

**Statistical Language Modelling and  
Novel Parsing Techniques for  
Enhanced Creation and Editing of  
Mathematical E-Content  
Using Spoken Input**



**Dilaksha Rajiv ATTANAYAKE**

A thesis submitted in partial fulfilment of the requirements of  
Kingston University for the degree of  
Doctor of Philosophy  
in Computer Science  
2014

## Acknowledgements

Many people helped this thesis along the way. First, I am grateful to my supervisors, Dr Eckhard Pflugel, Dr Gordon Hunter and Dr James Denholm-Price, for their supervision, valuable time and knowledge in the process of the study. Kingston University has constantly encouraged my studies by offering relevant training sessions and bursaries which supplemented well to this PhD.

During my time at the university, I was fortunate to meet wonderful people who helped me in many ways. I would like to thank Professor Andy Augousti, Dr Vincent Lau, Dr Souheil Khaddaj, Dr Jarinee Chattratchart, Dr Martin Colbert, and Alan Petty for their valuable insights and involvement at certain stages of the research.

I also like to thank Stuart Lunn and Susan Matthews for their encouragement and support during past few years. Special thanks to my sister Umanga and her family, my brother-in-law Naveendra for their kind support when I needed at times.

Throughout this project, I have been supported by some important people in my life and a special gratitude should go towards them, my loving wife Yasanthika, who stood by my side in all gains and losses, my mother Shelia, who actually inspired me to pursue a PhD, and Ananda and Thusitha Weerakoon, my in-laws who also financially supported my studies throughout the process. Hence, I dedicate this thesis to them.

## Abstract

The work described in this thesis aims at facilitating the design and implementation of web-based editors, driven by speech or natural language input, with a focus on editing mathematics.

First, a taxonomy for system architectures of speech-based applications is given. This classification is based on the location of the speech recognition, the speech, and application logic and the resulting flow of data between client and server components. This contribution extends existing system architecture approaches to take into account the characteristics of speech-based systems.

We then show, using statistical language modelling techniques, that mathematics, either spoken or typed, is more predictable than everyday natural languages. We illustrate how these models, in combination with error correction algorithms, can be used to successfully assist the process of creating mathematical expressions within electronic documents using speech. We have successfully implemented systems to demonstrate our findings, which have also been evaluated using standard language modelling evaluation techniques. This work is novel as applying statistical language models to the recognition of spoken mathematics has not been evaluated to this extent prior to our work.

We create a parsing framework for spoken mathematics, based on mixfix operators, operator precedences and non-deterministic parsing techniques. This framework can significantly improve the design and parsing of spoken command languages such as spoken mathematics. A novel robust error recovery method for an adaptation of the XGLR parsing approach to our operator precedence setting is presented. This greatly enhances the range of spoken or typed mathematics that can be parsed. The novel parsing

framework, algorithms and error recovery that we have designed are suitable for more general structured spoken command languages, as well.

The algorithms devised in this thesis have been implemented and integrated in a research prototype system called TalkMaths. We evaluate our contributions to the new version of this system by comparing the power of our parser with that contained in previous versions, and by conducting a field study where students engage with our system in a real classroom-based environment. We show that using TalkMaths, rather than a conventional mathematics editor, had a positive impact on the learning and understanding of mathematical concepts of the participants.



## Publications by Author

Some of the work described in this thesis are based on peer-reviewed journals/book chapters or conference papers listed below.

- D. R. Attanayake, G. J. A. Hunter, E. Pfluegel, J. C. W. Denholm-Price, “Using statistical language models and edit distance metrics for prediction and error correction in a novel interface for mathematical text”, U.K. Speech Conference, September 17-18, Cambridge, U.K. (2013)
- D. R. Attanayake, G. J. A. Hunter, E. Pfluegel, J. C. W. Denholm-Price, “Novel Multi-Modal Tools to Enhance Disabled and Distance Learners Experience of Mathematics” in International Journal on Advances in ICT for Emerging Regions (ICTER), 6(1), March, IEEE Computer Society (Asia Region). (2013)
- D. R. Attanayake, G. J. A. Hunter, J. C. W. Denholm-Price, E. Pfluegel, “Interactive error correction using statistical language models in a client-server interface for editing mathematical text” in Designing Inclusive Systems: Designing Inclusion for Real-world Applications, Edited by Langdon, P. et. al., Springer, pp. 125-132. ISBN 1447128664. (2012)
- D. R. Attanayake, G. J. A. Hunter, J. C. W. Denholm-Price, E. Pfluegel, “A Novel Web-Based Tool to Enhance Learning of Mathematical Concepts”, International Conference on Advances in ICT for Emerging Regions (ICTER), December 13-14, Colombo, Sri Lanka. ISBN 978-1-4673-5527-8. (2012)
- D. R. Attanayake, G. J. A. Hunter, J. C. W. Denholm-Price, E. Pfluegel, “Intelligent Assistive Interfaces for Editing Mathematics”, 1st Workshop on Future Intelligent Educational Environments (WOFIEEE12), Volume 13, IOS Press, June 2012, Guanajuato, Mexico, pp. 286–297. ISBN 978-1-61499-079-6. (2012)

- D. R. Attanayake, G. J. A. Hunter, J. C. W. Denholm-Price, E. Pfluegel, "SWIMS (Speech-based Web Interface for Mathematics using Statistical language models): An intelligent editing assistant for mathematical text", 8th International Conference on Intelligent Environments, IE2012, June 26-29, Guanajuato, Mexico, pp. 327-330. ISBN 978-1-4673-2093-1. (2012)
- D. R. Attanayake, E. Pfluegel, G. J. A. Hunter, J. C. W. Denholm-Price, "Use of a Novel Tool, TalkMaths, to Enhance Students Learning of Mathematical Concepts" in Quality Enhancement in Learning and Teaching: How Kingston University is improving the student experience, Vol. 2, pp 22 - 23, Kingston University Academic Development Centre, Kingston University, London. (2012)
- D. R. Attanayake, E. Pfluegel, J. C. W. Denholm-Price, G. J. A. Hunter, "Architectures for Speech-Based Web Applications", 4th International Conference on Semantic E-business and Enterprise Computing (SEEC2011), July 20-22, UK. (2011)
- D. R. Attanayake, J. C. W. Denholm-Price, G. J. A. Hunter, E. Pfluegel, "Talk-Maths - Developing a Speech User-Interface for Spoken Mathematics" in MSOR Connections, Vol 11(2), MSOR Network. (2011)
- A. M. Wigmore, E. Pfluegel, G. J. A. Hunter, J. C. W. Denholm-Price, M. Colbert, D. R. Attanayake, "Evaluating and improving the TalkMaths speech interface for dictating and editing mathematical text", 5th European Workshop on Mathematical and Scientific E-Contents, 9 Sept - 11 Sept 2010, Salamanca, Spain. DOI <http://fundacion.usal.es/5euworkshop/index.php>. (2010)

# Contents

<b>Contents</b>	<b>vi</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problems Addressed by this Thesis . . . . .	2
1.3 Research Aims and Objectives . . . . .	3
1.4 Contributions . . . . .	4
1.5 Thesis Organisation . . . . .	6
<b>2 Background</b>	<b>7</b>
2.1 Defining Spoken Mathematics . . . . .	8
2.2 Language Models and Algorithms . . . . .	9
2.2.1 Statistical Language Modelling . . . . .	9
2.2.2 Error Correction Algorithms for Text . . . . .	10
2.2.3 Predictive and Corrective Systems using SLMs and Algorithms .	11
2.3 Parsing . . . . .	12
2.3.1 Context-free Parsing Techniques . . . . .	12
2.3.2 Parsing Spoken Mathematics . . . . .	15
2.4 Speech-Driven Editing Systems . . . . .	16
2.4.1 Speech Recognition Technologies . . . . .	16
2.4.2 Speech-Driven Editing Approaches . . . . .	17

---

2.4.3	Speech-Driven Interfaces for Mathematics . . . . .	18
2.5	Other Related Approaches . . . . .	19
2.6	Summary . . . . .	20
<b>3</b>	<b>Architectures for Speech-based Applications</b>	<b>21</b>
3.1	Motivation . . . . .	21
3.2	Relevant Speech Technologies and Tools . . . . .	23
3.2.1	Post-ASR Technologies . . . . .	23
3.3	Classification of Architectures for Speech-based Systems . . . . .	24
3.3.1	Terminology . . . . .	25
3.3.2	ASR and PASRP Both Performed Locally . . . . .	26
3.3.2.1	Desktop Speech-Based (Architecture A) . . . . .	26
3.3.2.2	ASR and PASRP on Client (Architecture B) . . . . .	27
3.3.3	ASR on Client, PASRP on Server . . . . .	28
3.3.3.1	Speech Proxy (Architecture C) . . . . .	28
3.3.3.2	Application Proxy (Architecture D) . . . . .	29
3.3.4	ASR and PASRP on Server . . . . .	30
3.3.4.1	ASR on Same Server (Architecture E) . . . . .	30
3.3.4.2	ASR on Different Server (Architecture F) . . . . .	31
3.3.4.3	Distributed ASR using Same Server (Architecture G) . . . . .	32
3.3.4.4	Distributed ASR using Different Server (Architecture H) . . . . .	32
3.4	Choosing a Suitable Architecture for Speech-Enabled Web Applications	33
3.5	Summary . . . . .	37
<b>4</b>	<b>Theory of Statistical Language Models and Similarity Metrics</b>	<b>38</b>
4.1	Motivation . . . . .	38
4.2	N-grams – A Baseline Approach to Statistical Language Modelling . . . . .	39
4.3	Interpolation between N-grams . . . . .	43
4.4	Cache Models . . . . .	44
4.5	Interpolation between N-grams and Cache Models . . . . .	45
4.6	Evaluating the Performance of SLMs . . . . .	46
4.7	String Edit Distance Metrics and its Usage . . . . .	47
4.8	Applications to Spoken Mathematics . . . . .	48

---

<b>5</b>	<b>SLM Applications to Spoken Mathematics</b>	<b>49</b>
5.1	Introduction . . . . .	49
5.2	Developing and Evaluating SLMs for Spoken Mathematics . . . . .	50
5.2.1	Creation of the Dataset from Web Sources . . . . .	51
5.2.2	Creating Statistical Language Models . . . . .	52
5.2.3	Evaluation Results of Statistical Language Models . . . . .	53
5.3	Applications of SLMs to Prediction of Spoken Mathematics . . . . .	56
5.3.1	The SWIMS Prototype System . . . . .	57
5.3.2	Predictive Mathematics Interface . . . . .	58
5.3.3	Dependence of Prediction Success Rate on Number of Alternatives Offered (Experiments A and B) . . . . .	59
5.3.4	Dependence of Prediction Success Rate on Size of Training Dataset (Experiment C) . . . . .	63
5.4	Applications of Edit Distance Metrics to Error Correction . . . . .	65
5.4.1	How Successful the Correction System at Correcting Errors (Experiment D) . . . . .	65
5.5	Discussion of Results . . . . .	67
5.6	Summary . . . . .	69
<b>6</b>	<b>Error Recovery Strategies for Parsing Transcribed Spoken Mathematics</b>	<b>70</b>
6.1	Introduction . . . . .	70
6.2	Spoken Mathematics as Input Language . . . . .	72
6.2.1	Ambiguity . . . . .	72
6.2.2	Speech Templates . . . . .	73
6.3	Parsing Challenges . . . . .	74
6.3.1	Lexical Ambiguity . . . . .	75
6.3.2	Incomplete Input . . . . .	75
6.3.3	Incorrect Input . . . . .	76
6.4	Parsing Framework . . . . .	77
6.4.1	Underlying Grammar . . . . .	77
6.4.2	Operator Precedence Parsing . . . . .	80

---

6.5	Dealing with Lexical Ambiguity . . . . .	80
6.5.1	XGLR Approach . . . . .	82
6.5.2	Lexer Algorithm . . . . .	83
6.5.3	Parser Algorithm . . . . .	84
6.6	Error Recovery Strategies for XGLR Parsing . . . . .	84
6.6.1	Classification of Errors . . . . .	85
6.6.2	Main Idea . . . . .	86
6.6.3	Error Recovery at Lexer Level . . . . .	86
6.6.4	Error Recovery at Parser Level: Stage 1 . . . . .	88
6.6.4.1	The Parser Algorithm . . . . .	90
6.6.5	Error Recovery at Parser Level: Stage 2 . . . . .	90
6.6.5.1	Sort Argument Separator Parts . . . . .	92
6.6.5.2	Detangle Argument Separator Parts . . . . .	92
6.6.5.3	Complete Argument Separator Parts . . . . .	93
6.6.5.4	Eliminate Redundant Argument Separator Parts . . . . .	93
6.6.6	Error Recovery – Summary . . . . .	93
6.7	Summary . . . . .	94
<b>7</b>	<b>System Implementation and Evaluation</b>	<b>96</b>
7.1	Introduction . . . . .	96
7.2	The TalkMaths System . . . . .	96
7.2.1	System Architecture . . . . .	98
7.2.2	Natural Language Commands . . . . .	100
7.2.3	Editing Paradigms . . . . .	103
7.2.4	Natural Language Search-Driven Help Facility . . . . .	106
7.3	Parser Evaluation . . . . .	106
7.4	TalkMaths Field Study . . . . .	110
7.4.1	Experiment: The Learning Activities . . . . .	110
7.4.2	Design of Experiment . . . . .	111
7.4.3	Undertaking The Learning Activities . . . . .	112
7.4.4	Evaluation . . . . .	112

---

<b>8</b>	<b>Conclusions and Future Work</b>	<b>116</b>
8.1	Conclusions . . . . .	116
8.2	Future work . . . . .	119
	<b>References</b>	<b>124</b>
<b>A</b>	<b>SLM Training Data Comparisons</b>	<b>142</b>
<b>B</b>	<b>LaTeX to Spoken Mathematics Yapps2 Parser</b>	<b>147</b>
<b>C</b>	<b>Sample Parser Programming Code</b>	<b>156</b>
<b>D</b>	<b>TalkMaths Interface Screenshots</b>	<b>168</b>
<b>E</b>	<b>TalkMaths Field Study Material</b>	<b>177</b>

# List of Figures

3.1	Desktop Speech-Enabled Application Architecture (A) . . . . .	27
3.2	ASR and PASRP on Client Architecture (B) . . . . .	28
3.3	Speech Proxy Architecture (C) . . . . .	29
3.4	Application Proxy Architecture (D) . . . . .	30
3.5	ASR on Same Server Architecture (E) . . . . .	31
3.6	ASR on Different Server Architecture (F) . . . . .	32
3.7	Distributed ASR using Same Server Architecture (G) . . . . .	33
3.8	Distributed ASR using Different Server Architecture (H) . . . . .	34
5.1	Typical usage of CMU Toolkit [Clarkson and Rosenfeld, 1997] . . . . .	55
5.2	Building & Evaluating SLMs for Spoken Mathematics for Tutorial Web Site Data . . . . .	56
5.3	SWIMS Predictive Mathematics Interface System Architecture where “SM” refers to “Spoken Mathematics Format” . . . . .	58
5.4	Predictive Mathematics Interface in use. In the top-ranked suggestion, the SLM correctly predicts that <i>capital</i> will be followed by <i>charlie</i> . . .	59
5.5	One word ahead prediction success rate increasing with the number of suggestions offered to the user. The different points corresponding to the same number of suggestions indicate the results from the 10 different cross validation trials . . . . .	62
5.6	Two word ahead prediction success rate increasing with the number of suggestions offered to the user . . . . .	63
5.7	SWIMS Alternative/Corrective Mathematics System Architecture . . .	66
5.8	Alternative/Corrective Mathematics Interface in use. In the top-ranked suggestion, the OOV word <i>pluck</i> is replaced with <i>plus</i> . . . . .	67



---

7.1	TalkMaths system architecture . . . . .	99
7.2	TalkMaths used to create the formula for the sum of $n$ terms of an arithmetic progression . . . . .	101
7.3	Encoding of same mathematical expression in LaTeX, MathML, TalkMaths and SWIMS command languages respectively . . . . .	102
7.4	The van der Waals equation, read as stated above, rendered by TalkMaths104	
7.5	Different editing paradigms for editing mathematics by speech, each applied to one of equations of uniformly accelerated motion . . . . .	105
7.6	The TalkMaths Natural Language Driven Help Facility used to search to find information on fractions . . . . .	107
7.7	The ease of use of the system by the two groups . . . . .	113
7.8	Improved understanding . . . . .	114

# List of Tables

3.1	Architecture Classification for Speech-based Systems . . . . .	35
4.1	Change of 5-word cache content (marked in bold) over time . . . . .	45
5.1	Sample expressions from our spoken mathematics corpus. The names of individual letters are dictated using the NATO pronunciation alphabet ( <i>alpha, bravo, charlie, delta,...</i> ) with <i>x-ray</i> representing the letter <i>x</i> . . .	53
5.2	The most frequent words (unigrams) in our <i>spoken mathematical expressions</i> as percentages of all the words in the corpus. <i>x-ray</i> is the spoken form of the symbol <i>x</i> . . . . .	54
5.3	Most frequent bigrams and trigrams in our dataset . . . . .	54
5.4	Cross-validation Perplexity Calculations on Statistical Language Models of 3,194 spoken mathematical expressions using CMU Language Modelling toolkit . . . . .	57
5.5	Sample incomplete expressions we used for Experiment A, omitted word and predictions (using 5 suggestions, adding one word for each) . . . .	60
5.6	Sample incomplete expressions we used for Experiment B, omitted words and predictions (using 5 suggestions, adding two words for each) . . . .	61
5.7	Experiment A: Variation of success rates of one word prediction with number of suggestions offered to the user . . . . .	62
5.8	Experiment B: Variation of success rates of two word prediction with number of suggestions offered to the user . . . . .	63
5.9	Experiment C: Variation of success rate of one word ahead prediction with SLM size (5 suggestions per trial) – both training and test sets may vary in size . . . . .	64

5.10	Experiment C(2): Variation of success rate of one word ahead prediction with SLM size (5 suggestions per trial) keeping test set constant (400) . . . . .	64
5.11	Experiment D: Variation of success rate (%) of correction using Damerau-Levenshtein method (5 suggestions offered per trial) . . . . .	66
5.12	Experiment D: Examples of incorrect expressions derived from the same correct expression “yankee equals capital alpha x-ray to the power of begin minus three end” (by deleting, inserting and swapping characters). Words with errors are marked in bold font . . . . .	68
6.1	Examples of speech templates with different number of arguments . . . . .	74
6.2	Operator Precedence Table . . . . .	79
6.3	Additional Precedence Tables for Mixfix Operators . . . . .	79
6.4	Symbol table for spoken command “a times fraction b plus c over d end fraction”. The right most column contains additional information about each entries (for example, if the entry is the type of <i>op</i> , all possible operator classes it could take: i.e. $\star, \circ, \bullet$ ) . . . . .	82
6.5	Token Insertion Rules . . . . .	87
6.6	Lexer Error Recovery Examples . . . . .	88
6.7	Adjusted Operator Precedence Table . . . . .	89
6.8	Precedence Tables for Interlaced Mixfix Operators . . . . .	90
6.9	Summary of Error Recovery . . . . .	94
7.1	Sample results of parsing spoken mathematics corpus (complete expressions) using Yapps2-based parser ( $P_1$ ) presented by [Wigmore, 2011] , the Yapps2-based parser in our live TalkMaths web-based system ( $P_2$ ) and operator precedence-based parser ( $P_3$ ) . . . . .	109
7.2	Results of the post-task feedback on the ease of use of the tool . . . . .	113
7.3	Results of the post-task feedback on any improvements to understanding mathematical concepts . . . . .	115
A.1	Comparison of Overall words & most common words . . . . .	143
A.2	Most Common Bigrams . . . . .	144
A.3	Most Common Trigrams . . . . .	145
A.4	Perplexity scores as function of training set size . . . . .	146

# Chapter 1

## Introduction

### 1.1 Motivation

Until relatively recently, automatic speech recognition was only available to a small group of highly specialised scientists. Even around twenty five years ago, in order to use what was then considered state-of-the-art speech recognition software, highly expensive equipment was required. Today, even hand-held devices such as smartphones or tablet PCs and game consoles such as the Xbox and Nintendo Wii are often equipped with some form of speech recognition software. The reasons behind this rapid evolution are the tremendous progress that has been made in speech recognition technology, the advances in available memory and processing power of personal computers and the fact that complex and computationally intensive tasks can now be carried out with the help of powerful cloud-based applications.

At present, speech recognition is used in a large number of application domains. As a consequence, speech input plays an increasingly important role in making computer tasks accessible to users who wish or need to rely on input modalities other than conventional keyboard and mouse. This includes, for example, online (distance) learners, people working or studying “on the move”, relying on mobile devices, and disadvantaged users such as people with physical disabilities or other special needs.

Most speech recognition systems tend to focus on everyday natural language input in one of the “mainstream” languages such as English, Spanish or Japanese, including typical dialects of them. Often, these systems are not well-suited for recognising “arti-

ficial” languages such as formal languages for speaking mathematics or programming code. These languages are highly specialised and only used by a small community, and that might be why formal languages have been given less attention by research and industry.

## 1.2 Problems Addressed by this Thesis

The work described in this thesis, motivated by an enthusiasm for widening the accessibility of mathematics through speech input, addresses the following problems:

Editing mathematical text, which can already be a difficult and potentially error-prone process when using keyboard and mouse together with specialised mathematics editors, is particularly challenging for users who wish or need to engage in hands-free computing, relying exclusively on speech recognition.

Powerful speech recognition tools and products exist on the market, and these can be installed and used together with standard desktop applications, including reasonably user-friendly mathematics editors such as Microsoft Equation Editor. However, as we have found, this does not result in a satisfactory solution to the above problem.

Despite several documented and published attempts to remedy this situation [Elliott and Bilmes, 2007; Gould, 2001; Guy et al., 2004; Metroplex Voice Computing, Inc], we have found that the resulting outcomes are not very satisfactory. We have examined most of the systems that have been suggested in the literature, and came to the conclusion that they were not usable for the purposes we had in mind.

After some further investigation, and having reviewed the state-of-the-art of several research strands that we deemed relevant in the context of our overarching research problems, we identified the following problems:

Statistical Language Models (SLMs) are a widely adopted technique for implementing automatic speech recognition, and it appeared that no-one had evaluated sophisticated language models for spoken mathematics, covering a reasonably broad range of mathematics. In particular, we could not find evidence of any SLMs being used for improving the accuracy of recognition for this type of speech input languages. We are not aware of any system or mathematical editor that benefits from SLMs for this purpose.

Even if the spoken mathematics were recognised accurately, parsing the resulting

expressions at symbol level may pose significant problems. We have found specialist parsing techniques dealing with ambiguity for a related input language (spoken computer programming code) [Begel, 2005], but not for mathematics. Overall, the problem of error recovery as it arises in our situation did not seem to be addressed sufficiently in the literature, as standard techniques from compiler construction are not suitable.

The version of TalkMaths that was available prior to our work was limited in the amount of mathematics that could be created, and expressions were not always correctly parsed. Only the most basic of the mechanisms for editing mathematical expressions using speech within a GUI (Graphical User Interface) environment, as presented in [Wigmore, 2011], was implemented. Furthermore, this editing facility was poorly integrated in the system which made its use very difficult.

As a consequence, the TalkMaths system had only been used by a small number of individuals where it was difficult to gauge its usefulness and in a number of usability studies where the participants were helped by an expert. No use of TalkMaths in the classroom, for students without previous experience in using the system, had been undertaken.

### 1.3 Research Aims and Objectives

The work described in this thesis aims at facilitating the design and implementation of web-based editors, driven by speech or natural language input, with a focus on mathematical editing interfaces. It hence directly addresses the problems documented in the previous section by tackling the following research objectives:

1. To investigate and evaluate the potential benefit which statistical language models could provide, in the context of creating and editing mathematical text, for the prediction of future input and for the correction of errors.
2. To implement a predictive and corrective facility for speech-based mathematical editors using our statistical language models and an “edit distance” similarity metric.
3. To create a framework for defining and parsing spoken mathematics, based on mixfix operators, operator precedences and non-deterministic parsing techniques.

4. To recover from incomplete and incorrect spoken mathematics input, by devising novel error recovery strategies for the operator precedence parser.
5. To implement the novel parsing framework in the TalkMaths system and to evaluate the resulting enhancements.
6. To study the benefits that using the new version of TalkMaths offers to the understanding of mathematical concepts.

## 1.4 Contributions

This thesis contains several contributions to knowledge in a range of research domains. In this subsection, we motivate, describe and justify our contributions.

First, we start by giving a taxonomy for system architectures of speech-based applications. We classify a range of architectures by considering the location of the speech recognition, the speech and application logic, and the resulting flow of data between client and server components. We believe that our classification will be helpful for anyone needing to make a design choice for a planned speech-based systems, for example, based on frameworks such as [Gruenstein et al., 2008; Lau et al., 1997]. Also, our findings led us to fundamentally change the architecture of the TalkMaths system to a web-based one: prior to this work, TalkMaths was a stand-alone desktop-based application.

We then show, using statistical language modelling techniques, that mathematics, either spoken or typed, is more predictable than everyday natural language. We illustrate how these models, in combination with error correction algorithms, can be used to successfully assist the process of creating mathematical expressions within electronic documents using speech. Whilst some previous authors have used statistical language models of mathematical expressions, with the exception of the earlier work of [Wigmore, 2011] and [Wigmore et al., 2009b], we are only aware of these models being applied to the conversion of hand-written mathematical equations into electronic format using optical character recognition [Smirnova and Watt, 2008; Suzuki et al., 2009]. Hence, this present work applying statistical language models to the recognition of spoken mathematics is a novel contribution.

Our next contributions are concerned with the design and parsing of spoken mathematics, although our approach would also be valid for more general structured spoken command languages. Our starting point is a design framework using mixfix operators, leading to the construction of commands that we refer to as *speech templates*. This can significantly enhance the ease of design and maintenance of any spoken command language, and in particular, our proposed language for spoken mathematics improves upon previous attempts [Chang, 1983; Fateman, 2006, 2009], including that in earlier versions of the TalkMaths system.

The approach of speech templates greatly improves upon the parsing of the spoken input in the previous version of the TalkMaths system by designing the input language using an operator precedence grammar, which in turn is much simpler to parse while recovering from potential errors. To our knowledge, at the time of writing this thesis, considering operator grammars in the context of spoken command languages (and in particular, for spoken mathematics) is a novel approach.

We then devise a parsing method for our proposed language, based on operator precedences and non-deterministic parsing techniques. Our approach is inspired by work on spoken programming languages [Begel, 2005] and our main novel contribution here is an error recovery method for an adaptation of the XGLR [Begel and Graham, 2006] parsing approach to our operator precedence setting. As a consequence, the range of mathematics that can be parsed by TalkMaths has been significantly enhanced and, for the first time, our system can tolerate errors that might typically arise from the input of spoken or typed mathematics.

Finally, we evaluate our new version of TalkMaths by conducting some practical experiments with students using our system. Our implementation of the editing paradigms for spoken mathematics presented in [Wigmore, 2011] provided for the first time a persistent editing facility which subjects were able to use within a mathematics teaching and learning session. We show that the use of TalkMaths, compared to that of a conventional mathematical editor, had a positive impact on the learning and understanding of mathematical concepts of the participants.



## 1.5 Thesis Organisation

The remainder of this thesis is organised as follows: Chapter 2 reviews the related work that has been published in the literature. In Chapter 3, we present our classification of architectures for speech-based applications. The next chapter (Chapter 4) introduces the theory of statistical language models and additionally discusses similarity metrics. Moving to Chapter 5, we use statistical language models and “edit distance” metrics in order to improve recognition of spoken mathematics. Chapter 6 contains an important contribution of this thesis, a parsing framework for ambiguous spoken mathematics together with robust error-correction strategies. In Chapter 7, we give an overview of the TalkMaths system and explain the improvements in architecture and functionality that our research contributions have made. We then provide an evaluation of our parser implementation and finally present a field study demonstrating the benefits that using TalkMaths can have on the learning and understanding of mathematical concepts. In the conclusion of this thesis we critically reflect on our work and give some future directions. Finally, we include an appendix comprising a comparison of statistical language modelling training data, sample programming code from our implementation of the parser and other related tools, a selection of TalkMaths screenshots and some material from the TalkMaths field study.

# Chapter 2

## Background

The aim of this chapter is to provide background information for the reader in order to become more familiar with the main topical areas relevant to this thesis.

As noted in the introduction, our proposed system for creating and editing mathematical text has the following modules: an engine to implement the speech recognition (the ASR – we will be using the term “ASR” to denote “Automatic Speech Recogniser” and “Automatic Speech Recognising” as appropriate), spoken mathematical input, a statistical language model, an error correcting algorithm and a parser with error recovery. This chapter outlines work that has been published in the literature in each of these areas.

We will start with reviewing attempts to define how to unambiguously speak mathematical formulae, followed by a discussion of statistical language models and related algorithms for prediction and correction of text which can include transcribed versions of spoken mathematics.

We will then present an overview of different parsing techniques, leading to more specialised parsing of spoken mathematics. Subsequently, we discuss speech-driven editing systems by reviewing commercial and freely available tools, approaches for editing paradigms that use speech and finally speech-driven interfaces for mathematics. To conclude, we mention work that relates to some of the areas in which we have been carrying out research in.

## 2.1 Defining Spoken Mathematics

Defining a standard for spoken mathematics is an essential starting point when creating speech-based mathematical editing systems. In this section, we will review several approaches for this that have been taken in the past.

The question of how to define rules for speaking mathematics has been asked by several authors in the past. The motivation behind these attempts to find standards seems to be linked to three different contexts: dictating or describing mathematics to other human beings, parsing mathematics by computer systems in order to further process the input and defining rules for converting mathematical content to audio synthetic speech output (text-to-speech, TTS).

One of the first attempts to give a standard for speaking mathematical equations and objects in English appears to be Chang's handbook "Larry's Speakeasy" [Chang, 1983]. He defines spoken forms for a broad range of mathematics, from basic symbols, algebra, trigonometry, logic, geometry, statistics, calculus, linear algebra, topology to mathematical diagrams and graphs. Chang, himself being a blind mathematician, primarily focuses on dictating mathematics to other human beings.

As far as we are aware, the most complete investigation into spoken mathematics to date is presented in [Fateman, 2006, 2009]. He focuses on introducing a vocabulary that is intuitive and easy to learn by novices, and at the same time allows as little ambiguity as possible. This work was carried out by Fateman within the context of the Math Speak & Write system [Guy et al., 2004]. He provided a detailed analysis of how to speak numbers, non-numeric tokens, nested arithmetic expressions, integrals and sums. He also discusses the problem of ambiguity, to which in some cases no easy solution seems to be available.

Apparently unaware of this work, [Elliott and Bilmes, 2007] developed a similar approach, although their language design is oriented towards the use of "two-dimensional mathematics" in combination with an existing mathematics editor (*Scientific Notebook*).

In [Wigmore, 2011], spoken mathematics is also investigated, based on empirical evidence of how people – notably mathematical students and teachers – actually speak mathematical expressions. This was carried out by analysing transcriptions of recorded mathematics classes from the British National Corpus (BNC) [Burnard, 1995] and

an experiment where participants read out given expressions. The study focusses on the potential of prosodic information providing clues in order to resolve ambiguity. However, this did not influence the design of the language used in older versions of TalkMaths and Wigmore uses an approach very similar to that of Fateman.

Rules for spoken mathematics have also been developed for text-to-speech conversion of mathematics. For example, Raman, another blind mathematician, gave a framework in his AsTeR system [Raman, 1998], which synthesised LaTeX/TeX documents for blind users. Another initiative is the MathSpeak project [Schleppenbach, 2013] in which a set of grammar rules for speaking mathematics have been designed. However, rules for speaking mathematics that have been developed for text-to-speech appear less suitable for spoken input due to the tedious learning curve and usage. In other words, a text-to-speech system would use a more descriptive language with a much larger vocabulary to give the listener as much information as possible to explain the expression on screen which might be too long and complex for a user to dictate to insert a relatively smaller formulae [Fateman, 2009].

It should be mentioned that some commercial systems are gradually supporting spoken mathematics. The most popular example is the Computer Algebra System *Mathematica*, that actually provides a function `spokenString()` [Mathematica, 2014] for converting mathematical objects into their spoken language form.

## 2.2 Language Models and Algorithms

One of the primary goals of the work described in this thesis is to provide assistance to users of speech-driven mathematics editors, through predictive and corrective facilities. In this section we first explore the background of a particular technique for modelling language and algorithms which are useful in realising such facilities. Finally, we review both commercial and research systems that take advantage of those language models and algorithms.

### 2.2.1 Statistical Language Modelling

A language model can be described as an attempt to capture the properties of a language in order to be able to predict the next word(s) of a given sentence of that

language for the purpose of syntactic or semantic analysis, or to carry out corrections if it is malformed or otherwise incorrect. A variety of different approaches to language modelling exist in the literature, including those based on grammatical rules, statistics, or neural networks. Among such language models, Statistical Language Models (SLMs) [Rosenfeld, 2000] have been at the core of ASR systems for many years [Young, 1996, 2002]. These use statistics from *past experience* to predict the likelihood of what will be spoken next and combine this with evidence from the acoustic signal of the speech to decide what words were actually said.

The simplest types of SLMs are  $N$ -gram models, which use statistics of the occurrences of specific sequences of  $N$  consecutive words within a database (or *corpus*) of training material observed in the past. A more dynamic or adaptive approach is to use a cache model [Clarkson, 1999b], where a cache or buffer, of recently-occurred words is used to update the baseline  $N$ -gram models. Variants of these models have been successfully applied to various domains by other authors [Martins et al., 2008; Vaičiūnas and Raškinis, 2006]. We will explain the theoretical aspects of  $N$ -gram models and cache models in Chapter 4.

### 2.2.2 Error Correction Algorithms for Text

It has been noted that the majority of human typing and spelling errors are quite minor [Damerau, 1964; Pollock and Zamora, 1983], often involving just the omission or addition of a single character, typing two characters in the wrong order, or accidentally substituting one character for another (often one adjacent to the correct symbol on the keyboard [Grudin, 1983]). Correcting errors in text has been a widely researched area. There are three steps in correcting words in text: detecting non-words, isolated-word and context-dependent word error correction [Kukich, 1992]. In our setting, we only focus on the first two types.

The most researched technique of correcting isolated-word errors in text is the *minimum edit distance* [Wagner, 1974]. One of the pioneering approaches that use this technique, the Damerau-Levenshtein distance [Damerau, 1964; Levenshtein, 1966] between two character strings, measures how different those strings are by taking into account the minimum numbers of insertions, deletions, substitutions and transpositions of adjacent characters required to transform one of the strings into the other. [Wagner

and Fischer, 1974; Wagner and Lowrance, 1975] also introduced more computationally efficient spelling correction by using dynamic programming techniques [Nemhauser, 1966]. Some further extensions, such as allowing the exchange of non-adjacent characters given by [Wagner and Lowrance, 1975]. While the Damerau-Levenshtein distance method has been widely-used, research is being still carried out in this area – for example, [Shah et al., 2012] proposed a hybrid approach and [Lu et al., 2013] suggested a synonym-based approach. Although one of the original motivations for the development of this metric was to compare the similarity of short pieces of natural language text, it has also been applied in fields such as genetics, for example to study how similar two fragments of DNA are to each other [Troncoso-Pastoriza et al., 2007]. It has also been recently used in graph-matching [Cao et al., 2013], which is essential aspect to many graph searching, pattern recognition and machine vision tasks. It should also be noted that other similar metrics have been developed in the past, such as “direct threshold matching” described by Glantz [1957] in which the differences of two strings are matched in a position-for-position manner, but these are not as widely used as the Damerau-Levenshtein distance method.

### 2.2.3 Predictive and Corrective Systems using SLMs and Algorithms

More recently, SLMs have been incorporated into innovative systems for automatic translation between languages, such as Google Translate [Google, 2012]. A wide variety of existing technological systems employ prediction and/or correction methodologies in an attempt to make them more useful and usable. These include automatic (or semi-automatic) correction systems found in word processors and internet search engines (*Showing results for ... Search instead for...*) and the prediction systems used in ASR systems and SMS text message editors on mobile telephones. Although manufacturers of commercial products rarely reveal exactly their secrets, it is understood that correction systems look for *close matches* to what was entered within a database of common words or phrases, whilst prediction systems use statistical models. These models give probabilities of words and word sequences, using information from a large set of previously observed data and evidence from the current situation, together with an *inference rule*, such as a Bayesian framework, in order to combine information from

more than one source [Young, 1996].

## 2.3 Parsing

Spoken mathematics that has either been recognised by the ASR, or perhaps directly typed into a suitable user interface, needs to be parsed in order to lead to a meaningful action within the system. The classification of formal languages into recursively enumerable (Type-0), context-sensitive (Type-1), context-free (Type-2) and regular (Type-3) has been given in [Chomsky, 1957]. However, it is commonly assumed that parsing general natural languages is a “hard” problem – indeed, some authors [Kallmeyer, 2010] have noted that some natural languages, such as Dutch [Bresnan et al., 1987] and Swiss-German [Shieber, 1988], contain *cross-serial dependencies* which cannot be adequately modeled using context-free grammars. Thus natural languages are not in general Type-2 languages [van de Koot, 2013]. Hence for parsing our input, we will have to restrict ourselves to an appropriate sub-language, namely our standard of spoken mathematics, which has a well-defined vocabulary and relatively prescribed syntax. All of the approaches for spoken mathematics as discussed in Section 2.1, are based on context-free languages.

In this section, we will outline different key techniques for parsing context-free languages, followed by a discussion on suitable frameworks for parsing spoken mathematics. Note that this is considerably more difficult than the opposite task, the conversion of mathematical content into a spoken language representation, which can be carried out following one of the approaches mentioned in Section 2.1.

### 2.3.1 Context-free Parsing Techniques

Context-free parsing has been investigated in great detail in the past, mostly because of its use in computer science: most systems and compilers use context-free parsing. Even though any programming language that for example requires the declaration of variables (e.g., C or C++) is effectively context-sensitive, a multi-stage approach that is based on an initial context-free analysis, is generally assumed to be most effective.

Amongst the context-free languages, research has further identified several subclasses which are characterized by their ease of parsing, depending on the order in

which the resulting parse tree data structure is built (top-down or bottom-up) [Aho et al., 1986]. The class of  $LL(k)$  [Lewis II and Stearns, 1968] parsers proceeds top-down, usually following a recursive descent algorithm. Parsers written in this approach tend to be manually created and are easier to read for human beings.  $LR(k)$  parsers [Knuth, 1965] are driven by parse tables, constructing the parse tree bottom-up by following a stack-based shift-reduce scheme. They are fast and particularly suitable for automatic code generation. The parameter  $k$  is the value of the *look-ahead*, which indicates how many tokens have to be processed before the parser can decide which parsing action to take. Traditionally,  $k = 1$  was assumed to be sufficient.  $LALR(k)$  [DeRemer, 1969] languages are a subset of  $LR(k)$  which are particularly efficient to parse and are used for the design of popular modern programming languages. Specialist techniques exist [Pager, 1977; Pager and Chen, 2008] that can speed up general LR parsing, yielding parsers that are similarly efficient as LALR parsers [Sorkin and Donovan, 2011]. The class of Operator Precedence (OPrec) parsers, which is of particular relevance to the work in this thesis, is an attractive alternative as it is fast and results in a simple parser but is only applicable within a restricted domain.

GLR parsing is an extension of LR parsing to handle ambiguous languages (originally aiming at natural languages [Tomita, 1985]). The main technique is to use back-tracking of non-deterministic rules, in a breadth-first search manner. [Begel and Graham, 2006] in turn extends GLR parsing to handle lexical ambiguities arising from spoken input and embedded languages. The resulting parsing technique is referred to as XGLR in the original paper. In this thesis, we will see how to combine OPrec and GLR parsing techniques in order to obtain a suitable framework for the spoken languages we have in mind, which is an alternative to XGLR parsing.

Writing a parser can be a complex task and there are two different ways of designing and implementing parsing algorithms: “by hand”, i.e. coding manually appropriate functionality, or using a parser generator. As we have mentioned already in the previous paragraph, most automatically generated parsers are LR and LALR. One of the earliest tools was YACC [Johnson, 1975], which inspired the release of the Open Source tool Bison [Donnelly and Richard, 1998]. Both create native C code which, once compiled, results in efficient LALR parser applications. The LR parser generation system [Wetherell and Shannon, 1981] and various more recent tools [Chauveau and Bodin, 1998; Chen and Pager, 2008] are all essentially based on Pager’s algorithm



[Pager, 1977]. An interim version of the spoken mathematics parser included in the TalkMaths system used Yapps2 [Patel, 2009], a little used parser generator which produces recursive descent (LL) parsers in the scripting language Python. ANTLR [Parr and Fisher, 2012] is a modern tool (although in development since the late 80s) and is an exception as it has introduced efficient  $LL(\star)$  parsing, which denotes  $LL(k)$  parsing where  $k$  is not bounded a priori.

