Techniques for creating ground-truthed sketch corpora

by

Scott MacLean

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The problem of recognizing handwritten mathematics notation has been studied for over forty years with little practical success. The poor performance of math recognition systems is due, at least in part, to a lack of realistic data for use in training recognition systems and evaluating their accuracy. In fields for which such data is available, such as face and voice recognition, the data, along with objectively-evaluated recognition contests, has contributed to the rapid advancement of the state of the art.

This thesis proposes a method for constructing data corpora not only for handwritten math recognition, but for sketch recognition in general. The method consists of automatically generating template expressions, transcribing these expressions by hand, and automatically labelling them with ground-truth. This approach is motivated by practical considerations and is shown to be more extensible and objective than other potential methods.

We introduce a grammar-based approach for the template generation task. In this approach, random derivations in a context-free grammar are controlled so as to generate math expressions for transcription. The generation process may be controlled in terms of expression size and distribution over mathematical semantics.

Finally, we present a novel ground-truthing method based on matching terminal symbols in grammar derivations to recognized symbols. The matching is produced by a best-first search through symbol recognition results. Experiments show that this method is highly accurate but rejects many of its inputs.

Acknowledgements

Thanks to my supervisor, George Labahn, for giving me the opportunity to develop and evaluate my own ideas for many of the problems we have investigated, and for giving me guidance and support along the way. Thanks also to Ed Lank who has provided invaluable suggestions and advice regarding many aspects of my work and my attempts to communicate it clearly. As well, I wish to thank Michael Terry for making time to read my thesis and see my presentation.

This thesis would not have been possible without help from my MathBrush colleagues. Thanks to Mirette for sympathizing with my bug-ridden code and at least pretending to believe that my software would eventually work, and also to Dave not only for helping to run the collection study, but also for always being available to discuss ideas, give feedback, and head down to the Grad Club.

Thanks to my parents and Sarah, who have always supported me no matter how many years I spend in school. Knowing they will be behind any decision I make has been important to me these last few years. Finally, thanks to my good friends Nate, Kim, Karl, Colin - all the time we waste is time well spent.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Two present technology trends are the constant growth in power of modern computers and the increasing ubiquity of pen-based devices. The former trend has facilitated sophisticated software for modeling and solving scientific problems, while the latter has enabled users to easily sketch diagrams and equations on computers. It is natural to desire a bridge between these two developments: that is, a facility for interpreting users' sketches and using them to formulate input to software programs.

This bridge is a primary goal of the sketch recognition community. Over the past several decades, several recognition techniques have been invented and investigated, and systems have been developed for a variety of sketching domains, including flow charts [1], electrical schematics [14], molecular structures [25], UML diagrams [24], musical scores [5], etc.

This thesis focuses on mathematical expression recognition, the problem of deciding what math expression is represented by a sequence of digital ink strokes. There are many challenges in this recognition task, many of which are addressed by a recognizer under development as part of the MathBrush project. However, some challenges may be more easily addressed through the use of a ground-truthed corpus of handwritten math expressions. Other recognition domains, such as voice recognition, commonly make use of corpora for training and evaluating recognizer technologies, but they have not been used on a large scale in math recognition research. This thesis will describe the creation of a math expression corpus and some new techniques developed for its construction. As well, it explores some of the larger issues in corpus creation and corpus use, particularly as applied to math recognition.

1.1 Math recognition

Math recognition research has a history dating back at least forty years (e.g. [3, 30, 31]). In that time, many math recognition systems have been constructed

(see Chapter 2), but they have generally enjoyed little practical success. There is currently no recognizer available that is capable of interpreting a wide range of written mathematics. When contrasted with the remarkable success of conceptually similar technologies such as voice recognition, optical character recognition (OCR), and facial recognition, this lack of quality recognizers is all the more striking.

It is true that math recognition, and sketch recognition in general, poses significant technical challenges. Some challenges are shared with all recognition tasks involving symbols: for example, each user has a unique writing style, so symbols may be drawn in many different ways. Some symbols may be virtually indistinguishable (e.g. l,1,| or o,O,0). However, English text recognition is greatly aided by the fact that text is written from left-to-right, top-to-bottom. In this case, symbols can be linearized easily and there is a clear traversal order through the digital ink. Sketches, and math expressions in particular, tend to have much more complicated two-dimensional layouts. Linearization in these domains is extremely difficult, as the order in which symbols should be traversed is not obvious, and there are exponentially many possibilities. These difficulties are exacerbated by the uncertainty inherent in symbol recognition – different symbols may require different linearization strategies, leading to ambiguity and conflicting strategies.

Ambiguity is an issue common to all handwriting recognition tasks, but the difficulties it presents are exacerbated in math recognition. In text recognition, one can assume that the user has written words of some kind, and dictionary-based heuristics can be combined with grammar models incorporating verb conjugations, subject-verb agreement, etc. to narrow the range of feasible interpretations considerably. In math recognition, these techniques are applicable up to a point – one might consider the surrounding context of a symbol in order to restrict the number of feasible recognition results. However, the large-scale structures of written math are not as well-defined as for English grammar, so the available context is necessarily smaller. Math writing is also a concise notation lacking the redundancy that simplifies text recognition.

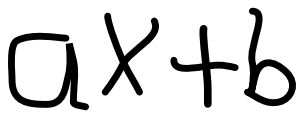

Figure 1.1: A short yet ambiguous handwritten math expression

For example, the math expression appearing in Figure 1.1 may be interpreted in at least six distinct ways: $ax + b, aX + b, a^x + b, axtb, aXtb, a^xtb$. It could be argued that knowing whether the "x" symbol was upper- or lower-case might help a recognizer to resolve the ambiguity between multiplication and exponentiation, but any of these six interpretations could conceivable fit equally well into a larger expression. Nothing is gained by expanding the notion of expression context beyond the size of local subexpressions, except perhaps to say that if the "t" symbol occurs elsewhere in the expression, the interpretations including "t" may become

more likely. On the other hand, the writer may still have intended to write an addition expression. Such assumptions about symbol reoccurence can be risky from a recognition point of view.

One encounters ambiguity in mathematical writing even in the absence of uncertain symbol recognition. For example, the symbol identities in the expression $u(x + y)$ are clear, but the intended interpretation is not: is u a function being applied to $x + y$, or is the expression a multiplication? Without knowing the intention of the writer, or having external information available (e.g. the user previously wrote $u : \mathbb{R} \to \mathbb{R}$, effectively declaring u's "type"), it is impossible to disambiguate between these interpretations because the syntax used to express them is identical.

This type of ambiguity is uncommon in other recognition domains. In text recognition, one finds the same syntax expressing different ideas – for example, "horse" the animal versus "horse" the gymnast's apparatus – but the goal of text recognition is typically to simply input the syntax of the language as a replacement for typing, not to communicate to the computer some notion of animals or sports. However, in math recognition, it is exactly this kind of communication of higher-level concepts that is often desired: it may be *convenient* to simply input math syntax using a pen rather than a keyboard, but it is more *useful* to have the computer interpret these inputs mathematically, so that it can actually perform the indicated mathematical operations for users. Indeed, this level of recognition is exactly what is required for the MathBrush project, described in Section 1.2.

Math recognition is a difficult problem, and difficulties arise at two distinct levels. Syntactic challenges include symbol classification and recognition of complicated two-dimensional expression structures. These difficulties have been well-studied and addressed by many different approaches to math recognition (see Chapter 2). Semantic challenges involve the resolution of ambiguities arising not only from the uncertainty inherent in the process of recognizing symbols and expression structures, but also from the overloading of math syntax. These difficulties are, at present, perhaps best addressed by incorporating intuitive and flexible user interfaces into recognition systems so that users can easily select the correct mathematical interpretation of their writing.

1.2 MathBrush

MathBrush is an experimental pen-based system for interactive mathematics. The system ([21]) allows users to write mathematical expressions as they would using a pen and paper, and to edit and manipulate the mathematical expressions using computer algebra system (CAS) functionality invoked by pen-based interaction. In order to formulate correct input for a CAS, MathBrush must resolve any ambiguities in symbol identity and semantic interpretation, as outlined in Section 1.1.

A major component of MathBrush is its math recognition engine. While it cannot currently disambiguate between mathematical interpretations having identical

syntax (e.g. $u(x+y)$), it does use local contextual information to guide and restrict possible symbol identity and subexpression semantics. To do so, the recognizer models well-formed math expressions using a formal grammar, and models the uncertainty arising during recognition by representing symbols and subexpressions as fuzzy sets. Each set member is a different interpretation of the same writing. Larger expressions are built up from smaller expressions by means of spatial relations satisfied by the expressions' constituent ink strokes.

This paradigm is intended to mimic one view of how humans interpret mathematics: by "chunking" a large expression into subexpressions, and assembling the whole expression by observing how those subexpressions are related to one another. Ambiguities in subexpression structures or symbol identity do not necessarily obscure the general structure of the whole expression. So it is with our math recognizer – large-scale expression structures can still be identified even if particular subexpressions cannot be accurately recognized. We currently rely on the user to select the correct interpretation in such cases of ambiguity.

During the creation of this math recognizer, we have found several potential uses for a corpus of math expressions, such as:

1. Training/evaluating the symbol recognizer component,
2. Training/evaluating the relation classifiers,
3. Comparing different techniques for symbol and relation classification,
4. Evaluating overall recognition accuracy,
5. Regression testing to ensure consistent recognition accuracy,
6. Comparing our recognizer to other current systems.

However, to the best of our knowledge, no publicly-available corpus of hand-drawn math expressions exists. We have therefore undertaken to create a large ground-truthed corpus of hand-drawn mathematical expressions to use for training and evaluating the math recognition component of MathBrush. In order to be of most use in improving our recognizer, our corpus possesses the following properties:

1. Broad coverage of symbol shapes (e.g. $1, R, \theta$ versus c, a, e versus ρ, q, y, etc.) and combinations of shapes.
2. Broad coverage of combinations of mathematical semantics.
3. Broad coverage of combinations of spatial relations.
4. Wide range of expression size and complexity.

