
      text_line("Rome was built in a day.")
      v_space(80)
      row(
        button("TRUE")
        h_space(100)
        button("FALSE")
      )
    )
  )
)

```

If we now run the RFV on the input file, then the block in "Component 3" will appear as is shown in Figure C.27.

When the participant responds to this block by selecting one of the buttons, we may wish to inform them about whether not they selected the correct button. This can be done by using the FEEDBACK parameter within the BLOCK parameter. A FEEDBACK parameter takes as its first argument the label of the button for which feedback is being provided. Therefore, if we wish to inform the participant that they chose an incorrect response, we would specify a feedback parameter for the "TRUE" button as follows.



Figure C.27: A simple two choice block.

```

:
component( "Component 3"
  block(
    display(
      text_line("Rome was built in a day.")
      v_space(80)
      row(
        button("TRUE")
        h_space(100)
        button("FALSE")
      )
    )
    feedback( "TRUE"
  )
)
)

```

This FEEDBACK parameter is now treated as a block that will appear if the participant selects the button labelled "TRUE". Inside the FEEDBACK parameter, after the relevant button label has been provided, it is possible to do almost anything that can be done inside a BLOCK parameter with the following exceptions. The BUTTON and the STIMULUS parameters cannot be defined inside the DISPLAY parameter of the feedback block, and a feedback block itself cannot contain a FEEDBACK parameter. This is because a feedback block is meant purely as a passive block providing information to the participant. It is not meant to contain interactive elements that might themselves require feedback.

Aside from these restrictions, feedback blocks can do anything else that can be done in a normal block. In our example, we will use the feedback block we have defined to inform the participant that their response was incorrect.

```

:
component( "Component 3"
  block(
    display(
      :
    )
    feedback( "TRUE"
      display(
        text_line("Incorrect"
          foreground(255 0 0) )
        text_line("Rome wasn't built in a day!"
          font("SansSerif" PLAIN 20) )
      )
    )
  )
)

```

Now if we run the RFV, if when we get to the block in "Component 3" we select the "TRUE" button, the RFV window should appear as in Figure C.28.

You will have no doubt noted that the RFV does not require feedback for every button defined. If the program is run again and the "FALSE" button is selected, the program will then just go on with the next block, since there is no feedback specified for this button response. Should we desire that feedback is also given for a correct answer, we can do this by adding another feedback block to the input file.

```

:
component( "Component 3"
  block(
    display(
      :
    )
    feedback( "TRUE"
      :
    )
    feedback( "FALSE"
      display(
        text_line("Correct")
        response_time()
      )
    )
  )
)

```

In the DISPLAY parameter of this new FEEDBACK block, we have used a new parameter. This is the RESPONSE.TIME parameter. It works just like a TEXT.LINE parameter, except that a label is not explicitly given. Instead, the label automatically contains the total time the participant spent viewing the block before giving a response, measured in milliseconds. (This does not include the time taken to load the block.) The RESPONSE.TIME parameter can only be used within a feedback block, and just as with the TEXT.LINE parameter it is possible to specify the FONT and the FOREGROUND colour of the text. Figure C.29 provides an example of the RFV window when the correct button has been selected after the block in Figure C.27 has been visible for about three and half seconds.

If no buttons have been specified within a block, it is still possible to assign feedback by using the NO.BUTTONS keyword in place of the label of a button as the first argument of the FEEDBACK

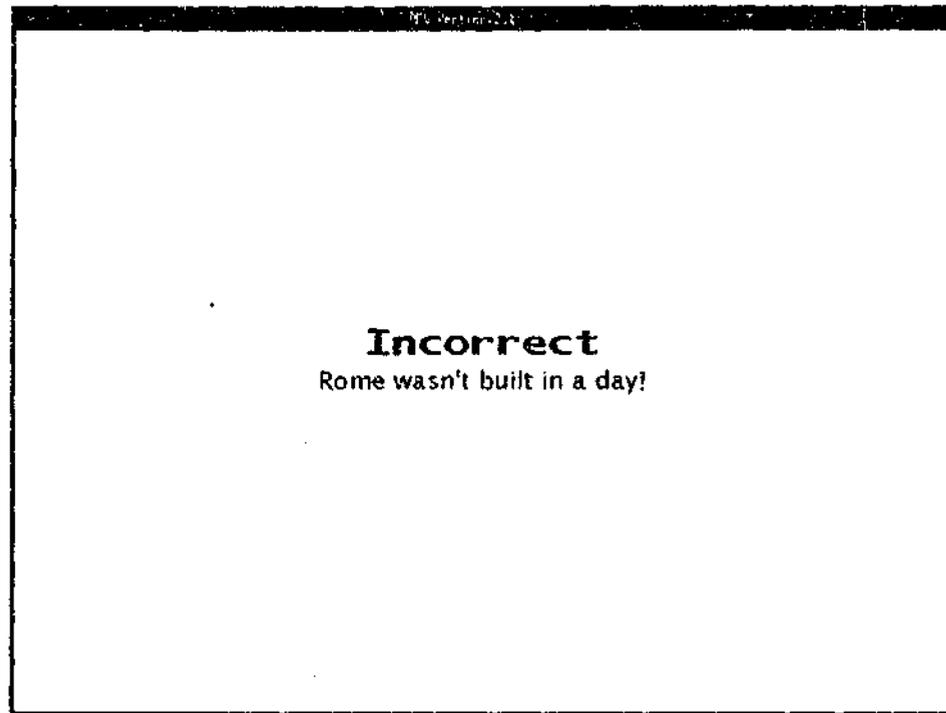


Figure C.28: Feedback for an incorrect response.

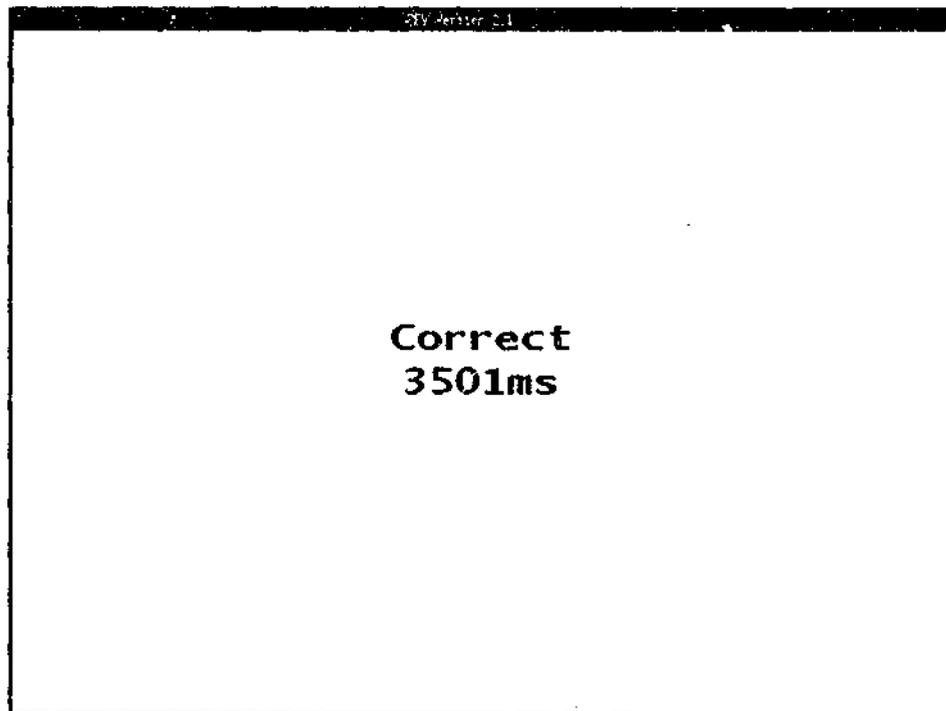


Figure C.29: Feedback that includes the time taken to respond.