Any language is defined by a grammar, and in particular grammars that accept context-free (and their refined classes  $LL(k)$ ,  $LR(k)$  and  $LALR(k)$ ) languages are referred to as context-free ( $LL(k)$ ,  $LR(k)$  and  $LALR(k)$  respectively) grammars. A grammar consists of terminals, non-terminals, productions and a start symbol. We will further explore these aspects when we introduce OPrec grammars in Chapter 6.

Once a suitable grammar and corresponding parser have been created and put into place, one might face an additional challenge: input that deviates from correct syntax cannot be parsed. In this situation, rather than simply detecting this and rejecting the input, one might wish for the parser to take appropriate actions to correct input, and also inform the user. This is commonly referred to as *error recovery*. In the literature, a substantial amount of work on error recovery has been carried out in the context of parsing programming languages. Comprehensive survey articles reviewing these early works are [Ciesinger, 1979; Sippu, 1981]. The seminal paper [Graham et al., 1979] introduces practical strategies for error recovery in LR parsing. Their approach is easy to use, even for adding recovery support to existing parser generators. This method is further developed in [Burke and Fisher, 1987], to give a method for LR and LL syntactic error diagnosis and recovery. Error recovery schemes for OPrec parsers are given in [Graham and Rhodes, 1975; Leinius, 1970]. These methods are computationally efficient and are still used in modern compilers. However, [de Jonge et al., 2010] indicates that more work could be done in order to improve the quality of error recovery. Strategies such as narrowing down regions where parser errors might reside and taking into account formatting (for example, indentation levels in source code) are used by [de Jonge et al., 2010] in order to improve error recovery for the Java programming language. Another approach is to provide structural information about the expected input, prior to parsing. In [Ugen, 2010], XML is used to define structural properties about documents containing JavaScript code, helping to correct erroneous data entry in the declaration of objects. However, more research has to

be conducted before these techniques will influence the error recovery capabilities of mainstream parsers.

For our purposes, none of these methods are totally suitable. In a system such as TalkMaths, wrong user input can arise for several reasons (in Chapter 6 we will look into this in detail). Error recovery needs to be extremely resilient, interactive and flexible, perhaps customisable by the user. In [Suhm et al., 1996], these requirements were formulated and various error recovery methods for speech interfaces are stated: repair by respeak, by spelling, by selection amongst alternatives and using an alternative input mode – in the original paper, the use of handwriting. If one is concerned with recovering from recognition errors, most of these strategies are available through the ASR tool, and users of TalkMaths will be able to benefit from this, as well. Our work on error recovery in the context of speech-driven systems is at the syntactical level during parsing. We have not found evidence that this aspect has been treated in the literature.

### 2.3.2 Parsing Spoken Mathematics

The work on spoken mathematics by Fateman, as presented earlier, was also interested in defining spoken mathematics using grammars, hence enabling a computer to process spoken input, using a parser. His research is interested in parsing suitably defined spoken forms of syntactically correct mathematical expressions, or else converting syntactically incorrect input to a format for display or further processing.

Given a syntactically correct spoken mathematical expression, it can be parsed using a grammar. This results in a parse tree that is useful for additional manipulations. As mentioned in [Fateman, 2006], it is difficult to specify a grammar that will parse the full set of mathematics that one may typically encounter in research papers, text books or lecture notes. An acceptable compromise might be the restriction to expressions that can be parsed with a context-free grammar, combined with some pre-processing of the input. In [Fateman, 2009], this approach is adopted in order to specify a subset of spoken mathematics, taking into account the fact that mathematical expressions use prefix, infix or suffix conventions and have operator precedences that are sometimes ambiguous.

In [Wigmore, 2011], an attribute grammar [Kastens, 1980; Knuth, 1990] for the

Yapps2 parser generator is described that recognises spoken mathematics at an elementary level. This was used in one of the earlier versions of TalkMaths to successfully parse complete mathematical expressions.

When attempting to add support for incomplete expressions to this aforementioned attribute grammar, we experienced difficulties formulating appropriate rules and concluded that novel directions for the problem of robustly parsing mathematics might be needed.

## 2.4 Speech-Driven Editing Systems

One of the main motivations behind our work is to make speech-driven systems as suitable as possible for users who typically might wish to engage in hands-free computing. In this section, we will outline contributions made by other authors to systems with a similar intention. After briefly reviewing existing commercial and freely available speech recognition tools, we discuss past work on speech-driven editing approaches, followed by documenting the extent to what the literature contributes towards realising speech-driven editors for mathematics. We pursue these further in Chapter 6.

### 2.4.1 Speech Recognition Technologies

Automatic speech recognition is the process of converting human spoken words to human- or computer-readable text [Young, 1996]. Commercial ASRs tend to be marketed as speech recognition applications or packages. The term “voice recognition” also appears in places, however, strictly speaking, this is incorrect terminology. Most ASRs require to be trained for a particular user in order to provide the best and most accurate recognition.

There are a few commercial speech recognition packages available in the current market, with the most widely available ones being products sold by Nuance and Microsoft. With claims of 99% word accuracy under good conditions, Nuance Dragon NaturallySpeaking (DNS) [Nuance Communications, 2013] is the market leader. It was originally developed to run on Windows operating systems, although recently Nuance has released versions for Mac OS X. Microsoft now includes Windows Speech in Windows 7 & 8, and their speech recognition solution is likely to be included and

further improved in future versions of their operating systems. At present, it remains unclear which of these two main players will eventually dominate the market.

A free alternative to the commercial products is Sphinx, an Open Source toolkit for speech recognition [Carnegie Mellon University, 2008] developed at Carnegie Mellon University. With Sphinx, additional data such as prosodic and intonation information can be captured. Currently, the recognition accuracy is inferior to that of the commercial ASRs.

More and more, vendors are adding speech support to their software. For example, Google has enabled speech recognition in its Chrome web browser, specifically designed for speech-enabled applications, such as “Google Voice Search” [Google]. In particular, mobile devices and gaming consoles are starting to be speech-enabled, and this trend is only likely to continue.

Typically, a speech recognition tool provides a *dictation mode* and a *command mode*. The former is used for the purpose of dictating textual information into an application such as a word processor or an email client, and the latter is for speaking structured commands which will trigger specific actions such as selecting menu items or clicking on GUI elements. Some allow the user to create custom spoken commands in addition to the default pre-defined commands.

Usually, these tools also supply an application programming interface (API), such as Microsoft’s speech API (SAPI) in order to add more sophisticated commands, load user files or restrict the vocabulary to increase the recognition accuracy for specific domains such as the medical industry [Mohr, 2009].

### 2.4.2 Speech-Driven Editing Approaches

Any system for creating mathematical content would be of limited value if it did not also allow the editing of existing material. Even for ordinary text documents, a person may spend a very substantial amount of time editing what he/she has typed or correcting mistakes [Sears et al., 2001]. Also, the existing cursor control methods are still difficult to use [Haque et al., 2013] when editing text by speech. This is likely to be just as much the case with mathematical expressions in documents. For speech-based systems, this appears only to have received limited attention in the literature. When using speech-controlled text editing, in a standard editor, it is relatively straightforward to

develop spoken commands for moving the cursor, plus a limited range of actions such as “backspace”, selecting words or a sentence. This is relatively inefficient and difficult to use for editing ordinary text, but of even less utility for editing mathematical content, which (as noted previously) often has a rather complicated two dimensional layout in conventional notation.

A mechanism for moving the mouse pointer using speech is attributed to [Pugliese and Gould, 1998]. This is available as the “mouse grid” in DNS, and Windows Speech provides a similar facility. Some research [Bickel et al., 2010; Christian et al., 2000; Dai et al., 2004; Karl et al., 1993; Sears et al., 2003] exists, which improves this approach for editing ordinary text via speech, exclusively based on an “anchor and target” approach. Whilst reasonably efficient for general mouse moving or text editing, it could be improved for the editing of structured documents. Begel [Begel and Zafrir, 2002] introduces a “context-sensitive” mouse grid, in the context of speech-enabled programming environments. This idea is further extended for the use in a mathematics editor in [Wigmore, 2011], where novel speech editing paradigms via specialist grids are introduced. In TalkMaths, some of these grids are available and can be used in order to assist the user with editing by speech.

### 2.4.3 Speech-Driven Interfaces for Mathematics

We are aware of a number of systems translating spoken mathematical input to different output formats and displaying the structure of the mathematical expressions. These systems work together with ASRs installed on the user’s machine.

MathTalk [Metroplex Voice Computing, Inc] is a commercially available system that implements speech macros for use within the Scientific Notebook environment. The functionality of MathTalk, even when compared with the other academic prototype systems mentioned below, is quite limited.

Fateman has undertaken work leading to Math Speak & Write, a multimodal approach combining spoken input with handwriting recognition. Unfortunately, the ambitious aim of simultaneous multimodal input was not achieved. Another system is CamMath, described in [Elliott and Bilmes, 2007], which needs the same support environment as MathTalk but seems to offer a better developed command language.

Previous approaches to allowing spoken input of mathematics include the research

prototype systems of Bernareggi & Brigatti [Bernareggi and Brigatti, 2008] (which only works in Italian) and Hanakovič and Nagy [Hanakovič and Nagy, 2006] (which is restricted to use with the Opera web browser due to it using X+V (XML + voice) technology).

We conclude that all these systems are not yet robust enough for day-to-day use by (potentially inexperienced) users.

## 2.5 Other Related Approaches

In the previous sections, we have explored different approaches to defining, recognising, predicting, correcting and parsing spoken mathematics. This section briefly reviews other work that relates to one or more of these aspects, including assistive systems for creating and editing mathematics since TalkMaths, from its early stages, has always been intended to be a highly usable and beneficial piece of assistive software.

There have been a variety of systems attempting to provide synthetic speech descriptions of mathematical text, including AsTeR (Audio Systems for Technical Readings) [Raman, 1998], MathGenie [Jacobs, 2006], REMathEx [Gaura, 2002], the commercial system MathPlayer [Design Science, 2013b], and AudioMath [Ferreira and Freitas, 2004]. The latter system is open-source, but unfortunately only functions in Portuguese.

In the context of verification mechanisms within OCR (*Optical Character Recognition*) for mathematical text, creating SLMs and context-free grammars for parsing mathematics in symbolic notation turns out to be highly relevant. Some previous authors [Suzuki et al., 2003; Watt and Xie, 2005] have developed systems for recognising and processing mathematical symbols using an OCR approach. However, two remaining issues are how to deal with the possibilities of misidentified symbols (potentially a big problem, since many people have poor *on-screen*, or on tablet handwriting) and mistakes by the user. Previous researchers have used syntactic [Fujiyoshi et al., 2008; Suzuki et al., 2009] or statistical [Mazalov and Watt, 2013; Smirnova and Watt, 2008] approaches in attempts to resolve these issues. Research in this area is continuing, and systems such as Math Input Panel (MIP) [Radakovic et al., 2011] provided by Microsoft Windows 7 and later versions of Windows, and MathPad 2 [Jr. et al., 2007] are more recent examples of OCR applications for converting hand-written mathematics in

conventional notation into its electronic equivalent (MS Word and MathType [Design Science, 2013a] equations and Matlab code respectively). However, adding speech as an input modality to these systems seems to be difficult.

Assistive systems such as those developed in the Lambda project [Edwards et al., 2006] and [Crombie et al., 2004] convert MathML into Braille format, to help blind users to read mathematical expressions within electronic documents such as web pages. However, such systems also require use of a Braille output device. They also require parsing of mathematics and use similar techniques to the ones discussed so far.

Although we are not aware of these approaches having been applied to the editing of mathematical text, severely disabled people, including tetraplegics, can only interact with computers using systems which monitor motion of their head, eyes, or possibly even facial muscles [Zielinski, 2013]. One such system of particular note is Dasher [Vertanen and MacKay, 2010] which can use a mouse pointer, possibly controlled by eye or head movement, to select characters from a menu on the screen. This uses statistical language models (see Chapter 4) to allocate an appropriate area of a display screen according to the likelihood of a particular character, word or symbol displayed there being the next item in the input sequence. Dasher can be controlled using a mouse, pointer or any motion or gaze tracking system. It should be possible to adapt Dasher for use with mathematical editors, and in principle, with speech.

## 2.6 Summary

In this chapter, we have primarily reviewed previous work in a number of key areas: defining languages for spoken mathematics, statistical language models and error correction algorithms focusing on predictive and corrective systems using those SLMs and algorithms, syntax analysis using context-free parsing techniques in general, but then also with a focus on spoken mathematics. We have also reviewed several systems: speech-driven editing systems implemented in the past by other authors that have similar aims as the TalkMaths system, and other non speech-based assistive systems for creating and editing mathematics.

# Chapter 3

## Architectures for Speech-based Applications

### 3.1 Motivation

The Naturally-preferred method for human-human communication method is speech, whereas that is not the case in human-computer communication. In this, we tend to rely on non-intuitive interfaces for input, such as a keyboard and mouse. However in recent times, speech has become a realistic alternative method to interact with computers [McTear, 2002], especially for users who have limited or no access to keyboard or mouse due to some kind of disability [Karshmer, 2008], or people “on the move” [Cuartero-Olivera et al., 2012]. In effect, speech recognition has now become a viable option for many users of desktop applications to use to input commands and data.

There does not appear to be a set of formal definitions in the literature of different types of applications using speech. It seems natural to define a computer application relying on speech as the primary input method as a *speech-based application*.

With current state-of-the-art speech recognition technology being more available, various speech-based systems exist either as research prototypes or as commercial products. A range of example systems can be given, from systems that are used in the medical industry such as Voice Activated Medical Tracking Application [Durling and Lumsden, 2008] or systems used in cars [Lee et al., 2001], to mobile phone intelligent personal assistants such as Siri [Dery, 2012] and Iris [Cheyer et al., 2005]. Such speech-



based applications can either run entirely on one computer – thus free from network and data transmission constraints – or use a client-server architecture.

The Internet is an important medium for modern applications in many fields. Whilst it should be noted that desktop-based applications still have their place, web-based applications have numerous advantages over other networked applications, such as location and platform independence, ease of upgrading and logging user interactions for various purposes. Although there are some non-trivial barriers one has to overcome in dealing with browser limitations, thick-client web applications built with more control over the client machine (using plug-ins such as Flash Player [Allaire, 2002] or the latest standard HTML5) are becoming increasingly popular.

For many users of speech recognition, possibly relying on mobile devices with limited processing power, a speech-based web application also has the advantage that it can be used whenever a speech-web client is available. Despite a number of research-prototypes being developed, not many practical systems combining automatic speech recognition and web technologies are available at present.

The motivation behind the work in this chapter was the fact that, for speech-based applications, the system architecture is a vital aspect that needs to be considered carefully in the design stages in order to implement a robust, reliable and efficient system. Our goal in this chapter is to describe and discuss different architectural approaches suitable for speech-based applications and identify the one most suitable for the TalkMaths system we will be presenting in Chapter 7. This chapter starts by investigating speech-based applications and their underlying technologies. We give an overview of different architectures suitable for speech-based applications, discuss their various merits and deficiencies, and then present a suitable architecture for web-based applications that use speech as the primary input mode. The remainder of this chapter is organised as follows: We first review relevant speech technology and tools. Then we propose the following possible architectures for speech-based applications:

- Desktop Speech-Based
- ASR and Post-ASR Processor (PASRP) on Client
- Speech Proxy
- Application Proxy

- ASR on Same Server
- ASR on Different Servers
- Distributed ASR using Same Server
- Distributed ASR using Different Servers

and finally choose a suitable architecture for the type of speech-enabled web applications of interest in this project.

## 3.2 Relevant Speech Technologies and Tools

Speech-based applications rely on a variety of tools and technologies. In this chapter we focus on architectural aspects and hence will not discuss the mechanisms behind speech recognition, speech synthesis or speech markup. We refer the reader to [Holmes and Holmes, 2001; Young, 1996] for more details of these. Here we mention some tools and technologies that are available, which may not be widely known.

### 3.2.1 Post-ASR Technologies

There are many technologies that can be used to build speech-enabled applications using the output from ASR. Here, we mention some of these technologies. Developed by Microsoft Corporation, the Speech Application development Interface (SAPI), provides a framework for speech application developers to design and build speech-enabled desktop applications. SALT (Speech Application Language Tags) is also a Microsoft technology that is developed specifically for the use of speech-enabled and voice-input only browsers [Microsoft Corporation]. SALT can be used to integrate simple speech interaction capabilities into existing web pages with minimal effort. A SALT enabled web page interacts with Text-to-Speech (TTS) and speech recognition software by specifying an XML-based grammar called Speech Recognition Grammar Specification (SRGS) [Hunt and McGlashan].

A Python-based macro system called NatLink has been developed by Joel Gould [Gould and Gould] as a subsystem for DNS. NatLink provides an interface to DNS and is available under an open source licence. Another speech macro system that works

with DNS is called Vocola [Mohr, 2009]. These macro systems require Python installed on the local machine hosting the application.

A more recent approach on building a framework for speech-based applications in a client-server setting is the Web-Accessible Multimodal Interfaces (WAMI) toolkit [Gruenstein et al., 2008]. WAMI provides a platform for developing speech-based web applications where the speech recognition happens at the server-side by allowing transfer of audio data and application data between client and server using two different connections, where one connection is established by a GUI controller and the other by an audio controller (both installed locally). It supports application-specific language models and is built with the aim of supporting multi-modal input systems. WAMI was developed to help interface developers to build systems that can be accessed by any standard web browser. Applications built with the WAMI toolkit use a browser-embedded Java applet that serves as an audio data transfer device, while a web-based GUI acts as the non-speech interface. Additionally, WAMI supports system development aimed at a variety of platforms such as tablet PCs, mobiles and laptops.

### 3.3 Classification of Architectures for Speech-based Systems

The World Wide Web Consortium (W3C) defines a speech interface framework [Larson, 2002] for speech-enabled web applications. This framework is concerned with defining markup languages suitable for integrating with ASR and standardising client-side communication methods. The W3C speech interface framework also allows for touchtone and pre-recorded audio input, as well as speech synthesis (text-to-speech) of output, which are not considered in this thesis. Whilst the W3C framework focuses only on the client-side, we are interested in ways of communication between client and server for speech-based applications. This will be the focus of the different architectures we will present later in this section.

We identify three components of a speech-based application: ASR, application logic and PASRP. We will now look at each of these components before moving on to possible system architectures.

Carrying out speech recognition solely on the client machine is called *embedded*

*speech recognition* (ESR), whilst sending raw audio data to the server and performing recognition there is called *network speech recognition* (NSR) [Zhemin and Philipos, 1999]. Splitting speech recognition between server and client is defined as *distributed speech recognition* (DSR) [Holada, 2003; Rajput and Nanavati, 2012] wherein performing a certain amount of processing on the client side (e.g. spectral analysis and/or data compression) reduces the amount of data to be transferred by only extracting necessary “features”, such as Mel-Frequency Cepstral Coefficients (MFCC) [Davis and Mermelstein, 1980; Holmes and Holmes, 2001] of the speech signal and sending these to the server [Flynn and Jones, 2012; Holada, 2003]. DSR is often used rather than NSR to save bandwidth of the network, in particular client-server speech recognition systems with low bandwidth network connections. Holada [2003] also states that feature extraction on client-side before transmission is better than compressing and decompressing speech on client and server sides respectively due to the decrease in the quality of speech. In any case, the processing and resource demands are very high for speech recognition tasks, as several processes involving signal processing, noise reduction and applying acoustic and language models are required. This intensive requirement on resources makes the choice of which of ESR, NSR or DSR strategies to adopt, a decisive factor for speech recognition. In particular, due to limited processing power, complex speech recognition in mobile phones, mobile hand-held and similar devices is best carried out at a remote server using DSR [Burke and Yacoub, 2010] strategies. However simple speech recognition tasks (such as digit or keyword recognition for dialing by voice) requiring less processing power can still be carried out solely on these devices (embedded speech recognition) using software available from vendors such as [Fonix.com; Rubidium.com; SensoryInc.com] and [Speechfxinc.com].

### 3.3.1 Terminology

Before moving on, let us define some terminology used to describe different architectures in this chapter. When considering the processing requirements of a given speech-enabled application, we can identify two aspects: the actual *application logic* that controls the behavior of the application independent from any speech input, and the Post-ASR Processor (PASRP). By the latter, we understand those parts of the logic that deal with interpreting input in the form of spoken commands. Due to these com-

mands a variety of actions can be taken, such as triggering specific application logic, enabling or disabling other speech commands, and generally maintaining a certain state within the current execution of the application (i.e. session management).

In a similar way to the ASR, the PASRP can reside on either client or server (or be split between both) and this will have an important impact on the design of the architecture. Furthermore, the (speech-enabled) application can also entirely run on one single machine or be based on a client-server design.

We will now present different system architectures for common speech-based applications, using diagrams to illustrate the different structures and characteristics, i.e. the location of the ASR, application logic and the PASRP. We denote by *speech audio data* ( $d_{sa}$ ) the electronic signal formed from the audio stream of a spoken utterance originating from the user via a microphone, possibly having been pre-processed and/or compressed. The term *recognised speech data* ( $d_{st}$ ) will be used for text string output data created from  $d_{sa}$  by the ASR with or without the assistance of an API. *Application specific data* ( $d_{app}$ ) will refer to all data other than the speech-related data as described above. Finally,  $f(d_{sa})$  denotes *features from speech audio data* extracted locally for some kind of DSR strategy.

### 3.3.2 ASR and PASRP Both Performed Locally

The following two architectures are based on ASR and PASRP being carried out locally, either on a stand-alone PC, or client-side in a web application, for example, with the aid of embedded speech recognition systems.

#### 3.3.2.1 Desktop Speech-Based (Architecture A)

The Desktop Speech-Based architecture is illustrated in Figure 3.1 (Architecture A). Applications based on this architecture rely on the ASR being installed entirely on the local machine. A spoken utterance is transmitted from the user as an audio signal to the microphone and is then converted into an (digital) electronic signal ( $d_{sa}$ ) which is passed to the ASR. The recognised string (denoted as  $d_{st}$ ) output from the ASR then passes to the main application. The main application subsequently processes the string and displays the output for the user. Depending on the level of sophistication of both the ASR and the application, PASRP can be implemented to some extent within both

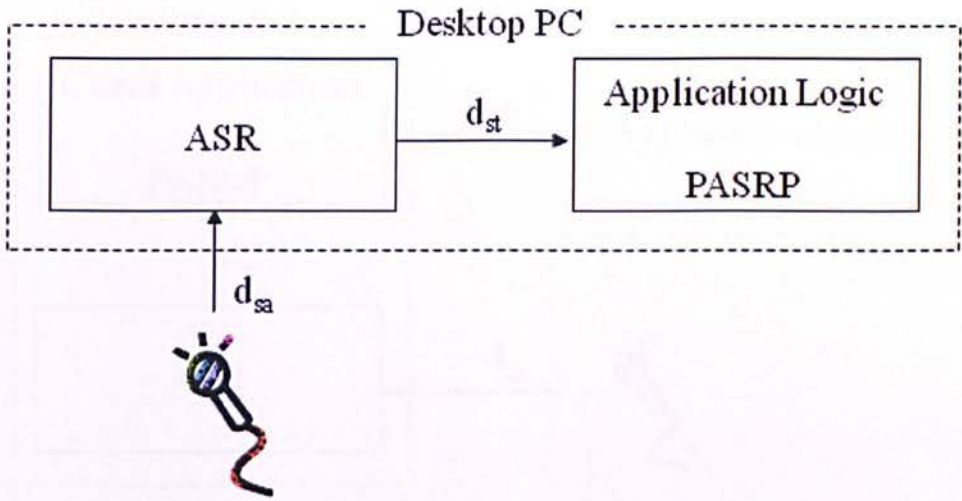


Figure 3.1: Desktop Speech-Enabled Application Architecture (A)

components.

As an example, consider a user creating a Microsoft Word document by speaking commands that are recognised by the ASR installed on his machine.

### 3.3.2.2 ASR and PASRP on Client (Architecture B)

As we have already mentioned, the introduction of server-side technology into a speech-enabled application can be done in several ways. Perhaps the most straightforward approach is to use a server to implement most of the application logic, with the ASR installed on the client machine. Recent advances in web technologies allow replacing the custom client application with a browser. Ultimately, this removes the necessity for the user to install additional software. Architecture B caters for this and is probably the one most commonly used for web applications.

Figure 3.2 shows that, as in Architecture A, the ASR sends recognised spoken commands ( $d_{st}$ ) as text to the client application, which exchanges application data ( $d_{app}$ ) with the server, following a request-response principle. Most modern web browsers are compatible with commercial ASRs such as DNS. A typical scenario where a user browses the web using speech commands that are recognised by DNS and translated into browser actions, would follow this architecture.



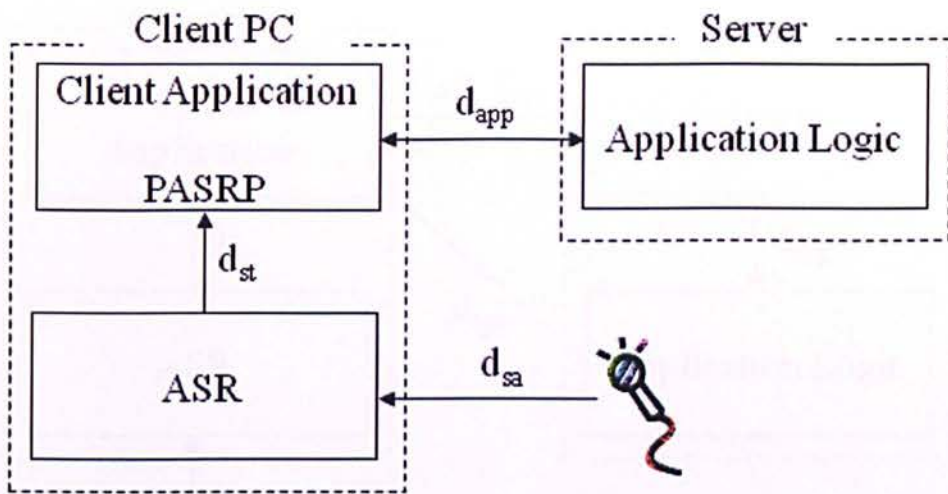


Figure 3.2: ASR and PASRP on Client Architecture (B)

### 3.3.3 ASR on Client, PASRP on Server

The next step would be to move the PASRP to the server whilst retaining the ASR on the client machine. This can be done in two different ways depending on the flow of data between PASRP and application logic.

#### 3.3.3.1 Speech Proxy (Architecture C)

The first approach to achieving this is through having the PASRP as a proxy between client and server application logic. A visual representation of this architecture is illustrated in Figure 3.3. Due to the fact that any incoming data from the client first passes through the PASRP proxy, and hence this data must have originally come from spoken input, we can see that any application based on this architecture is a speech-driven application. This architecture is of particular advantage if an existing client-server application can be easily turned into a speech-driven one by simply inserting the proxy. PASRP and application logic may or may not reside on the same host.

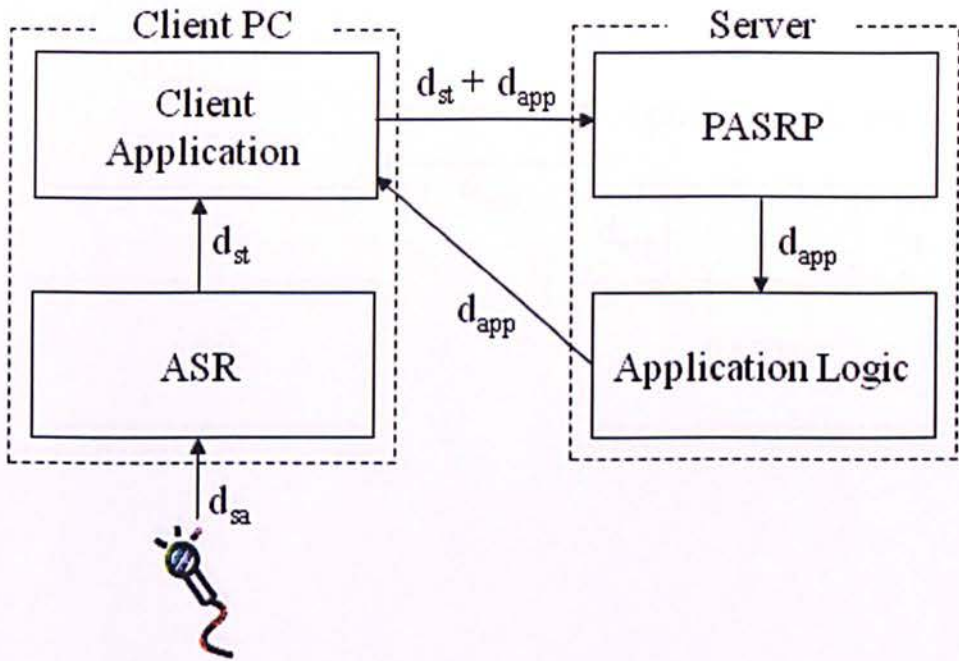


Figure 3.3: Speech Proxy Architecture (C)

### 3.3.3.2 Application Proxy (Architecture D)

The second approach of moving PASRP to the server can be achieved by exchanging the order in which the data flows through the server-side components (see Figure 3.4). In this architecture, the recognised speech data is forwarded by the application logic component to the PASRP. The PASRP then processes it and returns the results back to the application logic, which prepares and returns the response to the client. Compared to the previous architecture (C), it is necessary to exchange more data in case of the Architecture D, which will have a negative impact on the performance of the application. On the other hand, rather than having a purely speech-driven application, Architecture D allows for a speech-enabled approach as the application logic can decide to ignore incoming data  $d_{st}$  and process application data  $d_{app}$  only.

As further detailed in Chapter 7, the TalkMaths application is based on the above architecture.



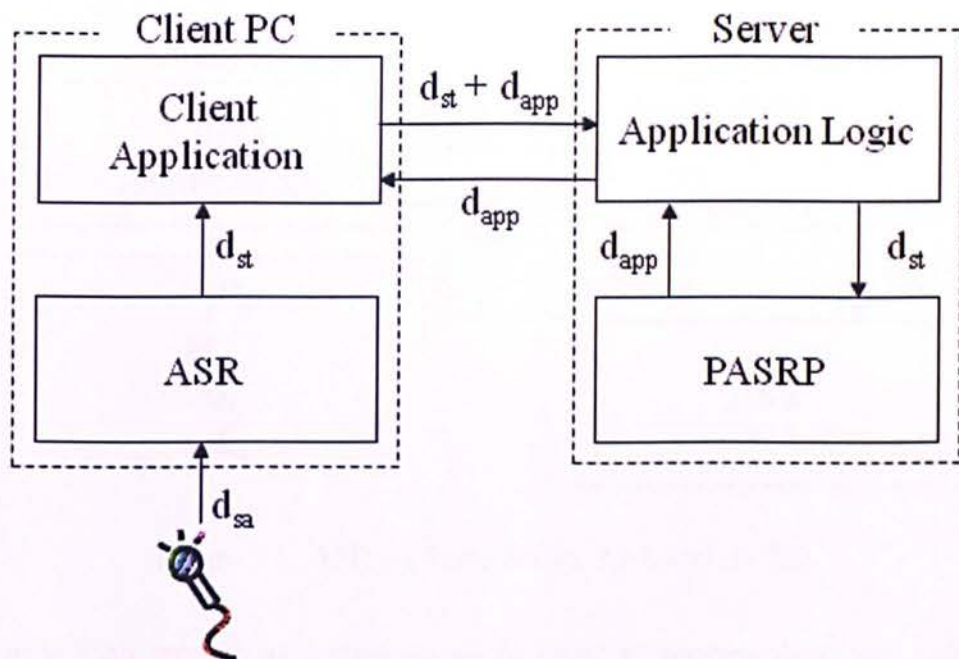


Figure 3.4: Application Proxy Architecture (D)

### 3.3.4 ASR and PASRP on Server

Moving speech recognition entirely from the client machine to the server yields another possible architectural approach (called *network speech recognition* in [Zhemin and Philipos, 1999]). Again, two different varieties exist, depending on the number of servers involved. An example appropriate to such architectures would be an application that is built using the WAMI toolkit [Gruenstein et al., 2008]. Further distinctions are possible, for example when taking into account the location of the PASRP, but we shall not further explore this aspect further in this chapter as that would deviate from the main focus of this thesis.

#### 3.3.4.1 ASR on Same Server (Architecture E)

In this architecture, both the server application and ASR coexist on the same server. As Figure 3.5 indicates, the user only has to install one client-side application that sends an audio stream  $d_{sa}$  to the server application. ASR will be residing in the server and receives  $d_{sa}$  from the server application for recognition. It then provides the server

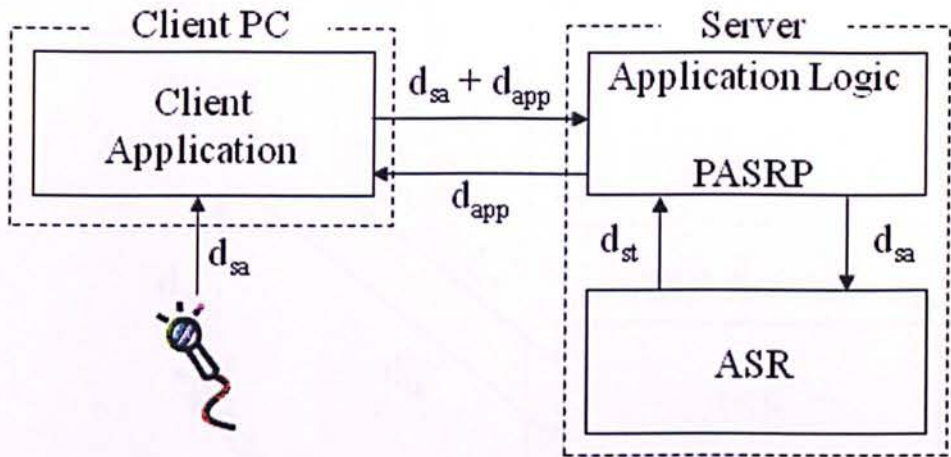


Figure 3.5: ASR on Same Server Architecture (E)

application with recognised commands  $d_{st}$  in order to process them and finally the server application sends  $d_{app}$  back to the client application.

#### 3.3.4.2 ASR on Different Server (Architecture F)

Having the server application and ASR located on different servers creates this architecture. The flow of data for this architecture type is indicated in Figure 3.6. The client application sends the audio data  $d_{sa}$  directly to the ASR, which resides on a remote server, for recognition. The recognised data  $d_{st}$ , is then fed back to the client application which interacts with the server application hosted on a different server to the ASR.

As cloud-based infrastructure for speech recognition is becoming more and more available, Architecture F experiences increasing popularity. Players such as Nuance and Google already provide ASRs based on this architecture. This is particularly suitable for mobile clients (eg. mobile phone and tablet PC users) due to their restricted resources. Also, WAMI-based online games such as Voice Race [Gruenstein et al., 2009] are examples for this kind of applications.

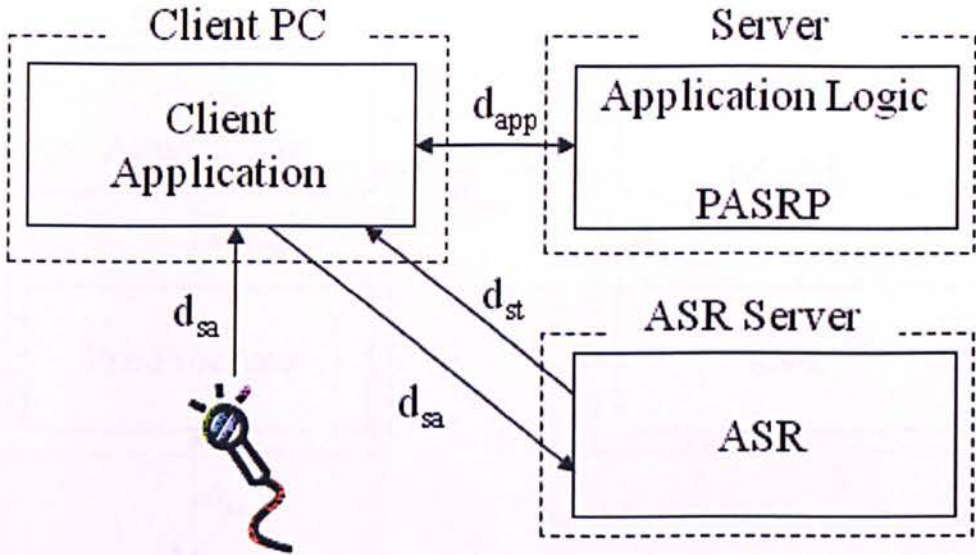


Figure 3.6: ASR on Different Server Architecture (F)

### 3.3.4.3 Distributed ASR using Same Server (Architecture G)

In this architecture, both server application and ASR coexist on the same server as in Architecture E. However, the raw audio signal  $d_{sa}$  is preprocessed at the client-side into a set of features,  $f(d_{sa})$ , which are then sent to the server so that the contents of the original speech signal can be recognised. Figure 3.7 illustrates the structure of this architecture.

### 3.3.4.4 Distributed ASR using Different Server (Architecture H)

In a similar way to Architecture F, here the server application and ASR reside on different servers. The only difference is that like Architecture G, the client machine only sends a selected set of features of the speech audio signal,  $f(d_{sa})$ , to the server that carries out ASR. Figure 3.8 illustrates this type of architecture.



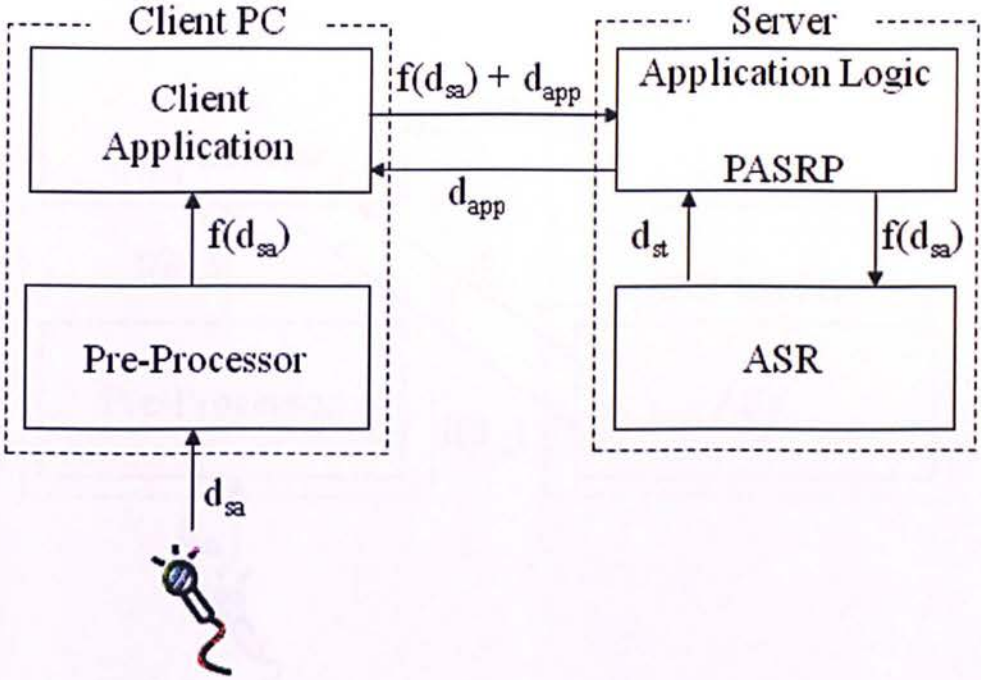


Figure 3.7: Distributed ASR using Same Server Architecture (G)

### 3.4 Choosing a Suitable Architecture for Speech-Enabled Web Applications

In this section, we first discuss the advantages and the disadvantages of the different architectures A – H, then analyse the architectural needs of the TalkMaths system prior to our work, and finally justify our decision to chose Architecture D.

Speech-enabled desktop applications are resource-intensive and require higher processing power than their web-based counterparts. In addition, the former require the installation of custom client-based software. However, these desktop applications can potentially be faster than corresponding client-server architecture versions since a desktop application does not have to rely on the speed and availability of the network.