Clearly, our corpus tends more toward a broader coverage of notation rather than a more accurate reflection of the notation's use in the real world. This tendency is due to our rule-based approach to recognition. It has some advantages in terms of training recognizers similar to MathBrush, but also limits the potential usefulness of our corpus for other recognition paradigms. These issues are addressed in Section 2.3.

Our corpus may find use outside of the MathBrush project as training and testing data for other math recognition systems. It may also be used as a common basis of comparison in order to evaluate the relative strengths and weaknesses of various approaches to math recognition. To date, researchers have generally reported recognition accuracy results using private data sets and their own particular evaluation criteria. Under these conditions, it is challenging to assess the benefits of a given recognition algorithm. Any common data set facilitating comparative evaluation of recognizers is useful.

1.3 Corpora for training and evaluation

Section 1.1 outlined some of the important challenges facing math recognition researchers. However, other recognition domains have their own unique challenges, yet they have been solved to the extent that their technology has been deployed on a large scale. Face, voice, and document recognition software programs are commonly used by governments and other large organizations. What lessons can be learned from their success?

It is interesting to note that each of these fields has enjoyed the availablility of large, high-quality corpora of data accurately labelled with ground-truth information ("ground-truthed"). For example, speech recognition corpora are available from the Linguistic Data Consortium (http://www.lds.upenn.edu), the National Institute for Standards and Technology (NIST) (http://www.itl.nist.gov/iad/mig), the European Language Resources Association (http://www.elra.info), and others. NIST operates yearly standardized evaluations in a variety of application domains (e.g. language identification, speaker identification, content extraction, etc.) As an authoritative and objective third party, these evaluations promote friendly competition between research groups.

The Facial Recognition Technology (FERET) Database comprises over 14000 images for training and evaluation of facial recognizers. It was collected by NIST, which also manages standardized evaluations, as with its speech recognition programme. The evaluation methodology for the 1996 evaluation consisted of a standardized protocol in which the tester specifies a training set and a testing set, the testee generates a matrix of values representing the similarity between each training and testing image, and the tester computes evaluation scores from this matrix based on a number of criteria reflecting various use cases [33]. (For example, "is the top match correct?", "is the correct answer in the top n matches?", etc.) Other, smaller corpora are also available (e.g. [16], [34]).

NIST also offers OCR databases (http://nist.gov/srd/optical.htm). At the University of Washington, an initiative led by Phillips, Chanda, and Haralick produced three ground-truthed corpora for recognition of technical documents. The first included a large number of English articles, the second included English and Japanese articles, and the third comprised technical drawings and mathematical equations. Unlike speech and facial recognition, there is no standardized, external evaluation process for document analysis systems.

These examples demonstrate that researchers in other fields have found it useful to:

1. Create large, ground-truthed corpora to use for training and testing recognition technology,
2. Standardize the problem statement for recognition, and
3. Hold competitions to evaluate the absolute and relative performance of a variety of approaches to the recognition problem.

These steps have not been taken in the math recognition community, nor in the sketch recognition community at large. While math recognition systems have been designed and evaluated largely on an ad-hoc basis, available corpora in other areas have given researchers in other fields the ability to:

1. Train sophisticated machine-learning techniques and parametrized models on real, representative data, rather than choosing parameters experimentally or using small or machine-generated data sets; and
2. Objectively measure recognition accuracy against well-established benchmarks, facilitating comparison against the state of the art and easy identification of areas of weakness.

These two steps may be performed only through the use of large, ground-truthed corpora. Furthermore, they are crucial to the development of highly-accurate recognition systems. Step 1 enables system adaptation; without it, a recognition system is likely to perform poorly for many users, even if it works well for others. Step 2 fosters continual improvement in the state of the art and encourages cooperation amongst researchers. Without it, researchers are more inclined to create a new recognizer from the ground up rather than to build on proven technology and address its shortcomings. Large datasets and benchmarks for evaluating and comparing recognizers have helped many recognition domains to achieve real-world success; there is no reason to suppose they would not be helpful to math recognition.

Creating these datasets and benchmarks requires research into three key questions:

1. How can correct and incorrect recognition can be measured independently of the technical details of a specific approach?

2. What would a corpus useful as a common basis of comparison need to consist of?

3. How can the recognition problem be phrased to ensure appropriate experimental conditions and to enable comparison between approaches?

The present work does not address these questions in their full generality. Indeed, it is beyond the scope of a single recognition project to do so. The corpus we have created is naturally biased toward the requirements of our own recognizer. However, other researchers may still find our corpus useful. The corpus can be used to train recognizers using a similar recognition paradigm (i.e. based on expression geometry) and as a testing suite for any recognizer. It can act as a publicly-available common test set, facilitating comparison between recognizers.

1.4 Contributions

This thesis provides some theoretical tools and practical observations to aid in the creation of large, ground-truthed corpora that are generated by transparent mechanisms.

One main contribution is a theoretical technique for automatically generating template expressions, applicable to any domain that can be expressed by a general class of grammars.

Another main contribution is a publicly-available corpus of hand-drawn math expressions, fully annotated with ground-truth. This corpus was created using the techniques described in this thesis. It is available at the URL:

http://www.cs.uwaterloo.ca/scg/mathbrush/mathdata

Secondary contributions include:

- A general methodology for creating sketch corpora based on theoretical and practical considerations, and
- A simple technique for ground-truthing mathematical expressions.

1.5 Thesis organization

The remainder of this thesis is organized as follows:

The next chapter reviews previous work in math recognition and evaluation metrics used by different authors. It also surveys related work in automatic ground-truthing from other fields.

Chapter 3 proposes a general process for corpus creation and argues for its legitimacy by comparison with alternative methods. It goes on to illustrate how this process was used in our creation of a large corpus of mathematical expressions.

Chapter 4 details our approach to automatically generating template expressions. It introduces a general grammar formalism and shows how unbiased expressions can be obtained by deriving random parse trees.

Chapter 5 gives a simple but accurate algorithm for ground-truthing mathematical expressions. The algorithm is illustrated with examples, and its accuracy is measured and discussed.

Finally, Chapter 6 explores some limitations to the techniques presented and identifies areas requiring further research. The thesis concludes with a summary of our contributions and observations.

Chapter 2

Related work

Mathematical expression recognition has a rich history of active research. Under various guises (visual languages, sketch recognition, system-construction projects, etc.), the problem has been tackled from many angles by many researchers. That so many have tried so much with relatively little practical success until recent years (likely more due to faster hardware than to any other factor) speaks to the challenge of formulating a computationally-feasible solution giving adequate recognition accuracy. In this chapter, we survey some approaches taken by existing math recognition systems, review their evaluation methodologies, and describe related work in ground-truthing corpora from other fields.

2.1 Math recognition systems

More exhaustive surveys of the many approaches to math recognition can be found in [9] and [4]. Here, we focus on recent research, particularly those publications in which metrics for evaluating recognition accuracy were introduced.

2.1.1 Chan and Yeung

Chan and Yeung proposed a rule-based approach to mathematics recognition based on Definite Clause Grammars [7]. Their approach combines rewriting techniques with heuristics such as operator dominance and hard decisions about subexpression boundaries.

Chan and Yeung used 60 expressions manually transcribed from a table of common formulae to evaluate their system. In [8], they analyze the many classes of errors that may arise in their parsing scheme, and introduce heuristics for repairing some common recognition errors they encountered. They go on to propose an evaluation metric integrating symbol recognition with structural recognition, namely the ratio of the sum of correctly recognized symbols and mathematical operators

to the total number of symbols and operators. This metric assigns equal weight to symbol recognition and structural recognition, and assumes that every error is equally important.

2.1.2 MathPad

The MathPad project at Brown University ([28, 43]) is one of the products of a long-running research programme for creating math sketching systems. Its high-level goals differ from those of MathBrush in that the Brown project aims to facilitate learning and problem-solving through interactive sketches controlled by mathematical formulae, while MathBrush intends to be a research tool for mathematicians and students. However, many software features required to attain each of these goals are similar, and so MathPad includes a math recognition system.

MathPad's recognition system may be viewed as an informal application of grammar-based parsing. Rules governing how symbols are arranged into math expressions are explicitly coded into the program. Input is processed in a way analagous to how a particular organization of grammar productions could be naively parsed. The recognition system includes sophisticated heuristics specialized to particular mathematical structures (eg. fractions, integrals, etc.). While this approach is highly tunable, it is less flexible than less explicitly-specified approaches.

A study of the recognition accuracy of MathPad was presented by LaViola [26]. In the study, 11 subjects individually provided the system's symbol recognizer with 20 samples of their handwriting for each supported symbol. Each subject then provided 12 additional samples of each symbol as well as drawings of 36 particular math expressions as test data. The expressions were partly taken from Chan and Yeung's collection [8] and partly designed by the author. This data was used to test MathPad's symbol recognizer and math expression parser, respectively.

LaViola used a different accuracy metric for each of these two systems. To measure symbol recognition accuracy, the number of correctly-identified symbols was divided by the total number of symbols. To measure parser accuracy, the number of correct parsing decisions was divided by the total number of parsing decisions. A parsing decision arises in LaViola's system whenever symbols must be grouped together (or not) into a subexpression (e.g. *tan* as multiplication versus the function name tan), or a choice must be made about the type of a subexpression (e.g. superscript vs. inline).