parameter. The RFV will give a warning if a FEEDBACK parameter has been defined with an invalid or irrelevant first argument, such as a label that does not match any button labels.

There is one more parameter that can be applied to a block that we have not yet discussed. In many circumstances, it is desirable that a participant can only see a certain block for a limited amount of time. The RFV allows this to be specified by using the TIME_LIMIT parameter. This parameter takes as its argument an integer that is the maximum number of milliseconds that the block should be visible for. For example, we can make the block in "Component 3" visible for only five seconds by adding the following.

```
      :
component( "Component 3"
          block(
              time_limit(5000)
              display(
                  :
              )
              feedback( "TRUE"
                  :
              )
              feedback( "FALSE"
                  :
              )
          )
      )
```

Although we have added the TIME_LIMIT parameter at the beginning of the block, it could also have been placed after the DISPLAY parameter or after the FEEDBACK parameters with the same effect. If the RFV is now run on "Component 3", it is still possible to select one of the buttons (and thus see the resulting feedback block) if the selection is made before the time limit expires. However, if the time limit expires before a button has been selected (or the mouse has been clicked anywhere if the block contains no buttons), then the block will automatically end and the program will just continue with the next block.

Of course, the participant may find it disconcerting to have the block suddenly finish without them having done anything. Therefore, it may be prudent to provide feedback when the time limit expires. This can be done by specifying a feedback block with the TIME_LIMIT_EXPIRED keyword in place of the label of a button as the first argument.

```
    :  
    component( "Component 3"  
      block(  
        :  
        feedback( "TRUE"  
          :  
        )  
        feedback( "FALSE"  
          :  
        )  
        feedback( TIME_LIMIT_EXPIRED  
          display(  
            text_line("Sorry - the time limit expired")  
          )  
        )  
      )  
    )
```

Now if we run the RFV and let the time limit expire on the block in "Component 3", the RFV window will appear as shown in Figure C.30.

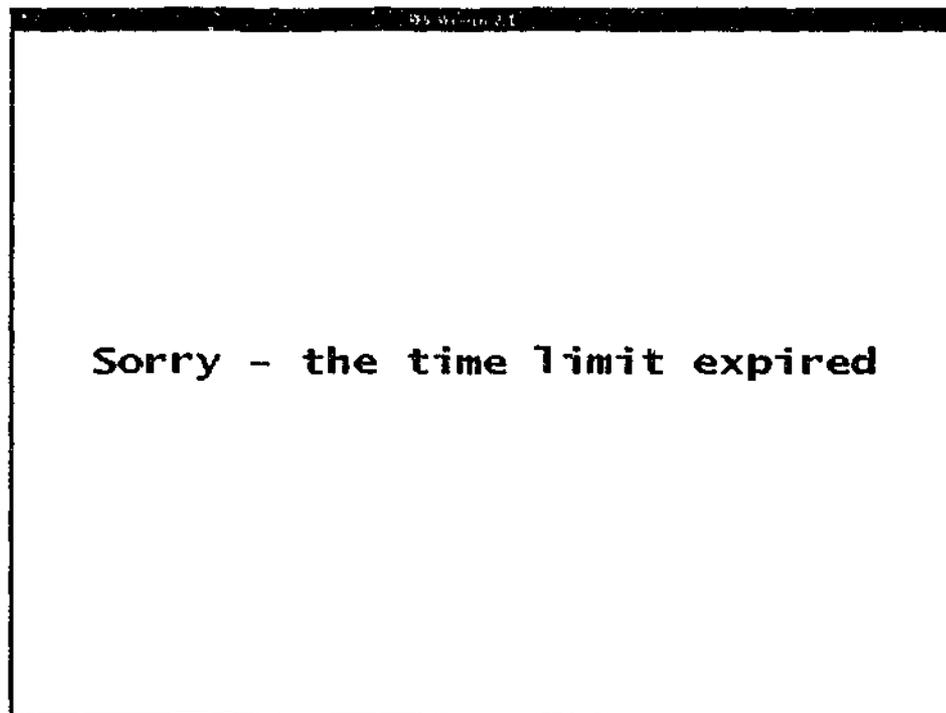


Figure C.30: Feedback to indicate that the time limit expired.

Occasionally, we may want to force the participant to view a block for the exact time limit; that is, we may not want to allow them to end the block early by selecting a button, or if there are no buttons, clicking the mouse anywhere in the RFV window. This can be done by adding the `STRICT` keyword as an optional argument to the `TIME_LIMIT` parameter, placing it after the integer that specifies the number of milliseconds the block should be visible for.

```

      :
      component( "Component 3"
        block(
          time_limit(5000 STRICT)
          display(
            :
          )
          :
        )
      )

```

If the RFV is now run on the input file, the block specified within "Component 3" will be visible for exactly five seconds, regardless of whether the a button is selected or not. As a result, the feedback block that appears can only be the one with `TIME_LIMIT_EXPIRED` as the first argument. Note that `FEEDBACK` blocks can also have a `TIME_LIMIT` parameter defined as an argument. This can be convenient if the feedback is only meant to be presented briefly before the participant continues on with the next block.

In this example, we composed the input file a bit at a time, and then tested it to make sure it did what we expected. This is a good approach to take when creating input files. Also, if unsure about what effect a particular parameter might have in a certain situation, try it and see. If it doesn't work as expected, it is easy to remove it from the input file or try it in a different place.

C.6 RFV Output Data File

Each time the RFV is run, an output data file is produced. The name of this file is the subject ID label plus the extension `.rfvd` (which stands for RFV Data). The file is produced in the directory that the RFV is run from. If a file with the same name already exists in that directory, it will be overwritten.

The RFV output data files record the responses given by a participant for each block in the experiment, and the time they spent looking at each block (which does not include the time to load the block). The output data files can also be used by the Replayer program to replay the way the mouse was moved over the RFV window if a block contained a blurred RFV stimulus. In order that the Replayer program can successfully read the output data files, they should not be modified in any way.

The header of the Stimulus Data File contains four lines. The first line simply identifies the file as an RFV Data File. The second line contains the subject ID label. The third line contains the name of the Input File that has been read in by the RFV. The final line records the date when experiment was conducted (taken from the system).

As the RFV is run, the participant responses and response times are recorded in the output data file for each block of each component presented. If the block contains one or more stimuli, the location of each stimulus in the RFV window is recorded (the x co-ordinate followed by the y co-ordinate), and then the mouse movements within the window are also recorded. Each updated mouse movement is recorded on a line containing five values. The first is the number of the stimulus the mouse was over (or zero if the mouse was not over a stimulus). Second is the time (in milliseconds) that has elapsed since the block was first displayed. The next two values are the x and y co-ordinates of the centre of the focus window (relative to the top left corner of the RFV window). The final piece of information is a flag to determine whether or not the focus window was motion blurred or not (a letter `F` indicates in focus, while a `B` indicates blurred). If the mouse was not over a stimulus, this final flag is not included.

The following example is taken from an RFV output data file. It contains the header, and the output from two blocks within a single component. The second of these two blocks contains two stimuli.

```
RFV Version 2.1 Output Data File
Subject:      test
Input Filename: sample.rfv
Date:        Jan 27, 2001

Component:   Multiple Stimulus
Block 1
  Response:  No Buttons Present
  Time (ms): 882
Block 2
  Stimulus 1: Top left corner at <162, 318>
  Stimulus 2: Top left corner at <612, 168>
  0      66          454  447
  1     1000         218  357  F
  1     1040         215  351  F
  1     1075         211  345  B
  1     1133         209  342  F
  1     1175         208  340  F
          :
  2    37222         716  387  B
  2    37441         669  505  B
  0    37654         652  568
  0    37816         653  573
  Response:   True
  Time (ms): 37839
```

C.7 The Replayer

The Replayer is a companion program to the RFV. It can read in a output data file generated by the RFV, and for any block that contains a stimulus, it can replay the way that the RFV participant moved the focus window over the stimulus. This can benefit the experimenter in two important ways.