With a client-server architecture, the increased availability over the internet of the application and the ability to store in, and retrieve data from different locations are advantages. Also, client-server architectures reduce the amount of software that needs to be installed on the client machine and can dramatically reduce the requirements for

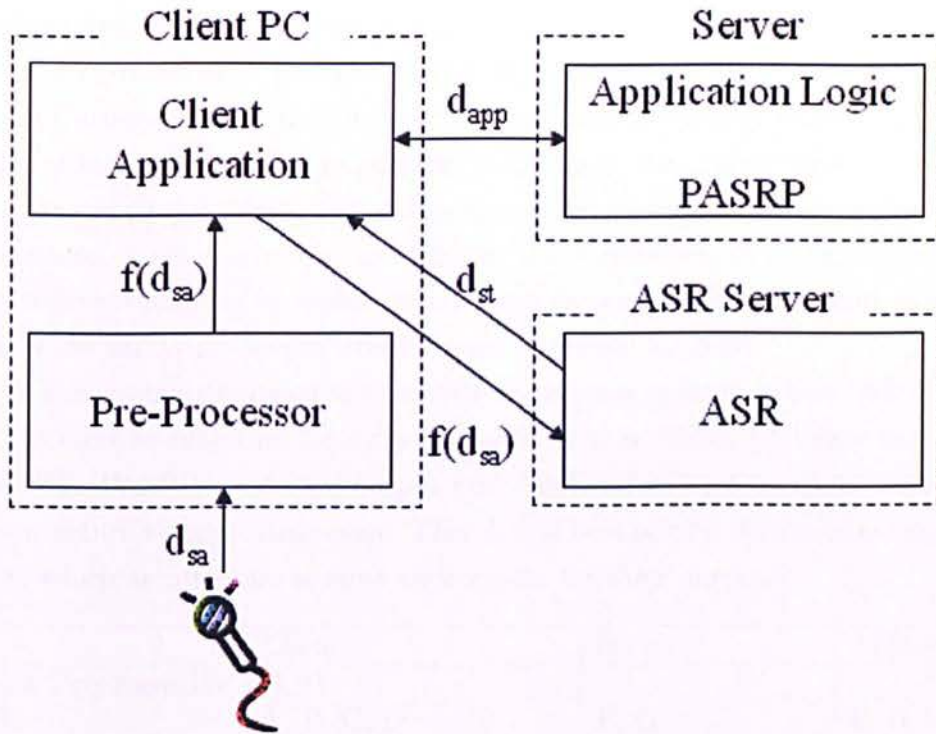


Figure 3.8: Distributed ASR using Different Server Architecture (H)

high processor speed and power consumption.

Due to the fact that the user has to have the speech recognition software installed locally for a speech-enabled desktop application (Architecture A) and other similar architectures (B, C and D), there are fewer choices available to satisfy these requirements at the client side and hence a *speech recognition on the server* architecture may be advantageous. On the other hand, when ASR resides on the server, as we discussed previously, the client machine has to send an audio stream to the server for recognition. However, there are some important technical challenges associated with sending large amounts of continuous speech audio data over a network. One solution is for the speech audio to be compressed before being sent, for example, as described in [Holada, 2003], by extracting only essential features (such as spectral coefficients) and only sending these as packet data to the server, where they are decoded for recognition. This approach allows for the system to be accessed by many devices of limited processing power, such as Personal Digital Assistants (PDA) and mobile phones, in addition to

personal computers. We described how this approach can be used in a client-server(s) setting in architectures G and H. Nuance SDK server edition [Nuance Communications] and Carnegie Mellon University's Sphinx [Carnegie Mellon University, 2008] are examples of server-side speech recognisers available at the present time.

As stated in [Bayer, 1996], embedding speech recognition inside the browser on the client machine is not convenient as it limits the application to a specific browser or requires different plug-ins or applets for different browsers. This is difficult to maintain and limits the utility of Google's cloud-based approach for ASR.

Given the options discussed in this chapter, one can appreciate how difficult it is to choose the right architecture for a speech-based system. Table 3.1 below shows where each of ASR, PASRP, rendering output and Application logic components reside for each architecture we presented above. This should be a helpful matrix to aid developers to decide which architecture is most appropriate for their purpose.

	Client	Server	ASR Server
Speech Preprocessing	G, H		
ASR	A, B, C, D	E, G	F, H
Application Logic	A, C, D	B, E, F, G, H	
PASRP	A, B	C, D, E, F, G, H	
Rendering Output	A, B, C, D, E, F, G, H		

Table 3.1: Architecture Classification for Speech-based Systems

In Table 3.1, the capital letters refer to the respective architectures, which we have listed here again for convenience.

- A: Desktop Speech-Based
- B: ASR and PASRP on Client
- C: Speech Proxy
- D: Application Proxy
- E: ASR on Same Server
- F: ASR on Different Server
- G: Distributed ASR using Same Server

- H: Distributed ASR using Different Server

For all the applications that we have in mind (such as the TalkMaths system or the SWIMS interface), we are concerned with the creation and modification of mathematical content, primarily by speech. We will be using client-side ASR both as a means of issuing commands for our systems, and to dictate (mathematical) text that needs to be processed and displayed.

The architectural choice for the TalkMaths system has been changed over the course of the project. The architecture of TalkMaths version prior to the start of this PhD work was a standard desktop-based one (Architecture A) [Wigmore, 2011]. As explained in Section 1.2, some limitations of this early version of TalkMaths led us to move to a web-based architecture. With this decision, all architectures apart from A, described in this chapter were potential candidates for our new prototype systems. Our initial TalkMaths parser was based on a parser generator written in the Python language - that requires Python to be installed on the machine that carries out the PASRP. When moving over to a client-server architecture, one requirement was to minimise the amount of software which to be installed on the client PC. Also, carrying out the PASRP on the server means that we can use high processing resources, such as cloud processing. Architecture B was therefore rejected for these reasons.

As we were more interested in good PASRP performance and speed rather than pure speech recognition performance, our focus was not on enhancing speech recognition at the signal processing level or making that “distributed” between client and server. Hence, for our work, a standard commercial client-side ASR tool such as Dragon Naturally Speaking or Microsoft Speech should be sufficient. This choice eliminated Architectures E – H as options. However elimination of these options could have compromised the overall speed of the system.

The final choice for our system is now between Architecture C and D. Although speech is considered to be the primary input method, we also allow keyboard and mouse modalities. The rationale behind this approach is that a considerable portion of our target users (individuals suffering from various disabilities, people relying heavily on on-line learning systems or using portable devices) may prefer to revert to keyboard and mouse for certain tasks, especially when ASR repeatedly fails to recognise dictated commands and/or corrections. If speech proved inadequate for input, this

multi-modality guarantees the user another option. With this multi-modal aspect, a command originating from the client-side in our setting is not necessarily speech input, and it makes sense not to process it as a speech command straight away. To allow for this, the best possible architecture was the Application Proxy (D) Architecture, as that allows the input of both modalities, rather than Speech Proxy architecture (C). Then, it was Architecture D which was eventually selected for use in the new web-based version of TalkMaths.

### 3.5 Summary

In this chapter, we were concerned with architectures for applications using speech as input. We further distinguished between speech-based, speech-enabled and speech-driven applications, depending on how speech input is processed. We also emphasised the usefulness of speech-based applications and reviewed relevant speech technologies and tools. We then gave a taxonomy for system architectures of applications using speech as input, by classifying possible architectures based on whether ASR and/or PASRP are being hosted on the server or on the client. Each architecture was explained and examined in terms of dataflow between client and/or server(s) and example systems were provided whenever possible. We then discussed advantages and disadvantages of each of these architectures, characterised by their requirements for resources and availability. A summary in the form of a table was also given to aid the process of making a design choice for speech-based systems. Finally, we chose an architecture that is suitable for speech-enabled web applications such as TalkMaths and SWIMS. In the next chapter we will explore the use of Statistical Language Models (SLM) which are later used in conjunction with the Application Proxy Architecture (D) in Chapter 5 to develop the SWIMS system.



# Chapter 4

## Theory of Statistical Language Models and Similarity Metrics

### 4.1 Motivation

Processing a language (natural or artificial) is a complex task for us human beings, as there are complex syntactic, semantic and pragmatic aspects of every language. As people, we constantly have to process the meaning of spoken and written (or typed) text in our daily life. Modelling or trying to model this process is called *language modelling*. Some researchers have focussed on syntactic [Chomsky, 1957, 1965; Lappin and Leass, 1994] or semantic [Purver and Ginzburg, 2004] aspects of language. However, one particularly successful approach has been to use statistical patterns found within a given language. We call this type of language model a “Statistical Language Model”. The assumption made is that we can in some sense encode syntactic and semantic rules in the statistics derived from these patterns [Hunter, 2004; Young, 1996]. The simplest types of Statistical Language Model (SLM) are based on sets of estimated probabilities corresponding to the  $N$ -grams (see next section) of the underlying language.

These models are primarily used in Automatic Speech Recognition (ASR) and have been very successful. In the ASR process, SLMs are used in conjunction with acoustic modelling and speech signal processing (spectral analysis, etc). Such language models based on word probabilities tend to be more effective for acoustically very similar words and lowest for acoustically very different words [Holmes and Holmes, 2001] since the

former are more likely to be confused by acoustic models unlike the latter. However SLMs are also useful in other disciplines such as machine translation [Marino et al., 2006], cryptography [Hasinoff, 2003], text prediction and auto correction tasks as well. Although SLMs were originally created for natural languages, they can equally well be applied to artificial languages such as computer programming languages or, as it will turn out in the next chapter, for language that arises from describing mathematical content in a spoken manner. In this chapter we present the theory of SLM and *edit distance metric*, which are then used in the next chapter in order to improve the recognition of spoken mathematics.

The remainder of this chapter is organised as follows: In the next section, we discuss  $N$ -gram SLMs and then look into the ways of interpolating them to create more sophisticated language models. Subsequently, an approach to allow SLMs to adapt to the current context called cache models, based on an analogy with “short-term memory”, and their interpolation with  $N$ -grams will be discussed. We then briefly discuss methods for the evaluation of SLMs before moving into a section which explains string edit distance metrics and their uses. We conclude this chapter by introducing how these can be applied to spoken mathematics which will be the theme of Chapter 5.

## 4.2 N-grams – A Baseline Approach to Statistical Language Modelling

The most common and easiest to understand approach to statistical language modelling is the use of  $N$ -grams [Bahl et al., 1983; Holmes and Holmes, 2001; Jelinek et al., 1990]. In this approach, we call all possible individual words (i.e the vocabulary) unigrams (where  $N = 1$ ), pairs of consecutive words bigrams ( $N = 2$ ) and sequences of three consecutive words trigrams ( $N = 3$ ) of the language. In general,  $N$ -grams are all possible  $N$  word sequences. In  $N$ -gram SLMs, we use probabilities of these  $N$ -grams within the language as the basis of the language model.

To create  $N$ -gram language models, we need to obtain estimates of the probabilities of all the possible  $N$ -grams. To do this, we require a suitable dataset, called *training data* (or a *training corpus*). We assume that this training data contains adequate

information to determine realistic  $N$ -gram probabilities of the underpinning language. Practically, with a large number of possible permutations and combinations of words, this is never the case for any natural language with a considerable vocabulary. If the language is assumed to have a vocabulary of  $V$  words where  $V$  may be several thousand, then there will be a total of  $V^N$  theoretically possible  $N$ -grams. In order to have appropriate probabilities for even the most rare but possible sequences of words, we require a large enough dataset which we call a representative sample of the language. The estimation of  $N$ -gram probabilities for common word sequences can be easily carried out by counting frequencies of these sequences from the training data to obtain maximum-likelihood estimates. For example, in a word sequence  $W = w_1, w_2, \dots, w_n$ , we can define the bigrams as  $(w_{k-1}, w_k)$  for  $2 \leq k \leq n$  and trigrams as  $(w_{k-2}, w_{k-1}, w_k)$  for  $3 \leq k \leq n$ . Then we can estimate conditional probabilities:

$$\hat{P}(w_k|w_{k-1}) = \frac{C(w_{k-1}, w_k)}{C(w_{k-1})} \quad (4.1)$$

(namely a bigram model) and

$$\hat{P}(w_k|w_{k-2}, w_{k-1}) = \frac{C(w_{k-2}, w_{k-1}, w_k)}{C(w_{k-2}, w_{k-1})} \quad (4.2)$$

(i.e. a trigram model), where the  $C(x)$  is used to represent the count of number of examples of  $x$  found in the training corpus [Holmes and Holmes, 2001].  $\hat{P}$  represents the “best” estimate of the probability.

Thus, it is possible to produce  $N$ -gram models for a language once a suitable dataset has been obtained. However, as already noted, the number of theoretically possible  $N$ -grams for this language of vocabulary  $V$  grows as  $V^N$  and it becomes difficult to obtain reliable estimates of  $N$ -gram probabilities, particularly for rare word sequences, for larger values of  $N$ . Furthermore, attempting to estimate these for large values of  $N$  would require very large amount of training data and be highly computationally expensive. Hence, in most cases,  $N$ -gram models are only created up to trigrams ( $N = 3$ ) as the performance at larger values of  $N$  usually does not improve significantly over trigram models [Clarkson, 1999b].

$N$ -gram language models are also useful for applications in text prediction, such as those used in SMS messaging in many mobile phones and in some assistive systems to

help disabled people type, such as *Dasher* [Vertanen and MacKay, 2010]. Typically, the task is to predict the next word in a sequence of  $n$  words, based on the previous  $n - 1$  words. This can be simply achieved by using  $N$ -grams on their own – for example, using only unigrams, only bigrams or only trigrams and so on. It is also possible to combine variants of these  $N$ -gram probabilities and use a more complex model to obtain better results. See Section 4.3 for an example.  $N$ -gram based SLMs are quite successful in predicting one or two words ahead. However these are not normally reliable for predicting beyond that (some of our work on this topic, relating to a study on the prediction accuracy of SLMs in the domain of spoken mathematics, can be found in the next chapter). Similarly, these models can be useful for correcting small errors in text documents (e.g. correction of spelling, such as is carried out by the automated spell checkers incorporated into many word processing systems and syntax checkers in state-of-the art programming editors).  $N$ -gram models can also be used in the correction of ill-formed word sequences [Pereira, 2000] or out-of-vocabulary (OOV) word errors. Using  $N$ -gram probabilities, we can rank the likelihood (probability) of a given sequence of words within a sentence and, if the probability is less than a given threshold, we can suggest that there is some sort of error in the sentence. Similarly, when a word used is not in the normal vocabulary or the current word sequence is highly impossible, we can find the most probable within-vocabulary words to substitute at that point in the sentence in order to suggest possible corrections. As mentioned in the Section 4.7 later, these types of errors can be corrected using the edit distance metric method as well.

Unigrams on their own have no context knowledge about the language they were generated from, only how common each word is. Bigram models are based on word pairs, and thus have a one word context knowledge, and in turn trigrams have a two word context. In general,  $N$ -gram models contain an  $(N - 1)$  word context, in terms of providing information about the properties of the language. However, this is purely from a statistical perspective, and can only provide the most basic semantic, syntactic and pragmatic information. This is in contrast to knowledge-based approaches, which would attempt to take more detailed account of these aspects. On the other hand, such approaches tend to be more complicated without necessarily yielding good performance in applications [Huckvale, 1996].

In natural languages, some word sequences may be theoretically possible, yet either

occur very rarely or do not occur at all in even very large training corpuses [Witten and Bell, 1991], for example, “a” and “the” are unlikely to occur next to each other in normal English text. (However, due to people hesitating while speaking, such pairings maybe less uncommon in transcriptions of spontaneous speech. For example, “I saw a the dog”, where the speaker hesitated between “a” and “the” to correct himself). In such cases, obtaining reliable probabilities for these rare or unseen  $N$ -grams may be near impossible even using a very large amount of training data. To deal with this problem, we estimate probabilities for these sparse  $N$ -grams by “borrowing” probability from more common  $N$ -gram probabilities [Hunter, 2004]. The rationale for this redistributing probabilities from “seen”  $N$ -grams to “less seen” and “unseen”  $N$ -grams is so that we can assume sum of all  $N$ -gram probabilities across all possible  $N$ -grams should be exactly equal to 1, whereas the sum of probabilities of the  $N$ -grams found in the training data should be less than 1, as this represents a subset of all possible events [Holmes and Holmes, 2001]. These techniques are called smoothing, discounting and backing-off [Katz, 1987; Ney et al., 1994]. Amongst many other smoothing techniques, the “Good-Turing” [Good, 1953] discounting method is one of the most-utilised on  $N$ -gram models. The Good-Turing method essentially involves altering the counts of  $N$ -grams such that an  $N$ -gram which occurs  $r$  times in the training data will be counted as though it had occurred  $d(r)$  times:

$$d(r) = (r + 1) \frac{n_{r+1}}{n_r} \quad (4.3)$$

where  $n_r$  is the number of  $N$ -grams that occur exactly  $r$  times in the training data [Chen and Goodman, 1996]. There are other discounting schemes such as “Witten-Bell” [Witten and Bell, 1991] in which the discounting coefficient is determined by a particular context, for example in case of a bigram “ $AB$ ” the context would be counts of distinct “ $A*$ ” (where  $*$  is any word within the vocabulary), and the context for a trigram “ $PQR$ ” would be “ $PQ*$ ” and so on [Clarkson, 1999b]. The concept of backing-off is to use a less-specific model when a high order language model fails to give an accurate probability estimate, for example using bigrams in place of trigrams. In general, an  $N$ -gram model can be backed-off to an  $(N - 1)$ -gram model in cases where the former has too little data to estimate the probability of a particular event.

### 4.3 Interpolation between N-grams

Increasing the performance of simple  $N$ -gram models can be done by combining them in several ways. One such approach, called a “deleted interpolation trigram model” combines unigram, bigram and trigram models together and was introduced by Lau [1994]. In this, the probability of the  $k^{\text{th}}$  word  $P(w_k)$  in a sequence can be estimated by using the unigram, bigram and trigram counts of occurrences of the word  $w_k$  and the two previous words  $w_{k-1}$  and  $w_{k-2}$  in the training data:

$$P(w_k|w_{k-1}, w_{k-2}) = \lambda_0 + \lambda_1 \frac{C(w_k)}{N} + \lambda_2 \frac{C(w_{k-1}, w_k)}{C(w_{k-1})} + \lambda_3 \frac{C(w_{k-2}, w_{k-1}, w_k)}{C(w_{k-2}, w_{k-1})} \quad (4.4)$$

where  $N$  is the number of words and  $C(x)$  is the count of an item  $x$  in the training data. The  $\lambda$ 's are optimised with respect to held-back data in order to minimise the *perplexity* (see Section 4.6).

When there is more than one set of training data, one could combine all the training sets into one and create an  $N$ -gram model from that. This technique is called the brute-force approach [Hsu, 2007]. Unless these training sets are of same size, relevance and coverage relative to the type of target language to which the model would be applied, this method will not be expected to give particularly satisfactory results [Rosenfeld, 2000]. Hence, some meaningful method of combining these data sources is required. One simple solution to this is *linear interpolation* [Chen and Goodman, 1996; Katz, 1987; Ney et al., 1994]. Here, each training set ( $T_i$ ) is used to create an  $N$ -gram model ( $M_i$ ) and in application the probability of the next word  $w$  given information  $a$  is estimated as:

$$P(w|a) = \sum_i \lambda_i P_i(w|a) \quad (4.5)$$

where  $\{\lambda_i\}$  are weights for each model. Note that  $\sum_i \lambda_i = 1$ . For example, the deleted interpolation trigram model proposed by Lau [1994] can be regarded as a special case of this. Typically the  $\lambda$ 's are chosen in order to optimise the performance of the resulting interpolated model [Hsu, 2007]. This way we can adapt the relevance of each training set to the target domain in a sensible way. In other words, with reference to a given target domain, priority is given to models trained on more relevant data over the models trained on less relevant data.

## 4.4 Cache Models

$N$ -gram models work well if enough training data is available to provide a representative sample of the domain it is intended to be applied to [Moore, 2001, 2003] – for example, for use in transcribing news broadcasts. In practice, we might need to create and apply language models to scenarios, such as dialogues and other spontaneous conversations between people and formal or informal speeches, where a suitable training corpus may not necessarily be available. In these circumstances, a language model needs to adjust while it is being used. Such *dynamic* language models are referred to as *adaptive* language models [Clarkson, 1999b], which use probabilities of recently encountered words – on the assumption that these are likely to influence the probabilities of words occurring in the near future – to update the existing word probability estimates. The distinction is that whilst  $N$ -gram models are static and use fixed probabilities based on a training corpus, adaptive models can dynamically change word probabilities according to the current situation. One often used adaptive language modelling technique is “Cache-based modelling”, that takes into account the fact that, in most natural languages, recently-appeared words are quite likely to re-appear [Holmes and Holmes, 2001] in the near future. Using this assumption, we can update the current word probabilities from those of a baseline  $N$ -gram model by giving higher weights to recently-appeared words and lesser weights to others. These tend to use a “cache-buffer” or “recent history”, usually over a predefined length – such as the 20 most recent words. (however some authors have adjusted the weights used according to a word’s position in the cache, e.g. Bellegarda [2004]; Clarkson and Robinson [1997]; Iyer and Ostendorf [1999]). If we define a word stream as  $w_1^i = w_1, w_2, \dots, w_i$ , such that  $w_i^j = w_i, w_{i+1}, \dots, w_j$ , then, using a cache of  $K$  words the cache-based conditional probability of the  $k^{\text{th}}$  word  $w_k$  in the sequence is given by:

$$\hat{P}_{\text{Cache}}(w_k | w_1^{k-1}) = \frac{1}{K} \sum_{j=k-K}^{k-1} I_{\{w_j=w_k\}} \quad (4.6)$$

where  $I_\epsilon$  is an indicator function which equals to 1 if  $\epsilon$  occurred and 0 otherwise [Clarkson, 1999b; Kuhn and De Mori, 1990]. This approach is analogous to the “short-term memory” of people [Baddeley, 2003; Fletcher, 1994] and the cache memory used in computers. Kuhn and De Mori [1990] have shown that this technique can be success-

fully used in ASR. Table 4.1 illustrates how a dynamic cache buffer could be maintained

Step	Sentence
1	<b>This is just another sentence</b> with
2	<b>This is just another sentence</b> with some
3	<b>This is just another sentence</b> with some words
4	<b>This is just another sentence</b> with some words in
5	<b>This is just another sentence</b> with some words in it

Table 4.1: Change of 5-word cache content (marked in bold) over time

over time. The words in bold font will be used in each step as the cache buffer, where at each step the left-most word in the cache is dropped-out and a new word from the text stream is inserted at the right-most edge.

## 4.5 Interpolation between N-grams and Cache Models

Since cache models on their own are based on limited information about language, it proves to be highly impractical to use them for recognition tasks on their own. However, they do reflect the word patterns over the short range – which is useful for prediction tasks as discussed earlier. In contrast,  $N$ -gram models contain the overall “global” word probabilities but not local patterns. To use both of these features together for better recognition performance, it is necessary to combine cache models and  $N$ -gram models in an appropriate manner. We can do this in many ways as the outputs of both models are in the same form (i.e. conditional probabilities). Generally, the linear interpolated cache- $N$ -gram model would be:

$$\hat{P}(w_k|w_1^k) = \lambda \hat{P}_{Cache}(w_k|w_1^{k-1}) + (1 - \lambda) \hat{P}_{N-gram}(w_k|w_{k-N+1}^{k-1}) \quad (4.7)$$

where  $\hat{P}_{Cache}$  is defined by equation 4.6. The goal is to find the optimal value of the coefficient  $\lambda$  to calculate probabilities of the resultant model such that its performance is maximised, and is better than that of each individual model. The *CMU SLM Toolkit* [Clarkson, 1999a; Clarkson and Rosenfeld, 1997] is a very useful tool to determine these optimal coefficients using algorithms such as the “Expectation Maximisation” (EM)



algorithm [Dempster et al., 1977], which estimates these coefficients on the basis of some testing data.

## 4.6 Evaluating the Performance of SLMs

The performance of a SLM is measured with respect to its domain of application by applying the model to a previously “unseen dataset”, which we call “test data”. Usually, a subset of the training data is held back for this purpose and is hence called “held-back data”. Suppose that we have a language model  $L$  trained on training data  $T$  and with held-back test data  $S$  exists, composed of a word sequence  $W = w_1, w_2, \dots, w_K$ . (Ideally both  $S$  and  $T$  should be representative of the domain which the model will be applied.) Using  $L$ , we can find the probability of the word sequence  $P(W) = (w_1, w_2, \dots, w_K)$ . The higher  $P(W)$ , the better the language model  $L$  with respect to the training set  $S$ . We call the reciprocal of the  $K^{\text{th}}$  root of this probability the *Perplexity score* [Holmes and Holmes, 2001] of the model  $L$  with respect to the test set  $S$ . In general, perplexity can be written as:

$$PP(W) = [P(w_1, w_2, \dots, w_K)]^{-1/K} = \left[ \prod_{k=1}^K P(w_k | w_1, \dots, w_{k-1}) \right]^{-1/K} \quad (4.8)$$

where  $PP(W)$  is the perplexity score of the model  $L$  with respect to a test set  $S$  having a word sequence with  $K$  words. We can use this technique to evaluate a language model with respect to a test dataset that needs to be recognised. The perplexity can also be regarded as the average number of equally probable words possible, and therefore which need to be considered, at each step in a “guess the next word” game [Hunter, 2004; Shannon, 1951]. Hence the lower the perplexity score, the better the language model will be in predicting the test data.

Given a set of SLMs, choosing the best possible model can also be done using *Maximum Entropy* method [Lau et al., 1993]. Given an optimal language model, its entropy (sometimes called cross entropy) is the average, minimum number of bits required to encode a word [Shannon, 1951]. So, for a given probability distribution  $p$ , the entropy

$H(p)$ :

$$H(p) = - \sum_i p_i \log_2(p_i) \quad (4.9)$$

where  $p_i$  is the probability of the system when it is in the state  $i$  [Hunter, 2004; Shannon, 1951]. The entropy of a language model can be interpreted as the amount of non-redundant (i.e. useful) information produced by it [Rosenfeld, 1996; Shannon, 1951] and hence the quality of a language model with respect to a target domain can be maximised by increasing the entropy. One such method of finding the best language model is called “maximum entropy method” [Berger et al., 1996; Lau et al., 1993; Rosenfeld, 1996]. The relationship between entropy  $H(p)$  and perplexity  $PP(W)$  can be given as [Bahl et al., 1983]:

$$PP(W) = 2^{H(p)} \quad (4.10)$$

Since the perplexity and entropy are dependent on test data, it depends on how the test data was chosen. To minimise this bias, we use a “Cross Validation” technique [Kohavi et al., 1995] when evaluating a language model. This is done by splitting the training corpus into several subsets and training the model on all but one subset, and then applying the model to the skipped subset for evaluation. We then repeat this process by skipping one subset at a time for all the subsets, averaging perplexity or entropy over the different trials. Sometimes we also look at the variation of measurements between trials to estimate the uniformity of the training corpus.

## 4.7 String Edit Distance Metrics and its Usage

As explained in Section 2.2.2, it has been noted that the majority of human typing and spelling errors are quite minor [Damerau, 1964], often involving just the omission or addition of a single character, typing two characters in the wrong order, or accidentally substituting one character for another (often one adjacent to the correct symbol on the keyboard [Grudin, 1983]). The Damerau-Levenshtein distance [Damerau, 1964; Levenshtein, 1966] on its own, or with some additional contextual knowledge [Mays et al., 1991] can be used to find the nearest correct within-vocabulary words to correct such spelling errors. The Damerau-Levenshtein distance between two character strings measures how different the strings are by taking account of the minimum numbers of

insertions, deletions, substitutions and “single transpositions” of characters required to transform one of the strings into the other. (Note that single transposition refers to the interchange of two adjacent characters [Damerau, 1964; Wagner and Lowrance, 1975]). Here we assume each such operation corresponds to equal “distance”, i.e. insertion=1, deletion=1, substitution=1 and transposition=1, where each “1” is the same unit of distance. For example, the edit distance of the words “string” and “strong” will be just 1 as changing the character “i” into “o” would suffice (1 substitution). Some approaches regard a substitution as 2 units: 1 deletion + 1 insertion. An efficient algorithm to calculate Damerau-Levenshtein distance between two words was introduced by Wagner and Lowrance [1975].

Edit distance metrics can be used in order to correct training data for errors prior to creation of SLMs and also to correct target data before applying SLMs to them for prediction. In this way the SLMs will no longer need to deal with OOV words. A more sophisticated method would be to rank the candidate words to be replaced in place of an incorrect word using a combination of SLM and edit distance metric in a meaningful way.

## 4.8 Applications to Spoken Mathematics

As mentioned in Section 4.1, SLMs have many applications in predicting and correcting both natural and artificial languages. One such artificial language is “Spoken Mathematics” [Attanayake et al., 2011a; Chang, 1983; Fateman, 2009], which is used to dictate mathematics into electronic documents using ASR on a computer. Although many applications of SLMs to the processing of different natural languages can be found in the literature, we have not come across any evidence of these being used to enhance the recognition of spoken mathematics, apart from studies by Wigmore [2011] in the earlier stages of the TalkMaths project. Regarding the use of edit distance metrics, it may be obvious that it can be used with any language, yet has not previously been employed in correcting typed mathematics equations. Based on the theories discussed in this chapter, the next chapter will present our work on prediction and correction of spoken mathematics using SLMs and an edit distance metric.

# Chapter 5

## SLM Applications to Spoken Mathematics

### 5.1 Introduction

As noted in Chapter 3, SLMs and edit distance metrics have proved to be very useful in prediction and correction tasks. In this chapter, we describe how these techniques were applied in the TalkMaths project, with a view to assisting its users in editing mathematical text documents.

Proficiency in mathematics, at least at an elementary level, is essential for success in a wide range of scientific, technical and commercial fields. However, it is a subject which many students find difficult, in part due to its specialized language and notation. These make working with mathematical equations and formulae a problem for a large proportion of people. This is more notable when the mathematics expressions to be manipulated are to be included in electronic documents. Typing and editing ordinary text can be both slow and error-prone for non-experts, which is even more the case for mathematical text, with its non-alphanumeric symbols and often rather complicated two-dimensional layout. Furthermore, creating, editing and reading mathematical text (in its conventional form) is particularly difficult for individuals suffering from various disabilities [Karshmer, 2008], on-line distance learners and people who are often working “on the move”. (It has been noted that a growing proportion of studying and academic exercises are being carried out using mobile devices such as smartphones

and notebook or tablet computers, often “on the move” or in potentially noisy public places such as cafés [Cuartero-Olivera et al., 2012]). Spell checkers, automated correcting facilities and predictive text have been familiar features of word processing and text messaging (e.g. SMS on mobile phones) systems for several years. These have aimed to provide *intelligent assistance* to the user in order to make the task of creating and editing ordinary text easier. In this chapter, we discuss the development of similar features using SLMs and edit distance metrics for two mathematical text editors, TalkMaths and SWIMS, which allow input of mathematical text using spoken mathematics. We also present the results of three experiments that were used to evaluate our prediction and correction implementations. The first part of this chapter is devoted to describing the creation and evaluation of SLMs from a suitable dataset collected from relevant domains on the World-Wide-Web. The next part will then describe three experiments using these SLMs for spoken mathematics prediction tasks and the analysis of their results. Subsequently, details of another experiment and its results on error correction for spoken mathematics using a string edit distance metric are given. The chapter will be concluded by a discussion section.

## 5.2 Developing and Evaluating SLMs for Spoken Mathematics

As part of the TalkMaths project, our approach to building a substantial database of mathematical text – incorporating equations, formulae, etc – has been evolving continuously over the last few years. During the earlier stages in this project [Wigmore, 2011], the first attempt at obtaining data on the ways in which mathematical equations are spoken by people was based on a part of the British National Corpus (BNC) [Burnard, 1995] which consisted of transcriptions of conversations from school and college mathematics classes [Wigmore, 2011; Wigmore et al., 2009a,b]. The vocabulary (number of different words) found in this dataset was 4,355 (the total number of words was 123,821) and the perplexity of the  $N$ -gram SLM obtained was rather high, due to both the relatively large vocabulary and the high proportion of non-mathematical words and “general chat” in the conversations, making the data somewhat unpredictable. The second attempt at modelling mathematical text was to manually populate a dataset using

trigonometric equations from some mathematical school text books [Wigmore, 2011; Wigmore et al., 2010]. Not surprisingly, both the vocabulary (102 distinct words) and the dataset overall (the total number of words was 7,857) were smaller for this trigonometric dataset than for the BNC-based data. The statistical language models built from the second (trigonometry) dataset had much lower perplexity scores, indicating that this data was more predictable. Interestingly, consistent with previous studies on more general speech data [Hunter and Huckvale, 2006; Moore, 2001, 2003], the results of these first two attempts at creating statistical language models from mathematical material confirmed that, *for data of a given type*, if the training dataset is increased in size, the perplexity of the resulting language model decreases and hence its predictive power is increased. Since the second dataset was much smaller, the necessity to create a substantial yet relevant dataset was evident.

### 5.2.1 Creation of the Dataset from Web Sources

In the current study, we have created a much larger, high quality dataset containing over 4100 expressions of mathematical expressions (the total number of words was 77,824 with a vocabulary of 100 words) on which to base new SLMs. We identified and *crawled* a handful of publicly available *tutorial* web sites containing mathematical equations at a similar level of complexity to those which TalkMaths is currently capable of processing. For this work, we identified a number of mathematical tutorial web sites in the public domain covering material at roughly GCSE & GCE A-Level or “Senior High School level” mathematics. We developed a *web-crawler* that can identify LaTeX or MathML content within the source code of a web site and applied this to the tutorial sites we had found. This mathematical content is then extracted into a database. A *filtering script* was then applied to remove display instructions from LaTeX code and illegal characters from the equations. Finally, a LaTeX to *spoken mathematics format* converter for expressions was designed and developed using the Yapps2 [Patel, 2009] parser generator. The implementation of this converter can be found in the Appendix B. In order to ensure that the mapping from LaTeX code to spoken mathematics was *one-to-one* rather than *many-to-one* (i.e. any given valid expression in spoken mathematics should be the result of converting a unique piece of LaTeX code) the converter introduced additional keywords *begin* and *end* for de-

noting sub-sections (such as fractions or square roots) within the *linearised spoken descriptions*<sup>1</sup> of each mathematical expression, to create a word string which would be identical to the *correct* way in which a TalkMaths user would dictate that expression. Some example expressions from this corpus are shown in Table 5.1. Table 5.2 shows the most frequent words (unigrams) found in this corpus of spoken mathematics. Table 5.3 illustrate most frequent bigrams and trigrams in our spoken mathematical expressions corpus respectively. A more detailed comparison of how our dataset compared to other related corpora in  $N$ -gram frequencies can be found in the Appendix A, Tables A.1, A.2 and A.3.

### 5.2.2 Creating Statistical Language Models

Once the dataset has been created by using the web-crawler, we then used the Carnegie Mellon University Statistical Language Modelling (CMU SLM) Toolkit [Clarkson and Rosenfeld, 1997] (with Good-Turing discounting [Good, 1953]) to build various trigram-based SLMs using samples of our corpus (3,194 mathematical expressions containing a total of 61,479 words with a vocabulary of 100 words). Figure 5.1 illustrates the typical usage of the CMU Toolkit which we have adopted to suite our needs. The theoretical principles behind these models were described in the previous chapter. An overview of the process of creating SLMs and measuring perplexities from web-crawled spoken mathematics corpus is illustrated in Figure 5.2 which covers three main processes, namely data collection, conversion and SLM production & evaluation. Here “text2wfreq”, “wfreq2vocab”, “text2idngram”, “idngram2lm” and “evallm” are names of programs or routines within the CMU toolkit. text2wfreq converts a passage of text to a table of word frequencies, wfreq2vocab produces a vocabulary list from that table of word frequencies, text2idngram takes in a text stream together with a vocabulary

---

<sup>1</sup>When speaking mathematics, in general, the spatial aspects of mathematical quantities are usually omitted. For example,  $a^{b+c} + d$  could be spoken as “a to the power of b plus c plus d”, where a pause may be used between  $c$  and  $d$  to indicate the end of the superscript. However, without seeing the written expression, this may be ambiguous for the listener since that utterance could also mean  $a^b + c + d$  or  $a^{b+c+d}$ . Here, what we mean by “linearised spoken mathematics” is the introduction of some suitable spoken delimiters to indicate such spatial aspects (end of a superscript, end or beginning of a numerator/denominator, etc.) to a mathematical expression. For the example here, the linearised spoken mathematical version would be “a to the power of begin b plus c end plus d” which clearly indicate the beginning and end of the sub-expression corresponding to the power of  $a$ . We refer to the next chapter for a more detailed discussion of spoken mathematics.

Spoken Mathematical Expression	Written form
delta victor equals cos x-ray delta x-ray	$dv = \cos x dx$
hotel of x-ray equals one minus x-ray to the power of two	$h(x) = 1 - x^2$
equals one over alpha begin integral of delta theta end	$= \frac{1}{a \int d\theta}$
equals theta over alpha plus capital charlie	$= \frac{\theta}{a+C}$
golf of x-ray equals square root x-ray semi-colon	$g(x) = \sqrt{x} :$
hotel of x-ray equals one minus x-ray to the power of three	$h(x) = 1 - x^3$
open bracket one over two close bracket	$(\frac{1}{2})$
four less than or equal to x-ray less than or equal to five	$4 \leq x \leq 5$
square root begin one minus x-ray to the power of two end	$\sqrt{1 - x^2}$
papa of x-ray equals foxtrot of x-ray plus capital charlie	$p(x) = f(x) + C$
uniform equals one plus begin alpha to the power of two end	$u = 1 + a^2$
india index zero equals echo to the power of x-ray	$i_0 = e^x$
begin minus bravo end over begin two alpha end	$\frac{-b}{2a}$
minus bravo over begin two alpha to the power two	$\frac{-b}{2a^2}$
two over five x-ray to the power of five plus capital charlie	$\frac{2}{5x^5+C}$

Table 5.1: Sample expressions from our spoken mathematics corpus. The names of individual letters are dictated using the NATO pronunciation alphabet (*alpha*, *bravo*, *charlie*, *delta*,...) with *x-ray* representing the letter *x*

list and generates a numerically sorted mapping between *N*-grams and the vocabulary list. `idngram2lm` then takes in the output of `text2idngram` and a vocabulary list to generate the language model, `evallm` evaluates the perplexity of a language model with respect to a specific passage of test text.

### 5.2.3 Evaluation Results of Statistical Language Models

The experiments to investigate the quality of the web-crawled data, and the SLMs obtained using them confirmed the trends noted by previous studies [Hunter and Huck-



Unigram	Frequency %	Unigram	Frequency %
end	7.58	over	3.21
begin	7.57	minus	2.53
x-ray	7.30	one	2.52
of	7.26	delta	2.42
two	4.82	close	1.81
to	4.32	open	1.80
the	3.93	three	1.66
power	3.93	alpha	1.65
equals	3.79	index	1.62
bracket	3.61	plus	3.34

Table 5.2: The most frequent words (unigrams) in our *spoken mathematical expressions* as percentages of all the words in the corpus. *x-ray* is the spoken form of the symbol  $x$

Bigram	Frequency %	Trigram	Frequency %
the power	3.93	to the power	3.93
to the	3.93	the power of	3.93
power of	3.93	power of two	2.05
of two	2.1	x-ray to the	1.66
over begin	1.85	end over begin	1.2
of begin	1.76	delta x-ray end	0.88
end over	1.68	power of begin	0.73
x-ray to	1.66	close bracket end	0.72
close bracket	1.55	of two end	0.67
open bracket	1.54	begin open bracket	0.63
of x-ray	1.42	over begin delta	0.63
x-ray end	1.39	open bracket x-ray	0.58
begin delta	1.17	of x-ray equals	0.52
delta x-ray	1.11	end to the	0.51
end end	1.08	foxtrot of x-ray	0.49
x-ray plus	1.07	of two plus	0.47
two end	1.07	begin delta x-ray	0.46
end equals	1.06	x-ray end equals	0.39
equals begin	0.9	power of three	0.39
begin x-ray	0.81	x-ray close bracket	0.39

Table 5.3: Most frequent bigrams and trigrams in our dataset

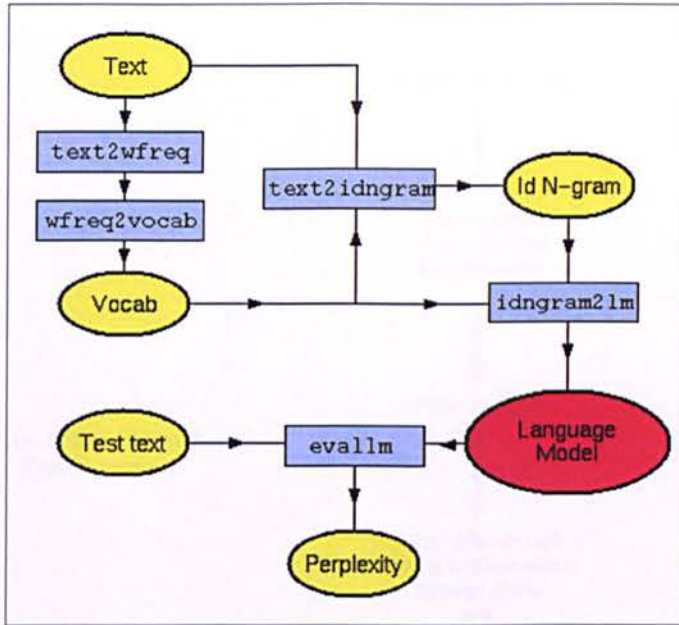


Figure 5.1: Typical usage of CMU Toolkit [Clarkson and Rosenfeld, 1997]

vale, 2006; Wigmore, 2011; Wigmore et al., 2010]. As mentioned previously (in Section 4.6), in a similar manner to other authors we used perplexity as a measure of the quality of our models, with lower perplexity indicating a better model. We also minimised any potential bias in the perplexity tests by using the cross-validation technique also explained in Section 4.6. The results from this new study are summarised in Table 5.4. Interestingly, the predictive power of the models based on the samples of our new dataset tested was better (as shown by lower perplexity values) than those of both earlier studies [Wigmore, 2011; Wigmore et al., 2010] on spoken mathematics. There are several possible reasons for this improvement. The vocabulary is still relatively small and the training datasets used in our latest study are considerably larger than the trigonometric dataset used in Wigmore et al. [2010]. Perhaps the most important reasons could be the higher quality and increased amount of training data.