2.1.3 Garain and Chaudhuri

In [17], Garain and Chaudhuri describe an online math recognizer along with an evaluation scheme. Their recognizer combines chain-code and Hidden Markov Model symbol recognizers with a two-stage exprssion parser. The first stage uses symbol positions to extract baseline information and identify certain non-nested

subexpressions. In the second stage, these subexpressions are segmented using projection profiles. LaTeX markup is generated for the expression as the segments are merged. A grammar is used to ensure that the generated LaTeX, and hence the recognized expression, is syntactically valid.

To evaluate their system, Garain and Chaudhuri collected a corpus of 5500 expressions in which each unique expression is duplicated 20 or 40 times (written twice by either 10 or 20 writers). They selected expressions for transcription from a number of pre-University textbooks and one Ph.D. thesis. The selections were transcribed by students and math researchers using an external writing tablet. Of the transcriptions, roughly 80% were used for training their recognizer, with the rest used for testing. Unfortunately, this corpus does not seem to be generally available.

Garain and Chaudhuri argue that recognition errors in highly-nested subexpressions are not as critical as those on the main expression baseline. They propose an integrated evaluation metric in which structural recognition accuracy is weighted based on subexpression nesting level. Under their metric, a subexpression at level i has weight $1/(i+1)$, so errors in more deeply-nested subexpressions are considered "less incorrect" than errors closer to the main expression baseline. To determine recognition accuracy on an expression, the sum of the number of symbol recognition errors and (weighted) structural errors is divided by the total number of symbols and (weighted) structural components at each level. The average value of these ratios over all nesting levels in the expression is called the "performance index" of the recognizer on the expression.

Using this metric, Garain and Chaudhuri present impressive recognition results, but the correct interpretation of their results is not obvious. The definition of "level" used by the authors is quite specific to the authors' own recognition technique. Furthermore, as the number of levels in sample expressions increases (from 1 to 11), so too increases the gap between percentage of samples correctly recognized and average performance index (from roughly 10% to 70%).

2.1.4 Freehand Formula Entry System

The Freehand Formula Entry System (FFES) was developed over several years by Zanibbi, Smithies, and others [38, 41, 42]. In multiple parsing passes, FFES identifies baselines in math input and uses operator dominance and tree transformation rules to convert linearized baselines into trees representing spatial layout, math syntax, and finally math semantics.

Zanibbi et al evaluated the recognition component of FFES, DRACULAE, against typeset data using the UW-III database of technical documents [42]. They measured two types of errors: incorrect baselines, meaning any baseline containing incorrect tokens, nested relative to an incorrect token, or at the wrong level of nesting; and misplaced tokens, meaning any token appearing on the wrong baseline (e.g. inline rather than a superscript).

With these types of errors, DRACULAE was shown to to have a moderate error rate with respect to misplaced symbols (roughly 10%) while recognizing less than 40% of expressions completely correctly. This gap is somewhat less than that of Garain and Chaudhuri, but the interpretation of these error rates is similarly unclear.

2.1.5 MathBrush

DRACULAE inspired an expression parser developed by Ian Rutherford [36]. Rutherford streamlined the approach taken by FFES by supporting single-pass parsing, eliminating the tree transformation step, and attempting to correct symbol recognition errors. This error correction used a notion of *structural confidence*, which measures the fit of a particular symbol into a parse tree based on the symbol's position and idealized shape. MathBrush uses Rutherford's parser in conjunction with a symbol recognizer tailored for mathematics, a user interface supporting interactive mathematical operations and expression plotting, and tools to integrate with computer algebra systems [22].

Accuracy evaluation of MathBrush's math recognizer has largely been performed independently on the symbol recognizer and expression parser. MacLean described a comparison of several symbol recognition algorithms [29]. Two scenarios were tested: in the first, the training and testing sets contained samples from the same writer; in the second, both sets contained samples from multiple writers. The evaluation used two accuracy measurements: top-1, in which the top-ranked candidate must be correct, and top-5, in which the correct symbol label must appear within the 5 most highly-ranked candidates.

Rutherford reported parsing accuracy under several scenarios designed to isolate the parser from symbol recognition errors. Rutherford's uses a pass/fail evaluation metric in which an expression must be parsed completely correctly to be correct, otherwise it is counted as incorrect. The test corpus is not described in detail, but its average expression size was 6.5 symbols, so the results are not useful for predicting recognition accuracy on typical math inputs.

Through experimentation with MathBrush, we have found approaches to math recognition using only baseline extraction to be fragile and difficult to implement effectively. Exclusive reliance on global baseline information means that small local variations in symbol placement can lead to poor recognition, or even failure to generate a candidate expression at all. Recognition errors of this type were shown to create usability problems for users, particularly new users [23].

2.1.6 Microsoft Math Recognition Panel

The upcoming release of Microsoft Corporation's Windows 7 operating system will include the Math Recognition Panel [6]. This Panel uses a sophisticated mathematics recognition engine which combines grammar-based parsing of expression layouts

with machine learning-based symbol and subexpression relation classifiers. We are not aware of any publications describing the training or evaluation procedures for this system, or of the metrics used.

2.2 Ground-truth generation

We are not aware of any work on tools and algorithms to aid in the generation and automatic ground-truthing of large, varied, on-line sketch corpora. However, Schwarz and Matousek used grammar-based methods to generate template expressions for use in training an automated telephone dialogue system [37]. As spoken language is much less formal than handwritten mathematics and their application was somewhat specialized, they were more concerned with handling a large amount of ambiguity within a small coverage area than with ensuring broad coverage of an entire domain.

Automatic ground truthing has generally not been attempted in sketch recognition domains, but OCR researchers have developed several techniques for automatically generating ground-truthed training and testing data. These techniques generally either generate perfectly ground-truthed synthetic data (eg. [18], [32]), or match real inputs to separate ground-truth created by hand, potentially with mistakes in both matching and ground-truthing (eg. [2]). Occasionally aspects of both approaches are combined as in [20].

As noted in the introduction, research in handwritten mathematics recognition has been ongoing for decades and the field is still an active area of research. It is becoming standard practice to study accuracy and usability by asking participants to transcribe a collection of expressions, as in [27], [38], and [23]. However, we are not aware of any large, publicly-available corpora of handwritten mathematical expressions. Most studies have collected a modest set of predetermined expressions, making it difficult to predict the recognizer's capacity to generalize to more varied expression sets.

2.3 The value of our corpus

The recognition systems described in Section 2.1 are representative in that they were evaluated on unavailable and largely unknown data sets using correctness metrics specific to the authors. Under these conditions, it is very difficult to make meaningful observations about the strengths or weaknesses of a particular recognition technique. Our corpus of math expressions facilitates the comparison of math recognition systems by providing a common set of testing data available to all researchers. Using the same test data will make published accuracy figures more transparent to the research community and will enable conclusions to be drawn more objectively about the benefits and drawbacks of any particular approach to recognition.

Our corpus may also be used during the construction of new recognition systems as training data. The systems cited above use geometric and spatial information to analyze the layout of math expressions. The expressions in our corpus can provide useful examples for training and testing such systems. However, we will see in Chapter 4 that the math expressions present in our corpus are uniformly distributed over different mathematical semantics. For this reason, it is not practical to use our corpus with recognition systems that rely on approximating the real distribution of math expressions (e.g., an adaptation of Chou's stochastic grammar approach [11] to handwritten input, or an application after [19] using Markov models).

During the creation of our corpus, we were careful to ensure that many combinations of symbol and expression shapes were represented. These varied expression shapes may allow researchers to develop more sophisticated models of expression layout taking into consideration how the shape of one part of an expression influences the position of another. (For example, the relative placement of the n in A^n versus e^n, or of the x in $\int x\ dx$ versus $\int_{1-\cos(t)} x\ dx$.) The researchers cited in this chapter generally created their own corpora by extracting math content from books, so the variations in expression and symbol shape in their corpora depend upon the contents of the particular books they used. In our corpus, this breadth of coverage is built-in by design.

Although our corpus may not be directly useful for training and evaluating all math recognizers, the techniques developed to create it are broadly applicable. The distribution of math semantics in corpus expressions may be tuned to be more realistic or to satisfy particular needs of individual researchers (for example, to test deeply-nested fractions). Our techniques may also be used to generate corpora for other structured sketching domains such as circuit diagrams, molecular diagrams, UML, etc.

Chapter 3

Corpus creation

As described in Chapter 1, the math recognition community currently suffers from a lack of corpora of ground-truthed expressions. This lack has led to ad-hoc training strategies and opaque accuracy measurements, making it difficult to directly compare the strengths and weaknesses of various approaches to recognition. In this chapter, we investigate the requirements of corpus creation, showing that they are contradictory and that no corpus can satisfy them perfectly. We evaluate a number of potential corpus creation methodologies, proposing some general strategies which offer the best available balance between practical and theoretical considerations. Finally, we illustrate one strategy through its concrete application to our own corpus creation project, which produced a publicly-available, ground-truthed collection of nearly 5000 hand-drawn mathematical expressions.

3.1 Requirements for corpus creation

The purpose of a corpus of hand-drawn sketches is threefold:

1. To provide training data during the construction of recognition systems.
2. To provide testing data during the evaluation of recognition systems.
3. To facilitate comparison of systems through a common set of testing data.

The generality of a recognition system is thus constrained by the domain coverage of the corpus used for training, as is the validity of an accuracy evaluation. For both training and testing, the corpus is taken to represent objective truth, so any errors in the corpus may lead to errors in recognition or evaluation results. These observations suggest that, to be of use in training and evaluating recognizers, a corpus must posess two properties:

1. It must capture the relevant properties of the sketch domain and the natural variations on those properties in a complete and representative way.

2. It must be ground-truthed with very high accuracy so that it may be taken to be objectively "correct".

Both of these requirements merit further discussion.