Firstly, the Replayer can be used as a data analysis tool. It can replay the movement of the focus window over a stimulus, at either normal speed, or faster or slower speeds. This is done in Focus Window mode. It can also be set to Scanpath mode. In this mode the stimuli are in full focus, and the Replayer draws a scan-path line over the RFV window, based on the locations of the mouse cursor. An example of a scan-path can be seen in Figure C.31.

Secondly, the Replayer is a very useful tool in eliciting retrospective verbal protocols from participants. Since participants can become distracted if they are giving verbal protocols during a complex task, many experimenters have begun to use retrospective verbal protocols in which the participant is required to explain what they were thinking or doing after the task is complete (Ericsson & Simon, 1993). However, this too presents problems as quite often participants can forget or be unsure of what they did or why they did it. By using the Replayer, this problem is overcome since it is clear at any point in time exactly what the participant was looking at, and where they moved to next. The flexibility in the speed of the replay can also help in, for example, allowing participants to explain complicated actions that took place in a brief period of time. This allows more extensive protocols to be obtained.

In the bottom right hand corner of the Replayer window shown in Figure C.31, there is a small black status region. This shows that the replayer is currently paused (the Replayer uses the standard symbols for play, pause and stop that are found on most video recorders or CD players). The red dot above the line on the right also shows that the current replay speed is normal (the dot is above

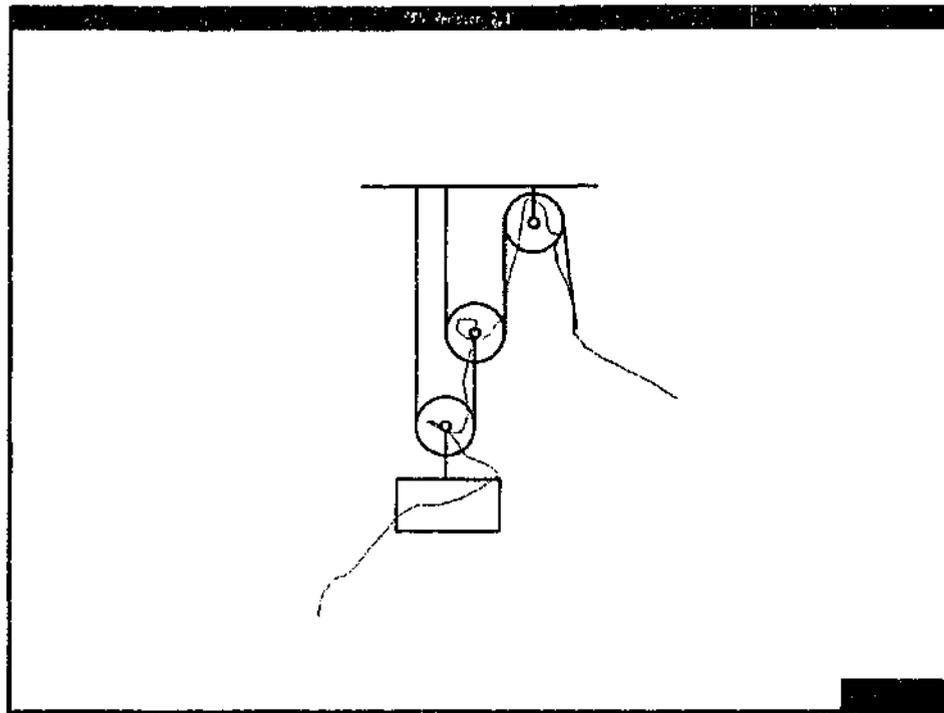


Figure C.31: Example of the Replayer in Scanpath mode.

the centre position). If the replayer speed is increased a level, the red dot moves to the right, and if it decreased a level, the red dot moves to the left.

C.7.1 Running the Replayer

Since it is also a Java 2 application, the Replayer program is launched in the same manner as the RFV program is. The program has been bundled into a Java JAR file called `Replayer.jar`. If we had previously run the RFV with the subject ID 'test', then there should be an output data file called `test.rfvd`. The Replayer can be run with this data file, using the command:

```
java -jar Replayer.jar test.rfvd
```

The `java` command at the beginning indicates that we wish to run a Java program (remember, it must be software version 1.2 or higher). The `-jar Replayer.jar` means the program is to be found in the JAR file called `Replayer.jar`, and `test.rfvd` is the name of the output data file we wish to use.

As with running the RFV, this command line assumes that all the relevant files are in the current directory. If not, the directory paths should be included along with the filename. The contents of the output data file should not have been altered since being produced by the RFV (although the filename may be changed). The output data file also includes in the header the name of the input file that was used to generate the data. If this cannot be found, the Replayer program will report an error. It is recommended that the Replayer be run from the same directory that the RFV was run from, with the input file used and its corresponding images in the same locations as when the RFV was run.

When the Replayer is run with the above command line, a block selection window similar to that shown in Figure C.32 should appear. The subject ID appears at the top of this window, with two lists below it. The list on the left contains the components that may be selected. These will be listed in the order that they were presented in the experiment. When a component is selected, the list on the right will contain a list of the blocks in that component (which are numbered according to their position within the component). If a block contains a stimulus, then an asterisk (*) will appear next to the block label, indicating that the Replayer can be used on that block. By selecting a block, the number of data points corresponding to that block will be reported in the region below

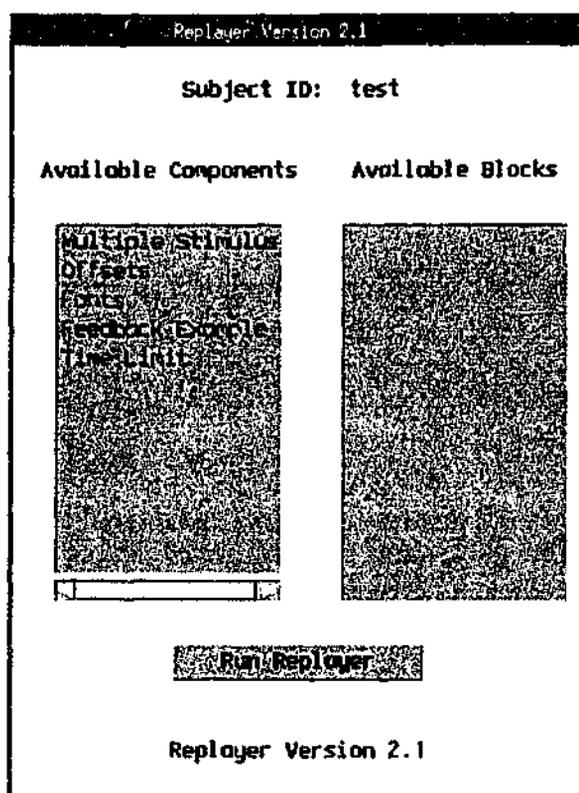


Figure C.32: The Replayer program's block selection window.

the 'Run Replayer' button (which is also where error messages relevant to the selection window will be reported). An example of selecting a component and a block is shown in Figure C.33.