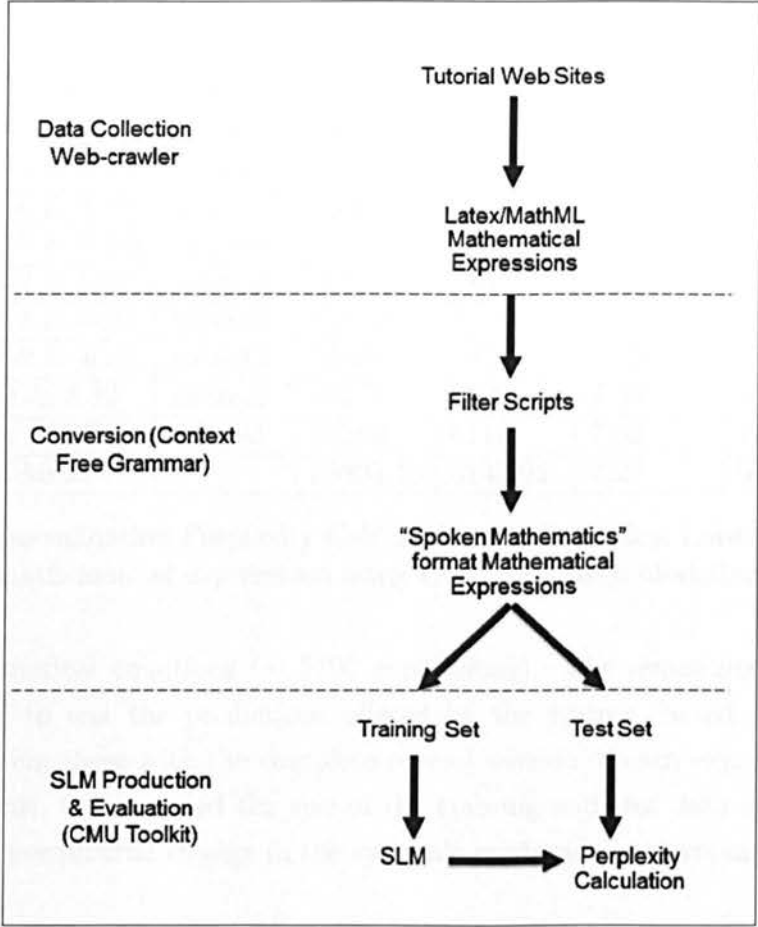


Figure 5.2: Building & Evaluating SLMs for Spoken Mathematics for Tutorial Web Site Data

### 5.3 Applications of SLMs to Prediction of Spoken Mathematics

As noted in previous section, the SLMs we built have the potential to be useful for prediction of mathematical text. In this section, we first introduce a prototype system (Speech-based Web Interface for Mathematics using SLMs, or *SWIMS*), that we used to empirically evaluate our SLMs when put into practice. Then we report results of three experiments used to evaluate the predictive power of the SLMs: For the first two experiments, A and B, we trained a SLM using 90% of our database of

Training set	Test set	Training words	Test words	Perplexity	Vocabulary (words)
subsets [1-9]	subset10	54907	6572	7.07	100
subsets [1-8 & 10]	subset9	54968	6511	7.17	100
subsets [1-7 & 9-10]	subset8	55294	6185	7.11	100
subsets [1-6 & 8-10]	subset7	55688	5791	7.31	100
subsets [1-5 & 7-10]	subset6	55172	6307	7.25	100
subsets [1-4 & 6-10]	subset5	55597	5882	7.74	100
subsets [1-3 & 5-10]	subset4	55340	6139	7.01	100
subsets [1-2 & 4-10]	subset3	55805	5674	7.65	98
subsets [1 & 3-10]	subset2	55177	6302	7.53	100
subsets [2-10]	subset1	55363	6116	7.02	100
Grand Mean		55331.10	6147.90	7.29	99.80

Table 5.4: Cross-validation Perplexity Calculations on Statistical Language Models of 3,194 spoken mathematical expressions using CMU Language Modelling toolkit

spoken mathematical equations ( $\sim 3700$  expressions). The remaining 10% ( $\sim 400$ ) was then used to test the predictions offered by the system, based on the trained model, comparing these with the complete correct version of each expression. For the third experiment, C, we varied the size of the training and test data sets in order to monitor the consequential change in the system's prediction performance.

### 5.3.1 The SWIMS Prototype System

SWIMS was developed as a separate module which can be later integrated into the TalkMaths system following successful evaluation. The goal of SWIMS is to assist the user by predicting and/or correcting his/her input using SLMs prior to parsing. Parsing is required in order to display the output on the screen using suitable mathematical rendering technology such as MathML. For ease of evaluation and for better performance, SWIMS has been divided into two units, one to predict the next word(s) in the input and the other to correct user mistakes. The former interface is called *Predictive Mathematics* and the latter *Alternative/Corrective Mathematics*. We note that requiring a user to read a large number of possible alternatives offered to them by the system imposes a high cognitive load. Therefore, we also provide previews of each of these alternatives, rendered into standard mathematical format. This should make the

user’s task of identifying the correct version easier, removing the burden of reading long expressions in spoken mathematics format. We used JSON to store trigram probabilities for our SLM, JavaScript for calculating probabilities, and the jQuery JavaScript library to communicate between the browser and the TalkMaths parsing server.

### 5.3.2 Predictive Mathematics Interface

The predictive mathematics interface predicts one or two words ahead of the currently typed or dictated mathematical text. In order to predict one word ahead, the system uses the last two words of the input, trying to match the first two words of trigrams found in our SLM. When such matches are found, suggestions for possible completions, i.e. the third word of such trigrams, are offered to the user as alternatives, ranked according to the trigram probabilities. In cases where the input is less than two words, or there is no matching trigram, then the system will back-off [Katz, 1987; Kneser and Ney, 1995] to bigrams and their probabilities. Similarly, if this is not successful, unigram probabilities will be used. Two word prediction is a recursive extension of the one word prediction mechanism, where only the first word of each trigram is used as input. The system architecture of this predictive interface is shown in Figure 5.3, where the dashed line indicates the system boundary of SWIMS. Figure 5.4 shows this

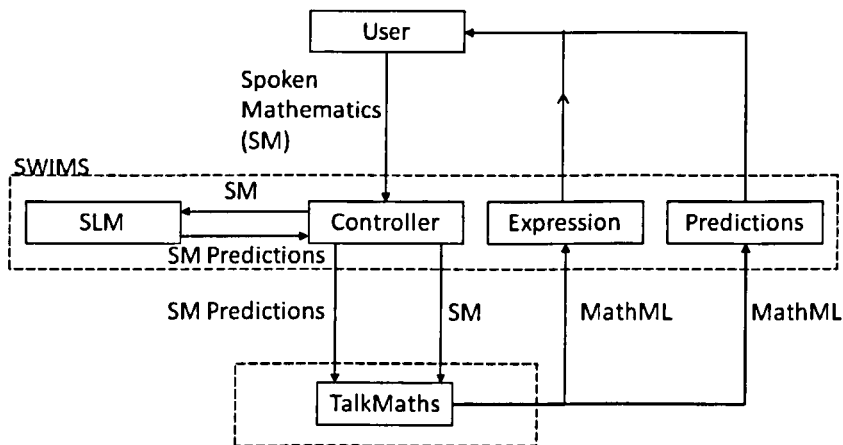


Figure 5.3: SWIMS Predictive Mathematics Interface System Architecture where “SM” refers to “Spoken Mathematics Format”



system applied to the spoken mathematics form of the formula,  $V = Ee^{-\frac{t}{RC}}$ , for the voltage across a capacitor which is being discharged through a resistor, where SWIMS has correctly predicted the next word (circled) that should come after the last word dictated/typed by the user.

**SpIMS**  
Speech-based Web Interface for Mathematics using SLM

Home Predictive Maths Alternative Maths About

$V = Ee^{-\frac{t}{RC}}$

Predict Level:  
 One Word  
 Two Words

Type or dictate your command:

Rendered Predictions:

- 1)  $V = Ee^{-\frac{t}{RC}}$
- 2)  $V = Ee^{-\frac{t}{RC}}$
- 3)  $V = Ee^{-\frac{t}{RC}}$
- 4)  $V = Ee^{-\frac{t}{RC}}$
- 5)  $V = Ee^{-\frac{t}{RC}}$

Spoken Predictions:

- 1) capital victor equals capital echo echo to the power of begin open bracket minus tango over begin capital romeo capital **charlie**
- 2) capital victor equals capital echo echo to the power of begin open bracket minus tango over begin capital romeo capital tango
- 3) capital victor equals capital echo echo to the power of begin open bracket minus tango over begin capital romeo capital sierra
- 4) capital victor equals capital echo echo to the power of begin open bracket minus tango over begin capital romeo capital bravo
- 5) capital victor equals capital echo echo to the power of begin open bracket minus tango over begin capital romeo capital alpha

Figure 5.4: Predictive Mathematics Interface in use. In the top-ranked suggestion, the SLM correctly predicts that *capital* will be followed by *charlie*

### 5.3.3 Dependence of Prediction Success Rate on Number of Alternatives Offered (Experiments A and B)

Each expression in the test set was run through the predictive mathematics interface, with the last one (Experiment A) or two (Experiment B) word(s) omitted. We then observed the next word(s) predicted by the system, to see if one of highest ranked (by probability) predictions contained the actual missing word(s). Some example expressions we used for these experiments are illustrated in Table 5.5 (Experiment A) and Table 5.6 (Experiment B). The word *end* is normally used as a *context cue* within our specialized language for spoken mathematics, resulting in it being the most common word (see Table 5.2) in our dataset. Hence, we did not test expressions ending with *end* (A) or ones which had *end* in the last two words (B). Table 5.7 illustrates the percentages of times the correct prediction was included in the list of  $M$  “best” suggestions being offered to the user, and how this varied with  $M$ . In order to check that

Incomplete Expression	Omitted next word	SWIMS predictions
alpha to the power of two plus bravo to the power of	two	<b>two</b> , begin, three, x-ray, four
begin x-ray minus one end over begin x-ray minus three end equals one plus two over begin x-ray minus	two	one, <b>two</b> , three, charlie, yankee
three delta over begin delta x-ray end open square bracket x-ray	to	close, plus, <b>to</b> , minus, index
begin delta over begin delta x-ray end end of begin charlie foxtrot end equals charlie begin delta foxtrot end over begin delta	x-ray	<b>x-ray</b> , yankee, tango, to, over
one minus two alpha plus bravo equals one minus two bravo plus	alpha	charlie, <b>alpha</b> , or, foxtrot, bravo
zero less than x-ray minus charlie less than delta index	two	<b>two</b> , one, x-ray, begin, yankee
alpha x-ray to the power of two plus bravo x-ray plus charlie less than	zero	or, x-ray, delta, three, one
alpha index november equals november to the power of begin minus begin november plus	one	<b>one</b> , alpha, begin, x-ray, papa

Table 5.5: Sample incomplete expressions we used for Experiment A, omitted word and predictions (using 5 suggestions, adding one word for each)

the results obtained were consistent, we performed 10 fold cross validation. The results of Experiment A are also represented graphically in Figure 5.5, which shows that the success of the one word ahead prediction increased as the number of suggestions shown to the user was increased, but with diminishing return. However, with 10 suggested alternatives, the system correctly predicted one word ahead over 75% of the time.

Incomplete Expression	Omitted last two word(s)	SWIMS predictions
alpha to the power of two plus bravo to the power	of two	<b>of two</b> , of begin, end over, end end, begin delta begin x-ray
begin x-ray minus one end over begin x-ray minus three end equals one plus two over begin x-ray	minus two	to the, to bravo, plus one, plus two, index zero
three delta over begin delta x-ray end open square bracket	x-ray to	times close, times open, begin integral, begin delta, foxtrot of
begin delta over begin delta x-ray end end of begin charlie foxtrot end equals charlie begin delta foxtrot end over begin	delta x-ray	<b>delta x-ray</b> , delta yankee, x-ray to, x-ray plus, two x-ray
four x-ray to the power of three plus four x-ray to the power of	two plus	<b>two end, two plus</b> , begin x-ray, begin two, three plus
foxtrot of x-ray equals x-ray to	the power	<b>the power</b> , the end, bravo of, bravo foxtrot, alpha of
alpha x-ray to the power of two plus bravo x-ray plus charlie less	than zero	than or, than x-ray, end over, end end, begin delta
alpha index november equals november to the power of begin minus begin november	plus one	minus one, minus two, <b>plus one</b> , plus alpha, index sierra

Table 5.6: Sample incomplete expressions we used for Experiment B, omitted words and predictions (using 5 suggestions, adding two words for each)



	Training Set Size	Test Set Size	Number of Suggestions				
			5	10	15	20	25
Minimum	3684	407	62%	74%	79%	85%	88%
Mean	3691.8	410.2	63.2%	77.6%	84.4%	88.9%	91.1%
Max	3695	418	66%	81%	87%	91%	94%

Table 5.7: Experiment A: Variation of success rates of one word prediction with number of suggestions offered to the user

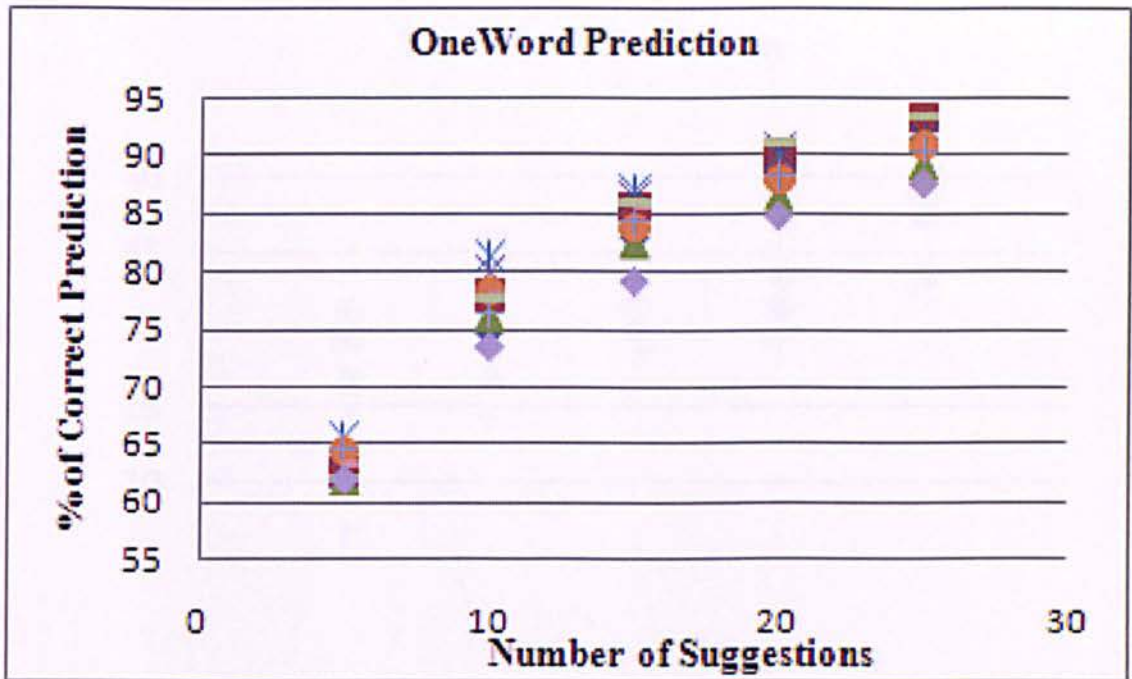


Figure 5.5: One word ahead prediction success rate increasing with the number of suggestions offered to the user. The different points corresponding to the same number of suggestions indicate the results from the 10 different cross validation trials

Experiment B evaluated the *two words ahead* prediction facility within SWIMS, in a similar manner to Experiment A. The results are summarized in Table 5.8, and shown graphically in Figure 5.6. The trend is similar to that for experiment A, but the success rates in experiment B are lower for a given number of suggestions, and it would appear unlikely that prediction success rates in excess of 50% could be achieved.

	Training Set Size	Test Set Size	Number of Suggestions				
			5	10	15	20	25
Minimum	3684	407	17%	24%	28%	29%	30%
Mean	3691.8	410.2	24.3%	30.2%	33.6%	35.2%	36.2%
Max	3695	418	31%	36%	39%	42%	44%

Table 5.8: Experiment B: Variation of success rates of two word prediction with number of suggestions offered to the user

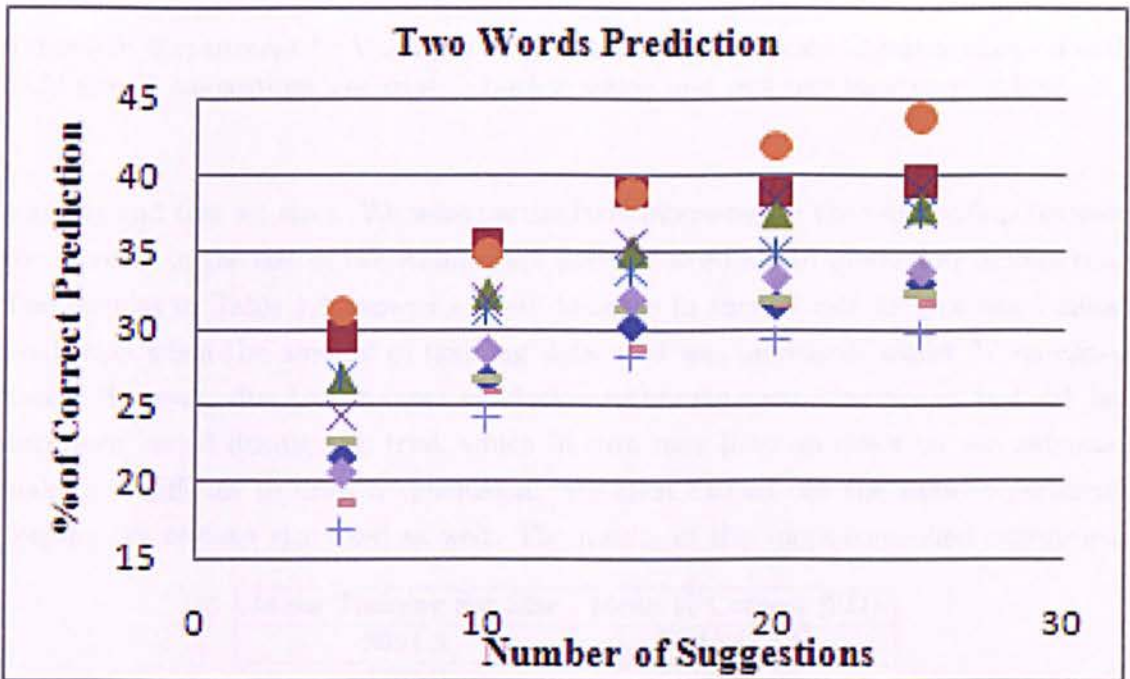


Figure 5.6: Two word ahead prediction success rate increasing with the number of suggestions offered to the user

### 5.3.4 Dependence of Prediction Success Rate on Size of Training Dataset (Experiment C)

In Experiment C, we studied how the success rate for one word ahead prediction varied as different sized datasets were used to train the SLM. The results are summarized in Table 5.9. We kept the number of suggestions offered,  $M$ , as 5 and varied the size of the

Mean Training Set Size	Mean Test Set Size	Mean % Correct (SD)
3691.8	410.2	63.10 (1.32)
3281.6	820.4	62.60 (1.69)
2871.4	1230.6	62.30 (1.58)
2461.2	1640.8	62.10 (1.60)
2051	2051	61.30 (1.62)
1640.8	2461.2	60.30 (1.49)
1230.6	2871.4	59.00 (1.31)
820.4	3281.6	56.50 (1.44)
410.2	3691.8	52.70 (1.77)

Table 5.9: Experiment C: Variation of success rate of one word ahead prediction with SLM size (5 suggestions per trial) – both training and test sets may vary in size

training and test set sizes. We were particularly interested in the relationship between the increase in the size of the training set and one word ahead prediction success rate. The Results in Table 5.9 showed a small increase in success rate for one word ahead prediction when the amount of training data used was increased, whilst  $M$  remained fixed. However, due to the cross validation technique used, the size of test set has also been varied during this trial, which in turn may have an effect on the outcome, making it difficult to draw a conclusion. We then carried out the same experiment, keeping the test set size fixed as well. The results of this more controlled experiment

Mean Training Set Size	Mean % Correct (SD)
3691.8	63.11 (1.14)
3281.6	61.36 (2.02)
2871.4	61.88 (2.57)
2461.2	62.99 (2.47)
2051	61.32 (3.16)
1640.8	59.18 (2.15)
1230.6	58.97 (3.88)
820.4	56.86 (3.97)
410.2	52.94 (3.03)

Table 5.10: Experiment C(2): Variation of success rate of one word ahead prediction with SLM size (5 suggestions per trial) keeping test set constant (400)

shown in Table 5.10 re-confirmed the relationship between training dataset size and the prediction accuracy shown by the less-controlled initial experiment.

## 5.4 Applications of Edit Distance Metrics to Error Correction

To realise correction of errors (in this case, small typing errors), in the SWIMS system, we implemented another web interface called Alternative/Corrective Mathematics. Each time an *out of vocabulary* (OOV) word is detected, the Damerau-Levenshtein algorithm [Damerau, 1964; Levenshtein, 1966] is used to calculate the Levenshtein distance of the typed word relative to each word in the vocabulary, in order to find suitable candidates for correction of the OOV word in question. Once a list of such candidates has been obtained, the Levenshtein distance and/or SLM probabilities can be used to re-rank the resulting new sequences of words. To illustrate this concept, we designed three variants of correction methods in the Alternative/Corrective Interface of SWIMS. These use Damerau-Levenshtein only, SLM only, and both in combination, respectively. In this project, we have only implemented and evaluated the first of these. In experiment D, the edit distance-based correction algorithm of the SWIMS system has been evaluated by artificially introducing a controlled selection of mistakes into *otherwise correct* expressions. Figure 5.7 illustrates the system architecture of SWIMS alternative/corrective mathematics interface. The *D-L Engine* is based on the Damerau-Levenshtein algorithm which provides alternatives for OOV words in the input.

An illustration of the SWIMS Alternative/Corrective Mathematics in use can be seen in Figure 5.8, where a user made a mistake while dictating/typing the word “plus” and this was corrected in the alternatives suggested by the system.

### 5.4.1 How Successful the Correction System at Correcting Errors (Experiment D)

In order to evaluate the performance of the correction algorithm, we artificially introduced some controlled errors each of 100 expressions (spoken mathematics form)

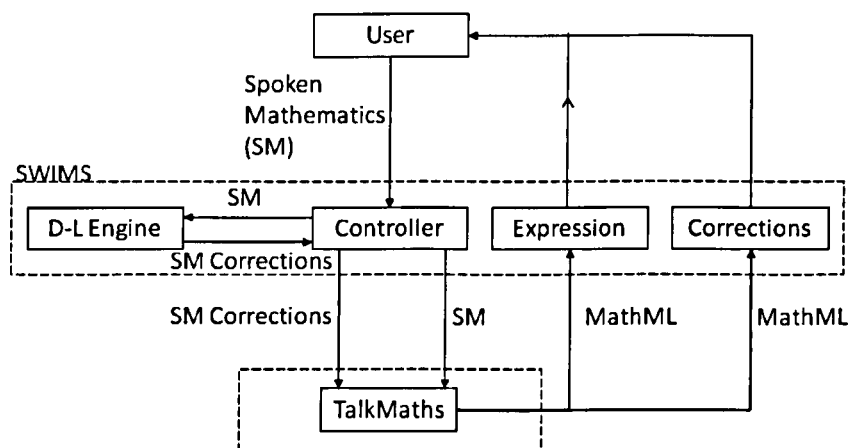


Figure 5.7: SWIMS Alternative/Corrective Mathematics System Architecture

selected from the set of test expressions, then observed the proportion of these where the *correct* version was found within the 5 top ranked alternatives offered by our correction system. This was carried out for each of introducing an extra  $R$  characters per expression, deleting  $R$  characters per expression and swapping  $R$  pairs of adjacent characters, for each of  $R = 1, 2, 3$ . Some example expressions created for this experiment are shown in Table 5.12. The percentage of expressions which were successfully corrected using this approach for each trial are shown in Table 5.11. It can be seen that our method is extremely successful in correcting up to 3 insertions or transpositions of characters per expression, and fairly successful in correcting cases where up to three characters have been deleted from an expression.

Number of Changes	1	2	3
Deletion of characters	95	92	68
Insertion of characters	100	98	97
Swapping pairs of adjacent characters	100	95	91

Table 5.11: Experiment D: Variation of success rate (%) of correction using Damerau-Levenshtein method (5 suggestions offered per trial)



**SpIMS**  
Speech-based Web Interface for Mathematics using SLM

Home Predictive Maths Alternative Maths About

$f(x) = (2x^5 - x) (3xpluck1)$

Suggested Corrections:

- 1)  $f(x) = (2x^5 - x) (3x + 1)$
- 2)  $f(x) = (2x^5 - x) (3x\alpha 1)$
- 3)  $f(x) = (2x^5 - x) (3x\alpha close 1)$
- 4)  $f(x) = (2x^5 - x) (3x\alpha equal 1)$
- 5)  $f(x) = (2x^5 - x) (3x\alpha 1)$

Correction Method \*

- Damerau - Levenshtein Only
- SLM Only (TBD)
- Damerau - Levenshtein Plus SLM (TBD)

Type or dictate your command y close bracket end of begin three x-ray pluck one end

Suggested Corrections:

- 1) foxtrot of x-ray equals begin open bracket two x-ray to the power of five minus x-ray close bracket end of begin three x-ray plus one end
- 2) foxtrot of x-ray equals begin open bracket two x-ray to the power of five minus x-ray close bracket end of begin three x-ray alpha one end
- 3) foxtrot of x-ray equals begin open bracket two x-ray to the power of five minus x-ray close bracket end of begin three x-ray close one end
- 4) foxtrot of x-ray equals begin open bracket two x-ray to the power of five minus x-ray close bracket end of begin three x-ray equal one end
- 5) foxtrot of x-ray equals begin open bracket two x-ray to the power of five minus x-ray close bracket end of begin three x-ray four one end

Figure 5.8: Alternative/Corrective Mathematics Interface in use. In the top-ranked suggestion, the OOV word *pluck* is replaced with *plus*

## 5.5 Discussion of Results

In this chapter, we first explained our method for developing and evaluating SLMs for spoken mathematics using a web-crawled spoken mathematics corpus. These evaluations were carried out by using perplexity calculations. The results showed that spoken mathematics is indeed highly predictable compared with ordinary text and the spoken mathematics corpus we created should therefore be useful. These observations justify our use of language models based on this corpus, for prediction tasks. Then we presented results from several prediction and correction experiments, namely A, B, C and D. From Experiments A and B, we observe that one word ahead and two word ahead prediction success rates can be improved by increasing the number of alternatives,  $M$ , suggested to the user. However, the rate of increase of success rate diminishes as  $M$  increases, and it would appear that the maximum possible rates are about 90% for one word prediction, but around just 40% for two word prediction. However, if the user has to read too large a number of suggestions, the cognitive load imposed will be very great. Thus, the number of options displayed must be limited. Based on our results, we propose that between 5 to 10 suggestions should be offered for one word prediction, giving success rates from 63 to 80%. However, two word prediction is rather less useful unless a large number of suggestions are presented. Experiment C showed that a small increase in success rate for one word ahead prediction could be achieved by increasing

yankee equals capital alpha x-ray to the power of begin minus three end
yankee equals capital alpha x-ray to the power of <b>bgin</b> minus three end
yankee equals <b>cpital</b> alpha x-ray to the <b>pwer</b> of begin minus three end
<b>yakee</b> equals <b>cpital</b> alpha x-ray to the <b>pwer</b> of begin minus three end
yankee equals capital alpha x-ray <b>too</b> the power of begin minus three end
yankee equals capital alpha x-ray <b>too</b> the power of <b>begins</b> minus three end
yankee equals capital alpha x-ray <b>too</b> the power <b>off</b> <b>begins</b> minus three end
yankee equals capital <b>alpha</b> x-ray to the power of begin minus three end
yankee equals <b>cpaital</b> alpha x-ray to the <b>opwer</b> of begin minus three end
<b>yaknee</b> equals <b>cpaital</b> alpha x-ray to the <b>opwer</b> of begin minus three end

Table 5.12: Experiment D: Examples of incorrect expressions derived from the same correct expression “yankee equals capital alpha x-ray to the power of begin minus three end” (by deleting, inserting and swapping characters). Words with errors are marked in bold font

the amount of training data used, whilst keeping  $M$ , the number of suggestions offered to the user, fixed. This is consistent with other studies of the predictive power of models based on other types of text [Jelinek, 1991; Kneser and Ney, 1995].

Finally, according to the results we obtained in Experiment D, the Damerau-Levenshtein based correcting method is highly successful at correcting up to 3 character-level errors in an expression. However, when the number or complexity of such errors is increased, the efficiency of correction declines. At present, the suggested corrections are limited to words within the vocabulary of the SLM. This implies that each time a valid new word (in our case, a spoken name of a mathematical entity) is encountered it will need to be added to the system’s vocabulary. This should be relatively straightforward for the Damerau-Levenshtein based method. However, in order to modify the SLM, the corpus of mathematical expressions will have to be extended to reflect the change. Although possible in principle, this is not straightforward as one would have to find a considerable additional amount of data in order to update the model. Online learning within an adaptive system may be the solution to this issue, however we do not investigate this further here.

Our work to date has indicated that the prediction/correction assistive facilities incorporated into SWIMS have potential to help make mathematical editing systems, including TalkMaths, more powerful and user-friendly. Improving such systems in this manner should in turn make writing and editing mathematics in electronic documents

much easier, particular for three target groups – the disabled, on-line (particularly *at a distance*) learners and people relying heavily on the use of portable devices – for whom these tasks are currently very difficult or even near impossible.

## 5.6 Summary

As seen in Section 5.2.3, our perplexity experiments have indicated that spoken mathematics seems to be relatively predictable, so SLMs should aid in its prediction and correction. In order to investigate this further, we carried out three prediction experiments in Sections 5.3.3 and 5.3.4 and the results of these confirmed our hypotheses on SLM-driven prediction of spoken mathematics. The results of the first three experiments suggested that, while one word ahead prediction is efficient and can be improved by increasing the size of the training dataset used, two words ahead prediction was rather poor. The fourth experiment was an error correction experiment using a method based on the Damerau-Levenshtein distance method, where character level errors were artificially introduced into otherwise correct spoken mathematics expressions. We observed that the correction success rate declines as the errors become more complex, suggesting that Damerau-Levenshtein distance method is only useful in correcting relatively small numbers of errors in spoken mathematics expressions. Our method should be easily adaptable to use in several other closely related domains. For example, computer algebra systems such as Maple and Mathematica have their own language and syntax for mathematical expressions. It should be fairly straightforward for our predictive and corrective methods to be integrated with these systems, assuming that enough data on past usage is available to train the necessary models. In summary, both these predictive and corrective methods can be used to assist users of TalkMaths and possibly other systems to create and edit mathematical text efficiently.



## Chapter 6

# Error Recovery Strategies for Parsing Transcribed Spoken Mathematics

### 6.1 Introduction

The art of compiler construction, essential to the success that modern computer science has experienced, was developed throughout the 1960s and 70s and most of its techniques are well understood [Aho and Ullman, 1972; Aho et al., 1986, 2007]. Its main use is to design tools such as compilers or interpreters that convert source code to object and/or executable code. The ease by which these tools are available nowadays has led to a proliferation of programming, markup and other formal languages. Usually, the functionality of a compiler is divided into several phases: lexical analysis (carried out by a *lexer*), syntax analysis (the corresponding tool is called a *parser*), semantic analysis and code generation.

Apart from writing compilers, some of these phases are also relevant when designing *structure editors* (sometimes also referred to as *language-sensitive editors* in the literature). A structure editor is an editor that is aware of the underlying document structure and supports the user to maintain document integrity while entering or manipulating the document. Creating such a structure editor, that is efficient and user-friendly, is commonly regarded as being difficult. Most commercial tools for developing computer

code (so called *Integrated Development Environments, IDEs*) do not offer the same level of functionality as a structure editor.

More recently, the rise of web-based interfaces and the need to manage vast amounts of information available on the internet has led to another challenge: more and more often, users want to use natural language to phrase queries for search engines and the use of speech input is becoming more mainstream (as we have already discussed in Chapter 3). Parsing natural language is still a difficult problem, and traditional context-free parsing algorithms [Chomsky, 1957, 1965] are not powerful enough. Hence, command languages that are based on a relatively small subset of natural language might be a viable option.

Our work on the TalkMaths system, as it is relevant to this thesis, effectively touches all these aspects. TalkMaths is a web-based structure editor, that accepts natural language commands in order to control the editing process. We will assume that spoken mathematics commands have been recognised by the ASR, or perhaps have been directly typed into a suitable user interface. As a consequence, we have a string of *transcribed spoken mathematics* which we would like to further analyse with the ultimate goal of displaying it in our system.

In this chapter, we first investigate some properties of previously suggested command languages for spoken mathematics. We then propose an improved framework based on the idea of speech templates. Next, we elicit technical challenges that arise when designing and implementing a robust parser that is able to analyse our language. A framework is developed that enables us to use mixfix operators in order to construct our input language. We then use operator precedence parsing for the syntactical analysis of this language. In order to deal with ambiguities at lexical level, we adapt the XGLR parsing algorithm given in Begel and Graham [2006] to our operator precedence setting. Finally, we devise robust error recovery strategies for our parser.

We have fully implemented our parsing approach as part of a new prototype version of TalkMaths, denoted by  $P_3$  in Chapter 7, where we have provided an evaluation of our implementation, that documents the resulting improvements. The error recovery strategies have been partially implemented with an emphasis on treating incomplete input (as this seemed the most frequently occurring error).

## 6.2 Spoken Mathematics as Input Language

In this section, we present the main characteristics of the language for spoken mathematics that we use in the TalkMaths system and for which we propose a parser and error recovery strategies.

Our language consists of a set of spoken forms for operators and operands. This approach is not new and arguably, most previous attempts, as explained in Chapter 2, follow the same scheme. However, we improve several aspects of spoken mathematics, as explained in the remainder of this section.

### 6.2.1 Ambiguity

As already stated in the literature [Fateman, 2009; Wigmore, 2011], spoken mathematics often contains ambiguity. Precedence rules, normally expected to be known by mathematically trained users, can help with resolving this ambiguity. For example, if we hear “a plus b over c”, either  $a + \frac{b}{c}$  or  $\frac{a+b}{c}$  could be meant. If a display of the expression was available – perhaps in a teaching session or during a mathematical talk – there would be no confusion. Otherwise, if we need to interpret the transcribed spoken mathematics without any additional clues, our system should follow the rule that division (the fraction operator) binds more strongly than addition. Hence, following this precedence rule, we would decide on the first expression rather than the second.

However, this is not always an obvious solution. Consider the transcribed spoken mathematics “square root of b squared minus a”. This could either be interpreted as  $\sqrt{b^2} - a$  or  $\sqrt{b^2 - a}$ . No obvious operator precedence rule would help in this example. There is no common standard that indicates exactly how much input is supposed to be the argument of the root function.

The approach taken by Fateman [Fateman, 2009] is to use additional “locutions” (spoken delimiter/marker commands) to clearly indicate boundaries between arguments. The first expression would be spoken as “square root of b squared all minus a”. Here, the term “all” acts as an “invisible” end marker in order to separate the square root function from the subtraction operation. An additional construction (“quantity”) implements a corresponding begin marker. The expression  $a + \frac{b+c}{d}$  for example would be dictated as “a plus quantity b plus c all over d”. Note that these commands could

also be used as a spoken form for brackets, as for example in the input “a plus b all times c” which is the spoken form of  $(a + b) \times c$ .

The emphasis on the simple design of spoken mathematics in [Fateman, 2009] is on simplicity and convenience for the user. For these reasons, the author does not consider the use of additional commands other than “quantity” and “all”, although he acknowledges that this might lead to artificial examples, see the example with “all all” in Section 6.2.2.

### 6.2.2 Speech Templates

We propose an extension of this scheme, the use of *speech templates*<sup>1</sup>. Speech templates are (usually, relatively short) spoken natural language commands. A speech template consists of one or several groups of words. Between each group, any other spoken language elements (including speech templates) might occur. Effectively, these groups act as “boundaries”. Hence, a speech template can take zero or more arguments and can be nested.

For example, suppose we would like to speak the expression  $(a + \frac{b}{c+d}) \times e$  (see Section 3.6 in Fateman [2009]). Using his approach, we would say “a plus b over quantity c plus d all all times e”. A speech template for fractions could be defined by

```
"fraction" .. "over" .. "end fraction"
```

Using this speech template, the expression is now pronounced as “a plus fraction b over c plus d end fraction all times e”. This appears more natural, as it avoids the repeated “all all” construction.

Speech templates without any arguments can also make sense. These consist of a single command such as “edit expression” or “what can I say”. Such commands can be useful, for example, in a system that offers a user interface with additional functionality, such as editing mathematics expressions or displaying help information.

As an example of a speech template with more than two arguments, we define a speech template for speaking or typing integrals as:

```
"integral from" .. "to" .. "of" .. "end integral"
```

---

<sup>1</sup>This is not to be confused with the type of acoustic templates used for pattern matching in speech signal processing.

The expression  $\int_{x=a}^b (f(x) + g(x)) dx$  would be spoken as “integral from x equals a to b of f of x plus g of x all d x end integral”. Table 6.1 illustrates some currently used and potential examples of speech templates.

Speech Template	Arguments
edit expression	0
clear expression	0
select numbers	0
integral of .. end integral	1
third root of .. end root	1
to the power of .. end power	1
edit .. end	1
begin .. end	1
function .. of .. end function	2
fraction .. over .. end fraction	2
integral from .. to .. of .. end integral	3
sum from .. to .. of .. end sum	3
matrix row .. and .. row .. and .. end matrix	4
limit of .. of .. as .. approaches .. end limit	4
matrix row .. and .. and .. row .. and .. and .. end matrix	6

Table 6.1: Examples of speech templates with different number of arguments

Finally, we remark that there are still other aspects of ambiguity in spoken mathematics which seem difficult to deal with, and we refer to Fateman [2009] for a discussion.

### 6.3 Parsing Challenges

Before presenting our parsing approach, we illustrate the challenges that typically arise during the syntax analysis of transcribed spoken mathematics or, potentially, any spoken command language for creating structured content. Essentially, we have to deal with lexical ambiguity, incomplete or syntactically incorrect input, as well as any combination of these.

### 6.3.1 Lexical Ambiguity

Ambiguity at the lexical level may arise for two reasons. In the context of spoken input, ambiguity is caused by the presence of homophones in natural languages (the English language in our case)<sup>1</sup>. For example, both “four” and “for” sound the same and an ASR might not be able to identify the correct utterance. Hence, the lexer will be misled by this type of ambiguity. Furthermore, due to its design, any formal language may contain ambiguity, and this is also the case in our language for spoken mathematics.

In the context of lexing, [Begel and Graham, 2006] classify four kinds of input streams.

- (i) Single spelling, single lexical type.
- (ii) Multiple spellings, single lexical type.
- (iii) Single spelling, multiple lexical types.
- (iv) Multiple spellings, multiple lexical types.

In this classification, the last three are ambiguous. Here, multiple spellings indicates the phenomenon of ambiguity due to homophones and multiple lexical types refers to ambiguous language.

### 6.3.2 Incomplete Input

We refer to incomplete input as syntactically incorrect input, that can be made correct by inserting additional content. Due to the input being a spoken command language, we will frequently encounter such incomplete input primarily due to the following two aspects.

- (i) The fact that all input originates from utterances that are spoken by the user. In general, humans find it easier to break down complex instructions into smaller chunks as it reduces cognitive load [Christian et al., 2000]. We assume here that the command language is designed in such a way that this “chunking” is permitted.

---

<sup>1</sup>This is even more likely to happen in some other natural languages, for example, in French.

- (ii) The desire to omit keywords in the input, as long as this does not create semantic problems. We have already seen in the context of spoken mathematics that specific keywords are necessary in order to resolve ambiguity. However, especially in reasonably short utterances, they might become optional. It greatly alleviates the physical and cognitive effort needed to use the system, if they can be omitted.