3.1.1 Capturing relevant domain properties

Within a sketch domain, the "relevant properties" which must be captured may differ between recognizers. For example, the math recognizer used in MathBrush is based largely on spatial analysis of input expressions, so the relevant properties include the relative positions of symbols and subexpressions. A math recognizer based on statistical models may use symbol co-occurrence rates and other frequentist measurements as relevant properties. Ideally, a corpus would be so complete that all possible properties could be reliably extracted from it; in practice, a corpus will have maximum utility with respect to one particular set of properties based on the methods used to construct it.

Furthermore, there is a natural conflict between the requirements of a "complete" and "representative" corpus. To be complete implies that a corpus contains all variations and combinations of syntax for a particular domain, but to be representative implies that a corpus reflects the natural distribution of these syntactic elements in real-world usage. Time and price considerations constrain the extent to which these two properties can co-exist within a corpus; however, a more representative corpus may lack examples of uncommon syntactic patterns, while a more complete corpus may be unrealistic. The appropriate balance of these two concerns, again, depends significantly on the recognition techniques the corpus is to be used with.

These points demonstrate that, in practice, a single corpus is unlikely to satisfy the requirements of all its potential users. This observation points out the utility of techniques for automating corpus creation. By reducing the costs associated with generating corpora, we may reduce the value of keeping them private and encourage the proliferation of several public corpora for different use cases, enabling standardized training procedures for recognition systems, and increasing the transparency of accuracy evaluations.

3.1.2 Correct ground-truth

It is obvious that the ground-truth labels in a corpus should be correct. However, in complex domains like mathematical expressions, what the correct label is can often be ambiguous. In part, this ambiguity is due to the aforementioned issues with selection of relevant properties, as these are the properties which must be identified in ground-truth labels. (e.g. symbol identity, spatial relationship between subexpressions, etc.) However, ambiguity also arises due to the complexity of the domain itself.

Figure 3.1: An expression where ground-truth assignment is arbitrary.

For example, the expression shown in Figure 3.1 was intended to be $\sqrt{QB}$. However, exactly which strokes comprise the B symbol is not clear. Ground-truth, if assigned at all, is somewhat arbitrary at best. In our corpus, ground-truth was assigned whenever the math expression was deemed legible by the operator providing the ground-truth. Expressions such as the one shown in Figure 3.1 thus have some arbitrariness in their labels.

Another source of ambiguity in ground-truth is the ambiguity inherent in written mathematics, described in Section 1.1. If an expression, like the one in Figure 1.1, has multiple valid interpretations, how should they be represented in the ground-truth? If a person is unable to uniquely identify the written expression, is it reasonable to assume that the ground-truth labels are objectively correct? The approach that we have taken is to assume that the writer's intentions represent objective truth (i.e. if they intended to write $ax+b$, then what they wrote objectively represents that expression). This approach means that our corpus is self-consistent even if it does contain only a single "correct" interpretation for expressions which may have multiple reasonable interpretations.

3.2 Tasks in corpus creation

Corpus creation is complicated – limited resources and conflicting requirements make it impossible to create a perfect general-purpose corpus for a complex sketch domain. It is therefore worthwhile breaking the overall process of corpus creation into general steps so that effective methodologies can be proposed. It is likely that many corpora will be required for domains like math recognition, and such methodologies can ease the creation of good-quality corpora.

A corpus consists of examples labeled with ground-truth. There are three general tasks required to attain these labeled examples:

1. The (abstract) sketches which will comprise the corpus must be selected.
2. The selected sketches must be rendered into concrete drawings.

3. The drawings must be annotated with ground-truth labels.

We will refer to these tasks as selection, rendering, and labeling, respectively.

Each of these tasks may be implemented by either manual or automatic procedures. For each task, we will evaluate the general characteristics of these two implementation styles with respect to criteria derived from the requirements for corpus creation from Section 3.1. Using these evaluations, we will propose some generally useful methodologies for corpus creation.

Note that another possibility is to use an interactive task implementation. That is, an automatic technique that is guided or honed by an operator (e.g. an automatic ground-truthing program in which a user resolves ambiguities or interactively corrects errors). We do not investigate these hybrid approaches here.

3.2.1 Selection

The selection task is the process by which sketches are chosen to be part of the corpus. In manual approaches, such as the one taken by Garain and Chaudhuri (as cited in Chapter 2), one typically amasses a collection of source materials (e.g. books, papers, etc.) and extracts the relevant sketches from the sources. Automatic approaches entail the synthesis of sketches from an abstract model of the sketch domain by some generative procedure.

Breadth of coverage can be difficult to achieve with manual approaches because sketches are taken from existing materials with their own biases. Each source provides only limited domain coverage, and the source selection may be biased by the corpus designer's own domain expertise or knowledge of recognizer tendencies. These biases can subtly and unintentionally affect the utility of selected sketches.

Conversely, an automatic approach can be designed so as to ensure that as many syntactic variations and combinations as possible are represented in the corpus. Such an approach cannot be said to be unbiased since bias is inherent in the generative models and algorithms used. However, this systematic bias is easily discerned by examining the generation algorithm. Such transparency makes it easier for researchers to evaluate the suitability of a particular corpus for use with their own recognizer.

Manually-selected sketches will naturally be realistic because they are taken from real-world examples, and this suggests that they are more representative of the sketch domain than their automatically-generated counterparts. This is not necessarily the case. Representativeness should be judged with respect to the "relavent properties" captured by the corpus, not by superficial similarity to natural examples. For example, the math recognizer in MathBrush considers only the overall shape of expressions, so unrealistic expressions can still be representative examples. But a stochastic approach may require that subexpression occurence rates in the

corpus are similar to those rates in real-world practice – in such a case, realism and representativeness coincide.

So, although manual sketch selection yields more realistic expressions, it does not necessarily yield more representative expressions. Representativeness is also difficult to achieve by automatic processes because to do so would require an accurate model of sketch domain properties on a very large scale. These properties are best determined by empirical evaluation, which would likely require a large corpus of real-world sketches. In other words, there is a bootstrapping problem.

The selection task is the most difficult of the three tasks to evaluate because it is where the complexities detailed in Section 3.1 are most prevalent. Sketch selection is where the tradeoff between breadth and realism must be made. The criteria for evaluating whether a particular method is appropriate will therefore depend on the indended uses for the corpus.

Based on the analysis above, we may draw some general conclusions. in cases where realism coincides with representativeness, it is preferable to use a manual approach to ensure that the corpus is suitable. In other cases, an automatic approach is preferable because it guarantees broad coverage. It is most preferable of all to use an automatic approach which can be tuned to approximate any distribution of sketches. With such an approach, one can generate sketches more and more representatively as knowledge of the large-scale properties of the sketch domain become clearer.

Automatic selection processes have some other, less crucial, advantages over manual processes. It is fast and easy to generate arbitrarily many unique sketches using an algorithm, but tedious and time-consuming to do so manually. The use of unique sketches can have an effect on training and testing with a particular corpus. If some form of user-omission is used to test, i.e. train on $n-1$ users and test on the remaining user, then using multiple drawing of the same sketch means that the system has been tested on sketches (from one user), but it has experienced each of these sketches before as training data from the other $n-1$ users. Leaving out sketches rather than users means that the system has been trained on that specific user's handwriting, which may also provide a higher than realistic performance measure.

Automatic processes can also be easier to extend. If new areas of the sketch domain need to be represented in an existing corpus, it may be easier to adjust the generation model than to gather a new collection of source materials and extract more sketches manually.

3.2.2 Rendering

Rendering is the process by which an abstract sketch is converted into a concrete drawing. In a manual approach, this process is transcription (i.e. a person draws a copy of the sketch). Automatic approaches would use an algorithm to convert an abstract sketch representation into a pictorial representation.

For this task, realism is the most important consideration: the rendered sketches must resemble how they might be drawn by hand. Clearly, the manual approach is ideal here, as the sketches are drawn by hand. It has also been shown that whether a person is transcribing a predetermined sketch or synthesizing a novel sketch has little effect on the utility of the sketch for training and testing purposes [15], strengthening the validity of a transcription-based approach.

Generally, automatic sketch generation is faster than manual transcription. Using an automatic approach also obviates the labeling task, as ground-truth can be generated along with the sketch rendering itself. However, we are not aware of any work in rendering sketches from a complex domain – it is very difficult to create a generative sketching model that produces realistic output. Furthermore, models for synthesizing handwritten symbols are often "reversible" in the sense that the same models can also be used to recognize symbols (e.g. [13], [10]). This fact suggests that using synthesized data to train a recognizer will not gain any new recognition capacity beyond that already posessed by the generative model.

For these reasons, it is preferable to be sure that rendered sketches are realistic by using manual transcription, at least until synthesis models can effectively produce a wide range of complex, realistic sketches.

3.2.3 Labeling

Labeling is the process of annotating a rendered sketch with ground-truth. Manual approaches entail the examination of each rendered sketch by a human operator who labels the appropriate parts of each sketch with the pertinent information, while an algorithm provides these labels in an automatic approach. In our study, the ground-truth labels identify symbols as well as the spatial relations connecting symbols and subexpressions together to form the entire math expression (e.g. containment, super-/sub-script relations, adjacency, etc.)

The most important consideration for this task is correctness. Ground-truth labels are taken to be objectively correct by training and testing procedures, so any errors may have far-reaching consequences affecting a recognizer's performance.

In this respect, manual approaches are preferable because people are experts at making sense of visual information and can therefore provide very accurate ground-truth. Manual labeling sketches with ground-truth is time-consuming, tedious work, though, and no labeler is perfectly accurate, so automatic approaches which can achieve near-human levels of accuracy are helpful as a time-saving measure.

Care is needed when using automatic approaches. If a recognizer R is used to analyze rendered sketches and label them with ground-truth, only sketches recognizable by R will be labeled, and some labels may be incorrect if R makes errors. Then the following situations may arise:

1. A recognizer trained on the labeled data will be trained on errors where R made errors.

2. When evaluating a recognizer on the labeled data, the recognizer's output will be compared against erroneous data where R made errors, invalidating the evaluation to some extent.