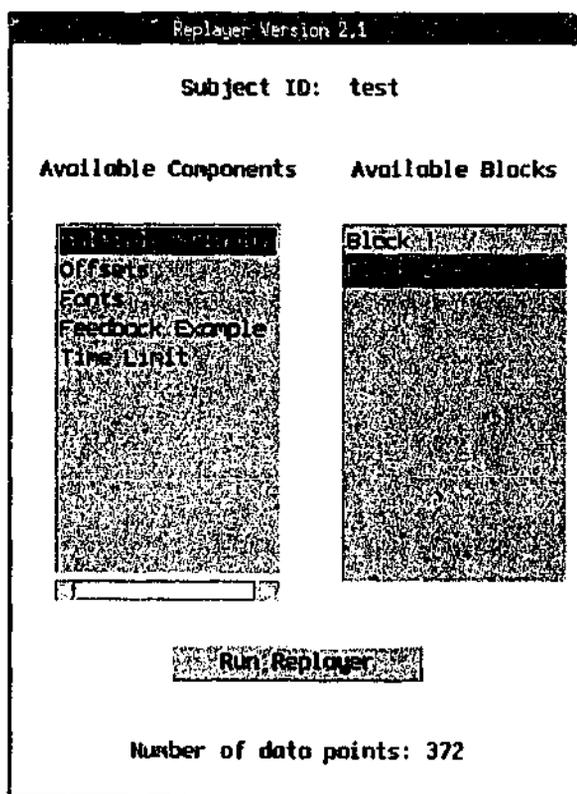


Figure C.33: Selecting a block to be replayed.

Once a block with more than zero data points has been selected, the 'Run Replayer' button can be pressed. The block selection window will disappear and the Replayer window will appear. The Replayer window can be manipulated using the keyboard. Note the Replayer program can be quit from within the block selection window by pressing the 'Q' key.

C.7.2 Keyboard Commands

There is no mouse interaction used in the Replayer program. Instead the keyboard is used, with specific key strokes corresponding to certain commands.

Key: M

Function: Toggle between Modes

Description: This command toggles the Replayer between the Focus Window mode, and the Scanpath mode. The Focus Window mode replays the data exactly as the RFV participant saw it during the experiment, with the focus window moving over the blurred stimulus. The Scanpath mode draws a red line over the original stimulus image, showing the path of the centre of the focus window as the item was viewed. The modes can be toggled at any time, including while a replay is running or paused.

Key: S

Function: Start the Replay

Description: If the Replayer is not already running a replay, this command will start one. If a replay is already in progress, the command is ignored. The replay will begin running at the current speed setting. The status region will indicate to the user that the replay has begun. This command will work regardless of the mode that the Replayer is in.

Key: R

Function: Reset the Replayer

Description: This command resets the Replayer if a replay is running, and is ignored otherwise. Once the Replayer is reset, another replay may be started. This command will work in either mode, and will also work even if the current replay is paused.

Key: P

Function: Pause/Unpause

Description: If a replay is running and currently not paused, this command will pause the replay. If it is already paused, this command will unpause the replay. If no replay is in progress, the command is ignored. When paused, the Replayer will indicate this in the status region. This command will work regardless of the mode that the Replayer is in.

Key: F

Function: Forward One Data Point

Description: This command will only work while the Replayer is paused in a replay. It will advance forward the replay by one data point each time the key is pressed. Again, the command is independent of the mode.

- Key:* G
- Function:* Toggle Guide On/Off
- Description:* This command toggles on or off the guide for the focus window. When in Focus Window mode, the guide can be used to make the current focus position clearer. The guide is a red box that surrounds the focus window. Since the replay is essentially watched like a movie, this can make it easier to keep track of the current focus position during a replay. There is no guide for Scanpath mode. At any time or in any mode, the Focus Window guide can be toggled on or off.
- Key:* T
- Function:* Toggle Status Region On/Off
- Description:* This command toggles on or off the black status region in the bottom right corner of the Replayer. This region displays the current status of the Replayer (playing, paused or stopped) using symbols that are the same as is used on CD players and video recorders. It also displays the current replayer speed by displaying a red dot above the line on the right. The centre position indicates normal speed. If the replayer speed is increased a level, the red dot moves to the right, and if it decreased a level, the red dot moves to the left. This command can be used at any time and in any mode.
- Key:* < and >
- Function:* Change Replay Speed
- Description:* The Replayer has five speed settings. They are Quarter speed, Half speed, Normal speed, Double speed and Quadruple speed. The settings are self-explanatory, with the first two resulting in replays that are slower than the RFV participant was, and the latter two resulting in replays that are faster than the RFV participant was. The Replayer will start at Normal speed. Whenever the '<' key is pressed, the speed will be decreased by one setting. Similarly, if the '>' key is pressed, the speed will increase by one setting. If for example, the speed is already set at the maximum setting (Quadruple speed), pressing the '>' will then do nothing. This also applies to the '<' key for the minimum setting. The speed setting can be changed at any time.
- Key:* Q
- Function:* Quit
- Description:* This command simply quits the program. This command will work at any time, regardless of whether a replay is in progress or not.

C.8 Input File Parameter Specifications

This section gives the specifications of the parameters that are used in an RFV input file. To declare a parameter, the parameter name needs to be input, followed by an opening bracket. Then the required arguments must be included, followed by any optional arguments that are being used. The declaration is completed by a closing bracket. The arguments of a parameter may be labels, filenames, keywords, integers, or other parameters. Some parameters may require no arguments.

The specifications indicate which parameters a given parameter may be declared inside; that is, each parameter may be included as an argument of only a limited group of other parameters. When a parameter is declared, the required arguments listed must appear within the brackets of that parameter in the order listed. Optional arguments may then follow the required arguments, although these can be in any order. Optional arguments can only be specified once for each parameter (unless it is indicated otherwise).

Two of the parameters do not appear as arguments to other parameters. The input file requires the DEFAULTS parameter to be completely defined first, followed by multiple (at least one) instances of the COMPONENT parameter.

Parameter Name: BACKGROUND

Declared Inside: BLOCK, FEEDBACK, BUTTON

Required Arguments: Three Integers

Optional Arguments: None

Description: This parameter specifies the background colour to be assigned to either the RFV window (when declared inside BLOCK or FEEDBACK) or a BUTTON. The three arguments it takes are integers in the range 0 to 255 inclusive. These integers represent the red, green and blue (RGB) contributions to the colour. The RGB model is the standard one used by most computer graphics programs. This parameter will override any default background colour setting.

Parameter Name: BLOCK

Declared Inside: COMPONENT

Required Arguments: None

Optional Arguments: DISPLAY, BACKGROUND, TIME_LIMIT, Multiple FEEDBACKs

Description: A block is used to define the elements that appear in the RFV window at a given time. Each block determines the layout of its elements with the DISPLAY parameter, and can also specify a TIME_LIMIT on the display duration of the block. FEEDBACK based on the participant's response can be specified, as can the BACKGROUND colour of the block, overriding the default setting.

Parameter Name: **BUTTON**
Declared Inside: **DISPLAY, ROW, COLUMN**
Required Arguments: **One Label**
Optional Arguments: **FONT, FOREGROUND, BACKGROUND**
Description: This parameter defines a button with the specified label. The size of the button will depend on the font being used, with the width being adjusted to accommodate the label. If a least one button is declared within a block, then only a mouse click on a button will end that block. If no buttons are declared within a block, then a mouse click anywhere in the RFV window will end that block. Buttons can also be used in conjunction with the **FEEDBACK** parameter to provide feedback to participants based on the button that was selected in a block.

Parameter Name: **BUTTON_BACKGROUND**
Declared Inside: **DEFAULTS**
Required Arguments: **Three Integers**
Optional Arguments: **None**
Description: This parameter specifies the default button background colour. If not explicitly set, this colour will be light gray. See **BACKGROUND** for details about the arguments.

Parameter Name: **BUTTON_FONT**
Declared Inside: **DEFAULTS**
Required Arguments: **One Label, One Keyword, One Integer**
Optional Arguments: **None**
Description: This parameter specifies the default button font. If not explicitly set, a default system font will be used. See **FONT** for details about the arguments.

Parameter Name: **BUTTON_FOREGROUND**
Declared Inside: **DEFAULTS**
Required Arguments: **Three Integers**
Optional Arguments: **None**
Description: This parameter specifies the default button foreground colour. If not explicitly set, this colour will be black. See **FOREGROUND** for details about the arguments.