Let us give some examples to illustrate these two aspects. Firstly, the utterance “a plus”, followed by “square root of b” together form the syntactical correct expression  $a + \sqrt{b}$ . This is only known after the second command, and the parser needs to tolerate the incomplete first command.

In order to better understand the second aspect, consider the spoken command “begin a plus b end times c”. Here, the term “begin” can be omitted without any resulting ambiguity. Effectively, it is an optional term and users would probably prefer not to speak it.

### 6.3.3 Incorrect Input

In addition to incomplete input, as discussed in the previous section, one could observe other types of general syntactically incorrect input. In the context of spoken input we can identify the following main reasons for this:

- (i) Misrecognition – errors of the ASR could lead to violation of the syntax in the input. This introduces invalid or erroneous commands, words or characters.
- (ii) Human error – users might make errors due to negligence or insufficient knowledge of the command language, or due to hesitation or speech impediment, with similar consequences as above.

Often, the parser could simply ignore input such as “\$ # &” (assuming that we have a means of inputting these special characters). On the other hand, the input “plus plus plus” is at the borderline of an incomplete expression. The input “square root root of three” however leaves us wondering what sensible actions the system could take.

## 6.4 Parsing Framework

In order to parse and display the input to the TalkMaths system, we proceed as in standard compiler construction. We recall the following phases [Aho et al., 1986]:

- (i) *Lexical Analysis*: this is the first step in analysing the input. A tool (the *Lexer*) scans the input and converts it into meaningful entities (*tokens*). For input languages such as programming languages, this is normally a straight forward task. Tokens are defined using regular expressions, and this drives a pattern matching process.
- (ii) *Syntax Analysis*: a continued analysis, based on the token stream that is the output of the lexical analysis. The main goal in this phase is to construct a *parse tree*.
- (iii) *Semantic Analysis*: in this phase, the information in the parse tree, together with additional semantic rules, is used for various semantic tasks. For example, type checking of tokens in programming languages is done at this phase.
- (iv) *Code Creation*: finally, executable code is generated by using all the information and data structures that were created in all the previous phases.

In this chapter, we focus on the first and second phase. The semantic analysis and code creation phases are not explicitly covered by the work in this thesis but are under ongoing investigation within the TalkMaths research project [Wigmore, 2011].

### 6.4.1 Underlying Grammar

In this section, we explain our framework for parsing our language for spoken mathematics, denoted by  $\mathcal{M}$ . The idea is to map this language into an equivalent language  $\mathcal{L}$ , which consists of expressions containing operators and operands. By “equivalent” we mean that there is a one-to-one correspondence of elements in  $\mathcal{M}$  to elements in  $\mathcal{L}$ . Lexical analysis converts equivalent elements into the same token stream, and hence this will result in the same syntax tree.

In order to realise this map, we use operators in their most general form: mixfix operators [Danielsson and Norell, 2011] (referred to as *distfix* in [Annika, 1995]), that



are a generalisation of prefix, postfix and infix operators. Mixfix operators are used in specialised programming languages such as PROLOG [Clocksin and Mellish, 1984], ML [Paulson, 1996] and Agda [Bove et al., 2009]. For our work, we consider the special class of *closed* mixfix operators. These operators consist of a sequence of *operator parts*, enclosing *holes*. Each hole in turn contains another valid expression, which may contain other mixfix operators. The notation for such an operator of *arity* (number of operators)  $n$  is  $op_1 - op_2 - \dots - op_n$  [Danielsson and Norell, 2011] where we also allow the case of arity 0, i.e. no internal holes at all. This makes it evident that speech templates are in fact spoken forms for closed mixfix operators.

The language  $\mathcal{L}$  is recognised by the following grammar  $G = (N, \Sigma, P, S)$  where  $E$  is a symbol,  $N = \{E\}$  is the set of non-terminals,  $\Sigma$  contains the terminal symbols,  $P$  is a set of productions (grammar rules), as shown below, and  $S = E$  is the start symbol:

$$\begin{aligned}
 E &\rightarrow a && (R_1) \\
 E &\rightarrow E \star E && (R_2) \\
 E &\rightarrow \circ E && (R_3) \\
 E &\rightarrow E \bullet && (R_4) \\
 E &\rightarrow (E) && (R_5) \\
 E &\rightarrow (E \parallel E) && (R_6) \\
 &\vdots && \\
 E &\rightarrow (E \parallel \dots \parallel E) && (R_m)
 \end{aligned}$$

In this grammar, the rules  $R_2$ – $R_5$  are specifying expressions containing unary pre- and post-fix, or binary operators. Rule  $R_1$  yields constants (operator-free expressions), and rules  $R_6$ – $R_m$  (and, strictly speaking, also  $R_1$ ) are concerned with specifying valid speech templates. For example, in order to create the integral speech template presented in Section 6.2.2, we would have  $m = 8$  as the spoken integral template example corresponds to a closed mixfix operator with three holes, which would be covered by rule  $R_8$ .

In order to simplify notation, we state these rules generically:  $a \in \Sigma$  denotes any alpha-numerical character or a number, which are operands;  $\star, \circ, \bullet \in \Sigma$  are any binary, prefix and postfix operators respectively and any arbitrary closed mixfix operator is written using the open operator “(”, optionally, the argument separating operators

“||”, and the close operator “)”.

By design, this grammar is a context-free grammar. Moreover, since there are no productions with two adjacent non-terminals on their right side, it is an *operator grammar* [Aho et al., 1986, Section 4.6]. It is well known that operator grammars, together with precedence and associativity rules, can be analysed using an efficient parsing technique called *operator-precedence parsing*.

In [Annika, 1995], this has been done for a grammar which ours is a subset of. From this work, we deduce precedence relations between operators and operands as in Tables 6.2 and 6.3. This information will be used in the parsing algorithm as explained in the next section.

	*	<i>a</i>	(		)	\$
*		<	<	>	>	>
<i>a</i>	>	$E_1$	$E_2$	>	>	>
(	<	<	<			$F_1$
	>	<	<			$F_2$
)	>	$E_3$	$E_4$	>	>	>
\$	<	<	<	$F_3$	$F_4$	$S$

Table 6.2: Operator Precedence Table

Table 6.2 is a simplified representation of the complete precedence matrix, which is obtained by grouping entries of identical precedences together. The generic operator \* represents the operator classes  $\star, \circ, \bullet$ . The \$ symbol is an additional symbol. The top left missing entry needs expanding into a refined precedence matrix, detailing the precedences that exist between the various prefix, infix and postfix operators. The values  $E_1$  to  $E_4$  and  $F_1$  to  $F_4$  are error entries, which will be used during error recovery. The remaining empty entries need to be completed according to the following table.

	<sub><i>i</i></sub>	) <sub><i>i</i></sub>		<sub><i>j</i></sub>	) <sub><i>j</i></sub>
( <sub><i>i</i></sub>	<	=	( <sub><i>i</i></sub>	$G_1$	$G_2$
<sub><i>i</i></sub>	>	>	<sub><i>i</i></sub>	$G_3$	$G_4$

Table 6.3: Additional Precedence Tables for Mixfix Operators

The first table expresses the precedence relation between the same mixfix operator

parts, whereas the second table represents those of different operators. The additional entries  $G_1$  to  $G_4$  are also for error recovery, more precisely, they are used in Section 6.6.4.

We will have the convention that the closed mixfix operators that correspond to our speech templates have a higher precedence than any other operator (prefix, infix or postfix). Furthermore, all closed mixfix operators have the same precedence and are right associative.

### 6.4.2 Operator Precedence Parsing

In this section, we explain the standard shift-reduce operator precedence parsing algorithm as described in [Aho et al., 1986]. It is based on bottom-up, shift-reduce parsing and uses an operator precedence table to decide on the parsing action at each step. During the algorithm, the precedence between the token at the top of the stack and the current input token is determined. Depending on its value, either the current token is shifted onto the stack (if the precedence is “=” or “<”), the stack is reduced (precedence “>”), the algorithm terminates or error recovery is triggered.

The following description of the algorithm using pseudocode follows closely the presentation of Algorithm 4.5 in [Aho et al., 1986]. The stack is initialised with the \$ symbol. The input, appended by “\$”, is contained in the variable  $w$ ,  $ip$  points to the current symbol in the input and precedences between token values stored in the variables  $a$  and  $b$  drive the algorithm as explained in the previous paragraph. After execution of the algorithm, the output queue contains the *reverse polish notation* of the input. It is straightforward to create a parse tree from this – we will detail this further in the context of error recovery in Section 6.6.5.

## 6.5 Dealing with Lexical Ambiguity

In Section 6.3, we have identified the main challenges that our parsing framework has to respond to. We now tackle the first of these challenges, lexical ambiguity.

The general task of a scanner (or lexer) is to take care of the first stage of the syntax analysis. Usually this is based on matching token definitions represented using regular expressions with parts of the input, hence identifying lexemes. The output of this process is a token stream. Each token contains information about its lexical type,

---

**Algorithm 1:** Standard Operator Precedence Parsing Algorithm

---

```

set ip to point to the first symbol of w;
while forever do
  if $ is on top of the stack and ip points to $ then
    return
  else
    let a be the topmost terminal symbol on the stack and let b be the
    symbol pointed to by ip;
    if if  $a < b$  or  $a = b$  then
      push b onto the stack;
      advance ip to the next input symbol;
    else if  $a > b$  then
      repeat
        pop the stack into output queue;
      until the top stack terminal is related by  $<$  to the terminal most
      recently popped;
    else
      error();

```

---

and a reference to an appropriate data structure (symbol table) that remains accessible during later stages (for example, syntax analysis).

For example, a spoken command “a times fraction b plus c over d end fraction” that uses the fraction speech template would result in the following token stream:

$$(id, 1), (op, 2), (open, 3), (id, 4), (op, 5), (id, 6), (arg, 7), (id, 8), (close, 9)$$

Each token is defined by two entries: the token type, and the reference to an entry in the symbol table (see Table 6.4). We use generic types such as ‘op’, ‘id’ for operators and operands. By following the symbol table references, additional information can be retrieved – in this example, the third row of the symbol table stores an identifier for the fraction speech template, indicating that the token (*open*, 3) was created from the “fraction” lexeme.

Type	Lexeme	Position	..
<i>id</i>	“a”	1	..
<i>op</i>	“times”	2	..
<i>open</i>	“fraction”	3	..
<i>id</i>	“b”	4	..
<i>op</i>	“plus”	5	..
<i>id</i>	“c”	6	..
<i>arg</i>	“over”	7	..
<i>id</i>	“d”	8	..
<i>close</i>	“end fraction”	9	..

Table 6.4: Symbol table for spoken command “a times fraction b plus c over d end fraction”. The right most column contains additional information about each entries (for example, if the entry is the type of *op*, all possible operator classes it could take: i.e.  $\star, \circ, \bullet$ )

### 6.5.1 XGLR Approach

The solution we have implemented for resolving lexical ambiguity is based on Begel’s XGLR framework [Begel and Graham, 2006], developed in the context of spoken programming languages, which in turn builds on GLR parsing [Lang, 1974; Rekers, 1992; Tomita, 1985].

GLR parsing is a knowledge-based, non-deterministic parsing method that can deal with ambiguities by creating all possible parse trees. In order to improve efficiency, common data structures should be shared. Inspired by Begel’s work, we have implemented this using an object oriented approach, where instead of backtracking multiple parses we have a lexer object that can fork into several instances whenever it encounters ambiguity. This is then propagated into the parser.

An extension of GLR parsing, that can handle ambiguous lookaheads arising from spoken input (due to multiple spellings, as mentioned in Section 6.3.1), is introduced in [Begel and Graham, 2006] as XGLR parsing. The mechanism is fairly similar to that in GLR parsing: whenever lexemes do not uniquely determine a token, the lexer explores all possibilities. This can also be implemented by forking the lexer object.

### 6.5.2 Lexer Algorithm

We now present our version of the lexer that implements the lexical analysis phase within the adapted XGLR framework. It is implemented as an object, remembering a certain state and offering a method `doWork()` which is illustrated below. This method is called repeatedly and, depending on the current state, scans the input for the next token (which might be ambiguous) or handles the case of ambiguous tokens by creating copies of itself for each possible token alternative. Initially, `self.state` is set to `new_token`. A list of currently active lexers is maintained and their respective `doWork()` methods are called successively. Effectively, we handle this process in a very similar way as it is explained in Begel's paper.

---

#### Algorithm 2: Lexer – `doWork()`

---

```

if self.state = check_ambiguity then
  if token is ambiguous then
    self.state = fork;
  else
    self.state = new_token;
else if self.state = fork then
  foreach non-ambiguous token in current token do
    create a copy of Lexer object where current token is replaced with
    non-ambiguous token;
    append new Lexer object into list of children;
  self.state = inactive;
else if self.state = new_token then
  if currentToken = endToken then
    self.state = terminate;
  else
    Scan input and return next (potentially ambiguous) token;

```

---

We give some examples in order to illustrate how the lexer proceeds.

The input “minus a” is ambiguous since the minus operator can be either unary prefix (the minus sign) or binary infix (the subtraction operator). Hence, two tokens will have to be considered during the lexical analysis. Eventually, the following two token streams are generated:

(*op*, 1), (*id*, 2)

and

$$(op, 3), (id, 2).$$

Note that here, the first tokens are different and the symbol table will store the information on the two different operators separately (in row 1 and 3). We remark that the token stream that corresponds to the binary subtraction operator represents a syntactically incorrect expression as there is a missing first argument. We will see later how this will be resolved during the error recovery.

As a second example, consider the intended input “x plus four” which the ASR misinterpreted as “x plus for”, and let us also assume the existence of a speech template starting with “for”. The lexer will return the token streams:

$$(id, 1), (op, 2), (id, 3)$$

and

$$(id, 1), (op, 2), (op, 4)$$

The difference is the interpretation of the third token as either an identifier (the numerical value 4 to be precise) or an initial operator word of the speech template.

### 6.5.3 Parser Algorithm

The parser algorithm is a slight extension of Algorithm 1 which takes into account the possibility that several instances of the lexer might be created, as discussed in the previous section. We have implemented this in a similar fashion, using objects, as in the lexer. At the beginning of the main loop, the parser calls the lexer’s `doWork()` method. After a successful return from this method, if we detect the presence of additional new lexer objects, we propagate this into the parser by creating a new child parser object.

## 6.6 Error Recovery Strategies for XGLR Parsing

Having addressed ambiguity at lexical analysis level, we continue to address the remaining parsing challenges of Section 6.3. We will see that this requires substantially more work, involving both the lexical and syntax analysis phases and suitable, robust

error recovery strategies. The task will be to parse the entire input, without rejection, followed by correcting as much as we can, and only ignoring input if really necessary. Some of this error recovery happens already during lexical analysis, whereas more “difficult” cases need to be addressed during the parsing.

### 6.6.1 Classification of Errors

In order to state precisely what the error recovery needs to cope with, we give a classification of possible errors that we can encounter. We include incomplete input in our discussion.

- (i) Missing operators and operands – we exclude mixfix operators from these considerations.
- (ii) Incomplete mixfix operators – the fact that one or several operator parts are missing.
- (iii) Interlaced mixfix operators – here, matters are made more complicated by the fact that operator parts from different mixfix operators appear “merged together” in the input stream.
- (iv) Shuffled mixfix operators – these are mixfix operators with correct open and close operator parts, containing an incorrectly permuted sequence of argument separator parts.
- (v) Mixfix operators with redundancy – multiple occurrence of the same argument separator parts.
- (vi) Otherwise incorrect mixfix operators – containing wrong identifiers that are not valid tokens.
- (vii) Any combination of any of the previous errors.

We now give some typical examples that are representatives of these error classes:

Class (i): “a b”, or “a plus”

Class (ii): “fraction a”, or even “fraction”



Class (iii): “function  $f$  over  $x$ ”

Class (iv): “integral  $f dx$  to  $b$  from  $a$  end”

Class (v): “function  $f$  of  $x$  of  $y$ ”

Class (vi): “fraction  $a$  hello  $b$ ”

### 6.6.2 Main Idea

The main idea for our parsing error recovery algorithm is as follows. Errors of type (i) can be dealt with at lexer level, based on the classical approach of inserting tokens [Burke and Fisher, 1987]. The remaining errors (ii) – (vii) are corrected in two stages: first, we parse an augmented language  $\mathcal{L}^*$ , followed by running a tree manipulation algorithm. Before we detail these steps in the following sections, let us define precisely  $\mathcal{L}^*$  by introducing some additional notations.

We shall denote by  $\mathcal{O}$  the set of mixfix operators that are contained in our language  $\mathcal{L}$ . Let  $\text{open}(m)$  ( $\text{close}(m)$  respectively) denote the initial (last respectively) operator parts of a mixfix operator  $m \in \mathcal{O}$ . Also,  $\text{args}(m)$  is a new mixfix operator, formed by the remaining, intermittent operator parts of  $m$ . We adopt the usual notation  $\text{open}(\mathcal{O}) := \{\text{open}(m) | m \in \mathcal{O}\}$  for the combined set of initial operator parts of all mixfix operators, and accordingly for  $\text{close}(\mathcal{O})$  and  $\text{args}(\mathcal{O})$ . Having this terminology in place, we can now characterise the augmented set of mixfix operators  $\mathcal{O}^*$  by

$$\mathcal{O}^* = \{o - a - c \mid o \in \text{open}(\mathcal{O}), a \in \text{args}^*(\mathcal{O}), c \in \text{close}(\mathcal{O})\}$$

and

$$\text{args}^*(\mathcal{O}) = \{a_0 - a_1 - \dots - a_n \mid a_i \in \text{args}(\mathcal{O}), n \in \mathbb{N}\}$$

The augmented language  $\mathcal{L}^*$  is  $\mathcal{L}$  together with all mixfix operators that are additionally contained in  $\mathcal{O}^*$ .

### 6.6.3 Error Recovery at Lexer Level

As already indicated in the previous section, we can deal with missing operators (other than mixfix) and operands at the level of lexical analysis. We insert missing operators

by using an appropriately defined *default invisible operator*  $\star_d$ . In the case of spoken maths, for example, this would be the “invisible times”. Missing operands are inserted using an *empty identifier*  $id_?$ . Due to the special property of the input language being recognised by an operator grammar, we can implement this based on a decision that involves comparing adjacent tokens. This is made explicit in Table 6.5, by stating the token which will be inserted between tokens that correspond to those in the row and column positions. The corresponding algorithm is given below. It takes as input the

	\$	$id$	$\star$	$\circ$	$\bullet$	(		)
\$	$id_?$		$id_?$		$id_?$			
$id$		$\star_d$		$\star_d$		$\star_d$		
$\star$	$id_?$		$id_?$		$id_?$		$id_?$	$id_?$
$\circ$	$id_?$						$id_?$	$id_?$
$\bullet$		$\star_d$		$\star_d$		$\star_d$		
(			$id_?$				$id_?$	$id_?$
	$id_?$		$id_?$				$id_?$	$id_?$
)		$\star_d$		$\star_d$		$\star_d$		

Table 6.5: Token Insertion Rules

last and current token and returns the token that needs to be inserted, otherwise, none

---

**Algorithm 3:** Lexer Error Recovery Algorithm – lexerErrorRecovery()

---

```

Lookup action in Table 6.5 by indexing with (lastToken, token);
if action =  $id_?$  then
    // Default empty operand to be inserted insertedToken =  $id_?$ ;
else if action =  $\star_d$  then
    // Default invisible operator to be inserted insertedToken =  $\star_d$ ;
else
    insertedToken = None;
return insertedToken;

```

---

This simple approach yields already quite powerful error recovery. We now give some examples which demonstrate this.

Input	Error recovery output
$abf(c + d)$	$a \star_d b \star_d f(c + d)$
$f(+d) \star$	$f(id_? + d) \star id_?$
$abf(+d)$	$a \star_d b \star_d f \star_d (id_? + d)$
$f(, x)y$	$f \star_d (id_?, x) \star_d y$

Table 6.6: Lexer Error Recovery Examples

### 6.6.4 Error Recovery at Parser Level: Stage 1

Let us remind that an attempt to parse incorrect input that contains any of the error types classified in Section 6.6.1 will mean that the operator precedence parser encounters, at some stage, one of the error entries  $E_1$  to  $E_4$ ,  $F_1$  to  $F_4$  or  $G_1$  to  $G_4$  given in Tables 6.2 and 6.3. One can easily see that the error recovery actions undertaken by the lexer as explained by the previous section eliminate the need to consider the entries  $E_1$  to  $E_4$ , and we can focus on addressing the remaining error entries.

We first consider the entries  $F_1$  to  $F_4$ . For each of these error entries, the “\$” symbol is involved which, after analysing the shift-reduce algorithm, implies that for error entries

- $F_1, F_2$ : the input has run out of symbols whilst we still have the mixfix operator words “(” or “|” on the stack. This is the case if a “)” is missing in the input.
- $F_3, F_4$ : the stack is empty, possibly after some reduce actions, and we encounter “|” or “)”. Here, we have a missing “(”.

We conclude that these error entries are concerned with errors in error class (ii) in Section 6.6.1. However, this does not cover all possible elements of this class. For example, in both cases above, there could also be missing “|” operator parts in the input. We will continue to deal with the same error class in Stage 2 of our error recovery.

In order to carry out suitable error recovery actions, we proceed as follows:

- Error entries  $F_1, F_2$ : since we have reached the end of the input, the only possible action is a reduce. This will remove elements from the stack into the output. Note that we will have to accept incomplete information. In order to construct a valid

parse tree, more work will have to be done. This is subject of the algorithm in Stage 2.

- Error entries  $F_3, F_4$ : using a similar argument, since the stack is empty, we carry out a shift. This will move tokens from the input into the stack and we are led to a similar situation as in previous case.

This leads to the following adjusted precedence table, where the error entries  $F_1$  to  $F_4$  have been replaced with precedence values that correspond to the discussion above.

	*	$a$	(		)	\$
*		<	<	>	>	>
$a$	>	$E_1$	$E_2$	>	>	>
(	<	<	<			>
	>	<	<			>
)	>	$E_3$	$E_4$	>	>	>
\$	<	<	<	<	<	$S$

Table 6.7: Adjusted Operator Precedence Table

We continue our analysis of error recovery actions by examining the error entries  $G_1$  to  $G_4$ . Rather than being able to give unique shift/reduce actions, we will see that we have to use (again) nondeterministic parsing.

- Error entry  $G_3$ : here, we can carry out a reduce action. In order to justify this, we recall that our aim is to parse the augmented language  $\mathcal{L}^*$ . By definition, this language consists of mixfix operators whose argument operator parts are the union of all possible such operator parts, in any order and repeated occurrences. This means that for this error entry, we can always accept the “generic” argument separator “||” by reducing it into the output. This will address the error class of interlaced mixfix operators.
- Error entries  $G_1, G_2$  and  $G_4$ : one can easily find examples (such as the input “(]”, which leads to  $G_2$ ) which show that shift and reduce are both equally meaningful possible actions. Inspired by the XGLR framework, we solve this by forking new parsers, which will carry out each of the actions.

This results in the following table, where we have used the symbol “<>” to indicate a shift-reduce conflict.

	<sub>j</sub>	) <sub>j</sub>
( <sub>i</sub>	<>	<>
<sub>i</sub>	>	<>

Table 6.8: Precedence Tables for Interlaced Mixfix Operators

### 6.6.4.1 The Parser Algorithm

We now give our parser algorithm including Stage 1 of the error recovery in pseudo-code.

### 6.6.5 Error Recovery at Parser Level: Stage 2

After carrying out the error recovery algorithm of the shift-reduce parser in Stage 1, we have created output data which can be converted into a syntax tree<sup>1</sup>. In our implementation, we create this syntax tree on-the-fly during each reduce step. The outcome is a partial syntax tree – we cannot completely construct complete nodes for mixfix operators at this stage. Such a mixfix operator node stores information on its arity, the individual operator parts and is turned into a tree by pointing to a list of child trees which correspond to expressions contained in the various holes of the operator (its operands).

Mixfix operator nodes in our parse tree might be incomplete in the following ways:

- (i) The node might consist of only one opening or closing operator part.
- (ii) It might have all of its argument separator operator parts missing.
- (iii) It might only consist of an argument separator operator part.

---

<sup>1</sup>Due to the fact that parsers can fork, we obtain a parse forest rather than single syntax tree. However, in this section, without loss of generality, we focus on an individual element of this parse forest.

---

**Algorithm 4:** Parser Algorithm with Error Recovery Stage 1 – doWork()

---

```
if self.state=check_ambiguity then
  if token is ambiguous then
    self.state=fork;
  else
    self.state=error_recovery;
else if self.state=error_recovery then
  if lexerErrorRecovery() returns a token then
    Insert new token into tokenBuffer;
    Insert new token as current token;
  Append token to token stream;
  self.state=new_token;
else if self.state=new_token then
  if token = endToken then
    self.state=terminate;
  else
    if tokenBuffer is empty then
      tokenBuffer := nextTokens()[0];
    lastToken := token;
    if tokenBuffer is not empty then
      token := tokenBuffer[0];
      Remove first element from tokenBuffer;
    else
      token := end token;
    self.state=check_ambiguity;
else if self.state=fork then
  if scanMode is all then
    foreach non-ambiguous token in token do
      Create a copy of GLRParser object where current token is replaced
      with non-ambiguous token;
      Append new GLRParser object into list of children;
    self.state=inactive;
  else
    Choose one non-ambiguous token, using some strategy;
    self.state=error_recovery;
```

---

Aspect (i) arises for incomplete input, with missing opening or closing operator parts, due to the error recovery reduce actions as described in Section 6.6.5.1.

Whenever the argument separator operator parts are reduced into the output, we create a new node, effectively interpreting them as binary operators and linking this node to the previous two existing trees in the output. This explains why the complete information for complete mixfix operator nodes is distributed into the nodes as sketched in (ii) and (iii).

The tree-manipulation algorithm will traverse the incomplete parse tree and, in combination with the algorithms introduced in the following 4 sections, re-instate a complete parse tree, possibly by adding placeholders.

#### 6.6.5.1 Sort Argument Separator Parts

The goal of this algorithm is to convert a mixfix operator node with (potentially interlaced, redundant or missing) argument separator parts in an arbitrary order into a node where the correct order has been reinstated. By this we mean

$$\sigma_i < \sigma_j$$

if either  $i \neq j$  and  $\sigma_i$  and  $\sigma_j$  denote mixfix operator parts that have been (initially) enumerated in a static order where  $\sigma_i$  appears before  $\sigma_j$ , or  $i = j$  and the argument separator parts  $\sigma_i$  is to the left of  $\sigma_j$ .

Due to the expected small number of argument separator parts, any reasonable searching algorithm can be used – we call the Python standard sorting algorithm *Timsort*[wikipedia.org].

#### 6.6.5.2 Detangle Argument Separator Parts

We assume that prior to this algorithm, the argument separator operator parts  $\sigma_i$  have been sorted as explained in Section 6.6.5.1. It is then straightforward to identify those that belong correctly to the individual mixfix operators. For all other operator parts  $\hat{\sigma}_i$ , several strategies are possible:

- (i) Remove all non-matching  $\hat{\sigma}_i$  including the content of the holes to their right.

- (ii) Complete non-matching  $\hat{\sigma}_i$  with their missing opening, closing and possibly argument separator parts and empty identifiers as content for the holes.
- (iii) A combination of removing and completing. If a certain amount of argument separator parts are present of a different mixfix operator  $\hat{\sigma}_i$  (say  $t$  out of  $m$  where  $t \geq \lfloor \frac{m}{2} \rfloor$  and  $m$  is the arity of  $\hat{\sigma}_i$ ), then proceed with completion as in (ii), otherwise as in (i).

### 6.6.5.3 Complete Argument Separator Parts

In this step, we add missing argument separator operand parts as well as default “empty” content for the resulting operator holes. This is straightforward provided the algorithms in Section 6.6.5.1 and 6.6.5.2 have been executed. We use the empty identifier “ $id_?$ ” as previously introduced.

### 6.6.5.4 Eliminate Redundant Argument Separator Parts

Finally, we need to eliminate redundant argument separator parts. Following the application of the algorithms from the previous sections, these can easily be identified if we adopt the convention that the first of a group of such redundant argument separator parts is kept.

The next step is to chose appropriate content for the hole that is immediately following the argument separator part in question. Here, several strategies could be envisaged, for example one could keep the first occurring operand that is not  $id_?$ , or the operand that was contained in the hole immediately following the first argument separator part in the group.

A more sophisticated strategy would be to attempt a completion of all redundant argument operator parts, but the simple strategy that we have sketched at first works quite well in practice.

## 6.6.6 Error Recovery – Summary

In this section, we have presented some strategies for error recovery that can be used to make our parser more robust. We have seen that the error recovery proceeds at both lexer and parser level (where we have further distinguished between two stages)



and the different types of errors, as classified in Section 6.6.1, are handled by several sub-algorithms (in particular, in Stage 2). The following table summarises this by listing, for all error types, the various algorithms that are used in order to recover from it. In our prototype implementation, we fully support error recovery at lexer level as

	Error Type	Lexer	Parser-Stage 1	Parser-Stage 2
(i)	Missing operators	Algorithm 3: All	–	–
(ii)	Incomplete Mixfix	Algorithm 3: Special case	$F_1, F_2, F_3, F_4$ : Missing “(”, “)”	Algorithm in 6.6.5.3: Complete “ ”
(iii)	Interlaced Mixfix	–	$G_3$ : Interlaced “ ” $G_1, G_2, G_4$ : Interlaced “(”, “)”	Algorithm in 6.6.5.2 Detangle “ ”
(iv)	Shuffled Mixfix	–	$G_3$ : Shuffled “ ”	Algorithm in 6.6.5.1 Sort “ ”
(v)	Mixfix with Redundancy	–	$G_3$ : Redundant “ ”	Algorithm in 6.6.5.4 Eliminate “ ”
(vi)	Otherwise incorrect Mixfix	Lexer token validator	–	–

Table 6.9: Summary of Error Recovery

well as parser Stage 1. For Stage 2, we have focused on the completion scenario described in Section 6.6.5.3 which, in our experience, is one of the most frequently occurring errors when considering input that is spoken mathematics. The remaining cases are relatively easy to implement and, upon completion and integration with the entire system, the error recovery strategies that we have devised in this section lead to a significant improvement of the TalkMaths parsing capabilities.

## 6.7 Summary

This chapter presented our approach to the design and parsing of spoken mathematics. The parsing framework we described and implemented was based on the idea of speech templates. This allowed us to define our language for spoken mathematics using an operator precedence grammar, containing mixfix operators, hence representing all elements of the language including editing commands as either operator or operand.

This approach is novel compared to previous approaches (described in Section 2.1) and could potentially be extended to other spoken command languages, for example spoken programming languages by the introduction of suitable rules following the grammar in Section 6.4.1.

# Chapter 7

## System Implementation and Evaluation

### 7.1 Introduction

All the contributions that have been described in the previous chapters of this thesis are ultimately aimed at improving the TalkMaths system. To this end, we have implemented as many as possible algorithms and methods that we have presented. Some of this work contributed to new prototype versions of TalkMaths and some were also released in the publicly available version of the system.

In this chapter, we will first give an overview of the TalkMaths system and explain how the theoretical contributions contained in this thesis have advanced earlier versions of TalkMaths. We then present an evaluation of the system by comparing the power of our parsing approach with that contained in the older versions of TalkMaths. Finally, we will present and discuss results of a user trial to evaluate TalkMaths as a tool to help improve students' understanding of mathematical concepts.

### 7.2 The TalkMaths System

The TalkMaths system is a result of an ongoing research project. Several versions have been released in the past, and our work has directly contributed to the current version.

TalkMaths is a web-based editing system for mathematical e-content that can be

controlled using a variety of modalities. Primarily, the system is designed to be used with spoken input. Users can also type their input, should they wish to use TalkMaths without speech. In addition, some functionality also supports the use of the mouse.

A demonstration video, showing TalkMaths in use, can be viewed at [www.youtube.com/talkmaths](http://www.youtube.com/talkmaths). We have also included some screenshots of the TalkMaths interface in Appendix D.

In order to use TalkMaths successfully by speech, a number of additional software applications need to be installed on the user's machine: an ASR, a speech front-end application and a web browser. Furthermore, there are certain compatibility requirements to be satisfied.

On the "client" (typically the user's desktop machine) speech-recognition and interaction with the user-interface are done using the ASR. This tool listens to the spoken input, performs the recognition and forwards the resulting transcribed text to the speech front-end.

The speech front-end fulfills several purposes. It enables the recognition of spoken mathematics. It also filters out words that do not belong to the vocabulary and converts certain spoken forms to symbol level (such as, for example, spoken "alpha" to the single letter "a") before it then sends the text stream to the browser.

To render the mathematical content of the TalkMaths web site, a web browser needs to display documents encoded in the Open MathML [Carlisle, 2003; W3C, 2010] standard format. Mozilla Firefox currently supports MathML [W3C, 2003] natively, Internet Explorer and other browsers require the MathJax plugins [Cervone, 2012]. However, the full power of the editing mechanisms can only be reached if CSS support for MathML is also implemented, as is the case in Mozilla Firefox. Currently, JavaScript needs to be enabled in the browser in order to obtain the best user experience of TalkMaths.

In the first version of TalkMaths, Dragon NaturallySpeaking was used as the ASR together with the free NatLink [Gould, 2001] library as speech front-end and the XUL-Runner [Mozilla Foundation, 2009] browser as a standalone GUI. In a subsequent intermediate version, NatLink was replaced with the commercial DNS SDK [Nuance Communications] and the TalkMaths web site was usable only with the Mozilla Firefox web browser. Although Dragon NaturallySpeaking provides probably the best and most accurate recognition amongst all available (commercial and free) ASRs, the latest

version of TalkMaths now uses Windows Speech Recognition. This made it possible to create a more sophisticated and flexible front-end.

Most ASR users are familiar with interacting with a web browser using their speech recogniser software with some help from the keyboard and mouse. Being web-based, TalkMaths is compatible with any speech recognition software on any platform, provided it supports the browser available on the client system.

### 7.2.1 System Architecture

The architecture of TalkMaths has evolved from that of [Wigmore, 2011], which was a standalone desktop application, towards a multi-component, distributed model, where the ASR and the TalkMaths web application (including the parser) are separate. This yields a web application which provides an online speech-based user-interface for mathematics that uses speech as the main input modality. In practise, this means that users speak into the ASR and the resulting transcribed text output is used as the input for the web application. To this purpose, an input field is provided within the editor page (see Figure D.5 in Appendix). An additional advantage of this web-based architecture is the fact that one can now enter commands via the keyboard, which was not the case in any of the earlier versions.

In Chapter 3, we have already introduced the general system architecture of the system. The final choice we made for the system architecture was the Application Proxy architecture 3.4. This architecture enabled the multi modality of the application (which was a limitation of the standalone desktop version) and it was compatible with the parser engine and would run on a web browser. It also allowed us to develop user interface, application logic and parser components in separate modules which makes the project more extensible. We will now provide more information on web-related aspects including the implementation.

The TalkMaths system is an interactive web application. The web server used is Apache, the server-side environment uses PHP and a MySQL database, and the Google App Engine service [Severance, 2009] written in the Python scripting language. The web server interacts with an application that resides on the Google Engine (currently, the URL is <http://talkmathsparse.appspot.com>). This application parses the input language and is effectively a (RESTful) web service. These two systems

exchange messages, including data in JSON format. All data is held in the database on the server, this ensures that the information that belongs to an editing session is persistent. Figure 7.1 illustrates the system architecture of the public TalkMaths web

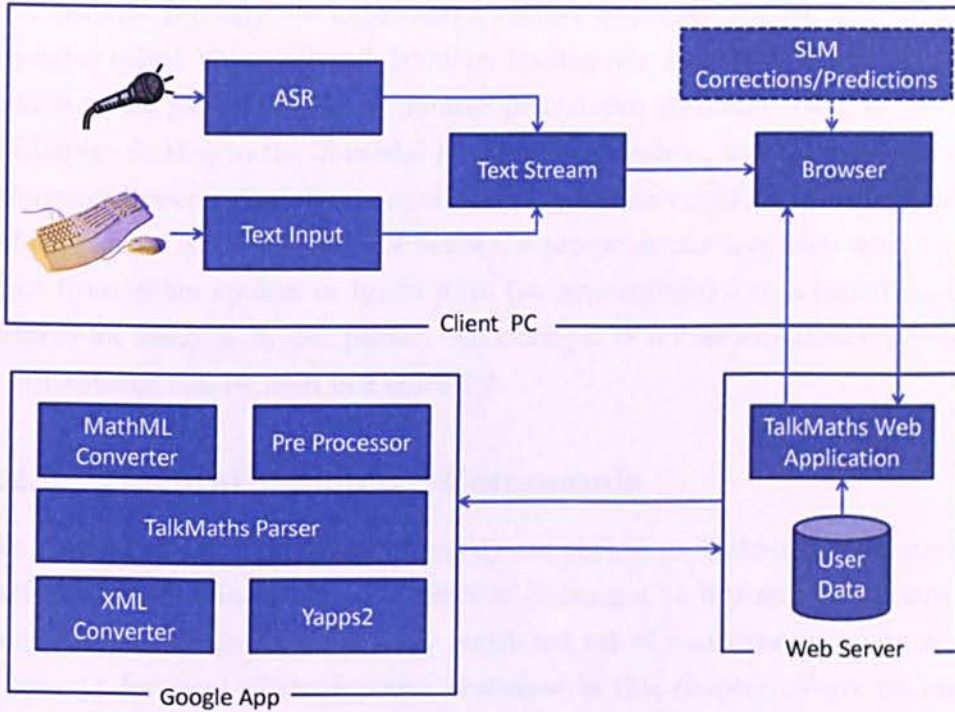


Figure 7.1: TalkMaths system architecture

application. Two input modalities are available, namely spoken input (using ASR) and textual input from a keyboard & mouse. Each of these result in a text stream which describes the mathematical expression of current interest in relatively natural language (see Section 7.2.2 below). The text stream is then passed through the browser interface, encapsulated in a HTTP request, to the web server. The SLM Corrections/Predictions component is currently not integrated to the TalkMaths system, however it is included in the figure to illustrate where the corrections and predictions of the text stream would take place (See SWIMS system in Section 5.3.1. The web application on this server then handles the application logic (covering general security, session management and storage tasks) [Attanayake et al., 2011b] and calls the parser, running on the Google App Engine server, which ultimately processes the input text stream, transforming it into

the desired format, such as MathML or XML. Again, web application requests and parser responses use HTTP. Upon receiving the parsed output from the Google App Engine server, the web application sends the output (in the appropriate marked-up form) to the browser on the client for rendering in conventional mathematical notation on a display. Initially, we have used a recursive-descent context-free grammar parser generator called Yapps2 [Patel, 2009] to develop our TalkMaths parser. Then we developed our own parser based on operator precedence grammar, that has been described in Chapter 6. Due to the bi-modal nature of our system, and the consequential (small) differences between the correct spoken and typed descriptions of any given mathematical expression (see Section 7.2.2 below), a pre-processor has been used to convert the input from either spoken or typed form (as appropriate) into a canonical form that is suitable for analysis by our parser. An example of a mathematical expression created by TalkMaths can be seen in Figure 7.2.

### 7.2.2 Natural Language Commands

The mathematical expressions currently supported in TalkMaths are standard arithmetic expressions including exponentials (raising a to a power), fractions, roots and functions. Although this is a fairly restricted set of mathematics, it provides a proof of concept for most of the features discussed in this chapter. Work on implementing speech templates has reached prototype stage and will be available in the online-system (of a public release) shortly.

Both SWIMS and TalkMaths employ an input language that is much closer to how people actually speak mathematics in a classroom environment compared to specialised mark-up or formatting languages for mathematics, such as LaTeX or MathML which take much effort to learn,. The rationale behind this is to make learning to command the system to be as easy and intuitive as possible for the user, keeping the “naturalness” of the task to a maximum. For example, Figure 7.3 shows encodings of the same mathematical expression, namely

$$\frac{n}{k(n-1)}$$

in LaTeX, MathML, the TalkMaths spoken input form and the TalkMaths and SWIMS keyboard input command language. Note that both forms of the TalkMaths command language are easy for a person to read, speak or type. They should be much more



talk{maths} *creating maths by  
using plain English*

The screenshot shows the TalkMaths web interface. At the top left, the logo "talk{maths}" is displayed with the tagline "creating maths by using plain English". The main workspace is a grid with the formula 
$$S_n = \frac{1}{2} n \left[ 2a + (n - 1) d \right]$$
 entered. The interface includes a "2d Maths" tab, "Import" and "Export" buttons, and a command input area at the bottom with a "Submit" button and a list of editing commands.