3. A recognizer trained on the labeled data will have incomplete domain coverage in the areas where R was unable to provide labeling.

4. A recognizer trained on the labeled data will not have significantly more recognition capacity than R.

So long as an automatic labeler's error rates are close to human standards, the first two situations are avoided somewhat, as they would naturally occur when human labelers make mistakes as well. The third situation is avoided if there is no particular domain area where the labeler consistently makes errors or fails to find a labeling. The fourth situation is more problematic as it suggests that it is unsuitable to use a sketch recognizer to label sketches.

One way to avoid the fourth situation is to change the labeling problem by introducing new input. For example, Chapter 5 describes a ground-truthing algorithm which takes as input a rendered sketch as well as a complete description of the idealized layout of the sketch. The algorithm would not be usable as a recognizer as it simply finds a matching between elements in the layout description and parts of the sketch. Any argument about degrading recognition capacity is moot.

In order to maintain correctness, manual ground-truthing is preferable unless an automatic approach is available which achieves near-human standards of accuracy.

3.3 Proposed methodology

Taking into consideration the analyses of the three tasks for corpus creation described above, we propose the following general methodology for creating sketch corpora:

1. Automatically select sketches using as flexible an approach as possible.

2. Manually transcribe the selected sketches.

3. Manually label the transcribed sketches with ground-truth, unless a viable automatic approach is available.

This is the process which we have endeavoured to follow in the creation of our mathematical corpus. Chapter 4 describes a flexible grammar-based approach for generating random mathematical expressions. The technique is applicable to any domain which can be modeled by a context-free grammar. Chapter 5 details our experiments with automatic ground-truthing using a simple matching algorithm.

This algorithm was highly accurate, but was not quite as accurate or complete as manually-generated ground truth, so the ground-truth published with our corpus was provided manually. The next section describes our manual transcription process.

3.4 Transcription process

To aid in the development of accurate math recognizers as a part of the MathBrush project (see Section 1.2), we created a large corpus of hand-drawn, ground-truthed mathematical expressions following the methodology outlined above. To obtain a large amount of data, we recruited students to transcribe math expressions for us. 20 participants transcribed expressions for roughly one hour each. Participants were reimbursed with a $10 Tim Horton's gift certificate. This study was reviewed by the Office of Research Ethics and received clearance to proceed (#15232).

The transcription process was carried out using various models of Tablet PCs running custom collection software on Windows XP. A screen shot of our data collection software is shown in Figure 3.2. After a sketch was generated by the automatic method described in Chapter 4, the resulting parse tree was converted to a LaTeX string, and then on to an image. This image was displayed to the participant for transcription, as shown in the screen shot. Participants' transcriptions were saved along with the grammar derivations, parse trees, LaTeX strings, and images files created during the sketch generation process. Examples of typical transcribed expressions are shown in Figures 3.3 and 3.4.

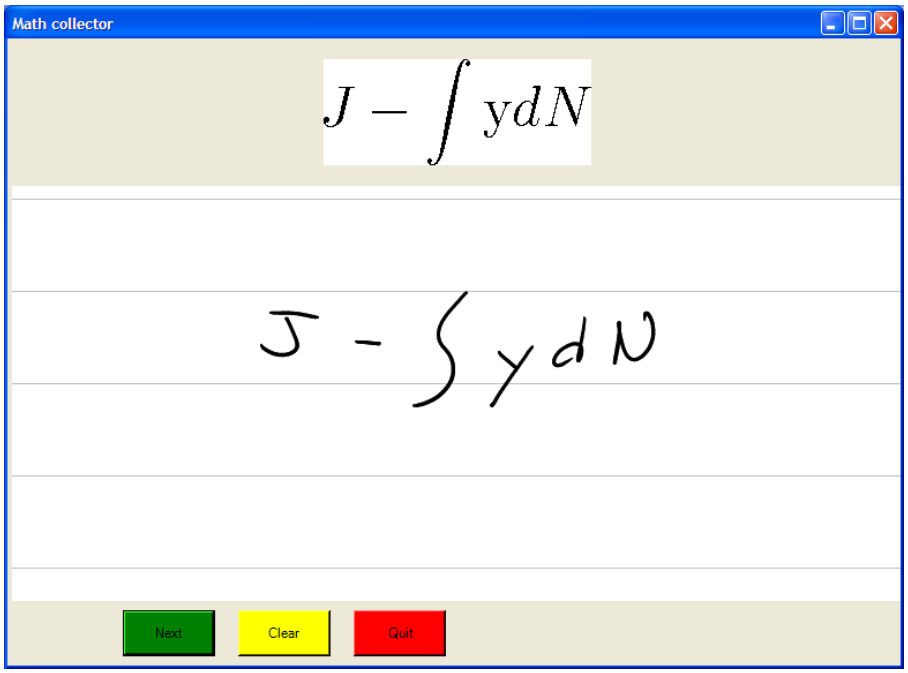

Figure 3.2: Our collection software in action.

Figure 3.3: A hand-drawn, randomly generated expression.

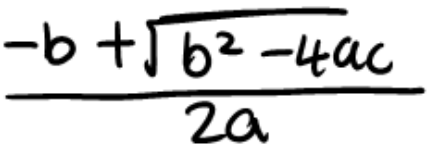

Figure 3.4: A hand-drawn, predetermined expression.

Fifty-three predetermined expressions from high-school and early University-level mathematics were also transcribed by each participant in the first 150 expressions displayed to them. These expressions were not automatically generated and provided a data set to "fall back on" in case the automatic processes failed to work well. An example of a predetermined expression is shown in Figure 3.5.

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

Figure 3.5: A prepared expression

As well as a short tutorial describing how to use the collection program, participants were given two instructions along before beginning the transcription process:

1. To write reasonably neatly (as if preparing an assignment for submission), and

2. To write math expressions and symbols as they naturally would, not necessarily how they were rendered in the image.

Many people have handwriting that is illegible to others and even to themselves. The first instruction was intended to reduce the effects of this illegibility. Guidelines were provided in our collection program to further encourage neatness. The second instruction was intended to improve the realism of our collected data and to discourage participants from simply copying the shapes of the symbols and expressions in the LaTeX rendering. Any transcription task must start with some initial rendering which may subtly influence participants' writing styles – by asking participants to consciously not copy the rendering style, we hope to have minimized this influence.

As can be expected in such a study, a number of samples were discarded. Our basic discard policy was that if an equation was incomplete or illegible to a human expert, it was discarded. An example of a discarded equation is shown in Figure

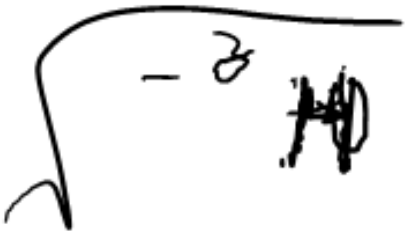

Figure 3.6: A discarded, illegible expression.

3.6. We felt it was unreasonable to expect an automated system to correctly infer the expected equation if a human expert was unable to do the same.

In all, 5119 transcriptions were collected from 20 users. Of these, 109 were blank and 355 were discarded, resulting in 4655 usable hand-drawn expressions. Comprising these expression are 25963 symbols drawn and 21264 relationships between subexpressions. This is a substantial amount of data and it was attained with a direct cost of only $200. Our collection study has shown that large and diverse corpora can be generated at a modest cost by using automated techniques.

Chapter 4

Template expression generation

To generate math expressions for transcription, we use an automated technique that performs a constrained walk through a context-free grammar describing mathematical equations. This approach has many advantages. First, expressions are created based on random sampling of symbols and relationships, ensuring that rare symbol combinations are reflected in the collected data. Secondly, this approach limits biases that a researcher may have in selecting testing data. Finally, this technique provides a limitless source of equations for participants to transcribe.

Our approach to math recognition uses a fuzzy relational context-free grammar to model the spatial structure of handwritten math. This chapter defines fuzzy relational context-free grammars and gives examples to illustrute how they may be applied to recognition and generation problems in the context of math expressions. It then shows how random derivations can be used to generate template expressions, and how the biases of the generation processes can be controlled.

4.1 Fuzzy relational context-free grammars

A natural approach to any parsing problem is to use a formal grammar to model the structure of the input. To model mathematical structure, we use a variant of the relational context-free grammars used by many authors ([39, 12, 35], etc.). This variant is called a *fuzzy relational context-free grammar* (fuzzy r-CFG).

In this scheme, a relational grammar models the visual structure of math expressions, and fuzzy sets model the uncertainty inherent in the recognition problem caused by special ambiguities. The grammar formalizes the spatial patterns encoding information in the input, ensuring recognition only produces valid mathematical interpretations. Fuzzy sets provide a formalism for representing the parse confidence of many alternative interpretations, helping to resolve ambiguities. We will review the basics of fuzzy set theory, then proceed to describe fuzzy r-CFGs.

4.2 Fuzzy set theory

The material in this section is introductory and is based on the original description of fuzzy sets by Zadeh in [40].

Definition 1. *A* fuzzy set $\tilde{X}$ *is a pair* (X, μ)*, where* X *is some underlying set and* $\mu : X \to [0, 1]$ *is a function giving the* membership grade in $\tilde{X}$ *of each element* $x \in X$*.*

A regular (ie. not fuzzy) set is called *crisp* and is equivalent to the universal set equipped with the membership function

$$\mu(x) = \begin{cases} 1 \text{ if } x \in X \\ 0 \text{ o.w.} \end{cases}$$

Note that fuzzy sets are typically denoted by the application of $\tilde{}$ to the name of the underlying set, as in the $\tilde{X}$ above. Here, the $\tilde{}$ symbol will be omitted whenever it is clear that a set is fuzzy.