Parameter Name: COLUMN
Declared Inside: ROW
Required Arguments: None
Optional Arguments: TEXT_LINE, IMAGE, BUTTON[†], STIMULUS[‡], ROW, V_SPACE, RESPONSE_TIME[‡]
[†] Cannot be used within a FEEDBACK block
[‡] Can only be used within a FEEDBACK block

Description: This parameter is used to vertically layout the elements of a region of the RFV window. It can contain visual elements, empty spaces and other regions in the form of ROWs.

Parameter Name: COMPONENT
Declared Inside: Input File
Required Arguments: One Label, Multiple BLOCKs
Optional Arguments: None

Description: This parameter is used to bind together a group of blocks in a set order. The first argument is a label to identify the component, which should be unique for each component that is defined. When the RFV program is run, the components can then be arranged in any order. This parameter can only be used in the input file after the DEFAULTS parameter. Each component must contain at least one BLOCK.

Parameter Name: DEFAULTS
Declared Inside: Input File
Required Arguments: FRAME_SIZE, FOCUS_WINDOW, MOTION_BLUR_SPEED
Optional Arguments: TEXT_FONT, TEXT_FOREGROUND, BUTTON_FONT, BUTTON_FOREGROUND, BUTTON_BACKGROUND, DISPLAY_BACKGROUND

Description: This parameter must be the first one to appear in the input file. It is used to specify the default settings for various attributes of the RFV's visual elements. It also contains the parameter which specifies the dimensions of the RFV window. If the optional parameters are not set, the TEXT_FOREGROUND and BUTTON_FOREGROUND colours will be set to black, the BUTTON_BACKGROUND colour will be set to light gray, and the DISPLAY_BACKGROUND colour will be white. The TEXT_FONT and BUTTON_FONT will set to a plain *Helvetica* font of point size 20, or if this is not available, a default system font.

Parameter Name: DISPLAY
Declared Inside: BLOCK, FEEDBACK
Required Arguments: None
Optional Arguments: TEXT_LINE, IMAGE, BUTTON[†], STIMULUS[†], ROW, V_SPACE, RESPONSE_TIME[‡]
[†] Cannot be used within a FEEDBACK block
[‡] Can only be used within a FEEDBACK block

Description: This parameter is used to vertically layout the elements of a block that are to be displayed. It defines the entire RFV window as a column, which can contain visual elements, empty spaces and other regions in the form of ROWs. Note that if declared as an argument to the BLOCK parameter, RESPONSE_TIME is not available as a visual element within the DISPLAY or any of its regions. Also, if declared as an argument to the FEEDBACK parameter, BUTTON and STIMULUS cannot be declared within the DISPLAY or any of its regions.

Parameter Name: DISPLAY_BACKGROUND
Declared Inside: DEFAULTS
Required Arguments: Three Integers
Optional Arguments: None

Description: This parameter specifies the default background colour for the RFV window. If not explicitly set, this colour will be white. See BACKGROUND for details about the arguments.

Parameter Name: FEEDBACK
Declared Inside: BLOCK
Required Arguments: One Label or One Keyword
Optional Arguments: DISPLAY, BACKGROUND, TIME_LIMIT

Description: This parameter is used to present different blocks of information to participants depending on their response to the block in which the parameter is defined. The first argument of this parameter is either the label of one of the buttons defined in the block, the keyword NO_BUTTONS (which can be used if no buttons were defined in the block and the participant can thus click the mouse anywhere to continue), or the keyword TIME_LIMIT_EXPIRED (which can be used any time a block uses a TIME_LIMIT). A feedback block will only be invoked if the participants response matches that of the first argument (for example, they selected the button with the corresponding label, or the time limit expired before they responded). If this happens, the feedback block acts similar to a normal block, with the following exceptions. The BUTTON and the STIMULUS parameters cannot be defined inside the DISPLAY parameter of the feedback block, and a feedback block cannot contain a FEEDBACK parameter. This is because a feedback block is meant purely as a passive block providing information to the participant, and is not meant to contain interactive elements that might themselves require feedback.

Parameter Name: FOCUS_WINDOW
Declared Inside: DEFAULTS, STIMULUS
Required Arguments: Eight Integers
Optional Arguments: FOCUS_WINDOW_OFFSETS
Description: This parameter takes in the dimensions of the four regions that define the focus window. The arguments thus effectively form four pairs of integers, specifying width followed by height (in pixels). The first pair gives the dimensions for the outermost transition region. The second pair, the middle transition region and the third pair, the innermost transition region. The final pair is the dimensions of the actual focus region inside the innermost transition region. Each pair of numbers should be no larger than the previous pair, otherwise an error will be reported. When the dimensions are set, all of the regions are centred around the dot that defines the cursor location within a stimulus.

Parameter Name: FOCUS_WINDOW_OFFSETS
Declared Inside: FOCUS_WINDOW
Required Arguments: Eight Integers
Optional Arguments: None
Description: This parameter allows the four regions of the focus window to be offset, so that they are not necessarily centred around the dot that defines the cursor location within a stimulus. The arguments form four pairs of integers, each consisting of a horizontal offset value followed by a vertical offset value (in pixels). These values can be positive or negative. The first pair of values applies to the outermost transition region. The second pair to the middle transition region and the third pair, the innermost transition region. The final pair is the offsets for the actual focus region inside the innermost transition region. If only one region needs to be offset, then the arguments for the other regions can be set to zero. Care needs to be taken when setting this parameter, since the RFV does no checking of the consequences of the settings provided. For most purposes however, this parameter will not need to be used.

Parameter Name: FONT
Declared Inside: TEXT_LINE, BUTTON, RESPONSE_TIME
Required Arguments: One Label, One Keyword, One Integer
Optional Arguments: None
Description: This parameter specifies the font to be used for a TEXT_LINE, BUTTON or RESPONSE_TIME. The first argument is the name of the font, which can be one of *Serif*, *SansSerif*, *Dialog*, *DialogInput*, *Monospaced* or *Symbol*. Some of these fonts might not be available on certain systems. The font name is given as a label. The second argument is the style of the font. This can be either PLAIN, BOLD, ITALIC or BOLD_ITALIC. Finally, the point size of the font must be specified. Just as some fonts are not available on all systems, not all fonts are always available in every size or style. This parameter will override any default font setting. However, if the RFV cannot load the font specified, it will revert to a default font.

Parameter Name: FOREGROUND
Declared Inside: TEXT.LINE, BUTTON, RESPONSE.TIME
Required Arguments: Three Integers
Optional Arguments: None
Description: This parameter specifies the foreground colour to be assigned to the text that is defined in a TEXT.LINE, BUTTON or RESPONSE.TIME declaration. The three arguments it takes are integers in the range 0 to 255 inclusive. These integers represent the red, green and blue (RGB) contributions to the colour. The RGB model is the standard one used by most computer graphics programs. This parameter will override any default foreground colour setting.

Parameter Name: FRAME.SIZE
Declared Inside: DEFAULTS
Required Arguments: Two Integers
Optional Arguments: None
Description: This parameter takes in the width followed by the height (in pixels) of the window that will contain each block. It is recommended that this window be the same size as the screen resolution being used. This will enable the RFV window to fill the entire screen, leaving no background remaining to distract the participant. The frame size also needs to be large enough to accommodate all of the visual elements that are presented in each block. If the frame size is not large enough, the program will inform the user of this and terminate.

Parameter Name: H.SPACE
Declared Inside: ROW
Required Arguments: None
Optional Arguments: One Integer
Description: With no arguments, this parameter is used to divide up extra horizontal space in a region of the RFV window. If this parameter appears with no arguments more than once in a region, then the extra horizontal space is divided evenly among each instance. This parameter can also be used with an integer as an argument. The integer specifies the width of a fixed horizontal space to be inserted into the region where the parameter is declared.