**Editing commands:**

<i>edit expression</i>	<i>select {n}</i>	<i>also select {n}</i>	<i>select all</i>	<i>delete</i>
<i>unselect {n}</i>	<i>unselect all</i>	<i>clear expression</i>	<i>finish editing</i>	<i>step back</i>

Figure 7.2: TalkMaths used to create the formula for the sum of  $n$  terms of an arithmetic progression



LaTeX:	$\frac{n}{k(n-1)}$
MathML:	<pre> &lt;mfrac&gt;   &lt;mi&gt;n&lt;/mi&gt;   &lt;mrow&gt;     &lt;mi&gt;k&lt;/mi&gt;     &lt;mo&gt;&amp;InvisibleTimes;&lt;/mo&gt;     &lt;mfenced close="" open=""&gt;       &lt;mrow&gt;         &lt;mi&gt;n&lt;/mi&gt;         &lt;mo&gt;-&lt;/mo&gt;         &lt;mn&gt;1&lt;/mn&gt;       &lt;/mrow&gt;     &lt;/mfenced&gt;   &lt;/mrow&gt; &lt;/mfrac&gt; </pre>
TalkMaths speech input :	<pre> november over begin kilo open bracket november minus one close bracket end </pre>
TalkMaths & SWIMS keyboard input:	<pre> n over begin k (n - 1) end </pre>

Figure 7.3: Encoding of same mathematical expression in LaTeX, MathML, TalkMaths and SWIMS command languages respectively

accessible and easy to learn for novice users than either LaTeX or MathML. Also, note that the TalkMaths speech input language requires use of the NATO alphabet [Law, 2009] for the dictation of single characters, due to issues of potential confusion between conventional letter names by ASR systems (e.g. “bee” (b), “cee” (c), “dee” (d)). An example of a simple mathematical expression is the equation for velocity under uniform acceleration  $v = u + at$ , which in our spoken mathematical language would be read as: “victor equals uniform plus alpha tango”. A more complex example is the formula for the solutions of a general quadratic equation:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (7.1)$$

which would be spoken as “minus bravo plus or minus square root of bravo squared minus four alpha charlie all over begin two alpha end”. Greek characters, such as  $\alpha$ ,  $\beta$ , etc. can be inserted using the prefix ‘greek’ before the name of the character. For example, the trigonometric identity :

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

would be read as “sine begin greek alpha plus greek beta end equals sine greek alpha cos greek beta plus cos greek alpha sine greek beta”. An example of a more complicated expression which can be interpreted by TalkMaths is the van der Waals equation from thermal physics, rendered as shown in Figure 7.4 below. This would be dictated as “open bracket capital papa plus begin november to the power of two alpha end over begin capital victor to the power of two end close bracket open bracket capital victor minus november bravo close bracket equals november capital romeo capital tango”.

### 7.2.3 Editing Paradigms

TalkMaths allows users to edit mathematical expressions that they have input, as rendered on the computer screen, by issuing relatively intuitive commands which are close to how a human-to-human interaction would deal with the same tasks. For example, editing the numerator of a fraction can be invoked using the “edit numerator” command. Here, the application “understands” that the user requires the editing of only a part (in this case, the top part) of the fraction. Another example is where

The screenshot shows the TalkMaths web interface. At the top left, the logo "talk{maths}" is displayed with the tagline "creating maths by using plain English". On the top right, there is a user greeting "Welcome, user" and a "Log Out" link. Below the logo, there are buttons for "Editor", "Download", and "Settings". The main content area features a large grid with the van der Waals equation rendered as 
$$\left( p + \frac{n^2 a}{V^2} \right) (V - nb) = nRT$$
. To the right of the grid is a "What can I do?" help panel with a "Hide this" button. The help panel lists various commands and their functions, such as "... over ..." for fractions, "... of ..." for functions, "plus" for binary operators, "begin" and "end" for sub-expressions, "open bracket...close bracket" for square brackets, "open square bracket...close square bracket" for square brackets, and "open curly bracket...close curly bracket" for curly brackets. Below the grid is a "Type Command Here:" input field with a "Clear field" button and a "Submit" button. Below the input field is a table of editing commands:

Editing commands:			
edit expression	select (n)	also select (a)	select all
delete	unselect (n)	unselect all	clear expression
finish editing	step back		

At the bottom of the page, there is a footer that reads "Talk Maths is developed at Kingston University by \_\_\_\_\_ & \_\_\_\_\_" with a "Contact Us" link, and a copyright notice "© 2011 Talk Maths".

Figure 7.4: The van der Waals equation, read as stated above, rendered by TalkMaths

the “edit functions” command will invoke editing of all available functions within the current mathematical expression. We refer to these types of edit commands as “semantic editing” commands, as they refer to the meaning of the mathematical structure/components which the user sees on the screen. Other types of commands include “selective editing” and “exhaustive editing”. The former is used to select specific sub-expression(s) within the full expression shown on the screen. For example, if a user wishes to change the sub-expression  $2a$  in the quadratic formula 7.1, he/she can use the command “edit two alpha”, which will make the denominator ( $2a$ ) of the above quadratic formula editable by placing indexed bounding boxes around all possible selections of the sub-expression  $2a$  – in this case, just the denominator. In contrast to this, exhaustive editing makes use of a displayed set of nested indexed boxes superimposed over the expression to allow the user to select the whole, or any part, of the expression for editing. Three such methods, highlighting all sub-expressions, all individual symbols and all operators, respectively, are illustrated in Figure 7.5 (corresponding commands are: “edit expression”, “edit symbols” and “edit operators” respectively). A more detailed explanation of these editing paradigms can be found in [Wigmore, 2011].

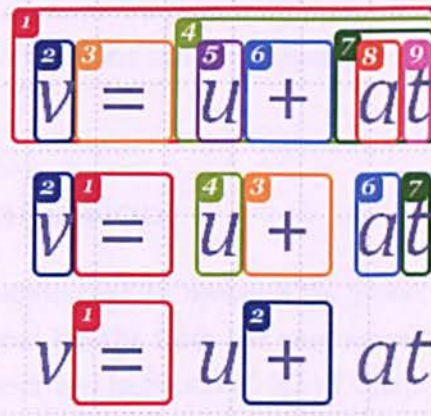


Figure 7.5: Different editing paradigms for editing mathematics by speech, each applied to one of equations of uniformly accelerated motion

### 7.2.4 Natural Language Search-Driven Help Facility

TalkMaths has a help facility that includes a natural language search option. A user can search through the help information using this tool, by entering search terms in natural language form. If the search phrase contains a word that is not in the vocabulary ( $V$ ) of the help content, we use the Damerau-Levenshtein algorithm [Damerau, 1964; Levenshtein, 1966] to calculate the Levenshtein distance between the entered word and each word in  $V$ . This distance is based on the minimum number of insertions, deletions, substitutions and transpositions of characters required to transform one string into the other. The word within  $V$  with the shortest distance from the entered word is selected as the “best guess” for what the user intended to enter. See Section 4.7 for a more detailed discussion.

The Damerau-Levenshtein method was originally introduced to compare the similarity of text strings, and we found that this strategy works well when the user makes minor misspellings when typing using the keyboard and mouse. For each help term, we assign a score based on its length, the length of the entered search term and how similar the terms are (based on the Levenshtein distance). Our mechanism assists the user to find appropriate commands using their existing mathematical knowledge. For example, typing, “How to edit a fraction” command will present all the TalkMaths commands associated with fractions and rank them according to how relevant they are believed to be (see Figure 7.6).

## 7.3 Parser Evaluation

We have carried out an experiment to measure the power of our current parser compared to its earlier versions. Firstly, from the very same spoken mathematics expressions corpus we created described in Section 5.2.1 of Chapter 5, we parsed 4308 spoken mathematics expressions using the Yapps2-based parser ( $P_1$ ) presented by [Wigmore, 2011] and the Yapps2-based parser in our live TalkMaths web-based system ( $P_2$ ). Of these, 1955 (45.38%) expressions were successfully parsed (865 had the expected parse tree and 1090 had a different one) by  $P_1$  and 3726 (86.49%) (2657 had the expected parse tree and 1069 had a different one) were successfully parsed by  $P_2$ . 1169 (27.14%) expressions were successfully parsed by both parsers. Table 7.1 shows a selected sam-



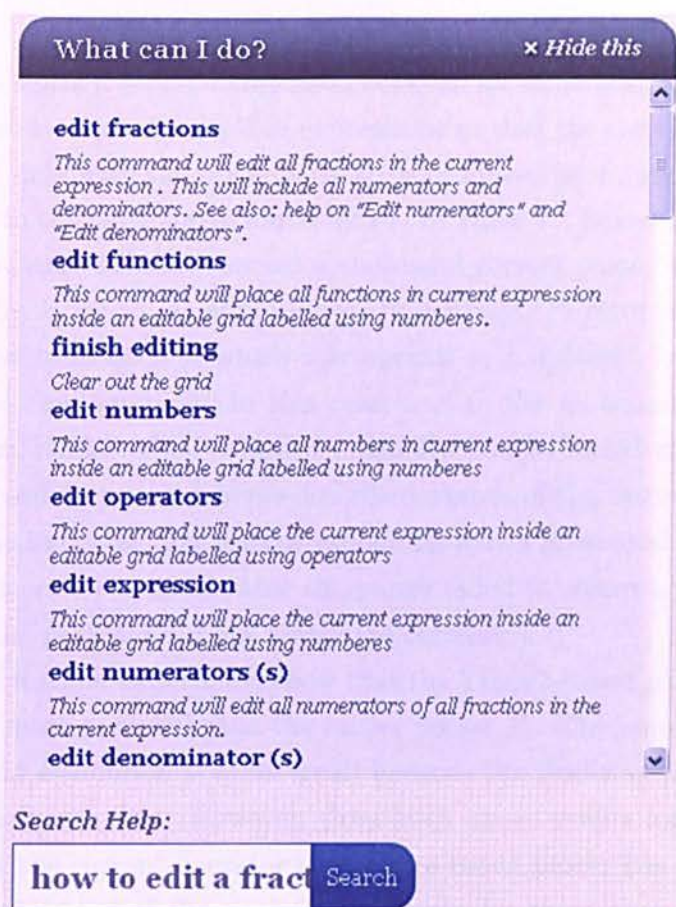


Figure 7.6: The TalkMaths Natural Language Driven Help Facility used to search to find information on fractions

ple of expressions and success or failure of their parsing by each parser. We then used this selected sample to evaluate our new operator precedence parser ( $P_3$ ). However, whereas  $P_1$  and  $P_2$  each return a single parse tree, which is either correct or incorrect,  $P_3$  actually returns a parse forest (a list of possible parse trees) which consists of different interpretations of ambiguous expressions. Of these, at most one parse tree from the forest may correspond to the correct parse of the expression, or none may be correct. We impose an ordering or ranking on the trees of the parse forest, such that simpler trees with fewer nodes are always ranked higher than more complex trees with more nodes. The ranking order of two trees with the same number of nodes is chosen

arbitrarily.

In order to perform a direct comparison between all three parsers, it was necessary to manually parse a selected sample of expressions so that the correct parse tree could be compared not only with the single parse trees produced by  $P_1$  and  $P_2$ , but also with each of the trees in the parse forest found by  $P_3$ . In Table 7.1 below, for  $P_1$  or  $P_2$ , a tick ( $\checkmark$ ) in the appropriate column denotes a successful correct parse, which is computed unambiguously by  $P_1$  or  $P_2$ , as appropriate. In contrast,  $P_3$  returns a parse forest, of one or more parse trees each of which corresponds to a different interpretation of an expression (due to ambiguities). In this case, a  $\checkmark$  in the  $P_3$  column means that the correct parse tree, namely that obtained manually, can be found within the first five trees, ranked according to the scheme described above, of the output parse forest. A “?” in any column indicates that the corresponding parser generated an incorrect parse tree(s), whereas a cross ( $\times$ ) means that the parser failed to return a parse tree for that expression (i.e. the program did not terminate correctly).

The results of the first experiment show that the Yapps2-based parser  $P_2$  we created in this project is more powerful than the earlier parser  $P_1$ . The number of expressions used in the second evaluation is quite small because the resulting parse tree from  $P_3$  have to be manually checked. However, from both above evaluations, we clearly can demonstrate that the current operator precedence-based parser has been significantly improved over the course of the project. It should be noted that in Table 7.1, the expression  $x', y''$  fails to be parsed by all three parsers, due to the fact that the “double prime” or “prime prime” mathematical operation was not defined in any of the parsers in question.

Expression	Spoken Format	$P_1$	$P_2$	$P_3$
$a^2 + b^2 = c^2$	alpha to the power of two plus bravo to the power of two equals charlie to the power of two	✓	✓	✓
$3x^2 + 3$	three x-ray to the power of two plus three	✓	✓	✓
$\text{delta}(X)$	delta echo lima tango alpha of capital x-ray	?	✓	✓
$f(g(h(x))) = e^{\sin(x^2)}$	foxtrot of begin golf of begin hotel of x-ray end end equals echo to the power of begin sin open bracket x-ray to the power of two close bracket end	?	?	✓
$S^n$	capital sierra to the power of november	✓	✓	✓
$x', y''$	x-ray prime comma yankee prime prime	×	×	×
$1_R$	one index capital romeo	×	✓	✓
$\sqrt[n]{\frac{ax+b}{cx+d}}$	november root of begin alpha x-ray plus bravo end over begin charlie x-ray plus delta end	×	×	✓
$\frac{d\theta}{dr}$	begin delta theta end over begin delta romeo end	×	✓	✓
$x = \frac{-7 \pm \sqrt{49+32}}{8}$	x-ray equals begin minus seven plus or minus square root begin forty nine plus thirty two end end over eight	×	✓	✓
$\delta x$	greek delta x-ray	?	✓	✓
$x < 4$	x-ray less than four	✓	✓	✓
$f(2 \times x)$	foxtrot of begin two times x-ray end	?	✓	✓
$y = f(x)$	yankee equals foxtrot of x-ray	?	✓	✓
[*]	open square bracket times close square bracket	✓	✓	✓
1	one	✓	✓	✓
$\frac{e}{m}$	echo over mike	?	✓	✓
$y = x^2$	yankee equals x-ray to the power of two	✓	✓	✓
$dB = 10 \log \frac{P_{out}}{P_{in}}$	delta capital bravo equals ten log begin capital papa index begin oscar uniform tango end end over begin capital papa index begin india november end end	×	?	✓
$4x^2 - 9y^2$	four x-ray to the power of two minus nine yankee to the power of two	✓	✓	✓

Table 7.1: Sample results of parsing spoken mathematics corpus (complete expressions) using Yapps2-based parser ( $P_1$ ) presented by [Wigmore, 2011], the Yapps2-based parser in our live TalkMaths web-based system ( $P_2$ ) and operator precedence-based parser ( $P_3$ )



## 7.4 TalkMaths Field Study

This section describes a field study on how users create and edit mathematical formulae using our TalkMaths system and natural language commands based on the approach in Section 6.2. For the purpose of this study, users typed their commands as opposed to speaking them, and they were using the TalkMaths editor in comparison to conventional equation editing software (such as the Microsoft Word equation editor). In particular, our goal is to evaluate whether such an approach can aid participants' understanding of particular mathematical concepts, such as the "numerator" and "denominator" of fractions, to reinforce their understanding of these and related ideas. Our findings also evaluate the usefulness of our developed language as a means of communicating mathematics using textual input only.

We carried this out through questionnaires, both to test participants' knowledge and understanding and to quantify their own perception of these, with respect to appropriate mathematical concepts. These questionnaires are shown in Appendix E. The current chapter describes a small pilot study of this type within an introductory non-specialist mathematics course and measures the students' performance in both pre- and post-task tests. We intend to use the results of such studies in the development of new teaching and support materials aimed at improving students' understanding of key ideas. These activities will be rolled-into more courses in the future, which should give useful insights and data for additional studies.

### 7.4.1 Experiment: The Learning Activities

In this section, we present the design, implementation and results of an experiment on using TalkMaths with real students in a real classroom environment. This was to assess the TalkMaths application in terms of its usability and impact on learning of mathematical concepts, compared with use of a conventional mathematical editor. A user evaluation had previously been carried-out on the original, desktop-based version of TalkMaths [Wigmore, 2011]. However, this focused on the system's ease of use and how fast and accurate users were in performing various mathematical editing tasks. It was found that the majority of participants, who did not have any disability, took longer and produced more errors using TalkMaths than when using a conventional key-

board & mouse based editor. Nevertheless, the only participant who did have a major disability (Duchenne Muscular Dystrophy) performed better, both in terms of speed and accuracy, when using TalkMaths, and out-performed many of the non-disabled group when using this modality [Wigmore, 2011]. This illustrated the potential benefits of TalkMaths to one of the user groups for which it was primarily designed. The present study investigates how the new version of TalkMaths influences the users' understanding of mathematical concepts related to the prescribed editing tasks, instead of speed and accuracy in performing the editing. The results of a preliminary evaluation of the new version have already been published elsewhere [Attanayake et al., 2013], but scrutiny of these revealed some weaknesses in the design of the original experiment – notably ambiguities in some parts of the questionnaires. These limitations were addressed and the revised materials were tested using a new group of students [Attanayake et al., 2012]. The details and results of this refined, follow-up study are presented below.

#### 7.4.2 Design of Experiment

We developed a set of classroom mathematical materials and learning tasks for undergraduate Life-Science students who were taking a basic mathematics module at Kingston University. The tasks to be carried out by the participants and the questions on their mathematical knowledge, were designed to be appropriate to their typical level of mathematical expertise. The volunteer student participants were allocated to two groups randomly. Both groups carried out the same tasks, but using two different tools. The first group (A) used a conventional mathematical editor (Microsoft Word Equation Editor) while the other group (B) used a research prototype version of TalkMaths. Note that all subjects had previously used MS Equation Editor but none had used the TalkMaths system before. Each participant was asked to complete three questionnaires (see the Appendix E). The first questionnaire was about the participants' own perception of their mathematical competence at the start of the experiment. The second was a diagnostic test related to the tasks they were about to carry out and the final questionnaire, given at the end of the exercise, was similar to the second, in order to assess improvements to the participants' understanding, but also included questions concerning their experience of using whichever system they were allocated. From our

initial pilot experiments, it was observed that some participants omitting to answer some of the questions – particularly in the post-task questionnaire – led to results of dubious reliability. Hence, in the follow-up study we instructed the subjects that all questions on both pre- and post-task questionnaires were mandatory. Furthermore, after detailed scrutiny of the original versions, the revised questions were re-designed and re-worded to be as unambiguous as possible. Details of the tasks and the questionnaires are given in Appendix E.

### 7.4.3 Undertaking The Learning Activities

The regular teaching staff and demonstrators for the module supervised the participants carrying out the tasks of the experiment, without actually instructing them. The students were required to learn by themselves, with the only resources available being the instructions given in the worksheet and the help facility of whichever tool they were assigned to. After completing the first two questionnaires, participants were required to undertake three tasks (Tasks 1, 2 & 3 – see Appendix E) creating and editing mathematical expressions involving fractions, functions and square roots respectively. Each task required them to create a specified equation and then carry out a minor modification to this using the editor they were assigned. As noted previously, all participants had used Microsoft Equation Editor in earlier practical sessions for the module but no prior training on TalkMaths was given. Thus, we expected “better” performance and possibly higher levels of satisfaction amongst the group using MS Equation Editor. All participants were encouraged to use the help facility of the editor to resolve any questions they might have while completing the tasks.

### 7.4.4 Evaluation

Table 7.2 presents the results of the post-task feedback from the participants on the ease of use of the tool they were using to complete the tasks. Figure 7.7 illustrates the same results graphically. The group using the MS Equation Editor seemed to find it easy to use, more-so than the group using TalkMaths. When these qualitative evaluations were each put onto a 5 point Likert scale, the differences in perceived ease of use of the system by the two groups proved just to be statistically significant ( $p \approx 0.047$  in a one-tailed t-test). However, this was to be expected due to the participants’

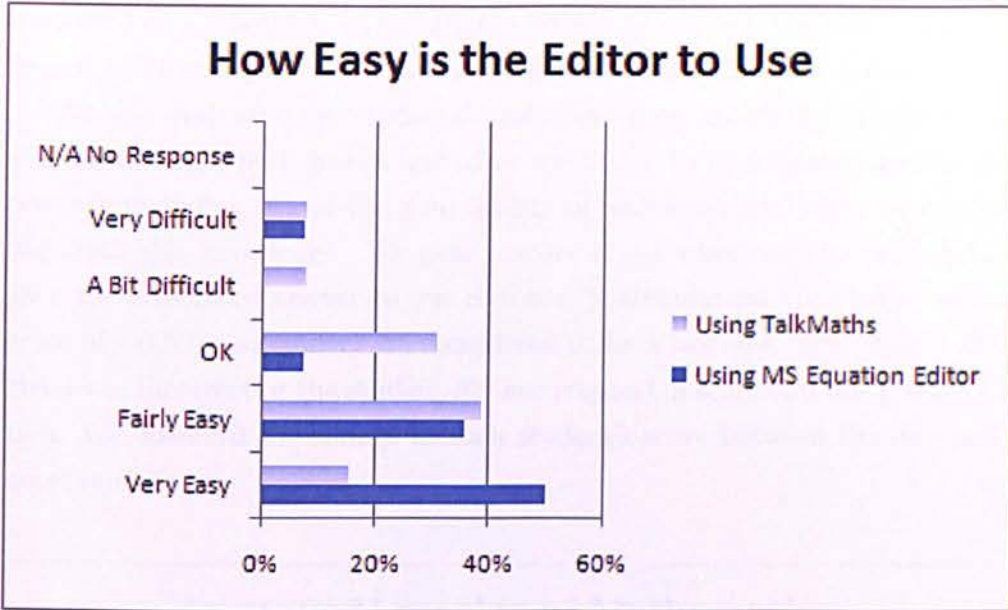


Figure 7.7: The ease of use of the system by the two groups

Editor Used	Participants	Very Easy	Fairly Easy	OK	A Bit Difficult	Very Difficult	No Response
MS Equation	14	50%	36%	7%	0%	7%	0%
TalkMaths	13	15%	38%	31%	8%	8%	0%

Table 7.2: Results of the post-task feedback on the ease of use of the tool

previous experience with MS Equation Editor, in contrast to none of them having used TalkMaths before.

Table 7.3 and Figure 7.8 present the results of the post task feedback from the participants regarding any improvements they perceived to their understanding of the mathematical concepts involved in the task after using the appropriate tool they were allocated.

Most users of TalkMaths seemed to believe it had improved their understanding, while only 35% of users of MS Equation Editor reported any improvement. In fact, 14% of users of MS Equation Editor thought it had impeded their understanding, whereas no TalkMaths users held this opinion. However, when these qualitative evaluations were

converted to a 5 point Likert scale, the difference between the two groups' perceptions proved to be not quite statistically significant ( $p \approx 0.065$  for a one-tailed t-test).

We also evaluated the students' performance on knowledge of relevant mathematical terminology, both before and after the tasks, to investigate whether the exercise, possibly including use of the help facility of whichever tool they were allocated, had improved this knowledge. We gave a score of +1 whenever the participant correctly gave the prescribed answer to one of these "mathematical knowledge" questions, or a score of +0.5 for an answer we considered to be a relevant "near miss". If the answer given was incorrect or the student did not respond, a score of 0 was given for that question. We observed the change to each student's score between the pre- and post-task questionnaires.

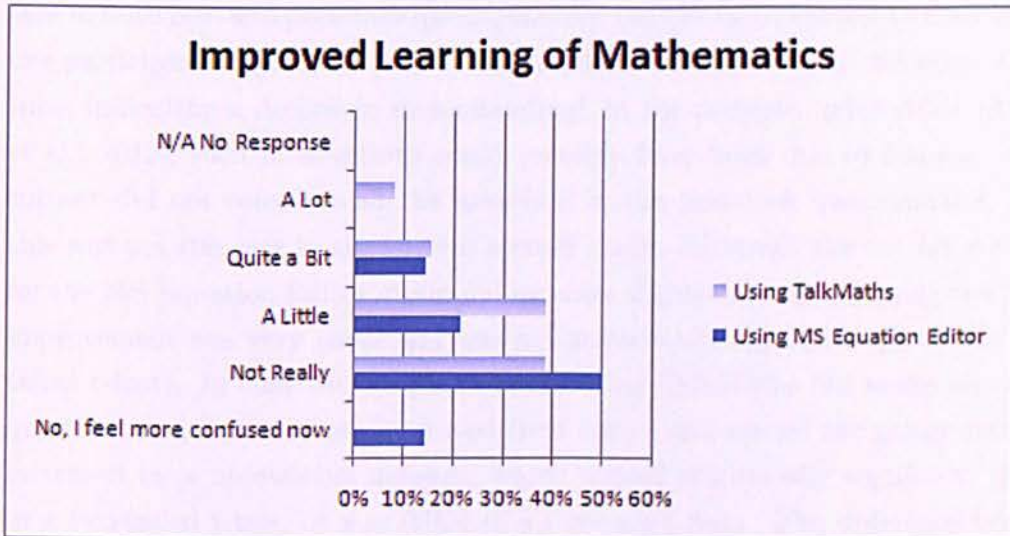


Figure 7.8: Improved understanding

Editor Used	Participants	No, I feel More confused now	Not Really	A Little	Quite a Bit	A Lot	No Response
MS Equation	14	14%	50%	21%	14%	0%	0%
TalkMaths	13	0%	38%	38%	15%	8%	0%

Table 7.3: Results of the post-task feedback on any improvements to understanding mathematical concepts

Although, in both groups, most individuals gave the same response to any given question in both pre- and post-task questionnaires, indicating no change to their knowledge, one participant in group A (MS Word Equation Editor) actually did worse the second time, indicating a decline in understanding! In the previous, pilot study [Attanayake et al., 2012], such observations could possibly have been due to laziness, where the subject did not complete all the questions in the post-task questionnaire. However, this was not the case in the current second study. Although the overall average score for the MS Equation Editor group did increase slightly after performing the tasks, this improvement was very small and was not statistically significant ( $p \approx 0.18$  in a one-tailed t-test). In contrast, no participants using TalkMaths did worse on the second questionnaire, five students improved their scores and overall the group average score increased by a noteworthy amount, which proved statistically significant ( $p \approx 0.011$  in a two-tailed t-test, or  $p \approx 0.006$  in a one-tailed test). The difference between the two groups in mean improvement after performing the tasks also proved to be weakly statistically significant ( $p \approx 0.048$  in a one-tailed t-test), where the TalkMaths group showed greater improvement. In summary, from these results, it can be concluded that due to their previous experience, users found the MS Word Equation Editor easier to use than TalkMaths, but on the other hand TalkMaths helped improving their understanding of mathematical concepts more than MS Word Equation Editor.



# Chapter 8

## Conclusions and Future Work

The work described in this thesis has devised, implemented and evaluated solutions for the problem of creating and modifying mathematical expressions by speech, using web-based applications. In this chapter, we conclude this work by critically analysing our findings and giving an outlook for future research directions.

### 8.1 Conclusions

The first phase of our research was motivated by our desire to improve the desktop-based TalkMaths system that was described and implemented in [Wigmore, 2011]. We started-off by investigating various architectures for realising general-purpose speech-based applications. This led to our literature review of this area and our architecture taxonomy. In particular, we have adopted the view that web-based architectures will eventually become more and more prevalent even amongst speech-based applications due to trends like the move to more cloud-based solutions and service-oriented architectures.

The amount of previously published work in the area of web-based architectures for speech-enabled systems seemed relatively small at the time of writing. However, we expect additional interest on speech-based applications amongst researchers in the foreseeable future, as speech technology becomes more established. As we explained in Chapter 7, our work resulted in TalkMaths being rewritten as a web-based system (and a web service), and our findings might persuade other system architects to proceed

similarly.

We have gained very encouraging results from the perplexity experiments carried out in Chapter 5, leading to the conclusion that spoken mathematics is relatively more predictable than everyday natural languages. Hence, by statistically modelling spoken mathematics, it is possible to (semi-) automatically predict or correct the user's input to a system such as TalkMaths. With this result in mind, we have implemented a prototype predictive system that can achieve over 90% one word ahead prediction accuracy, which can be used to assist in the creation of electronic mathematical content. We have also seen that one word ahead prediction accuracy can be improved by increasing the size of the training dataset used for the SLM. At present, the predictions are limited to the vocabulary of the SLM. This implies that each time a new word (in our case, a spoken name of a mathematical entity) is encountered, it would need to be added to the system's vocabulary. This in turn requires the corpus of mathematical expressions on which the SLM is based to be extended to reflect the change. Although possible in principle, this is not straightforward as one would have to find a considerable amount of additional data in order to update the baseline SLM. Adaptive online learning may be a solution to this issue.

We have also implemented and evaluated a prototype corrective system for spoken mathematics input using the Damerau-Levenshtein distance method. This was also proven to be highly successful at correcting up to 3 character-level errors in an expression. However, as expected, when the number or complexity of such errors increases, the efficiency of correction declines.

Our proposed framework for the design of spoken mathematics (and spoken command languages in general) uses mixfix operators. This leads to our construction of speech templates, resulting in (usually short) spoken commands that appear reasonably intuitive and "natural" to the user. They can be used for expressing commands of different types such as content, editing or help commands. Mixfix operators with no arguments (arity 0) are frequently used for any command that is not related to content such as editing and system commands. Overall, our framework has clearly resulted in a much improved input language for our new prototype version of the TalkMaths system, in particular, the increased range of mathematics that can be parsed as documented in our parser evaluation (Table 7.1).

The operator precedence-based GLR parsing technique that we have developed,



using multiple parses for resolving ambiguity, and our error recovery algorithm have resolved many problems and weaknesses in earlier systems and proposals. However, parsing incomplete or incorrect spoken commands remains a complex and challenging task. Whereas evidence from our evaluation suggests that our overall error recovery strategy is correct and can treat a wide range of problematic input, there exist a number of issues that could be addressed to further improve our algorithm. The error recovery algorithm that we have given in Section 6.6.5 is very comprehensive, but may not always be needed in practice. Restricting the range of tolerated incorrect input would speed up the algorithm. This leads us to discuss the main drawback of our approach, and certainly of the prototype implementation: due to the large number of possible interpretations of occurring ambiguities, an exponentially growing number of different parse trees might be generated. This can slow down the algorithm and make it less convenient to the user. We suspect that a careful analysis and profiling of our code would reveal bottlenecks in it, and it is clear that using revised data structures allowing partial sharing of the various parse trees could significantly improve its runtime behaviour. On the other hand, for relatively shorter input (and this will be the typical situation in a speech-based system) the algorithm appears practical.

Syntax analysis is only the first step of the classical approach to compiler-construction, and it would be interesting to investigate whether a more general framework for displaying spoken mathematical input can be found. An investigation into tools and techniques employed for semantic analysis and code generation would be needed, followed by an analysis of their usefulness and relevance for the kind of tasks and challenges that are required for our line of work.

Throughout the different phases of this PhD project, we have been keen to see the practical relevance of our research by implementing the algorithms even in their early stages. This has led to a number of prototypes, including re-implementation of earlier work. We recreated a version of the system described by [Wigmore, 2011] in order to have a baseline in our parser evaluation (see our comparison of the parsers  $P_1$ ,  $P_2$  and  $P_3$  in Section 7.3). Our implementation of some of the speech editing paradigms, as introduced in [Wigmore, 2011] was needed to make speech-driven editing possible in an interim version of TalkMaths, used for the learning and teaching user experiment in 7.4.1. To our knowledge, this is the first time that both the exhaustive and semantic grids (see Section 7.2.3) have been fully implemented, used and tested in

a real classroom environment.

In Chapter 7, we described a thorough evaluation of our new version of TalkMaths, by illustrating the improvements we made to the parsing algorithm. Our findings suggest that our new Operator Precedence parsing approach performs significantly better for both syntactically correct and incorrect expressions, compared with that used in [Wigmore, 2011]. On the other hand, there are still a number of problems, mainly linked to out of vocabulary words, out of scope grammar and Yapps2 grammar inflicted left recursion errors (only for  $P_1$  and  $P_2$ ).

The results of our field study, presented in Chapter 7, Section 7.4.1 are encouraging. Due to the novel web interface offering multimodal input, students were able to type “natural maths” commands. Removing the danger of misrecognition due to the lack of an ASR, they could focus on engaging with the mathematical content using descriptive natural language commands. In this way, as our findings suggest, the students were able to improve their understanding of specific mathematical concepts. We feel that there is a lot of potential in this approach. For example, online maths tutoring systems, educational chat rooms or forums could all hugely benefit from our techniques as they provide a first step towards a natural or multimodal user interface for online mathematics that is operating-system independent.

## 8.2 Future work

During the time of the PhD, numerous findings and insights reshaped the course of the TalkMaths project. However, there were other interesting research questions raised while answering the main research questions we were originally concerned with. Even though it would have been valid to carry out more research on each of these other aspects, the time available was limited. In this chapter, we briefly outline some of those aspects as future directions to our work.

Although the architecture choice we made for the TalkMaths project was facilitated by our classification of speech based architectures, we were constrained by some historical reasons and design limitations, for example the language used and the processing power available at the University. For these reasons, we were unable to compare the performance of our system between different possible architectures. It would have been better to experiment on the performance of our prototype systems using different pos-

sible architectures prior to implementation of the final solution which can be used by end users.

The SLMs we created were based on linearised mathematics language of the nature described by previous authors [Chang, 1983; Fateman, 2009; Wigmore, 2011]. Even though the experiments described in Chapter 5 demonstrated that spoken mathematics in general is quite predictable, the final proposal of the spoken mathematics language described in subsequent chapters (7 and 6) was slightly different from its predecessors (for example, using speech templates). As a direction for future work, creating and evaluating SLMs using a suitable dataset of spoken mathematics that use speech templates could further demonstrate that we can use prediction and correction facilities for the new spoken mathematics language that includes speech template components as well. It should be noted that such a dataset will be available through time as more and more people use the TalkMaths system.

As explained in Chapter 8, when new speech components are added to the system, the baseline SLM will have to be updated to incorporate the changes. This is the same case when a user wishes to alter or remove currently-used speech components in from the system. An adaptive language modeling technique, such as cache or trigger models, (see Chapter 4) could be useful to do this in a more dynamic way rather than recalculating word probabilities from each new training dataset. Investigating which adaptive language modeling technique is best for the underlying language and subsequent implementation of predictive and corrective facilities into the TalkMaths system will make the system more scalable.

Currently, the predictive and corrective systems are implemented as separate prototypes to the TalkMaths system. However, integrating these facilities into TalkMaths in the future is possible and shall be very useful. It would also be interesting to employ a user evaluation study of our predictive and corrective facilities once they have been integrated into the TalkMaths system and compare the resulting benefits with the usability of the system prior to integration – in particular, carrying out an evaluation similar to that described in Chapter 5.

We have shown that the error correction system based on Damerau-Levenshtein distance was quite good at correcting character-level errors during user input. This is extremely useful when the input method is via the keyboard. However, during speech input - by mistake (or due to different ways of speaking), a user may speak words in

a different order to what the system expect. A word-level (for example, words being deleted/inserted/swapped in a given sentence) distance calculation metric similar to the Damerau-Levenshtein method that we have applied could be of benefit in such circumstances. Also, this method can be extended to correction of phonetic error (errors in ASR) between similar sounding words – such as “theta”, “beta” and “eta”. More research on this topic should be of interest for any speech-based system, not just TalkMaths. Theoretically this would seem to be useful, however, carrying these out might be an excessively heavy computational task. SLMs could also be used to assess which sentences are highly unlikely to occur as a way of detecting word-level errors in a given sentence. It would be a valid experiment to compare these different methods with the input correction method described in this thesis.

As already mentioned in [Fateman, 2006], any speech-based system that wishes to offer the fullest flexibility and usability to its users needs to provide a facility for defining custom user-created commands. In our framework, this corresponds to adding new operators (in particular, mixfix operators corresponding to new speech templates), with appropriate precedence information, to the system. In principle, this does not pose any problem. However, a suitable user interface for this would have to be designed and implemented. We think that this desirable feature could be an important next step for improving TalkMaths as it would enable TalkMaths to become a generic, extensible editor for languages like Maths that are suitable for the parser and speech template approach implemented.

Additional future work could integrate a generic semantic analysis mechanism, for correctly interpreting arbitrary, potentially user-defined command types. This would improve the way the parser could distinguish between, e.g. editing, help or mathematical content commands and correctly identify the necessary action that the system has to undertake.

During the field study we have carried out to evaluate our TalkMaths interface, it was evident to us that the use of speech templates was quite natural for the users. The comments we received from the participants were encouraging and no one questioned the nature of the language they had to use. However, the focus of those experiments was not on the ease of use of the new spoken mathematics language, therefore it cannot be used to reach the conclusion that the speech templates were indeed easy and natural to users compared to earlier versions of spoken mathematics. An empirical evaluation

of the use of spoken mathematics that include speech templates will be highly beneficial in the future.

The editing paradigms implemented in TalkMaths produced much lively discussion at public demonstrations at Kingston University among post graduate user experience students. If we had time in hand, experimenting on the ease of use of those novel editing paradigms described in Chapter 7 would have been designed and carried out. Currently editing paradigms designed based on editing mathematics. It would also be a valid experiment to design and use our editing paradigms concept in the context of dictation of programming code and compare these with conventional speech editing paradigms (such as use of a mouse grid)

The prototype implementation of our parsing algorithm presented in Section 6.5.3 has been evaluated and its novel error recovery features have been illustrated and tested (see Section 6.6). Due to the complexity of the algorithm, we could not fine-tune all aspects of our implementation, and we did not integrate our code into the live TalkMaths system, publicly available on the web. Doing so would mainly require work relating to system integration and software engineering of a technical nature rather than academic research.

An interesting additional feature of our system could be an interactive mechanism to display various ranked alternatives of expressions offered to resolve ambiguity, as well as devising new ways of determining this ranking (currently, we use the number of nodes in the parse tree – the lower the better, as this gave us better results). For example, this mechanism could work similarly to how our SWIMS system offers suggested predictions and corrections to the users, and in principle, it would be possible to use the same JavaScript code to implement this for TalkMaths. However, this will also require additional processing power due to the need for analysis of possible outcomes using a suitable rule set, which therefore could affect the performance of the system.

A highly desirable direction for TalkMaths project is to make the system more accessible for people with visual impairments by incorporating facilities to output mathematics expressions on the screen using synthetic spoken text-to-speech (TTS) mathematics descriptions.

Ultimately, we believe that in future versions of TalkMaths, the concepts and algorithms that we have devised in this project and described in this thesis will form core components of the system, and will be further enhanced by additional improvements

and extensions. Eventually, the entire functionality described here will be available in the public interface and the power of our approach will be measured in terms of satisfied users, rather than academic case studies or prototype evaluations.