Definition 2. *A* fuzzy relation *on the fuzzy sets* $\tilde{X}_1, \ldots, \tilde{X}_n$ *(where* $\tilde{X}_i = (X_i, \mu_i)$*) is a fuzzy set* (X, μ)*, where* $X = X_1 \times \cdots \times X_n$ *and* $\mu : X \to [0, 1]$*.*

We will use fuzzy relations extensively to represent the spatial relationships between the symbols and subexpressions contained within any particular math expression.

The common set operations – union, intersection, and Cartesian product – can be applied in a fuzzy context. Letting $\mathcal{M}$ denote the set of all fuzzy membership functions, these are defined as follows:

Definition 3. *Let* $\tilde{A} = (A, \alpha), \tilde{B} = (B, \beta)$ *be fuzzy sets. Then:*

$$\begin{aligned} \tilde{A} \cup \tilde{B} &= (A \cup B, u(\alpha, \beta)) \\ \tilde{A} \cap \tilde{B} &= (A \cap B, n(\alpha, \beta)) \\ \tilde{A} \times \tilde{B} &= (A \times B, c(\alpha, \beta)) \end{aligned}$$

where $u, n, c : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$*.*

Typical choices for u, n, c are $\max, \min, \max$, respectively, although many other possibilities exist.

4.3 Fuzzy relational context-free grammars

The formal definition of a fuzzy r-CFG is given below. A more comprehensible interpretation in the context of math recognition follows.

4.3.1 Definition

Definition 4. *A* fuzzy relational context-free grammar *(fuzzy r-CFG) G is a tuple $(\Sigma, N, S, T, R, r_\Sigma, P)$, where:*

- *Σ is a finite set of terminal symbols,*
- *N is a finite set of nonterminal symbols,*
- *$S \in N$ is the* start symbol*,*
- *T is the set of observables,*
- *R is a set of fuzzy relations on (T, T),*
- *r_Σ is a fuzzy relation on (Σ, T), and*
- *P is a set of productions, each of the form*

$$A_0 \overset{r}{\Rightarrow} A_1 A_2 \cdots A_n$$

where $A_0 \in N$, $r \in R$, $A_1, \ldots, A_n \in \Sigma \cup N$.

Whereas typical context-free grammars deal with *strings*, the objects generated by fuzzy r-CFGs will be called *expressions*. Let us investigate the definition from the point of view of representing mathematical expressions.

Σ and N are similar to their namesakes in "regular" context-free grammars. They provide an alphabet out of which expressions can be constructed, and a set of labels for various types of subexpressions. S is a label for a valid math expression.

The productions in P have the same fundamental structure as the productions of regular context-free grammars in which the left-hand element derives the sequence of right-hand elements. However, the fuzzy relation r appearing above the $\Rightarrow$ production symbol indicates a requirement that the relation r is satisfied by adjacent elements of the RHS. More specifically, if $t(i)$ denotes the set of input elements used while parsing A_i, then to parse A_0 requires that $(t(i), t(i+1)) \in r, i = 1, \ldots, n-1$.

In our application, T is the set of all possible sketches that a user can draw. The set R is the set of all fuzzy relations that appear as constraints in P. These relations act on sketches: they model spatial or conceptual relationships between sketch elements. In particular, for math recognition, we use five spatial relations: Right-to, Up-right-to, Down-right-to, Down-to, and Contains. For brevity, these will be denoted as $\rightarrow$, $\nearrow$, $\searrow$, $\downarrow$, and $\odot$, respectively. The membership grades of pairs of sketches in these relations represent recognition confidence in a particular spatial arrangement. The grades are combined with symbol recognition results to obtain confidence scores for parse trees.

The relation r_Σ models the relationship between sketches and terminal symbols: it is a symbol recognizer. By including symbol recognition into the grammar formalism, the integration of recognition and structural analysis in our recognition system is made explicit.

4.3.2 Examples

We illustrate the use of this grammar formalism through some examples.

1. Suppose that `[ADD]` and `[EXPR]` are nonterminals, + is a terminal, and $\rightarrow$ is a relation. Then the meaning of the production

$$\texttt{[ADD]} \overset{\rightarrow}{\Rightarrow} \texttt{[EXPR]} + \texttt{[EXPR]}$$

   is clear: `[ADD]` derives two instances of `[EXPR]` separated by +, all connected by the $\rightarrow$ relation.

2. In the superscript production,

$$\texttt{[SUP]} \overset{\nearrow}{\Rightarrow} \texttt{[EXPR]}\ \texttt{[EXPR]},$$

   the first RHS token represents the base of the exponentiation, and the second represents the exponent.

3. The following pair of productions represent an integral expression.

$$\begin{aligned} \texttt{[LIMITS]} &\overset{\downarrow}{\Rightarrow} \texttt{[EXPR]} \int \texttt{[EXPR]} \\ \texttt{[INTEGRAL]} &\overset{\rightarrow}{\Rightarrow} \texttt{[LIMITS]}\texttt{[EXPR]}d\texttt{[VAR]} \end{aligned}$$

   This example demonstrates how expressions using more than one writing direction can be modeled. An integral expression (`[INT]`) is constructed by attaching the integral sign and limits of integration to the integrand and variable of integration using the $\rightarrow$ relation. The limits (`[LIMITS]`) are connected with the $\downarrow$ relation and consist of an upper limit, the integration symbol, and the lower limit.

4.3.3 Derivations

Recall that, for regular CFGs, a derivation sequence describes the order in which nonterminal symbols were expanded and what those expansions were. For example, in the simple grammar

$$\texttt{[S]} \Rightarrow \text{a}\ \texttt{[S]}\ \text{b} \mid \text{ab},$$

`[S]` $\Rightarrow$ a `[S]` b $\Rightarrow$ aa `[S]` bb $\Rightarrow$ aaabbb is a valid derivation sequence.

In fuzzy r-CFGs, these sequences are extended to describe subexpression structure using parentheses. The sequence above is written as `[S]` $\Rightarrow$ (a `[S]` b) $\Rightarrow$ a(a `[S]` b)b $\Rightarrow$ (a(a(ab)b)b).

A more useful example is the math expression $\frac{ab+b}{2}$. Suppose we are using the toy grammar defined by:

$$\begin{aligned}
\texttt{[S]} &\overset{\downarrow}{\Rightarrow} \texttt{[S]}\text{—}\texttt{[S]} \\
\texttt{[S]} &\overset{\rightarrow}{\Rightarrow} \texttt{[S]} + \texttt{[S]} \\
\texttt{[S]} &\overset{\rightarrow}{\Rightarrow} \texttt{[S]}\,\texttt{[S]} \\
\texttt{[S]} &\Rightarrow a|b|2
\end{aligned}$$

Then the derivation

$$\begin{aligned}
\texttt{[S]} &\Rightarrow (\texttt{[S]} \downarrow \text{—} \downarrow \texttt{[S]}) \\
&\Rightarrow ((\texttt{[S]} \rightarrow + \rightarrow \texttt{[S]}) \downarrow \text{—} \downarrow \texttt{[S]}) \\
&\Rightarrow (((\texttt{[S]} \rightarrow \texttt{[S]}) \rightarrow + \rightarrow \texttt{[S]}) \downarrow \text{—} \downarrow \texttt{[S]}) \\
&\Rightarrow^{*} (((a \rightarrow b) \rightarrow + \rightarrow b) \downarrow \text{—} \downarrow 2)
\end{aligned}$$

uniquely captures the structure of the example expression.

This expression demonstrates how any expression which can be modeled using a particular grammar is associated with a derivation which has a unique fully-parenthesized form.

4.4 Grammar representation

In our implementation of the grammar described in section 4.3, each grammar production is associated with a tree generator and a string generator. The tree generator produces an expression tree that describes how terminal symbols are combined using mathematical operations to represent the syntax of the math expression. The string generator produces a string representation (e.g. LaTeX, MathML) of the expression tree.

For extensibility, the grammar and associated generators are encoded in an external text file. For example, the following defines the grammar production for addition:

```
ADD :: [AT0] <R> plus <R> [RT]
       {ADD_EXPR(%1 'EXPR_LHS', %3 'EXPR_RHS')}
       `%1 + %3` ;
```

In this example, on the first line, `ADD` is the name of the production's LHS nonterminal, `AT0` and `RT` are two other nonterminals, `plus` is a terminal name, and `R` is the relation code for the RIGHT relation. The second and third lines represent the

tree and string generators, respectively. The second line of the production, between the braces, describes a tree with root label "ADD_EXPR" with two children. The first child is labeled "EXPR_LHS" (the left hand operand of the addition operation) and is linked to the tree output by the tree generator for AT0. The second child is labeled "EXPR_RHS" (the right hand operand) and is linked to the tree output by the generator for RT. The string generator is described on line 3 (between the back ticks). As with tree generation, the %n notation indicates where to insert the output of string generators associated with the left hand operand AT0 and the right hand operand RT.

In our grammar, semantic content is described by the root labels produced by tree generators. The ADD production above therefore has semantic type ADD_EXPR. Not all productions include tree generators, however. For example, consider the following three productions: (Pipe symbols are used on the RHS to separate distinct productions.)

```
RT :: [ADD] | [SUB] | [AT0] ;
```

The nonterminal symbol RT represents a collection of expression types with the same level of precedence, in this case, addition, subtraction, and an isolated addition term.

In this example, each of the three nonterminals that the symbol RT can produce have distinct semantic types. Specifically, ADD always produces an addition expression having semantic type ADD_EXPR, SUB always produces a subtraction expression having semantic type SUB_EXPR, and AT0 may produce many types of expressions (e.g. exponentiation, integration, multiplication, etc.), so it's semantic content is not known a priori.

Therefore, RT itself does not have a fixed semantic type. As parsing proceeds and trees are formed, any valid parse of RT inherits the expression tree (and hence the semantic type) given by the tree generator for the particular nonterminal it produces at that point in the parse.