- Parameter Name:* IMAGE
- Declared Inside:* DISPLAY, ROW, COLUMN
- Required Arguments:* One Filename
- Optional Arguments:* None
- Description:* This parameter allows a static image to be loaded and displayed by the RFV. The argument taken by this parameter is the filename for the image. The image filename should include the directory path if needed, and it is recommended that the full directory path be used if the image is in a different directory from where the program will be run. The image file must be in either the GIF or JPEG graphics file format (these files usually have a .gif or .jpg extension).
-
- Parameter Name:* MOTION_BLUR_SPEED
- Declared Inside:* DEFAULTS, STIMULUS
- Required Arguments:* One Integer
- Optional Arguments:* None
- Description:* This parameter takes in a single value, which is the mouse speed that determines the onset of motion blur. If the user moves the mouse at greater than this speed, then the focus window will be motion blurred. If the user then slows down to below this speed, or stops moving the mouse altogether, then full focus will return. The speed is measured in pixels per second. A value of 100 is a reasonable starting point in trying to determine an appropriate setting. A very low value will see motion blur occur for even slow mouse movements, whereas a high value will allow the participant to move the mouse at a reasonable speed and still maintain full focus.
-
- Parameter Name:* RESPONSE_TIME
- Declared Inside:* DISPLAY, ROW, COLUMN
- Required Arguments:* None
- Optional Arguments:* FONT, FOREGROUND
- Description:* This parameter defines a visual element that can only be used within a FEEDBACK block. It is similar to the TEXT_LINE parameter, except that it accepts no label as an argument. Instead, the text that it produces is the amount of time (in milliseconds) that the participant spent looking at the block in which the FEEDBACK parameter is defined. This is the amount of time between when the block first appears (not including loading time) and when the block disappears (due to either a mouse click or the expiration of a time limit). If, for example, this time is 1625 milliseconds, then using this parameter will be equivalent to using a TEXT_LINE parameter with the label "1625ms".

Parameter Name: ROW
Declared Inside: DISPLAY, COLUMN
Required Arguments: None
Optional Arguments: TEXT_LINE, IMAGE, BUTTON[†], STIMULUS[†], COLUMN, H_SPACE, RESPONSE_TIME[‡]
[†] Cannot be used within a FEEDBACK block
[‡] Can only be used within a FEEDBACK block

Description: This parameter is used to horizontally layout the elements of a region of the RFV window. It can contain visual elements, empty spaces and other regions in the form of COLUMNS.

Parameter Name: STIMULUS
Declared Inside: DISPLAY, ROW, COLUMN
Required Arguments: Five Filenames
Optional Arguments: None

Description: This parameter allows a set of stimulus images to be loaded and displayed by the RFV. The arguments taken by this parameter are the filenames for the five images needed for the stimulus. The image filenames should include the directory paths if needed, and it is recommended that the full directory path be used if the images are in a different directory from where the program will be run. The first label is the filename for the fully blurred image of the stimulus. The second is the filename for the image to be used for the outermost transition region. The third, for the middle transition region and the fourth, for the innermost transition region. The final label is the filename for the stimulus proper, that is, the stimulus in full focus. Each image therefore should be slightly less blurred than the previous one. All of the image files must be in either the GIF or JPEG graphics file format (these files usually have a .gif or .jpg extension).

Parameter Name: TEXT_FONT
Declared Inside: DEFAULTS
Required Arguments: One Label, One Keyword, One Integer
Optional Arguments: None

Description: This parameter specifies the default text font. If not explicitly set, a default system font will be used. See FONT for details about the arguments.

Parameter Name: TEXT_FOREGROUND
Declared Inside: DEFAULTS
Required Arguments: Three Integers
Optional Arguments: None

Description: This parameter specifies the default text foreground colour. If not explicitly set, this colour will be black. See FOREGROUND for details about the arguments.

Parameter Name: TEXT_LINE
Declared Inside: DISPLAY, ROW, COLUMN
Required Arguments: One Label
Optional Arguments: FONT, FOREGROUND
Description: This parameter defines a line of text that is specified by the label. The size of the text will depend on the font being used, with the text line only taking up as much space as is needed.

Parameter Name: TIME_LIMIT
Declared Inside: BLOCK, FEEDBACK
Required Arguments: One Integer
Optional Arguments: One Keyword
Description: Occasionally it may be desirable for there to be a maximum time limit that a block is present for. If this parameter is set within a block, then the RFV will only display the block for the time specified in the argument. The time specified is measured in milliseconds. This still allows the participant to respond with a mouse click, but only if the response can be made before the time limit expires. If the time limit is reached before the participant has responded, then the next block is automatically loaded. If the optional STRICT keyword is present, this will force the block to remain for the exact time limit specified by not allowing the participant to end the block early with a mouse click.

Parameter Name: V_SPACE
Declared Inside: DISPLAY, COLUMN
Required Arguments: None
Optional Arguments: One Integer
Description: With no arguments, this parameter is used to divide up extra vertical space in a region of the RFV window. If this parameter appears with no arguments more than once in a region, then the extra vertical space is divided evenly among each instance. This parameter can also be used with an integer as an argument. The integer specifies the height of a fixed vertical space to be inserted into the region where the parameter is declared.

References

- Abney, M., McPeck, M. S., & Ober, C. (2000). Estimation of variance components of quantitative traits in inbred populations. *American Journal of Human Genetics*, *66*, 629–650.
- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. N. Petrov & F. Czaki (Eds.), *Proceedings of the second international symposium on information theory* (pp. 267–281). Budapest: Akademiai Kiado.
- Akmajian, A., Demers, R. A., & Harnish, R. M. (1984). *Linguistics: An introduction to language and communication* (Second ed.). Massachusetts: MIT Press.
- Alpern, M., Lawrence, M., & Wolsk, D. (1967). *Sensory processes*. Belmont, California: Brooks/Cole Publishing Company.
- Anderson, R. H. (1968). Syntax-directed recognition of hand-printed two-dimensional mathematics. In M. Klerer & J. Reinfelds (Eds.), *Interactive systems for experimental applied mathematics*. New York: Academic Press.
- Anderson, R. H. (1977). Two-dimensional mathematical notation. In K. S. Fu (Ed.), *Syn-tactic pattern recognition: Applications* (pp. 147–177). New York: Springer-Verlag.
- Arnold, A., & Guessarian, I. (1996). *Mathematics for computer science*. New York: Prentice Hall.
- Ashcraft, M. H. (1992). Cognitive arithmetic: A review of data and theory. *Cognition*, *44*, 75–106.

- Baddeley, A. (1990). *Human memory: Theory and practice*. Boston: Allyn and Bacon.
- Baldi, P., Chauvin, Y., Hunkapiller, T., & McClure, M. A. (1994). Hidden markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences (USA)*, *91*, 1059-1063.
- Becker, C. A. (1980). Semantic context effects in visual word recognition: An analysis of semantic strategies. *Memory and Cognition*, *8*, 493-512.
- Bell, R. A., & Bevan, W. (1968). Influence of anchors upon the operation of certain gestalt organizing principles. *Journal of Experimental Psychology*, *78*, 670-678.
- Blackwell, A. F., Jansen, A. R., & Marriott, K. (2000). Restricted focus viewer: A tool for tracking visual attention. In M. Anderson, P. Cheng, & V. Haarslev (Eds.), *Theory and application of diagrams. Lecture notes in artificial intelligence (LNAI) 1889* (pp. 162-177). Springer-Verlag.
- Bower, T. G. (1967). Phenomenal identity and form perception in an infant. *Perception & Psychophysics*, *2*(2), 74-76.
- Brandt, S. A., & Stark, L. W. (1997). Spontaneous eye movements during visual imagery reflect the content of the visual scene. *Journal of Cognitive Neuroscience*, *9*(1), 27-38.