# References

- A.V. Aho and J.D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., 1972. 70
- A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, USA, world student series edition, 1986. ISBN 0201101947. 13, 70, 77, 79, 80
- A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: principles, techniques, & tools*, volume 1009. Pearson/Addison Wesley, 2007. 70
- J. Allaire. Macromedia Flash MXA next-generation rich client. *Macromedia white paper*, pages 1–2, 2002. 22
- A. Annika. Precedences in specifications and implementations of programming languages. *Theoretical Computer Science*, 142(1):3–26, 1995. 77, 79
- D. Attanayake, G. Hunter, J. Denholm-Price, and E. Pfluegel. TalkMaths—developing a speech user-interface for spoken mathematics. *MSOR Connections*, 11(2):44–46, 2011a. 48
- D. Attanayake, G. Hunter, E. Pfluegel, and J. Denholm-Price. Architectures for speech-based web applications. In *International Conference on Semantic E-Business and Enterprise Computing, 2011. SEEC 2012. Proceedings.*, pages 73–77, 2011b. 99
- D. Attanayake, G. Hunter, J. Denholm-Price, and E. Pfluegel. A novel web-based tool to enhance learning of mathematical concepts. In *Advances in ICT for Emerging Regions (ICTer), 2012 International Conference on*, pages 129–136. IEEE, 2012. 111, 115



- D. Attanayake, G. Hunter, J. Denholm-Price, and E. Pfluegel. Novel multi-modal tools to enhance disabled and distance learners experience of mathematics. *International Journal on Advances in ICT for Emerging Regions (ICTer)*, 6(1), 2013. 111
- A. Baddeley. Working memory: Looking back and looking forward. *Nature Reviews Neuroscience*, 4(10):829–839, 2003. 44
- L.R. Bahl, F. Jelinek, and R.L. Mercer. A maximum likelihood approach to continuous speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):179–190, 1983. 39, 47
- S. Bayer. Embedding speech in web interfaces. In *Proceedings of the Fourth International Conference on Spoken Language Processing, 1996. ICSLP 96.*, volume 3, pages 1684–1687. IEEE, 1996. 35
- A. Begel. *Spoken Language Support for Software Development*. PhD thesis, University of California, Berkeley, 2005. Report UCB-EECS-2006-8. 3, 5
- A. Begel and S.L. Graham. XGLR – an algorithm for ambiguity in programming languages. *Science of Computer Programming*, 61(3):211–227, 2006. 5, 13, 71, 75, 82
- A. Begel and K. Zafrir. SpeedNav: Document navigation by voice. unpublished, 2002. 18
- J.R. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech communication*, 42(1):93–108, 2004. 44
- A.L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:1–36, 1996. 47
- C. Bernareggi and V. Brigatti. Writing mathematics by speech: A case study for visually impaired. In *Proceedings of ICCHP 2008 (11th International Conference on Computers Helping People with Special Needs)*, pages 879–882, 2008. 19
- R.T. Bickel, O.E. Murillo, D. Mowatt, R.L. Chambers, and O. Scholz. Graphic user interface schemes for supporting speech recognition input systems, June 22 2010. US Patent 7,742,923. 18



- A. Bove, P. Dybjer, and U. Norell. A brief overview of Agda—a functional language with dependent types. In *Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009. 78
- J. Bresnan, R.M. Kaplan, S. Peters, and A. Zaenen. Cross-serial dependencies in Dutch. In *The Formal complexity of natural language*, pages 286–319. Springer, 1987. 12
- M.G. Burke and G.A. Fisher. A practical method for LR and LL syntactic error diagnosis and recovery. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(2):164–197, 1987. 14, 86
- P.M. Burke and S. Yacoub. Distributed speech recognition for mobile devices, November 24 2010. EP Patent 1,617,410. 25
- L. Burnard. User’s reference guide for the British National Corpus. <http://www.natcorp.ox.ac.uk/>, 1995. Accessed: 28/05/2013. 8, 50
- B. Cao, Y. Li, and J. Yin. Measuring similarity between graphs based on the Levenshtein Distance. *Appl. Math*, 7(1L):169–175, 2013. 11
- D. Carlisle. Mathematical markup language (MathML) version 2.0 (second edition). <http://www.w3.org/TR/MathML2>, 2003. Accessed: 6/8/2008. 97
- Carnegie Mellon University. Sphinx-4 A speech recognizer written entirely in the Java™ programming language. <http://cmusphinx.sourceforge.net/sphinx4/>, 2008. Accessed: 27/03/2013. 17, 35
- D. Cervone. Mathjax: A platform for mathematics on the web. *Notices of the AMS*, 59(2):312–316, 2012. 97
- L.A. Chang. *Handbook for spoken mathematics (Larry’s speakeasy)*. Lawrence Livermore National Laboratory, University of California, USA, 1983. 5, 8, 48, 120
- S. Chauveau and F. Bodin. Menhir: An environment for high performance MATLAB. In *Languages, Compilers, and Run-Time Systems for Scalable Computers*, pages 27–40. Springer, 1998. 13

- S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996. 42, 43
- X. Chen and D. Pager. LR (1) parser generator Hyacc. In *Proceedings of International Conference on Software Engineering Research and Practice*, pages 471–477, 2008. 13
- A. Cheyer, J. Park, and R. Giuli. IRIS: Integrate, relate. infer. share. Technical report, DTIC Document, 2005. 21
- N. Chomsky. *Syntactic structures*. de Gruyter Mouton, The Hague, Netherlands, 1957. 12, 38, 71
- N. Chomsky. *Aspects of the Theory of Syntax*. The MIT press, 1965. 38, 71
- K. Christian, B. Kules, B. Shneiderman, and A. Youssef. A comparison of voice controlled and mouse controlled web browsing. In *Proceedings of the fourth international ACM conference on Assistive technologies*, pages 72–79, 2000. ISBN 1581133148. 18, 75
- J. Ciesinger. A bibliography of error-handling. *SIGPLAN Notices*, 14(1):16–26, 1979. 14
- P. Clarkson. Statistical language modeling toolkit. <http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html>, 1999a. Accessed: 27/04/2013. 45
- P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of Fifth European Conference on Speech Communication and Technology (Eurospeech)*, pages 2707–2710, 1997. xi, 45, 52, 55
- P.R. Clarkson. *Adaptation of Statistical Language Models for Automatic Speech Recognition*. PhD thesis, 1999b. 10, 40, 42, 44
- P.R. Clarkson and A.J. Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-97. 1997*, volume 2, pages 799–802. IEEE, 1997. 44



- W.F. Clocksin and C.S. Mellish. Programming in PROLOG. 1984. 78
- D. Crombie, R. Lenoir, N. McKenzie, and A. Barker. math2braille: Opening access to mathematics. In *Computers Helping People with Special Needs*, volume 3118, pages 670–677. Springer, 2004. ISBN 978-3-540-22334-4. 20
- J. Cuartero-Olivera, G. Hunter, and A. Pérez-Navarro. Reading and writing mathematical notation in e-learning environments. *eLearn Center Research Paper Series, Universitat Oberta de Catalunya*, (4), 2012. 21, 50
- L. Dai, R. Goldman, A. Sears, and J. Lozier. Speech-based cursor control: a study of grid-based solutions. In *Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, pages 94–101, Atlanta, GA, USA, 2004. ACM. ISBN 158113911X. 18
- F.J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964. ISSN 0001-0782. doi: 10.1145/363958.363994. URL <http://doi.acm.org/10.1145/363958.363994>. 10, 47, 48, 65, 106
- N.A. Danielsson and U. Norell. Parsing mixfix operators. In Sven-Bodo Scholz and Olaf Chitil, editors, *Implementation and Application of Functional Languages*, volume 5836, pages 80–99. Springer, 2011. ISBN 978-3-642-24451-3. 77, 78
- M. Davies. The corpus of contemporary american english (COCA): 400+ million words, 1990-present. Available online at <http://www.americancorpus.org>, 2008. 142
- S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366, 1980. 25
- M. de Jonge, E. Nilsson-Nyman, L. Kats, and E. Visser. Natural and flexible error recovery for generated parsers. *Software Language Engineering*, pages 204–223, 2010. 14
- A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. 46

- F.L. DeRemer. *Practical translators for LR (k) languages*. PhD thesis, Massachusetts Institute of Technology, 1969. 13
- J. Dery. *Life With Siri*, volume 2. Jodi Dery, 2012. 21
- Design Science. MathType 6.9. <http://www.dessci.com/en/products/mathtype/>, 2013a. Accessed: 21/06/2013. 20
- Design Science. MathPlayer. <http://www.dessci.com/en/products/mathplayer/>, 2013b. Accessed: 20/06/2013. 19
- C. Donnelly and S. Richard. *Bison: The YACC-compatible Parser Generator (November 1995, Bison Version 1.25)*. Free Software Foundation, 1998. 13
- S. Durling and J. Lumsden. Speech recognition use in healthcare applications. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 473–478. ACM, 2008. 21
- A. D. N. Edwards, H. McCartney, and F. Fogarolo. Lambda: A multimodal approach to making mathematics accessible to blind students. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility, Assets '06*, pages 48–54, New York, NY, USA, 2006. ACM. ISBN 1-59593-290-9. doi: 10.1145/1168987.1168997. URL <http://doi.acm.org/10.1145/1168987.1168997>. 20
- C. Elliott and J.A. Bilmes. Computer based mathematics using continuous speech recognition. *CHI 2007 Workshop on Striking a C[h]ord: Vocal Interaction in Assistive Technologies, Games and More*, 2007. 2, 8, 18
- R. Fateman. 2-D Display of Incomplete Mathematical Expressions. <http://www.eecs.berkeley.edu/~fateman/papers/dispbad.pdf>, 2006. Accessed: 20/01/2011. 5, 8, 15, 121
- R. Fateman. How can we speak math? <http://www.eecs.berkeley.edu/~fateman/papers/speakmath.pdf>, 2009. Accessed: 25/07/2013. 5, 8, 9, 15, 48, 72, 73, 74, 120



- H. Ferreira and D. Freitas. Enhancing the accessibility of mathematics for blind people: The AudioMath project. In *Computers Helping People with Special Needs*, volume 3118, page 627, 2004. ISBN 978-3-540-22334-4. 19
- C.R. Fletcher. Levels of representation in memory for discourse. 1994. 44
- R. Flynn and E. Jones. Feature selection for reduced-bandwidth distributed speech recognition. *Speech Communication*, 2012. 25
- Fonix.com. <http://www.Fonix.com>. Accessed: 10/03/2013. 25
- A. Fujiyoshi, M. Suzuki, and S. Uchida. Verification of mathematical formulae based on a combination of context-free grammar and tree grammar. In *Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics*, volume 5144, pages 415–429, Heidelberg, 2008. Springer. ISBN 9783540851097. 19
- P. Gaura. REMathEx reader and editor of the mathematical expressions for blind students. In K. Miesenberger, J. Klaus, and W. Zagler, editors, *Computers Helping People with Special Needs*, volume 2398 of *Lecture Notes in Computer Science*, pages 486–493. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43904-2. doi: 10.1007/3-540-45491-8\_92. URL [http://dx.doi.org/10.1007/3-540-45491-8\\_92](http://dx.doi.org/10.1007/3-540-45491-8_92). 19
- H.T. Glantz. On the recognition of information with a digital computer. *J. ACM*, 4(2):178–188, April 1957. ISSN 0004-5411. doi: 10.1145/320868.320878. URL <http://doi.acm.org/10.1145/320868.320878>. 11
- I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264, 1953. 42, 52
- Google. Google speech demo. <http://www.google.com/intl/en/chrome/demos/speech.html>. Accessed: 27/03/2013. 17
- Google. Google translate. <http://translate.google.com/about>, 2012. Accessed: 27/03/2013. 11

- J. Gould. Implementation and acceptance of NatLink, a Python-based macro system for Dragon NaturallySpeaking. In *The Ninth International Python Conference*, pages 5–8, 2001. 2, 97
- J. Gould and D. Gould. Gould home page. <http://www.gouldhome.com/joel/>. Accessed: 27/03/2013. 23
- S.L. Graham and S.P. Rhodes. Practical syntactic error recovery. *Communications of the ACM*, 18(11):639–650, 1975. ISSN 0001-0782. URL <http://doi.acm.org/10.1145/361219.361223>. 14
- S.L. Graham, C.B. Haley, and W.N. Joy. *Practical LR error recovery*, volume 14. ACM, 1979. 14
- J.T. Grudin. Error patterns in novice and skilled transcription typing. In WilliamE. Cooper, editor, *Cognitive aspects of skilled typewriting*, pages 121–143. Springer, 1983. ISBN 978-1-4612-5472-0. URL [http://dx.doi.org/10.1007/978-1-4612-5470-6\\_6](http://dx.doi.org/10.1007/978-1-4612-5470-6_6). 10, 47
- A. Gruenstein, I. McGraw, and I. Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on Multimodal interfaces*, pages 141–148. ACM, 2008. 4, 24, 30
- A. Gruenstein, I. McGraw, and A. Sutherland. A self-transcribing speech corpus: collecting continuous speech with an online educational game. In *Speech and Language Technology in Education, 2011. SLaTE 2012 Workshop.*, 2009. 31
- C. Guy, M. Jurka, S. Stanek, and R. Fateman. Math Speak & Write, a computer program to read and hear mathematical input. Technical report, University of California, Berkeley, Electrical Engineering and Computer Sciences Department, 2004. 2, 8
- T. Hanakovič and M. Nagy. Speech recognition helps visually impaired people writing mathematical formulas. In K. Miesenberger, J. Klaus, W.L. Zagler, and A.I. Karshmer, editors, *Computers Helping People with Special Needs*, volume 4061, pages



- 1231–1234. Springer, 2006. ISBN 978-3-540-36020-9. doi: 10.1007/11788713\_177. URL [http://dx.doi.org/10.1007/11788713\\_177](http://dx.doi.org/10.1007/11788713_177). 19
- T. Haque, E. Liang, and J. Gray. The adjustable grid: A grid-based cursor control solution using speech recognition. In *Proceedings of the 51st ACM Southeast Conference*, pages 36:1–36:6, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1901-0. URL <http://doi.acm.org/10.1145/2498328.2500084>. 17
- S. Hasinoff. Solving substitution ciphers, a technical report. Technical report, University of Toronto, 2003. 39
- M. Holada. Internet speech recognition server. *Systemics, Cybernetics and Informatics*, 1(3):74–77, 2003. 25, 34
- J.N. Holmes and W.J. Holmes. *Speech synthesis and recognition*. Taylor and Francis, 2001. ISBN 0748408576. 23, 25, 38, 39, 40, 42, 44, 46
- B. Hsu. Generalized linear interpolation of language models. In *IEEE Workshop on Automatic Speech Recognition & Understanding, 2007, ASRU*, pages 136–140. IEEE, 2007. 43
- M. Huckvale. Learning from the experience of building automatic speech recognition systems. *Speech Hearing and Language-Work in Progress*, 9, 1996. 41
- A. Hunt and S. McGlashan. Speech recognition grammar specification version 1.0. *W3C Recommendation*, March. 23
- G.J.A Hunter. *Statistical language modelling of dialogue material in the British National Corpus*. PhD thesis, University College London, London, 2004. 38, 42, 46, 47
- G.J.A. Hunter and M. Huckvale. Is it appropriate to model dialogue in the same way as text? In *Proceedings of European Modelling Symposium*, pages 199–203, London, UK, 2006. ISBN 0-9516509-3-9. 51, 53
- R.M. Iyer and M. Ostendorf. Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *IEEE Transactions on Speech and Audio Processing*, 7(1):30–39, 1999. 44

- N. Jacobs. MathGenie : User's guide and teacher's manual. <http://logicalsoft.net/MathGenie.pdf>, 2006. 19
- F. Jelinek. Up from trigrams!-The struggle for improved language models. In *Proceedings of Second European Conference on Speech Communication and Technology (Eurospeech)*, 1991. 68
- F. Jelinek, B. Merialdo, S. Roukos, and M.I. Strauss. Self-organized language modeling for speech recognition. In *Readings in Speech Recognition*, pages 450–506. Morgan Kaufmann, 1990. 39
- S.C. Johnson. *Yacc: Yet another compiler-compiler*, volume 32. Bell Laboratories Murray Hill, NJ, 1975. 13
- LaViola Jr., J. Joseph, and R.C. Zeleznik. MathPad 2: a system for the creation and exploration of mathematical sketches. In *ACM SIGGRAPH 2007 courses*, page 46, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-1823-5. URL <http://doi.acm.org/10.1145/1281500.1281557>. 19
- L. Kallmeyer. *Parsing beyond context-free grammars*. Springer, 2010. ISBN 3642264530, 9783642264535. 12
- K. Karl, M. Pettey, and B. Shneiderman. Speech activated versus mouse-activated commands for word processing applications: An empirical evaluation. *International Journal of Man-Machine Studies*, 30(4):355–375, 1993. 18
- A. I. Karshmer. Access to mathematics and science. In Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer, editors, *Computers Helping People with Special Needs*, volume 5105, pages 873–874. Springer, 2008. ISBN 978-3-540-70539-0. URL [http://dx.doi.org/10.1007/978-3-540-70540-6\\_129](http://dx.doi.org/10.1007/978-3-540-70540-6_129). 21, 49
- U. Kastens. Ordered attributed grammars. *Acta Informatica*, 13(3):229–256, 1980. 15
- S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987. 42, 43, 58



- R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995*, volume 1, pages 181–184. IEEE, 1995. 58, 68
- D.E. Knuth. On the translation of languages from left to right. *Information and control*, 8(6):607–639, 1965. 13
- D.E. Knuth. The genesis of attribute grammars. In *Attribute Grammars and Their Applications*, pages 1–12. Springer, 1990. 15
- R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Lawrence Erlbaum Associates Ltd, 1995. 47
- R. Kuhn and R. De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990. 44
- K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439, 1992. 10
- B. Lang. Deterministic techniques for efficient non-deterministic parsers. In *Proceedings of the 2nd Colloquium on Automata, Languages and Programming*, volume 14, pages 255–269, 1974. 82
- S. Lappin and H.J. Leass. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535–561, 1994. 38
- J.A. Larson. *VoiceXML: Introduction to Developing Speech Applications*. Prentice Hall, USA, 2002. ISBN 0130092622. 24
- R. Lau. Adaptive statistical language modelling. Master's thesis, Department of Electrical and Computer Science, Massachusetts Insistute of Technology, Cambridge, MA, 1994. 43
- R. Lau, R. Rosenfeld, and S. Roukos. Trigger-based language models: a maximum entropy approach. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 45–48, Minneapolis, 1993. 46, 47

- R. Lau, G. Flammia, C. Pao, and V. Zue. Webgalaxy: beyond point and click - a conversational interface to a browser. *Computer Networks and ISDN Systems*, 29(8):1385–1393, 1997. 4
- G. Law. Phonetic alphabets (alpha bravo charlie delta). <http://www.faqs.org/faqs/radio/phonetic-alpha/full/>, 2009. Accessed: 22/01/2010. 103
- J.D. Lee, B. Caven, S. Haake, and T.L. Brown. Speech-based interaction with in-vehicle computers: The effect of speech-based e-mail on drivers' attention to the roadway. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 43(4):631–640, 2001. 21
- R.P. Leinius. Error detection and recovery for syntax directed compiler systems. 1970. 14
- V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady*, 10(8):707–710, 1966. 10, 47, 65, 106
- P.M. Lewis II and R.E. Stearns. Syntax-directed transduction. *Journal of the ACM (JACM)*, 15(3):465–488, 1968. 13
- J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. String similarity measures and joins with synonyms. 2013. 11
- J.B. Marino, R.E. Banchs, J.M. Crego, A. De Gispert, P. Lambert, J.A.R. Fonollosa, and M.R. Costa-Jussà. N-gram-based machine translation. *Computational Linguistics*, 32(4):527–549, 2006. 39
- C. Martins, A. Teixeira, and J. Neto. Dynamic language modeling for the European Portuguese. In *Computational Processing of the Portuguese Language*, pages 264–267. Springer, 2008. 10
- Mathematica. Spokenstring – Wolfram Mathematica 9 documentation. <http://reference.wolfram.com/mathematica/ref/SpokenString.html>, 2014. 9
- E. Mays, F.J. Damerau, and R.L. Mercer. Context based spelling correction. *Information Processing & Management*, 27(5):517 – 522, 1991. ISSN 0306-4573. doi:



- 10.1016/0306-4573(91)90066-U. URL <http://www.sciencedirect.com/science/article/pii/030645739190066U>. 47
- V. Mazalov and S. Watt. Recommendation systems in mathematical character recognition. In *Proceedings of MathUI*, Bath, UK, 2013. 19
- M. F. McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, 2002. 21
- Metroplex Voice Computing, Inc. mathtalk.com. <http://www.mathtalk.com/>. Accessed: 19/10/2013. 2, 18
- Microsoft Corporation. Speech application language tags (SALT). <http://msdn.microsoft.com/en-us/library/ms994629.aspx>. Accessed: 27/03/2013. 23
- R. Mohr. Vocola information pages. <http://vocola.net/>, 2009. Accessed: 27/13/2013. 17, 24
- R.K. Moore. There's no data like more data. In *Proceedings of the Institute of Acoustics*, volume 23, pages 19–26, 2001. 44, 51
- R.K. Moore. A comparison of the data requirements of automatic speech recognition systems and human listeners. In *Proc. Eurospeech, Geneva*, pages 2582–2584, 2003. 44, 51
- Mozilla Foundation. XULRunner. <https://developer.mozilla.org/En/XULRunner>, 2009. Accessed: 22/01/2010. 97
- G.L. Nemhauser. *Introduction to dynamic programming*. Wiley New York, 1966. 11
- H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8(1):1–38, 1994. 42, 43
- Nuance Communications. Dragon NaturallySpeaking 10 sdk server edition. [http://www.nuance.com/naturallyspeaking/products/sdk/sdk\\_server.asp](http://www.nuance.com/naturallyspeaking/products/sdk/sdk_server.asp). Accessed: 27/03/2013. 35, 97

- Nuance Communications. Dragon NaturallySpeaking Professional. <http://www.nuance.co.uk/for-business/by-product/dragon/dragon-for-the-pc/dragon-professional/>, 2013. Accessed: 27/03/2013. 16
- D. Pager. A practical general method for constructing LR (k) parsers. *Acta Informatica*, 7(3):249–268, 1977. 13, 14
- D. Pager and X. Chen. The lane table method of constructing LR (1) parsers. Technical report, Technical Report No. ICS2009-06-02, University of Hawaii, Information and Computer Sciences Department, 2008. 13
- T. Parr and K. Fisher. LL (\*): the foundation of the ANTLR parser generator. *ACM SIGPLAN Notices*, 47(6):425–436, 2012. 14
- A. Patel. Parsing with yapps. <http://theory.stanford.edu/~amitp/yapps/>, 2009. Accessed: 22/01/2010. 14, 51, 100, 147
- L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1996. 78
- F. Pereira. Formal grammar and information theory: together again? *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 358(1769):1239–1253, 2000. 41
- J.J. Pollock and A. Zamora. Collection and characterization of spelling errors in scientific and scholarly text. *Journal of the American Society for Information Science*, 34(1):51–58, 1983. 10
- P. J. Pugliese and J. M. Gould. Voice controlled cursor movement, 1998. US Patent No. 5818423. 18
- M. Purver and J. Ginzburg. Clarifying noun phrase semantics. *Journal of Semantics*, 21(3):283–339, 2004. 38
- B. Radakovic, G. Predovic, and B. Dresevic. Geometric parsing of mathematical expressions, November 22 2011. US Patent 8,064,696. 19



- N. Rajput and A.A. Nanavati. Distributed speech recognition. *Speech in Mobile and Pervasive Environments*, pages 99–114, 2012. 25
- T.V. Raman. *Audio system for technical readings*. Springer Verlag, Berlin, 1998. ISBN 3540655158. 9, 19
- J.G. Rekers. *Parser generation for interactive environments*. PhD thesis, University of Amsterdam, 1992. 82
- R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187–228, 1996. 47
- R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? In *Proceedings of the IEEE*, volume 88, pages 1270–1278, 2000. 10, 43
- Rubidium.com. <http://www.Rubidium.com>. Accessed: 10/03/2013. 25
- D.A. Schleppenbach. 2004 MathSpeak initiative.. <http://www.gh-mathspeak.com/>, 2013. Accessed: 28/06/2013. 9
- A. Sears, C. Karat, K. Oseitutu, A. Karimullah, and J. Feng. Productivity, satisfaction, and interaction strategies of individuals with spinal cord injuries and traditional users interacting with speech recognition software. *Universal Access in the Information Society*, 1(1):4–15, 2001. ISSN 16155289. 17
- Andrew Sears, Jinjuan Feng, Kwesi Oseitutu, and Clare-Marie Karat. Hands-free, speech-based navigation during dictation: Difficulties, consequences, and solutions. *Human-Computer Interaction*, 18:229–257, 2003. 18
- SensoryInc.com. <http://www.Sensoryinc.com>. Accessed: 10/03/2013. 25
- C. Severance. *Using Google App Engine*. O'Reilly, 2009. 98
- K.R. Shah, M. Patel, J. Sheth, and K. Lad. Hybrid approach for measuring string similarity and its usage in supply type questions answer evaluation. *IJCER*, 1(3): 81–87, 2012. 11
- C.F. Shannon. Prediction and entropy of printed English text. *Bell System Technical Journal*, 30:50–64, 1951. 46, 47

- S.M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th conference on Computational linguistics-Volume 2*, pages 614–619. Association for Computational Linguistics, 1988. 12
- S. Sippu. *Syntax error handling in compilers*. University of Helsinki, Department of Computer Science, 1981. 14
- E. Smirnova and S.M. Watt. Context-sensitive mathematical character recognition. In *Proc. IAPR International Conference on Frontiers in Handwriting Recognition, (ICFHR 2008)*, pages 604–610. Citeseer, 2008. 4, 19
- A. Sorkin and P. Donovan. LR (1) parser generation system: LR (1) error recovery, oracles, and generic tokens. *ACM SIGSOFT Software Engineering Notes*, 36(2):1–5, 2011. 13
- Speechfxinc.com. <http://www.Speechfxinc.com>. Accessed: 10/03/2013. 25
- B. Suhm, B. Myers, and A. Waibel. Designing interactive error recovery methods for speech interfaces. In *Proceedings of the CHI96 Workshop on Designing the User Interface for Speech Recognition Applications*. Citeseer, 1996. 15
- M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori. Infy: an integrated ocr system for mathematical documents. In *Proceedings of the 2003 ACM symposium on Document engineering*, pages 95–104. ACM, 2003. 19
- M. Suzuki, S. Uchida, and A. Fujiyoshi. Syntactic detection and correction of misrecognitions in mathematical ocr. In *Proceedings of The 10th International Conference on Document Analysis and Recognition, ICDAR 2009, Barcelona, Spain*, pages 1360–1364, 2009. 4, 19
- M. Tomita. *Efficient parsing for natural language: a fast algorithm for practical systems*, volume 8. Springer, 1985. 13, 82
- J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528. ACM, 2007.



- M. Ugen. Correcting errors by providing structural information to parsers. 2010. 14
- A. Vaičiūnas and G. Raškinis. Cache-based statistical language models of english and highly inflected lithuanian. *Informatica*, 17(1):111–124, 2006. 10
- H. van de Koot. Personal Communication, 2013. 12
- K. Vertanen and D.J.C. MacKay. Speech dasher: fast writing using speech and gaze. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 595–598. ACM, 2010. 20, 41
- W3C. Mathematical markup language (MathML) version 2.0 (second edition). <http://www.w3.org/TR/MathML2>, 2003. Accessed: 06/08/2008. 97
- W3C. Mathematical markup language (MathML) version 3.0. <http://www.w3.org/TR/MathML3/>, 2010. Accessed: 23/02/2011. 97
- R.A. Wagner. Order-n correction for regular languages. *Communications of the ACM*, 17(5):265–268, 1974. 10
- R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974. 10
- R.A. Wagner and R. Lowrance. An extension of the string-to-string correction problem. *Journal of the ACM (JACM)*, 22(2):177–183, 1975. 11, 48
- S. M. Watt and X. Xie. Recognition for large sets of handwritten mathematical symbols. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 740–744. IEEE, 2005. 19
- C. Wetherell and A. Shannon. LR automatic parser generator and LR (1) parser. *Software Engineering, IEEE Transactions on*, (3):274–278, 1981. 13
- A.M. Wigmore. *Speech-Based Creation and Editing of Mathematical Content*. PhD thesis, Kingston University, UK, 2011. xiv, 3, 4, 5, 8, 15, 18, 36, 48, 50, 51, 55, 72, 77, 98, 105, 106, 109, 110, 111, 116, 118, 119, 120, 142



- A.M. Wigmore, Hunter G., E. Pfluegel, and J. Denholm-Price. TalkMaths : A speech user interface for dictating mathematical expressions into electronic documents. In *2nd ISCA Workshop of Speech and Language Technology in Education (SLaTE 2009)*, page P3.4, 2009a. 50
- A.M. Wigmore, G. Hunter, E. Pfluegel, J. Denholm-Price, and V. Binelli. "let them TalkMaths !" Developing an intelligent system to assist disabled people to learn and use mathematics on computers through a speech interface : the TalkMaths and VoiceCalc systems. In *Proceedings of 5th AAAI International Conference on Intelligent Environments (IE'09)*, 2009b. 4, 50
- A.M. Wigmore, E. Pfluegel, G. Hunter, J. Denholm-Price, M. Colbert, and D. Atanayake. Evaluating and improving the TalkMaths speech interface for dictating and editing mathematical text. In *5th European Workshop on Mathematical and Scientific E-Contents*, 2010. 51, 55
- wikipedia.org. Timsort. <http://en.wikipedia.org/wiki/Timsort>. Accessed: 26/07/2013. 92
- I.H. Witten and T.C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991. 42
- S. Young. Large vocabulary continuous speech recognition: A review. *IEEE Signal Processing Magazine*, 13(5):45–57, 1996. 10, 12, 16, 23, 38
- S. Young. Talking to machines (statistically speaking). In *Seventh International Conference on Spoken Language Processing*, 2002. 10
- T. Zhemina and C.L. Philipos. Speech recognition over the internet using Java. In *Proceedings of the Acoustics, Speech, and Signal Processing*, volume 4, pages 2367–2370, 1999. ISBN 0780350413. 25, 30
- P. Zielinski. Opengazer: open-source gaze tracker for ordinary webcams. *Samsung and The Gatsby Charitable Foundation*, 2013. 20

# Appendix A

## SLM Training Data Comparisons

In Chapter 5, we have described how we have created a new spoken mathematics training data corpus. It is sensible to view our perplexity results compared to some already known data corpuses to see the full picture. The tables in this appendix illustrate different comparisons of our web crawled SLM training data with other known corpuses. The full description of these corpuses used in each tables are:

- COCA – Corpus of Contemporary American English (COCA) – non case sensitive [Davies, 2008] (419845044 words in total, containing 629895 different words)
- BNC Maths – selected subset of the BNC(The British National Corpus) dialogue material (15 files – 123821 words in total, containing 4355 different words), composed of transcriptions of audio recordings from school, college and university mathematics lessons, described in [Wigmore, 2011]
- Trig Maths Data – a hand-typed dataset created from symbols from the “trigonometry” chapters of some secondary school GCSE mathematics textbooks, described in [Wigmore, 2011] (7857 words in total, containing 102 different words)
- Web Crawled Maths – spoken mathematics dataset extracted from mathematical tutorial web sites in the public domain covering material at roughly GCSE & GCE A-Level or “Senior-High School level” mathematics, described in this thesis, in Chapter 5 (61479 words in total, containing 100 different words)

	COCA (Gen)	BNC (Maths)	Trig Maths Data	Web Crawled Maths
Total Words	419845044	123821	7857	61479
Vocab	629895	4355	102	100
1	the (0.05%)	one (1.39%)	= (10.03%)	end (7.58%)
2	of (0.03%)	two (0.87%)	2 (5.24%)	begin (7.57%)
3	to (0.03%)	three (0.67%)	° (4.77%)	x-ray (7.30%)
4	and (0.03%)	X (0.47%)	sin (4.37%)	of (7.26%)
5	a (0.02%)	four (0.41%)	^ (superscript or "to the power") (3.91%)	two (4.82%)
6	in (0.02%)	six (0.41%)	/ (3.37%)	to (4.32%)
7	that (0.01%)	five (0.40%)	x (2.60%)	the (3.93%)
8	I (0.01%)	hundred (0.39%)	cos (2.20%)	power (3.93%)
9	is (0.01%)	minus (0.36%)	a (2.07%)	equals (3.79%)
10	for (0.01%)	times (0.31%)	x (2.07%)	bracket (3.61%)

Table A.1: Comparison of Overall words &amp; most common words

	COCA (Gen)	BNC (Maths)	Trig Maths Data	Web Crawled Maths
Total Bigrams	286758206	123820	7856	61478
1	of the (0.90%)	a hundred (0.4%)	$^{\circ} =$ (5.24%)	the power (3.93%)
2	in the (0.71%)	hundred and (0.3%)	$\wedge 2$ (or $^2$ ) (3.55%)	to the (3.93%)
3	to the (0.37%)	take away (0.3%)	$2=$ (1.18%)	power of (3.93%)
4	on the (0.32%)	of a (0.2%)	$2+$ (1.04%)	of two (2.1%)
5	and the (0.26%)	an hour (0.2%)	$/2$ (0.88%)	over begin (1.85%)
6	to be (0.23%)	one of (0.2%)	$/\sin$ (0.85%)	of begin (1.76%)
7	at the (0.22%)	equal to (0.1%)	$=\sin$ (0.80%)	end over (1.68%)
8	for the (0.21%)	X squared (0.1%)	$C=$ (0.79%)	x-ray to (1.66%)
9	in a (0.19%)	a quater (0.1%)	$2-$ (0.74%)	close bracket (1.55%)
10	do n't (0.19%)	minus one (0.1%)	$x=$ (0.73%)	open bracket (1.54%)

Table A.2: Most Common Bigrams

	Corpus COCA (Gen)	BNC (Maths)	Trig Maths Data	Web Crawled Maths
Total Tri- grams	128125547	123819	7855	61477
1	i do n't (0.16%)	miles an hour (0.06%)	$\hat{2} =$ (or 2=) (1.13%)	to the power (3.93%)
2	one of the (0.13%)	hundred snd twenty (0.06%)	$\hat{2}+$ (or 2+) (1.02%)	the power of (3.93%)
3	a lot of (0.11%)	D Y by (0.04%)	$\hat{2}+$ (or 2-) (0.74%)	power of two (2.05%)
4	the united states (0.1%)	Y by D (0.04%)	$a\hat{2}$ (or a2) (0.66%)	x-ray to the (1.66%)
5	do n't know (0.06%)	take away a (0.04%)	$b\hat{2}$ (or b2) (0.61%)	end over begin (1.2%)
6	out of the (0.06%)	X plus two (0.04%)	1/2 (or ) (0.61)	delta x-ray end (0.88%)
7	as well as (0.06%)	Y equals X (0.04%)	$c\hat{2}$ (or c2) (0.57%)	power of begin (0.73%)
8	going to be (0.06%)	square root of (0.03%)	=1/ (0.56%)	close bracket end (7.2%)
9	some of the (0.05%)	hundred and eighty (0.03%)	2 - 2 (0.45%)	of two end (0.67%)
10	you do n't (0.05%)	two three four (0.03%)	x)= (0.34%)	begin open bracket (0.63%)

Table A.3: Most Common Trigrams

BNC (Maths)			Trig Maths Data			Web Crawled Maths		
Training Words	Test Words	Perplexity	Training Words	Test Words	Perplexity	100 Training Words	Test Words	Perplexity
61913	61913	238.85	3928.5	3928.5	18.97	54907	6572	7.07
82551.33	41275.67	221.61	5238	2619	17.18	54968	6511	7.17
92870.25	30956.75	210.13	5892.75	1964.25	16.46	55294	6185	7.11
99001.6	24765.4	206	6285.6	1571.4	16	55688	5791	7.31
103188.33	20637.67	200.05	6547.5	1309.5	15.47	55172	6307	7.25
106136.57	17689.43	194.05	6729.29	1122.43	14.84	55597	5882	7.74
108347.75	15478.25	185.27				55340	6139	7.04
110068.44	13758.56	196.18				55805	5674	7.65
111444.3	12382.7	188.57				55177	6302	7.53
117635.65	6191.35	181.3				55363	6116	7.02

Table A.4: Perplexity scores as function of training set size

# Appendix B

## LaTeX to Spoken Mathematics Yapps2 Parser

As described in Chapter 5, we have used the parser generator tool Yapps2 [Patel, 2009] to translate web-crawled LaTeX mathematical expressions into spoken mathematics. Below are the grammar rules written using the Yapps2 grammar specification. Some parts were omitted for readability.

```
parser Latex:
```

```
# tokens for templates
```

```
token FRACTION: "\\frac{"
```

```
token OVER: "\\over"
```

```
# tokens for operators used in arithmetic expressions
```

```
token PLUS: "+"
```

```
token MINUS: "-"
```

```
token PLUS_MINUS: "+-|\\pm"
```

```
token MINUS_PLUS: "-+|\\mp"
```

```
token TIMES: "*|\\times"
```

```
token DIVIDE: "/"
```

```
token POWER: "^"
```

```
token SQUARED: "_squared"
```



```
token CUBED: "_cubed"
token BEGIN: "_quantity"
token END: "$"
token ENDS: "_all"

token ASSIGN: ":@"
token EQUALS: "="
token COMMA: ","
token LESS_THAN: "<"
token GREATER_THAN: ">"
token LESS_THAN_E: "<=|\\\\leq"
token GREATER_THAN_E: ">=|\\\\geq"

# while using left and right as alternatives for open and close
token OPEN_BRACKET: "\("
token OPEN_CURLY_BRACKET: "\{"
token OPEN_SQUARE_BRACKET: "\["
token CLOSE_BRACKET: "\)"
token CLOSE_CURLY_BRACKET: "\}"
token CLOSE_SQUARE_BRACKET: "\]"

token OPEN_CURLY_ACTUAL_BRACKET: "\\\\{"
token CLOSE_CURLY_ACTUAL_BRACKET: "\\\\}"

# tokens for functions
token BEGINFUNCTION: "_function"
token OF: "_of"

# tokens for trigonometry
token SIN: "\\\\sin"
token COS: "\\\\cos"
token TAN: "\\\\tan"
token LOG: "\\\\log"
```

```
token LN: "\\ln"
token DEGREE: "_degree"
token THETA: "\\theta"
token DELTA: "\\delta"
token ALPHA: "\\alpha"
token PI: "\\pi"
token DOT: "\\cdot"
token PRIME: "\\prime"
#token IN: "\\in"
token SEMICOLON: "\\;|"
token MATHBF: "\\mathbf{"

# tokens for numbers
token AHUNDRED: "_hundred"
token ATHOUSAND: "_thousand"
token AMILLION: "_million"

#tokens for roots
token SQUARE_ROOT: "\\sqrt"
token NTHROOT: "_root"

#tokens for integral
token INTEGRAL: "\\int"
token FROM: "_from"
token TO: "_to"

token GREEKID: "\\&#x[0-9]+[A-Z][0-9];"
token EMPTY: ""
token PERCENT: "_percent"
token TILDE: "_tilde"
token OVERBAR: "_overbar"
token AND: "_and"
```

token SUPERSCRIPT: "\_superscript"

token SUBSCRIPT: "\_"

token IDENTIFIER: "[a-zA-Z]|\\"?"

token DIGIT: "[0-9]+"

ignore: "\\s+"

rule goal: expr END {{ return expr }}

# An expression is one or several arithmetic expressions, separated by  
 # relational operators (<, >, <=, >=, =) or the assignment operator (:=)  
 # these operators are of lowest priority

rule expr:

arithmetic\_expr

{{ v = arithmetic\_expr }}

( LESS\_THAN arithmetic\_expr

{{ v = ['<',v,arithmetic\_expr ] }}

| GREATER\_THAN arithmetic\_expr

{{ v = ['>',v,arithmetic\_expr ] }}

| LESS\_THAN\_E arithmetic\_expr

{{ v = ['<=',v,arithmetic\_expr ] }}

| GREATER\_THAN\_E arithmetic\_expr

{{ v = ['>=',v,arithmetic\_expr ] }}

| EQUALS arithmetic\_expr

{{ v = ['=',v,arithmetic\_expr ] }}

| ASSIGN arithmetic\_expr

{{ v = [':=',v,arithmetic\_expr ] }}

| COMMA arithmetic\_expr

{{ v = [',',v,arithmetic\_expr ] }}

)\*

{{ return v }}

# An arithmetic expression is a sum or difference of a

```

    # signed term and one or several unsigned terms.
# This means it is a signed term,
    # followed by '+', '-' or '+-' and an unsigned term
rule arithmetic_expr:
    (signed_term
    {{ v = signed_term }}
    | unsigned_term
    {{ v = unsigned_term }}
    )
    ( PLUS unsigned_term
    {{ v = ['+',v,unsigned_term] }}
    | MINUS unsigned_term
    {{ v = ['- ',v,unsigned_term] }}
    | PLUS_MINUS unsigned_term
    {{ v = ['+- ',v,unsigned_term] }}
    | MINUS_PLUS unsigned_term
    {{ v = ['-+',v,unsigned_term] }}
    )*
    {{ return v }}

# A signed term consists of a
    # sign ('+', '-' or '+-') followed by an unsigned term
rule signed_term:
PLUS unsigned_term
{{ return ['plus_sign',unsigned_term] }}
| MINUS unsigned_term
{{ return ['minus_sign',unsigned_term] }}
| PLUS_MINUS unsigned_term
{{ return ['plus_minus_sign',unsigned_term] }}
| MINUS_PLUS unsigned_term
{{ return ['minus_plus_sign',unsigned_term] }}

```

```

# An unsigned term is a product or division of factors
rule unsigned_term:
#FRACTION factor {{ v = factor }}
    CLOSE_CURLY_BRACKET OPEN_CURLY_BRACKET factor
    {{ v = ['fraction',v,factor] }} CLOSE_CURLY_BRACKET {{v=v}}
factor
{{ v = factor }}
(
    TIMES factor
    {{ v = ['*',v,factor] }}
    | DIVIDE factor
    {{ v = ['/',v,factor] }}
    | OVER factor
    {{ v = ['fraction',v,factor] }}
    | DOT factor
    {{ v = ['dot',v,factor] }}
    | factor
    {{ v = ['invisibletimes',v,factor] }}
)*

{{ return v }}

rule factor:
base
{{v=base}} (
POWER factor
{{v=['^',base,factor]}}
| SUPERSCRIPT factor
{{v=['superscript',base,factor]}}
| SUBSCRIPT factor
{{v=['subscript',base,factor]}}
| SQUARED
{{v=['^',base,2]}}

```

```

| CUBED
{{v=['^',base,3]}}
| )
{{return v}}

rule base:
  argument {{v=argument}}
    [OPEN_BRACKET arithmetic_expr
    CLOSE_BRACKET {{v=['function',v,arithmetic_expr]}}]
  {{ return v}}

rule argument:
  root
  {{ return root }}
  |fraction
  {{ return fraction }}
  |integration
  {{ return integration }}
  | number
  {{ return number }}
  | trig
  {{ return trig }}
  | OPEN_BRACKET arithmetic_expr (
  CLOSE_BRACKET
  {{b='()'}}
  | CLOSE_SQUARE_BRACKET
  {{b='[]'}}
  | {{b='()'}}
  )
  {{ return [b,arithmetic_expr]}}
  ...

| OPEN_SQUARE_BRACKET arithmetic_expr (