Unlabelled nonterminals like RT therefore represent different semantic types in different contexts. Unlabelled nonterminals can derive other unlabelled nonterminals (such as AT0 in the above example), so the semantic types they can assume are not always immediately apparent from their productions.

4.5 Random derivations

A random expression generator must balance two properties to be useful. It must generate samples typical of math expressions written on tablet computers (so that machine learning algorithms have a representative training set). It must also generate as many combinations as possible of bounding box shapes and relative positions

(so that algorithms can be trained on low-probability spatial relationships). However, only a relatively small number of examples can be collected from any subject. In our experience, approximately 225 equations can be written by a subject in a one hour session. As a result, there exists a tradeoff between accurately representing the frequency of various mathematical relationships and providing a broad enough coverage of various mathematical relationships to fully train and test a recognizer. Our approach opts for broader coverage at the expense of reflecting the true frequency with which particular mathematical relationships occur in real-world expressions. This means, for example, that in our generated expressions, exponents, subscripts, and fractions are more common than one would normally encounter.

Random expressions are generated using the r-CFG for mathematical expressions described above. Fuzzy relationship values are not relevant when using a grammar to generate expressions. The essential idea of generating a random equation is quite basic: given a current symbol, choose a grammar production arbitrarily and recurse on each nonterminal in the RHS. However, care must be taken to avoid three problems:

1. Recursing blindly may generate extremely large expressions. Expression length must be constrained since the expressions are to be transcribed by human users.

2. The structure of productions in the grammar definition can bias the distribution of mathematical structures generated when productions are chosen at random. We must take care to eliminate this bias.

3. Ascender (e.g. upper-case symbols, some lower-case symbols such as 'b'), descender (e.g. 'p'), and baseline (e.g. 'a', 'o') symbols provide different bounding box profiles for relationships. We must ensure that no type of symbol (ascender, descender, baseline) is over-represented in spatial relationships.

We address each of these issues in turn.

4.5.1 Managing expression length

To limit expression length, we introduce a parameter $0 < p_{\text{inc}} \leq 1$. The algorithm begins with $p = 0$. Instead of simply choosing a production, it draws x from a uniform distribution on $[0, 1]$. If $x < p$, a derivation leading to a single terminal symbol is selected if possible; otherwise p is incremented by p_{inc}, a random production is selected, and the algorithm recurses. This process does not guarantee a maximum expression size, but, by varying the value of p_{inc}, we can control the expected output size while still allowing a degree of variability in expression length.

4.5.2 Managing semantic distribution

The structure of a grammar can bias the distribution of generated expressions. To see why, consider the following toy grammar:

$$
\begin{aligned}
\texttt{[ADD]} &\overset{\rightarrow}{\Rightarrow} \texttt{[TERM]} + \texttt{[ADD]} \\
\texttt{[ADD]} &\Rightarrow \texttt{[TERM]} \\
\texttt{[TERM]} &\overset{\rightarrow}{\Rightarrow} \texttt{[VAR]}\,\texttt{[TERM]} \\
\texttt{[TERM]} &\overset{\downarrow}{\Rightarrow} \texttt{[ADD]}\text{—}\texttt{[ADD]} \\
\texttt{[TERM]} &\Rightarrow \texttt{[VAR]} \\
\texttt{[VAR]} &\Rightarrow a|b|\cdots
\end{aligned}
$$

In this grammar, `[ADD]` is the start symbol. The grammar is capable of generating four distinct types of mathematical expressions: addition (from `[ADD]`), multiplication and fractions (from `[TERM]`), and variables (from `[VAR]`).

Assume that the template generation algorithm chooses productions from a given nonterminal uniformly at random. Then, starting from `[ADD]`, it will generate an addition expression half the time. But, to generate a multiplication expression, it must first produce `[TERM]` (with probability 1/2), and then produce `[VAR]` `[TERM]` (with probability 1/3, since there are three productions with LHS `[TERM]`). Thus, multiplication expressions will only be generated 1/6 of the time. Clearly, we cannot rely on a mechanism like this to reflect a reasonable distribution of mathematical structures.

We do not attempt to model the "real" distribution of mathematical expressions. Instead, we take the following approach which distributes uniformly over all supported expression types. For each nonterminal symbol N, the template generator constructs a list of all mathematical expression types derivable from N. Then, whenever it needs to expand an instance of N, it selects an expression type uniformly at random, rather than a production.

To repeat the example above, suppose an instance of `[ADD]` requires expansion during template generation. Rather than choosing randomly between the two productions with LHS `[ADD]`, we now randomly select one of the expression types derivable from `[ADD]` (i.e. one of "addition", "multiplication", "fraction", or "variable") and create the intermediate derivation steps all at once. Suppose we choose "fraction". Then the derivation sequence

$$\texttt{[ADD]} \Rightarrow \texttt{[TERM]} \overset{\downarrow}{\Rightarrow} \texttt{[ADD]}\text{—}\texttt{[ADD]}$$

replaces the single instance of `[ADD]` in the derivation being generated.

By decoupling mathematical semantics from the syntactic description provided by a grammar, we achieve a uniform distribution over expression types. This approach can easily be extended so as to generate non-uniform distributions over

mathematical semantics or to generate expressions according to some other statistical process.

4.5.3 Managing bounding-box shape

Finally, to obtain broader coverage of relative bounding box positions, the Latin and Greek letter symbols in the grammar were grouped into classes based on their characteristic shape with respect to a baseline (ascender, descender, baseline). The grammar was modified so that each class is produced by a single non-terminal. In this way, we obtained a uniform distribution over symbol shapes rather than symbols.

4.6 Examples

Below are two examples of expressions generated by the above process.

$$\left[\frac{B}{7}\right]^{N+6} \qquad \beta + \frac{\mathrm{z} - L\left(\mathrm{v}\right)}{\sqrt{s}} \int 24 d\mathrm{X}_{\mathrm{h}}$$

Figure 4.1: Generated expressions

4.7 Output formats

The algorithm sketched above generates random derivations from a grammar. Using the tree generator, one can produce and serialize parse trees for use in recognizer testing. Using the string generator, one can generate LaTeX strings, which can be converted to images and displayed to subjects for transcription. Strings representing generated expressions for use in a computer algebra system such as Maple or Mathematica can also be generated. The latter is particularly useful for automatically evaluating recognizer accuracy via simple, built-in arithmetic. These structures are generated for each randomly-derived expression.

The algorithm also creates a *derivation string* describing the spatial layout generated by the derivation in the form described by Section 4.3.3. For example, suppose the expression $2 + m^a + \frac{b}{3}$ is generated. The corresponding derivation string is `2 _R_ plus _R_ (m _AR_ a) _R_ plus _R_ (b _B_ horzline _B_ 3)`. Here, the parentheses indicate nested subexpressions and the underscores delimit relation codes (`_R_`=RIGHT, `_AR_`=ABOVE-RIGHT, `_B_`=BELOW). The derivation strings are used to guide the automatic ground-truthing process which we will describe in section 5.

Chapter 5

Automatic ground-truthing

When generating a corpus of ground-truthed data to train and/or validate recognition systems, it is convenient to use a program to automatically ground-truth at least part of the collected data. In Section 3.2, we noted the existence of a "bootstrapping problem" associated with this approach and stated that one way to avoid this problem is to use additional layout description data associated with a hand-drawn expression. In our experiment, we have this data available: we computationally generated each template expression along with the derivation strings described in Section 4.7, which fully describe the expected spatial layout of transcribed expressions.

However, we do not know which strokes the participants drew correspond to which symbols in the derivation string. Ground-truthing is, therefore, a constrained recognition task in which the symbol labels appearing in the derivation string must be matched to groups of strokes in a handwriting sample. Because of the constraints on possible recognizer output, a weak recognition engine should generate useful ground-truth.

This chapter describes a simple algorithm that we developed to label hand-drawn math expressions with ground truth by matching groups of ink to elements in derivation strings. The algorithm's accuracy is evaluated with respect to three different measurement criteria, each applicable for different uses of corpus data.

5.1 Algorithm

We devised a naive greedy algorithm for matching strokes to derivation strings. At a high level, the algorithm may be viewed as a heuristic best-first search through symbol recognition results, with backtracking. To compute the heuristic, the algorithm combines confidence scores computed by a symbol recognizer and spatial relation classifiers. These tools were trained on a small bootstrap data set which was manually annotated with ground-truth. This bootstrap data is independent of our main corpus and was obtained by having several researchers in our group draw a

small set of expressions. The ground- truthing algorithm is based on an inaccurate but somewhat useful heuristic: given a derivation string, remove all the parentheses and match the derivation string to sketched equation symbol by symbol. For example, the derivation string from section 4.7,

```
2 _R_ plus _R_ (m _AR_ a) _R_ plus _R_ (b _B_ horzline _B_ 3)
```

is interpreted as

```
2 _R_ plus _R_ m _AR_ a _R_ plus _R_ b _B_ horzline _B_ 3
```

The algorithm considers a derivation string of the form $S_1 r_1 S_2 r_2 \ldots r_n S_n$ where the S_i are names of terminal symbols and the r_i are any of the spatial relations. Given such a string, suppose that symbols $S_1, \ldots, S_{m-1}$ have been matched to ink with confidence z that the match is correct. The ink used by symbols $S_0, \ldots, S_{m-1}$ is marked "used". All other ink is currently "unused". To match symbol S_m, the algorithm proceeds as follows:

Order all possible occurrences of S_m in the input ink in decreasing order of confidence
for each possible occurrence of S_m **do**
  Let c be the local confidence of S_m
  Mark ink strokes comprising this occurence "used"
  Recurse at S_{m+1} with score zc
  if a match was found **then**
    return the match
  Mark ink strokes comprising this occurence "unused"
return failure

In the $m = 1$ case (i.e. when the algorithm is matching against the first symbol), we match to the occurence of S_1 having the minimal sum of spatial relationship scores from all other possible symbols. That occurence is taken to be the "starting-point" of the expression and is, loosely speaking, the top-left symbol in the drawing. This choice works because the spatial relations used are unidirectional (e.g. we use a Right relation but not Left), so the matching process consumes symbols roughly in a top-left to bottom-right order.