- Butterworth, B. (1999). *The mathematical brain*. London: Macmillan.
- Cajori, F. (1928). *A history of mathematical notations* (Vol. 1 & 2). Open Court Publishing Company.
- Campbell, J. I. D. (1994). Architectures for numerical cognition. *Cognition*, *53*, 1-44.
- Chan, K.-F., & Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal of Document Analysis and Recognition*, *3*(1), 3-15.
- Chang, S.-K. (1970). A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences*, *2*, 253-272.

- Chang, S.-K. (Ed.). (1990). *Visual languages and visual programming*. New York: Plenum Press.
- Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton.
- Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge, Massachusetts: MIT Press.
- Clark, J. M., & Campbell, J. I. D. (1991). Integrated versus modular theories of number skills and acalculia. *Brain and Cognition, 17*, 204-239.
- Coren, S., Ward, L. M., & Enns, J. T. (1994). *Sensation and perception* (Fourth ed.). Harcourt Brace and Co.
- Crystal, D. (1987). *The cambridge encyclopedia of language*. Cambridge: Cambridge University Press.
- Descartes, R. (1637). *La géométrie*.
- Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press.
- Egan, D. E., & Schwartz, B. J. (1979). Chunking in recall of symbolic drawings. *Memory and Cognition, 7*(2), 149-158.
- Ericsson, K. A., & Simon, H. A. (1993). *Protocol analysis: Verbal reports as data* (Revised ed.). Cambridge, Massachusetts: MIT Press.
- Ernest, P. (1987). A model of the cognitive meaning of mathematical expressions. *British Journal of Educational Psychology, 57*, 343-370.
- Ferriera, F., & Clifton, C., Jr. (1986). The independence of syntactic processing. *Journal of Memory and Language, 25*, 348-368.
- Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28*, 3-71.

- Forster, K. I. (1998). The pros and cons of masked priming. *Journal of Psycholinguistic Research, 27*(2), 203-233.
- Forster, K. I., & Davis, C. (1984). Masked priming and frequency attenuation in lexical access. *Journal of Experimental Psychology: Learning, Memory and Cognition, 10*, 680-698.
- Forster, K. I., Davis, C., Schoknecht, C., & Carter, R. (1987). Masked priming with graphemically related forms: Repetition or partial activation? *Quarterly Journal of Experimental Psychology, 39*, 211-251.
- Frazier, L., & Fodor, J. D. (1978). The sausage machine: A new two-stage parsing model. *Cognition, 6*, 291-325.
- Frazier, L., & Rayner, K. (1982). Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology, 14*, 178-210.
- Frensch, P. A., & Geary, D. C. (1993). Effects of practice on component processes in complex mental addition. *Journal of Experimental Psychology: Learning, Memory and Cognition, 19*(2), 433-456.
- Fromkin, V., & Rodman, R. (1993). *An introduction to language* (Fifth ed.). Fort Worth, Texas: Harcourt Brace Jovanovich College Publishers.
- Futrelle, R. P., & Rumshisky, A. (2001). Discourse structure of text-graphics documents. In A. Butz, A. Krueger, P. Oliver, & M. Zhou (Eds.), *1st international symposium on smart graphics*. ACM Press.
- Gelb, I. J. (1963). *A study of writing* (Second ed.). Chicago: Chicago University Press.
- Ghosh, J. K., & Samanta, T. (2001). Model selection - an overview. *Current Science, 80*(9), 1135-1144.

- Glaisher, J. W. L. (1915). Logarithms and computation. In C. G. Knott (Ed.), *Napier Tercentenary Memorial Volume*. London: Longmans, Green & Co.
- Gleason, J. B., & Ratner, N. B. (Eds.). (1993). *Psycholinguistics*. Harcourt Brace College Publishers.
- Goldstein, E. B. (1996). *Sensation and perception* (Fourth ed.). Brooks/Cole Publishing Company.
- Groot, A. D. de. (1965). *Thought and choice in chess*. The Hague: Mouton.
- Gurr, C. A. (1998). Theories of visual and diagrammatic reasoning: Foundational issues. In G. Allwein, K. Marriott, & B. Meyer (Eds.), *AAAI fall symposium: Formalizing reasoning with visual and diagrammatic representations*. AAAI Press.
- Hayes, J. R. (1973). On the function of visual imagery in elementary mathematics. In W. G. Chase (Ed.), *Visual information processing* (pp. 177-214). New York: Academic Press.
- Hegarty, M. (1992). Mental animation: Inferring motion from static diagrams of mechanical systems. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 18(5), 1084-1102.
- Hegarty, M., Mayer, R. E., & Monk, C. A. (1995). Comprehension of arithmetic word problems: A comparison of successful and unsuccessful problem solvers. *Journal of Educational Psychology*, 87, 18-32.
- Heij, W. la, Dirkx, J., & Kramer, P. (1990). Categorical interference and associative priming in picture naming. *British Journal of Psychology*, 81(4), 511-525.
- Henderson, J. M., McClure, K. K., Pierce, S., & Schrock, G. (1997). Object identification without foveal vision: Evidence from an artificial scotoma paradigm. *Perception & Psychophysics*, 59(3), 323-346.

- Heyer, K. den, Briand, K., & Dannenbring, G. L. (1983). Strategic factors in a lexical decision task: Evidence for automatic and attention driven processes. *Memory and Cognition*, *11*, 374-381.
- Holmes, D. S. (1968). Search for "closure" in a visually perceived pattern. *Psychological Bulletin*, *70*(5), 296-312.
- Hong, E. L., & Yelland, G. W. (1997). The generality of lexical neighbourhood effects. In H. C. Chen (Ed.), *Cognitive processing of chinese and related asian languages* (pp. 187-203). Hong Kong: The Chinese University Press.
- Jackson, M., & Warrington, E. K. (1986). Arithmetic skills in patients with unilateral cerebral lesions. *Cortex*, *22*, 611-620.
- Jansen, A. R. (2000). *Restricted focus viewer (RFV) Version 2.0 User's manual* (Tech. Rep. No. 2000/78). School of Computer Science and Software Engineering, Monash University.
- Jansen, A. R. (2001). *Restricted focus viewer (RFV) Version 2.1 User's manual and tutorial* (Tech. Rep. No. 2001/87). School of Computer Science and Software Engineering, Monash University.
- Jansen, A. R., Marriott, K., & Yelland, G. W. (2000). Constituent structure in mathematical expressions. In L. R. Gleitman & A. K. Joshi (Eds.), *Proceedings of the twenty second annual conference of the cognitive science society* (pp. 238-243). Lawrence Erlbaum Associates.
- Johnson, N. F. (1968). The influence of grammatical units on learning. *Journal of Verbal Learning and Verbal Behaviour*, *7*, 236-240.
- Johnson, N. F. (1970). Chunking and organization in the process of recall. In G. H. Bower (Ed.), *The psychology of learning and motivation* (Vol. 4, pp. 171-247). New York: Academic Press.

- Johnston, M., Hayward, W., & Sujica, R. (1999). Masked priming of object classification decisions: Evidence for unconscious processing. *Australian Journal of Psychology, 48* (Suppl.), 32. (26th Australian Experimental Psychology Conference)
- Johnstone, A. H., & Kellett, N. C. (1980). Learning difficulties in school science - towards a working hypothesis. *European Journal of Science Education, 2*(2), 175-181.
- Judge, G. G., Griffiths, W. E., Hill, R. C., Lütkepohl, H., & Lee, T.-C. (1985). *The theory and practice of econometrics* (Second ed.). New York: John Wiley & Sons.
- Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology, 8*, 441-480.
- Just, M. A., & Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review, 87*, 329-354.
- Just, M. A., & Carpenter, P. A. (1987). *The psychology of reading and language comprehension*. Massachusetts: Allyn and Bacon.