```

```

CLOSE_SQUARE_BRACKET
  {{b='[]'}}
  | CLOSE_BRACKET
  {{b='()' }}
  |
  {{b='[]'}}
)
  {{ return [b,arithmetic_expr]}}
  | BEGIN arithmetic_expr ENDS
  {{return arithmetic_expr}}
  | MATHBF arithmetic_expr CLOSE_CURLY_BRACKET {{return arithmetic_expr}}
  | EMPTY {{ return '?' }}
  | IDENTIFIER {{v=IDENTIFIER}} ["\|" {{v=[v, '|']}}] {{ return v }}
  | "\|" IDENTIFIER {{ return ['|', IDENTIFIER] }}
  | THETA {{return 'theta'}}
  | DELTA {{return 'delta'}}
  | ALPHA {{return 'greek alpha'}}
  | PI {{return 'pi'}}
  | SEMICOLON {{return 'semi-colon'}}
  | PRIME {{return 'prime'}}

```

rule integration:

INTEGRAL

( ...

rule accents:

TILDE IDENTIFIER (END | )

{{ return ['~',IDENTIFIER] }}

| OVERBAR arithmetic\_expr (END | )

{{ return ['overbar',arithmetic\_expr] }}

rule root:

SQUARE\_ROOT



( ...

rule fraction:

```
FRACTION arithmetic_expr {{n=arithmetic_expr}}
      CLOSE_CURLY_BRACKET OPEN_CURLY_BRACKET arithmetic_expr
      {{d=arithmetic_expr}} CLOSE_CURLY_BRACKET
{{ return ['fraction', n, d] }}
```

rule trig:

```
SIN arithmetic_expr
{{ return ['sin', arithmetic_expr] }}
| COS arithmetic_expr
{{ return ['cos', arithmetic_expr] }}
| TAN arithmetic_expr
{{ return ['tan', arithmetic_expr] }}
| LOG arithmetic_expr
{{ return ['log', arithmetic_expr] }}
| LN arithmetic_expr
{{ return ['ln', arithmetic_expr] }}
```

rule number:

```
DIGIT {{v = eval(DIGIT)}} {{return v}}
```

# Appendix C

## Sample Parser Programming Code

### GLR Parser

This section gives our GLR Parser code from the TalkMaths web service. For readability, some parts are abbreviated. The language used is Python 2.5.

```
# Filename: GLRParser.py
# Creation Date: 16 April 2012, 14:33:33
# Last Modified $Date: 2013-12-13 15:57:46 +0000 (Wed, 13 Feb 2013) $
# Explanation: Constructs a GLR parser.
from copy import copy
from OperatorPrecedenceParser import *
class GLRParser:
    # constructor
    def __init__(self, SpokenLanguageData, logger):
        self.id = 'GLRParser' + str(id(self))
        self.SpokenLanguageData = SpokenLanguageData
        self.results = []
        self.logger = logger
        self.logger.addLog(...)

    def doParse(self, inputString, parseMode):
        self.parseSet = []
        self.parseSet.append(OperatorPrecedenceParser(...))
```

```
while self.parseSet <> []:
    updatedParserSet = range(0, len(self.parseSet))
    self.newParsers = []
    for i in range(0, len(self.parseSet)):
        p = self.parseSet[i]
        if p.state == 'READY_TO_TERMINATE':
            # This parser completed it's work.
            # Add to completed list
            updatedParserSet.remove(i)
            if type(p.parseForest) is ListType:
                for i in p.parseForest:
                    self.results.append(i)
            else:
                self.results.append(p.parseForest)
        elif p.state == 'READY_TO_PARSE':
            # This parser is ready to act, doWork
            p.doWork()
        elif p.state == 'READY_TO_FORK':
            # This parser is ready to fork
            p.doWork()
            self.newParsers = self.newParsers + p.children
    for j in updatedParserSet:
        self.newParsers.append(self.parseSet[j])
    self.parseSet = self.newParsers
    tmpParseSet = copy(self.parseSet)
    for p in tmpParseSet:
        # Remove all parent parsers which forked
        if p.state == 'READY_TO_BECOME_INACTIVE':
            self.parseSet.remove(p)
# We now run an analysis to count various things
for tree in self.results:
    tree.collectSeparators()
    tree.sort_Argument_Separators()
```

```
tree.detangle_Argument_Separators()
tree.complete_Argument_Separator_Parts()
tree.eliminate_Redundant_Argument_Separator_Parts()
# sort by number of mixfix operators
# check once for top most node
# and if it is separator(s), then upgrade to mixfix
if tree.node.getType() in ['separator', 'separators']:
    tree.node.upgrade('mixfix_operator', [])
tree.countMixFix()
self.results.sort(key=lambda x: x.mixfixOperatorCount)
return self.results
```

## OP Parser

The Operator Precedence Parser code from the TalkMaths web service is illustrated below. Similar to the GLR Parser code, some parts have been abbreviated for readability.

```
# Filename: OperatorPrecedenceParser.py
# Creation Date: 16 April 2012, 14:33:33
# Last Modified $Date: 2013-12-13 15:57:46 +0000 (Wed, 13 Feb 2013) $
# Explanation: Constructs a Operator Precedence Parser.
# Assume a token list is provided after Lexing
from OperatorPrecedence.Scanner import Scanner
from GoogleApp.ParseTree import *
from GoogleApp.Node import *
from copy import copy, deepcopy
from GoogleApp.Logger import *
import sys
import re
from Token import Token

class OperatorPrecedenceParser:
    def __init__(..., *args):
```

```
# This is the constructor for the operator precedence parser.
self.id = 'OPParser' + str(id(self))
self.parent = parent
self.spokenLanguageData = spokenLanguageData
self.inputString = inputString
self.parseMode = parseMode
self.logger = logger
self.children = []
self.pause = False
...

if len(args)== 0:
    # We create a new scanner object
    self.myScanner = Scanner(...)
    # We initialise the stack with begins token
    self.stack = [self.myScanner.token]
    # We let the scanner do some work
    # in order to skip begin token
    self.myScanner.doWork()
    # We let the scanner do some work until we can either
    # retrieve a unique token (which is not the begin token,
    # we will skip this), or the scanner needs forking
    while self.myScanner.state not in [...]:
        self.myScanner.doWork()
    # We initialise the stack with begins token
    self.state = "READY_TO_PARSE"
    #self.token = ('', '')
    self.token = Token(...)
    self.parseForest = []
    self.shift_reduce = ''
else:
    self.myScanner = args[0]
    self.stack = copy(args[1])
```

```
self.parseForest = deepcopy(args[2])
self.state = args[3]
self.shift_reduce = args[4]

def doWork(self ):
    if self.state == "READY_TO_PARSE":
        # We perform Operator precedence parsing,
        # involving shift and reduce actions depending on the
        # precedence relation between the token on
        # top of the stack and the current token
        while self.state == "READY_TO_PARSE":
            # Depending on the scanner state,
            # we do some specific actions
            if self.myScanner.state == 'READY_TO_RETURN_UNIQUE_TOKEN':
                self.token = self.myScanner.token
                # In this state, the parser is ready to act,
                # depending on the information on the precedence table
                self.topToken = self.stack[-1]
                if self.shift_reduce == '':
                    # A new parser created by forking in
                    # shift-reduce error
                    thisPrec = self.computePrecedence(...)
                else:
                    thisPrec = self.shift_reduce
                    # Set the shift_reduce to nothing to make sure
                    # next time thisPrec is calculated in the usual way
                    self.shift_reduce = ''
            if (thisPrec == '<') or (thisPrec == '='):
                # shift
                self.logger.addLog(...)
                self.stack.append(self.token)
                # Now we change state of the scanner
                self.myScanner.doWork()
```

```
# We scan until an external state is reached
while self.myScanner.state
not in [...]:
    self.myScanner.doWork()

elif thisPrec == '>':
    # In this case, we need to reduce
    self.logger.addLog(...)
    i = len(self.stack)-1
    while (self.computePrecedence(...)):
        i = i -1
    l = len(self.stack)
    temp = []
    for j in range(i, l):
        temp.append(self.stack.pop())
    # We call updateParseForest() with
    # all reduced tokens as input
    self.parseForest = self.updateParseForest(temp)

elif thisPrec == '<>':
    # We will fork in order to do
    # a shift and reduce in parallel
    self.state = "READY_TO_FORK"

elif thisPrec == 'STOP':
    # We have finished the parsing
    self.state = "READY_TO_TERMINATE"
else:
    # The precedence table indicates an error state
    self.errorRecovery(thisPrec)

elif self.myScanner.state == 'READY_TO_FORK':
    self.state = 'READY_TO_FORK'
```



```
# We continue to do some more work with the scanner
self.myScanner.doWork()

elif self.myScanner.state == 'READY_TO_TERMINATE':
    self.state = "READY_TO_TERMINATE"

else:
    self.myScanner.doWork()

elif self.state == "READY_TO_FORK":
    #if self.parseMode == 'all':
    if True:
        # now main script checks for shortest
        # bracket count for this
        #if thisPrec == '<>':
        if self.myScanner.children == []:
            # we fork because of different brackets case
            # Create a copy of self
            self.logger.addLog(...)
            # Call constructor by setting shift_reduce flag to shift
            myCopy1 = OperatorPrecedenceParser(...)
            self.children.append(myCopy1)
            # Call constructor by setting shift_reduce flag to reduce
            myCopy2 = OperatorPrecedenceParser(...)
            self.children.append(myCopy2)
        else:
            # We fork because of the ambiguity of the lexing
            for child in self.myScanner.children:
                # Create a copy of self
                self.logger.addLog(...)
                # Call constructor with each child scanner
                # and copy of stack and parseForest
                myCopy = OperatorPrecedenceParser(...)
```

```
        self.children.append(myCopy)
    # Since we only need forked parsers,
    # this parser is not needed anymore.
    # We set it to be inactive.
    self.state = "READY_TO_BECOME_INACTIVE"
    self.spokenLanguageData=[]
    self.inputString=''
    self.parent=[]
    self.myScanner=[]
    self.logger.addLog(...)
else:
    self.state = "READY_TO_PARSE"
elif self.state == "READY_TO_TERMINATE":
    pass

def updateParseForest(self, tokenList):
    self.logger.addLog(...)
    for token in tokenList:
        t = token.getType()
        if token.getType() == 'ID':
            # Create a tree that has one node only,
            # and added to the parse forest.
            mynode = Node('id', [token])
            mytree = ParseTree(mynode, [])
            self.parseForest.append(mytree)
        elif token.getErrorRecoveryTokenType() == 'binary':
            # Token is a binary operator.
            # Token is binary, both operands are
            # in the parseForest list. Use them as child one and two
            # first popped child is the second child as the
            # reverse polish notation is used
            child2 = self.parseForest.pop()
            child1 = self.parseForest.pop()
```

```
        mynode = Node('op', [token])
        mytree = ParseTree(mynode, [child1, child2])
        self.parseForest.append(mytree)

elif token.getErrorRecoveryTokenType() in ['prefix', 'postfix']:
    # Token is postfix or prefix,
    # an operand exist in the parseForest list.
    # Use it as the only child
    child1 = self.parseForest.pop()
    mynode = Node('op', [token])
    mytree = ParseTree(mynode, [child1])
    self.parseForest.append(mytree)

elif token.getType() == 'OPEN_BRACKET':
    theNode = self.parseForest[-1].node
    if theNode.getDetailedType() == 'close_bracket'
    and (theNode.getKey() == token.getKey()):
        # The token and the node of the top tree of the
        # parseForest are of the same speech template,
        # just update the top node of the top of parseTree
        # open bracket property to True
        theNode.setOpenBracket(token)
    else:
        # Either top tree of the parseForest node
        # is not bracket type, or if it is, then
        # its of a different speech template
        # than that of the token.
        # So create a new tree with brackets node
        mynode = Node('brackets', [token, None])
        mytree = ParseTree(mynode, [self.parseForest.pop()])
        self.parseForest.append(mytree)

elif token.getType() == 'ARGUMENT_SEPARATOR':
```

```
# add information on open bracket to self.functionNode
mynode = Node('separator', [token])
child1 = self.parseForest.pop()
child2 = self.parseForest.pop()
mytree = ParseTree(mynode, [child2, child1])
self.parseForest.append(mytree)

elif token.getType() == 'CLOSE_BRACKET':
    theNode = self.parseForest[-1].node

    if theNode.getDetailedType() == 'open_bracket'
    and (theNode.getKey() == token.getKey()):
        #case 1 - if we have same bracket type node
        # at the top with missing close bracket,
        # we merge
        theNode.setCloseBracket(token)
    else:
        # Case 2 - otherwise we always
        # create a new bracket node
        mynode = Node('brackets', [None, token])
        mytree = ParseTree(mynode, [self.parseForest.pop()])
        self.parseForest.append(mytree)
return self.parseForest

def computePrecedence(self, token1, token2, *args):
    # if both tokens are not open brackets, do as we used to
    opClass1 = token1.getOperatorClass()
    opClass2 = token2.getOperatorClass()
    if len(args) > 0 :
        # We are in reduce. No need to check bracket types
        return self.spokenLanguageData.precedenceTable[...]
    else:
        # We are checking for either reduce or shift.
```

```
# Need to check bracket types are same or not
if (opClass1 in [...]) and (opClass2 in [...]):
    # Check for function brackets by checking lexeme
    # according to which speech template this
    # bracket belongs to, action as required :)
    if token1.getKey() == token2.getKey():
        return self.spokenLanguageData.precedenceTable[...]
    else:
        #Now we know that we have two different bracket types
        if opClass1=='OPEN_BRACKET' and
        opClass2=='OPEN_BRACKET':
            return '<'
        if opClass1=='OPEN_BRACKET' and
        opClass2=='ARGUMENT_SEPARATOR':
            #return 'S1'    NOW G1
            return '<>'
        if opClass1=='OPEN_BRACKET'
        and opClass2=='CLOSE_BRACKET':
            #return 'S2'    NOW G2
            return '<>'
        if opClass1=='ARGUMENT_SEPARATOR'
        and opClass2=='OPEN_BRACKET':
            return '<'
        if opClass1=='ARGUMENT_SEPARATOR'
        and opClass2=='ARGUMENT_SEPARATOR':
            #return 'S3'
            return '>'
        if opClass1=='ARGUMENT_SEPARATOR'
        and opClass2=='CLOSE_BRACKET':
            #return 'S4'
            return '<>'
        if opClass1=='CLOSE_BRACKET'
        and opClass2=='OPEN_BRACKET':
```

```
        print 'stopping code'
        sys.exit()
    if opClass1=='CLOSE_BRACKET'
    and opClass2=='ARGUMENT_SEPARATOR':
        return '>'
    if opClass1=='CLOSE_BRACKET'
    and opClass2=='CLOSE_BRACKET':
        return '>'
else:
    # both are not speech templates
    if token1.getType() == 'OPERATOR'
    and token2.getType() == 'OPERATOR':
        return self.spokenLanguageData.precedenceTable[...]
    else:
        classOrType1 = opClass1
        classOrType2 = opClass2
        if token1.getType() == 'OPERATOR':
            classOrType1 = 'OPERATOR'
        if token2.getType() == 'OPERATOR':
            classOrType2 = 'OPERATOR'
        return self.spokenLanguageData.precedenceTable[...]
```

## Appendix D

# TalkMaths Interface Screenshots

TalkMaths is a web-based software which is free to use. However, it requires the user to register before being able to access the editor. The following figure shows the login page and Figure D.2 is the new user registration page. The screenshots are taken from the latest interface, that we have integrated into the web application which was created by us from scratch.

**talk {maths}** *creating maths by using plain English*

**Welcome to TalkMaths**

Do you want to create mathematical expressions by talking or typing plain English into your computer? Tired of pointing and clicking? Then check out our web site.

*How does it work? If you have speech recognition installed on your computer, you can download an application from our website. Our website provides a mathematics editor that you can use by speaking commands for inputting and modifying your mathematical content, once you have installed this application.*

Is it free? Yes, it is.

**Login**

*First time user?*  
**Create an account**

**Email address:**

**Password:**  
 **Login**

**Remember me?**    **Forgot password?**

Figure D.1: TalkMaths Login Page



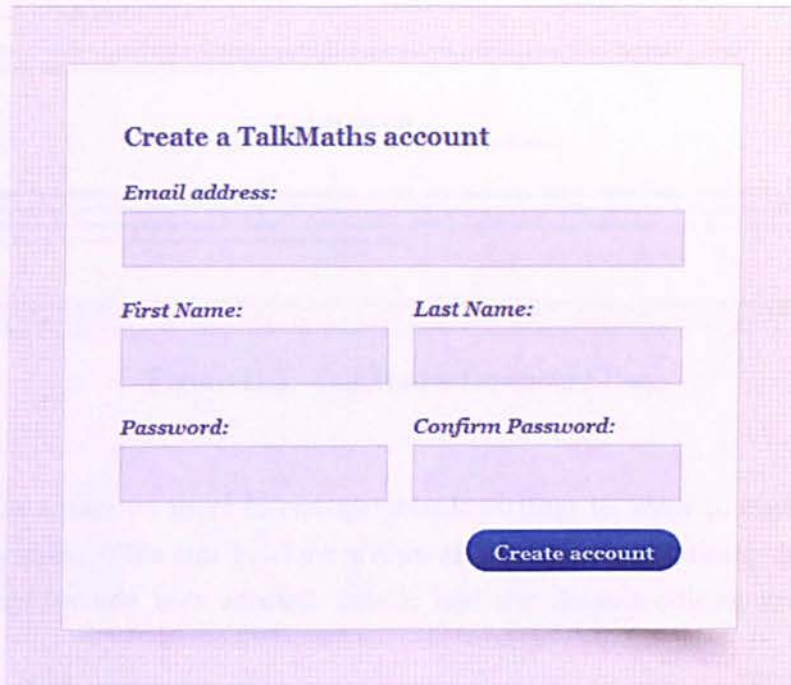
A screenshot of the TalkMaths registration page. The page has a white background with a light blue border. At the top, it says "Create a TalkMaths account". Below this, there are four input fields: "Email address:" (a single wide field), "First Name:" (a narrow field), "Last Name:" (a narrow field), "Password:" (a narrow field), and "Confirm Password:" (a narrow field). At the bottom right, there is a blue button with white text that says "Create account".

Figure D.2: TalkMaths Registration Page

Once registered, should the user wish to use the system with an ASR system, he/she can install the speech front-end which works with Microsoft Windows Speech ASR. Figure D.3 illustrates the download page of the TalkMaths system which gives the instructions for this and a download link to the front-end.



Figure D.3: TalkMaths Download Page

TalkMaths allows its users to change default settings in order to customise the usage of the system. This can be done within the Settings page shown in Figure D.4. These settings include user account details and the default editing grid type. The

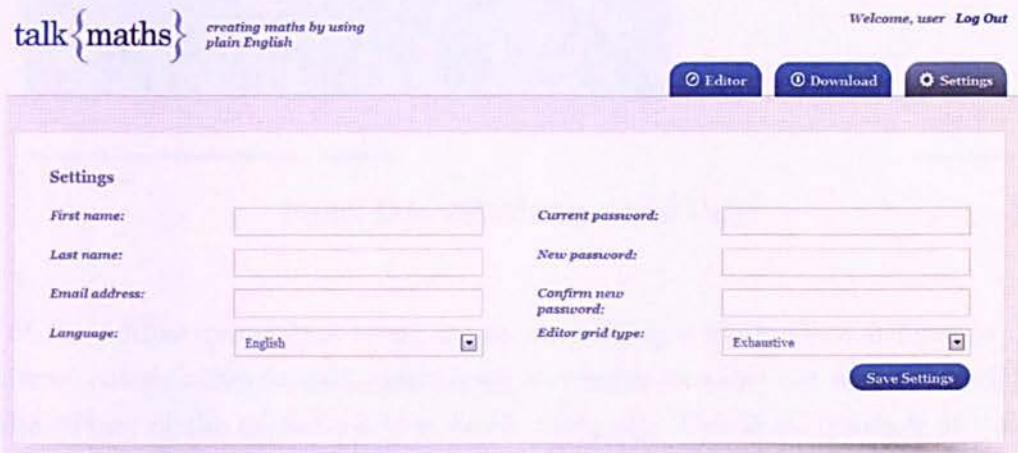


Figure D.4: TalkMaths Settings Page

TalkMaths system editor page (shown in Figure D.5), has an area to render mathematics expressions, two text boxes to dictate or type spoken mathematics and help



queries (As described in Section 7.2.4). The section titled “Editing Commands” gives (or activated by speech) links to frequently used commands, which when invoked, automatically trigger corresponding commands on the system. The Figure D.6 illustrates

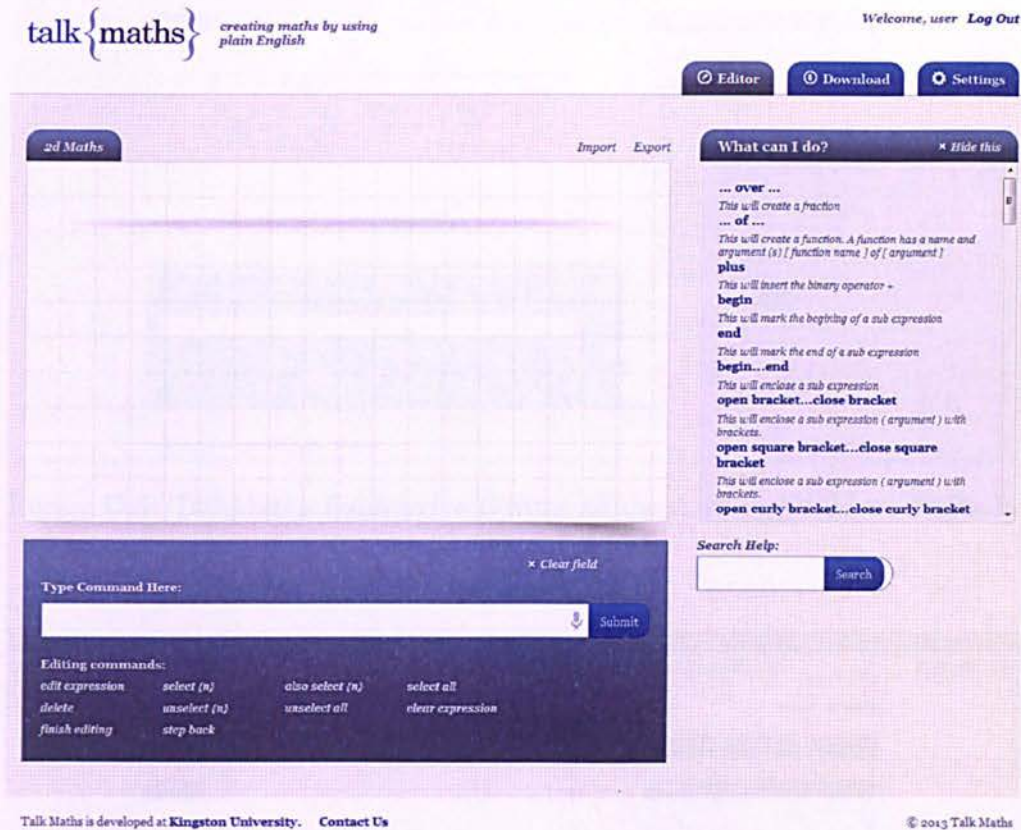


Figure D.5: TalkMaths Editor Page

one of the editing paradigms being in use on editing a mathematical formula. The numbered colour boxes around mathematical constructs allow the user to select parts (or the whole) of the expression that needs changing. This is an example of the “exhaustive” editing grid. Semantic editing is similar to the exhaustive editing. It still provides numbered colour boxes around mathematical constructs, however the user can decide which sub construct(s) within the expression on display should have them. For example, as seen in Figure D.7, the numbered boxes only invoked on symbols. This is less complicate in the eye of the user than the exhaustive editing. Similar to the



Figure D.6: TalkMaths Exhaustive Editing of the Arithmetic Series Formulae



Figure D.7: TalkMaths Semantic Editing – Symbols

semantic editing of symbols, Figures D.8, D.9, D.10, D.11, D.12, D.13, D.14 show examples of other types of semantic editing method such as editing operators, numbers, roots, fractions, functions, numerators and denominators respectively.



Figure D.8: TalkMaths Semantic Editing – Operators



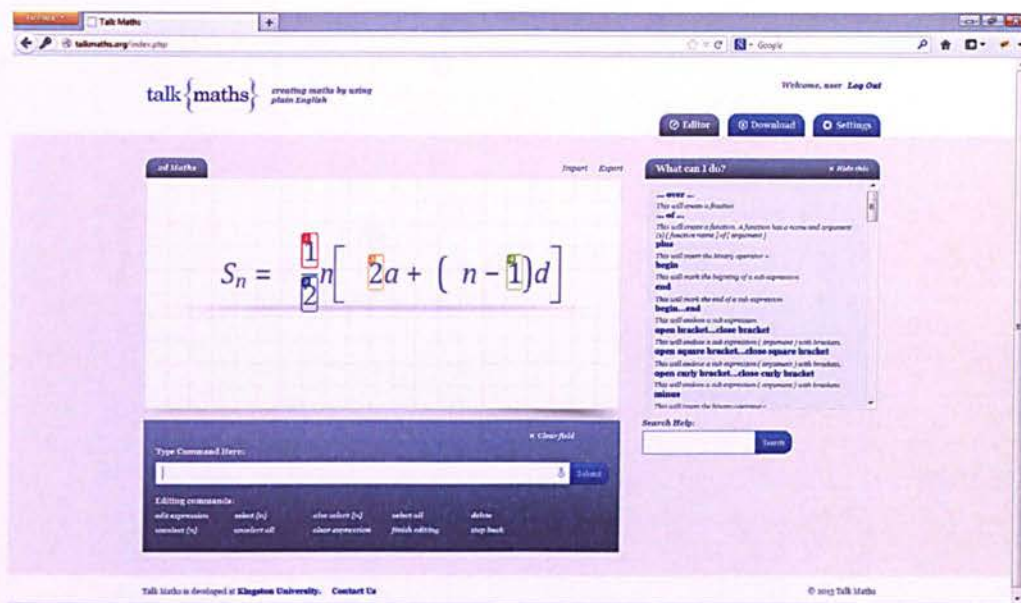


Figure D.9: TalkMaths Semantic Editing – Numbers

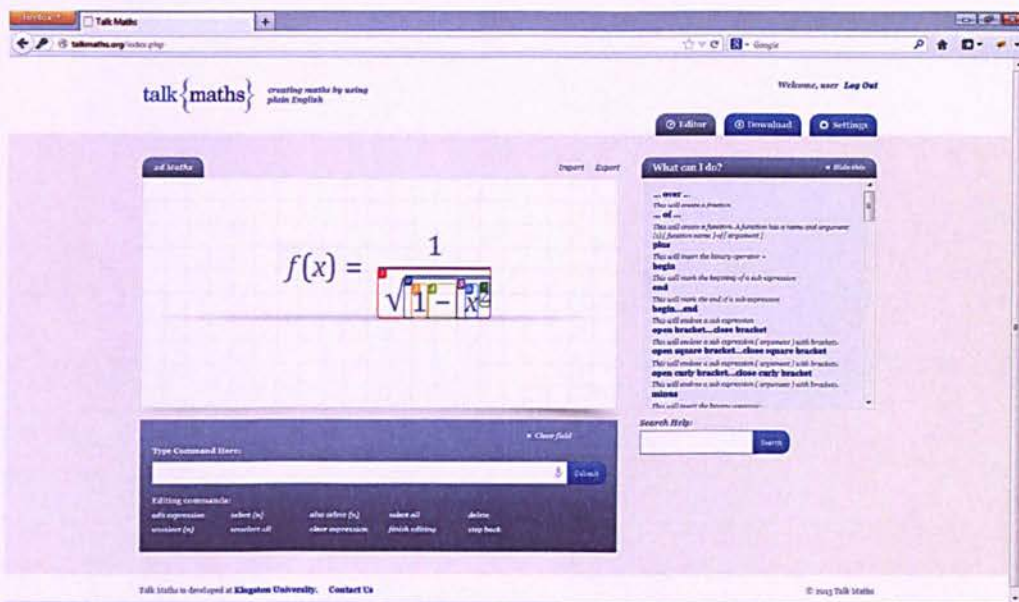


Figure D.10: TalkMaths Semantic Editing – Roots

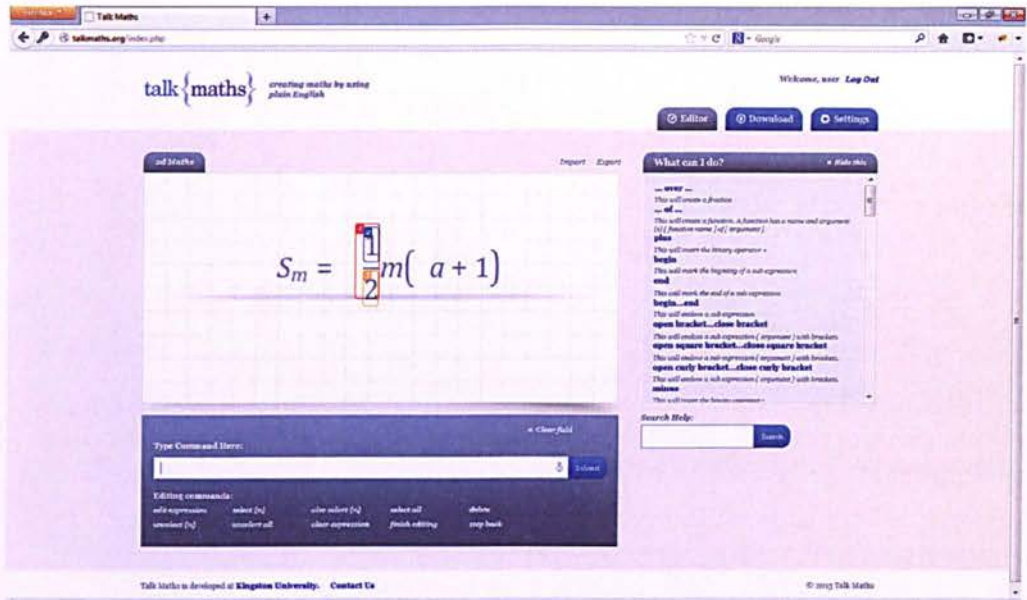


Figure D.11: TalkMaths Semantic Editing – Fractions



Figure D.12: TalkMaths Semantic Editing – Functions



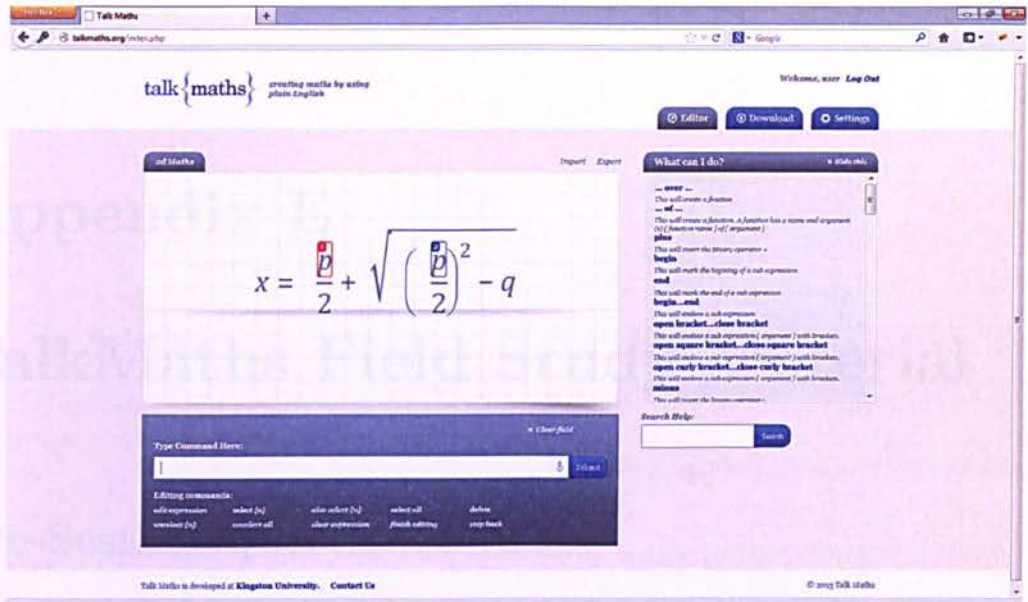


Figure D.13: TalkMaths Semantic Editing – Numerators

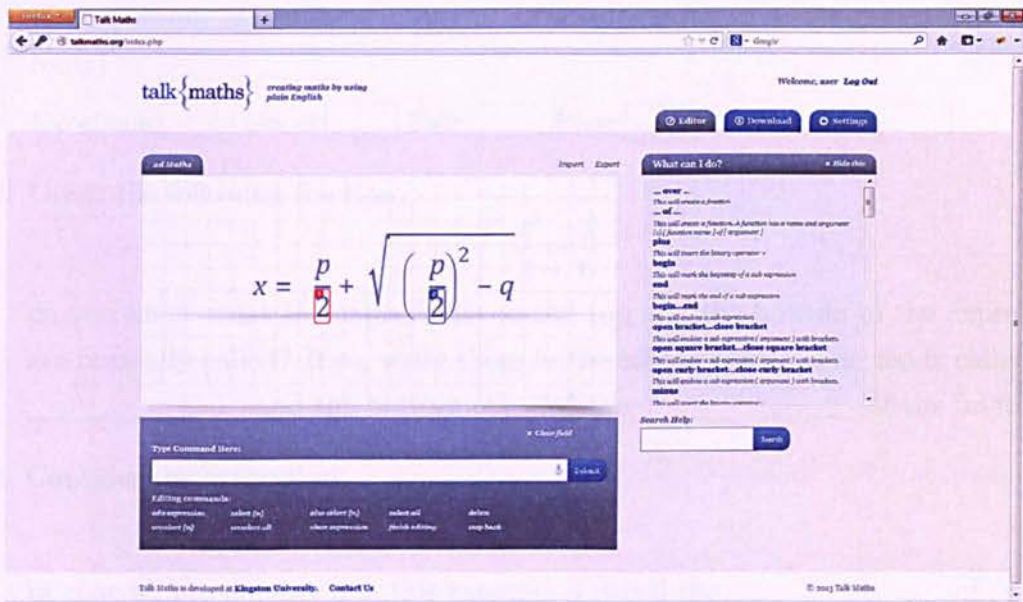


Figure D.14: TalkMaths Semantic Editing – Denominators

# Appendix E

## TalkMaths Field Study Material

### Pre-Session Questionnaire

This is the pre-session questionnaire for both groups of participants (the correct answers for Questions 2 to 6 are given in Section E below).

1. How competent would you regard yourself in terms of your basic maths skills? (Doing simple algebraic manipulations, involving fractions, functions and square roots)

Excellent  Good  Fair  Poor  Don't Know

2. Given the following fraction,

$$\frac{x^2 - 1}{x + 1}$$

do you know what the expressions on the top and the bottom of the expression are normally called? If so, write them in the blanks below: The top is called the \_\_\_\_\_ and the bottom is called the \_\_\_\_\_ of the fraction.

3. Consider the expression,

$$(a + b)^n$$

In general, the number n in this example is called the \_\_\_\_\_ of  $(a + b)$ .

4. The mathematical symbols  $+$ ,  $-$ ,  $\times$ ,  $\div$  represent the processes of addition, subtraction, multiplication and division respectively. We call them binary \_\_\_\_\_.

(Add a single word to complete the blank space above)

5. Given the following expression,

$$z + f(x^2 + y^2)$$

do you know how the expression inside the bracket associated with the function  $f$  is normally referred to? If so, write it in the space: The expression inside the bracket is called the \_\_\_\_\_ of the function of  $f$ .

6. Given the following square root,

$$y + \sqrt{x(1+x)^b}$$

do you know how the expression inside the square root symbol is normally referred to? If so, write it in the blank space below: The expression inside the square root is called the \_\_\_\_\_ of the root.

## Mathematical Tasks for Microsoft Equation Editor Group

Mathematical tasks given for participants who were allocated with Microsoft Equation Editor:

### *Task 1: Question on fractions*

- i For this exercise, you are supposed to create the following mathematical expression, using Microsoft Equation Editor:

$$a + b \frac{2+c}{x-y}$$

- ii Now, replace the expression  $x - y$  in (1) by  $w - z$ .

**Task 2: Question on functions**

- i Still using the Microsoft Equation Editor, create the following expression:

$$a + f(2x - 5)$$

- ii Now, replace the expression  $2x$  in (2) by  $y$ .

**Task 3: Question on square roots**

- i Still using the Microsoft Equation Editor, create the following expression:

$$a + \sqrt{x + 2y}$$

- ii Now, replace the expression  $2y$  in (3) by  $3z$ .

**Mathematical Tasks for TalkMaths Group**

Mathematical tasks given for participants who were allocated with TalkMaths:

**Task 1: Question on fractions**

- i Using the TalkMaths editor, create the following expression:

$$a + b \frac{2 + c}{x - y}$$

In order to do this, in the input field, type “a + b begin 2 + c end over begin x - y” and press enter.

- ii Now, replace the expression  $x - y$  in (1) by  $w - z$ . Hint: consult the help facility on fractions and selections. This will help you to edit the expression, select appropriate parts of it and replace the desired symbols.

**Task 2: Question on functions**

- i Still using the TalkMaths editor, clear the previous expression and create the following expression:

$$a + f(2x - 5)$$

In order to do this, in the input field, type “a + f of begin 2 x -5 end” and press enter.

- ii Now, replace the expression  $2x$  in (2) by  $y$ . Hint: consult the help facility on functions and selections. As in the previous example, this will help you to edit the expression, select appropriate parts of it and replace the desired symbols.

**Task 3: Question on square roots**

- i Still using the TalkMaths editor, create the following expression:

$$a + \sqrt{x + 2y}$$

In order to do this, in the input field, type “a + square root of begin x + 2 y end” and press enter.

- ii Now, replace the expression  $2y$  in (3) by  $3z$ . Hint: consult the help facility on roots and selections. As in the previous example, this will help you to edit the expression, select appropriate parts of it and replace the desired symbols.

**Post-Session Questionnaire**

This is the post-session questionnaire for both groups of participants (the correct answers for questions 3 to 7 are given in Section E below).

1. How competent would you regard yourself in terms of your basic maths skills?  
(Doing simple algebraic manipulations, involving fractions, functions and square roots)

Excellent  Good  Fair  Poor  Don't Know

2. Which of the computer-based tasks(s) from this session did you manage to complete? Please circle.

Task 1(i)    Task 1(ii)    Task 2(i)    Task 2(ii)    Task 3(i)    Task 3(ii)

3. Given the following fraction,

$$\frac{x^2 - 1}{x + 1}$$

do you know what the expressions on the top and the bottom of the expression are normally called? If so, write them in the blanks below:

The top is called the \_\_\_\_\_ and the bottom is called the \_\_\_\_\_ of the fraction.

4. Consider the expression,

$$(a + b)^n$$

In general, the number  $n$  in this example is called the \_\_\_\_\_ of  $(a + b)$ .

5. The mathematical symbols  $+$ ,  $-$ ,  $\times$ ,  $\div$  represent the processes of addition, subtraction, multiplication and division respectively. We call them binary \_\_\_\_\_.  
(Add a single word to complete the blank space above)
6. Given the following expression,

$$z + f(x^2 + y^2)$$

do you know how the expression inside the bracket associated with the function  $f$  is normally referred to? If so, write it in the space below: The expression inside the bracket is called the \_\_\_\_\_ of the function of  $f$ .

7. Given the following square root,

$$y + \sqrt{x(1+x)^b}$$

do you know how the expression inside the square root symbol is normally referred to? If so, write it in the blank space below:

The expression inside the square root is called the \_\_\_\_\_ of the root.

8. How easy did you find the Equation Editor system to use? Please tick the appropriate answer.

Very Easy  Fairly Easy  O.K.  A Bit Difficult  Very Difficult

9. Did you feel that using the TalkMaths/ Microsoft Equation Editor for doing this exercise improved your understanding of the relevant mathematical concepts? Please tick the appropriate answer.

No, I feel more confused now  Not Really  A Little  Quite a Bit  A Lot

10. Have you any other comments? If so, please write them below.

## Correct Answers

The prescribed correct answers for Questions 2 to 6 and 3 to 7 in pre- and post-questionnaires respectively (the numbers not in parentheses refer to the pre-questionnaire and those in parentheses to the post-questionnaire) are as follows.

- 2(3). The top is called the numerator and the bottom is called the denominator of the fraction.
- 3(4). In general, the number  $n$  in this example is called the power of  $(a + b)$ .
- 4(5). We call them binary operators.
- 5(6). The expression inside the bracket is called the argument of the function of  $f$ .
- 6(7). The expression inside the square root is called the radicand of the root.