- Katz, J., & Fodor, J. (1963). The structure of semantic theory. *Language, 39*, 170-210.
- Kirshner, D. (1989). The visual syntax of algebra. *Journal for Research in Mathematics Education, 20*(3), 274-287.
- Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14*, 511-550.
- Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994). Hidden markov models in computational biology: Application to protein modeling. *Journal of Molecular Biology, 235*, 1501-1531.
- Latimer, C. R. (1988). Eye-movement data: Cumulative fixation time and cluster analysis. *Behavior Research Methods, Instruments & Computers, 20*(5), 437-470.

- Lohse, G. L., & Johnson, E. J. (1996). A comparison of two process tracing methods for choice tasks. *Organizational Behavior and Human Decision Processes*, 68(1), 28-43.
- Marriott, K., Meyer, B., & Wittenburg, K. (1998). A survey of visual language specification and recognition. In K. Marriott & B. Meyer (Eds.), *A theory of visual languages*. Springer-Verlag.
- McCloskey, M., & Caramazza, A. (1987). Cognitive mechanisms in normal and impaired number processing. In G. Deloche & X. Seron (Eds.), *Mathematical disabilities: A cognitive neuropsychological perspective* (pp. 201-219). Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- McCloskey, M., Harley, W., & Sokol, S. M. (1991). Models of arithmetic fact retrieval: An evaluation in light of findings from normal and brain-damaged subjects. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 17(3), 377-397.
- McConkie, G. W., & Rayner, K. (1975). The span of the effective stimulus during a fixation in reading. *Perception & Psychophysics*, 17, 578-586.
- Meyer, D. E., & Schvaneveldt, R. W. (1971). Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *Journal of Experimental Psychology*, 90, 227-234.
- Miller, G., & Gildea, P. (1987). How children learn words. *Scientific American*, 257(3), 94-99.
- Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63, 81-97.
- Mitchell, D. C., & Green, D. W. (1978). The effects of context and content on immediate processing in reading. *Quarterly Journal of Experimental Psychology*, 30, 609-636.
- Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, New Jersey: Prentice-Hall.

- Osaka, N., & Oda, K. (1994). Moving window generator for reading experiments. *Behavior Research Methods, Instruments & Computers*, 26(1), 49-53.
- Payne, J. W., Bettman, J. R., & Johnson, E. J. (1993). *The adaptive decision maker*. Cambridge, England: Cambridge University Press.
- Peterson, L. R., & Peterson, M. J. (1959). Short-term retention of individual verbal items. *Journal of Experimental Psychology*, 58, 193-198.
- Pinker, S. (1994). *The language instinct*. New York: William Morrow and Co.
- Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257-285.
- Ranney, M. (1987). The role of structural context in perception: Syntax in the recognition of algebraic expressions. *Memory and Cognition*, 15(1), 29-41.
- Rayner, K. (1975). The perceptual span and peripheral cues in reading. *Cognitive Psychology*, 7, 65-81.
- Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3), 372-422.
- Rayner, K., & Pollatsek, A. (1989). *The psychology of reading*. Englewood Cliffs, New Jersey: Prentice-Hall.
- Rayner, K., & Pollatsek, A. (1992). Eye movements and scene perception. *Canadian Journal of Psychology*, 46, 342-376.
- Reicher, G. M. (1969). Perceptual recognition as a function of meaningfulness of stimulus material. *Journal of Experimental Psychology*, 81, 275-280.
- Rosbergen, E., Wedel, M., & Pieters, R. (1997). *Analyzing visual attention to repeated print advertising using scanpath theory* (Tech. Rep. No. 97B32). Research Institute SOM (Systems, Organizations and Management), University of Groningen.

- Sakamoto, Y., Ishiguro, M., & Kitagawa, G. (1986). *Akaike information criterion statistics*. Tokyo: KTK Scientific Publishers.
- Salvucci, D. D. (1999). Inferring intent in eye-movement interfaces: Tracing user actions with process models. In *Human factors in computing systems: CHI '99 conference proceedings* (pp. 254-261). New York: ACM Press.
- Salvucci, D. D., & Anderson, J. R. (1998). Tracing eye movement protocols with cognitive process models. In M. Gernsbacher & S. Derry (Eds.), *Proceedings of the twentieth annual conference of the cognitive science society* (pp. 923-928). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Saul, L. K., & Jordan, M. I. (1999). Mixed memory Markov models. *Machine Learning*, 37, 75-87.
- Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2), 461-464.
- Scinto, L. F. M., & Barnette, B. D. (1986). An algorithm for determining clusters, pairs or singletons in eye-movement scan-path records. *Behavior Research Methods, Instruments & Computers*, 18(1), 41-44.
- Sfard, A., & Linchevski, L. (1994). The gains and the pitfalls of reification - the case of algebra. *Educational Studies in Mathematics*, 26, 191-228.
- Shaw, A. C. (1969). A formal picture description scheme as a basis for picture processing systems. *Information and Control*, 14, 9-52.
- Stark, L., & Ellis, S. R. (1981). Scanpaths revisited: Cognitive models direct active looking. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye movements: Cognition and visual perception* (pp. 193-226). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Stark, L. W., Ezumi, K., Nguyen, T., Paul, R., Tharp, G., & Yamashita, H. I. (1992).

- Visual search in virtual environments. *Proceedings of the SPIE: Human Vision, Visual Processing, and Digital Display III*, 1666, 577-589.
- Steinke, T. R. (1987). Eye movement studies in cartography and related fields. *Cartographica*, 24(2), 40-73. (Studies in Cartography, Monograph 37)
- Stevens, R. D., Brewster, S. A., Wright, P. C., & Edwards, A. D. N. (1994). Design and evaluation of an auditory glance at algebra for blind readers. In G. Kramer (Ed.), *Auditory display: The proceedings of the second international conference on auditory display*. Addison-Wesley.
- Stone, M. (1974). Cross-validators choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2), 111-147.
- Suppes, P., Cohen, M., Laddaga, R., Anliker, J., & Floyd, R. (1983). A procedural theory of eye movements in doing arithmetic. *Journal of Mathematical Psychology*, 27, 341-369.
- Tortora, G. (1990). Structure and interpretation of visual languages. In S.-K. Chang (Ed.), *Visual languages and visual programming*. New York: Plenum Press.
- Tovée, M. J. (1996). *An introduction to the visual system*. Cambridge University Press.
- Treisman, A. (1987). Properties, parts and objects. In K. R. Boff, L. Kaufman, & F. P. Thomas (Eds.), *Handbook of perception and human performance*. New York: Wiley.
- Trueswell, J. C., Tanenhaus, M. K., & Garnsey, S. M. (1994). Semantic influences on parsing: Use of thematic role information in syntactic disambiguation. *Journal of Memory and Language*, 33, 285-318.
- Tuijl, H. F. J. M. van. (1980). Perceptual interpretation of complex line patterns. *Journal of Experimental Psychology: Human Perception & Performance*, 6(2), 197-221.
- Ummelen, N. (1997). *Procedural and declarative information in software manuals*. Unpub-

- lished doctoral dissertation, Universiteit Twente, Utrecht. (Studies in Language and Communication 7)
- Wertheimer, M. (1923). Principles of perceptual organization. In D. S. Beardslee & M. Wertheimer (Eds.), *Readings in perception* (pp. 115-137). Princeton, New Jersey: Van Nostrand-Reinhold.
- Wheeler, D. D. (1970). Processes in word recognition. *Cognitive Psychology*, 1, 59-85.
- Winer, B. J. (1962). *Statistical principles in experimental design*. New York: McGraw-Hill.
- Wouterlood, D., & Boselie, F. (1992). A good-continuation model of some occlusion phenomena. *Psychological Research*, 54(4), 267-277.
- Yarbus, A. L. (1967). *Eye movements and vision*. New York: Plenum Press.