The local confidence c computed by the algorithm is the product of relational and symbol recognition confidences.

This algorithm may take exponential time to report failure in the worst case. During testing the algorithm was stopped if it had not reported a match after two minutes.

5.1.1 Example

To make the algorithm more concrete, we demonstrate it on an actual handwriting sample collected in our study.

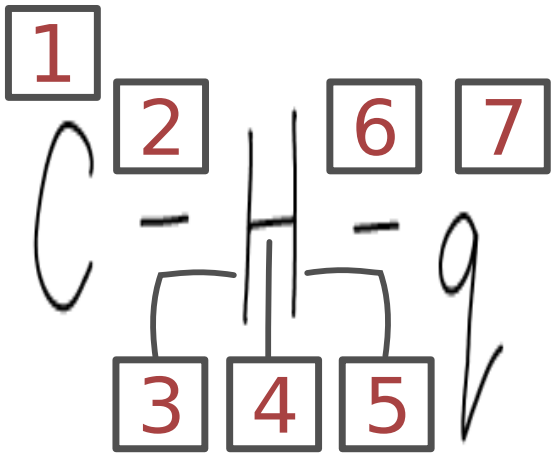

Figure 5.1: Ink collected from a study participant.

In this example, the automatically generated expression was

$$C - H - q$$

This corresponds to the derivation string

```
(C _R_ horzline _R_ (H _R_ horzline _R_ q))
```

Figure 5.1 shows the transcription submitted by a study participant. Note that the boxed numbers serve only as a way to identify symbols in the ink; they are not part of the transcription.

Using the algorithm's notation, we have $S_1 =$ `C`, $S_2 =$ —, $S_3 =$ `H`, $S_4 =$ —, $S_5 =$ `q`. Partial symbol recognition results are shown in Table 5.1, with recognition scores indicated in parentheses.

1	2	3	4	3,4
C (1.0)	— (1.0)	1 (0.20)	— (0.12)	t (0.19)
c (0.98)	$\surd$ (0.25)	l (0.18)	$\surd$ (0.04)	+ (0.14)
q (0.76)	w (0.14)	$\int$ (0.09)	w (0.02)	f (0.13)

5	4,5	3,4,5	6	7
1 (0.23)	+ (0.37)	H (0.77)	— (1.0)	q (1.0)
l (0.14)	t (0.23)	Π (0.30)	$\surd$ (0.21)	ϵ (0.72)
$\int$ (0.12)	T (0.22)	n/a	w (0.11)	[(0.69)

Table 5.1: Partial symbol recognition results during ground-truthing.

We begin with $m = 1$. The only occurence of `C` is at symbol 1 with score 1.0. Symbol 1 is marked "used" and we recurse.

$m = 2$. There are three occurences of —: at symbols 2, 4, and 6. The recognition and $\rightarrow$ relation scores for each symbol are computed and summarized in Table 5.2.

Stroke(s)	Reco. score	Rel. score	Product
2	1.0	0.95	0.95
4	0.12	0	0
6	1.0	0	0

Table 5.2: Score summary for occurences of —.

Based on the score summary, stroke 2 is the best match. Stroke 2 is marked "used" and we recurse.

$m = 3$. The only occurence of `H` comprises symbols 3, 4, and 5 and has recognition score 0.77. The $\rightarrow$ relation score from symbol 2 is 0.88, so the symbol is usable. Symbol 3 is marked "used" and we recurse.

$m = 4$. The only remaining occurence of — is at symbol 6 with score 1.0. Notice that the other occurences (at symbols 2 and 4) have already been marked "used" in steps 2 and 3. The $\rightarrow$ relation score from symbols 3,4,5 is 0.78. Symbol 6 is marked "used" and we recurse.

$m = 5$. The only occurence of `q` is at symbol 7 with score 1.0, as symbol 1 (which also has an occurence) was marked "used" in step 1. The $\rightarrow$ relation score from symbol 6 is 0.4, so the symbol is usable. As this is last symbol, the current match is returned with score

$$1.0 * 1.0 * 0.95 * 0.77 * 0.88 * 1.0 * 0.78 * 1.0 * 0.4 = 0.2$$

This example demonstrates that the ground-truthing algorithm is efficient and accurate for expressions with basic spatial layouts, provided that symbol recognition is correct.

5.2 Experiments and results

To test the accuracy of our algorithm, the entire corpus of 4655 expressions was annotated manually. Automatically-annotated data was then compared to the manually-annotated data using two scenarios.

5.2.1 Experimental scenarios

1. The algorithm described in Section 5.1.

2. Pre-training the symbol recognizer for each user on approximately 20% of their input data, selected randomly, and then running the algorithm from Section 5.1.

Scenario 2 was introduced because the algorithm frequently failed when it could not match a hand-drawn symbol to a symbol in the specified input string. The second scenario ensures that the symbol recognizer has the greatest possible chance to match user-drawn symbols against symbols in the generated expression tree.

5.2.2 Measures of accuracy

We plan to use parts of our expression corpus to aid in the development of classifiers for the relations in our grammar formalism. Our relational classifiers are based on bounding-box geometry, so it is natural to measure the accuracy of our ground-truthing algorithm by considering how similar labeled bounding boxes are between automatically- and manually- ground-truthed expressions.

Due to issues of ambiguity described in Section 3.1, manually-annotated samples occasionally had ground-truth assigned somewhat arbitrarily. Although it is unreasonable to expect a recognition system to correctly identify such transcriptions, their bounding boxes can still be useful from the point of view of training relational classifiers. In these cases, it is not always obvious when an automatically-annotated relation should be considered correct, so we have elected to use three accuracy measurements which permit varying degrees of differentiation in ground-truth assignment, as detailed below. For each scenario, three accuracy measurements are reported, normalized to give percentages:

1. *Full expression:* In this measurement, an annotated sample is considered correct if all symbols and relations are correctly annotated and all subexpression bounding boxes exactly match their counterparts in the manually-annotated ground-truth.

2. *Exact bounding-box:* This measurement concerns individual relations arising from the randomly-generated derivation. A relation between any two subexpressions is considered correct if its constituent symbols are correct and the automatically-annotated bounding boxes of both subexpressions exactly match their counterparts in the manually-annotated ground-truth.

3. *Bounding-box overlap:* This measurement also concerns individual relations. In it, each relation is assigned a score between 0 and 1. The score is computed by averaging bounding-box similarity measurements from each of the automatically-annotated relation's constituent subexpressions to their counterparts in the manually-annotated ground-truth. We define bounding-box similarity as the ratio of the area of intersection of the two boxes to the area of the larger box. Two boxes thus have similarity 0 if they are disjoint and similarity 1 if they are identical, with a range of possible values in between.

Figures 5.2 and 5.3 illustrate the difference between exact and overlap bounding-box accuracy.

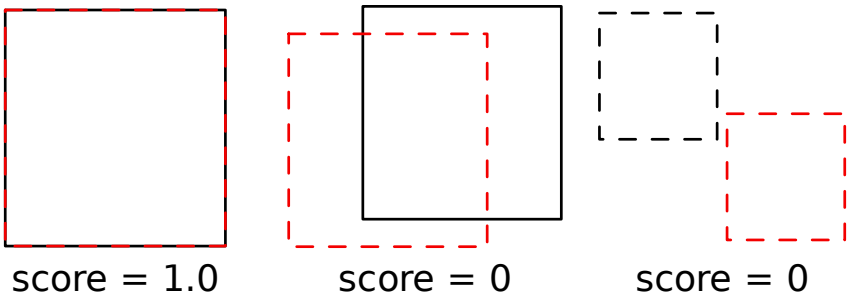

Figure 5.2: Exact bounding-box accuracy.

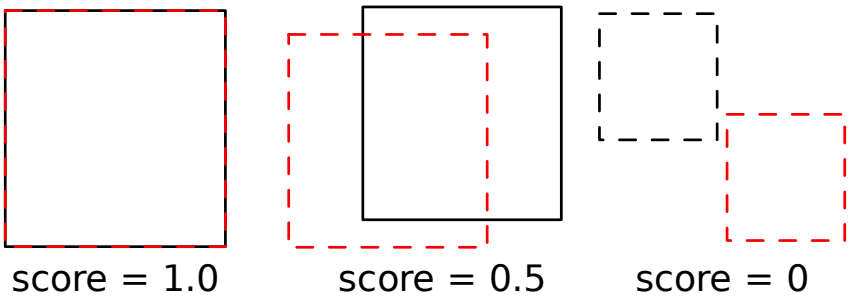

Figure 5.3: Bounding-box overlap accuracy.

These three measurements are defined in increasing order of permissiveness of match. Full expression accuracy requires every symbol, relation, and bounding box in an entire sample to be annotated precisely the same as in the manual ground-truth, while bounding-box similarity accuracy allows symbol bounding boxes to disagree slightly but still count as a close match.

The ground-truthing algorithm produces highly accurate ground-truth for hand-drawn input when compared to manual ground-truth, as shown by Figure 5.4 For scenario 1, the algorithm achieved 90% full expression accuracy. Exact bounding-box accuracy was 93.6% and overlap accuracy was 97.7%. The accuracy rates for scenario 2 are about the same. The error bars in the graph indicate the algorithm's accuracy on the data of the most- and least- accurately annotated subjects.

Scenarios 1 and 2 differ significantly in their reject rates. The algorithm either produces highly-accurate ground-truth or rejects the entire expression. For scenario 1, the reject rate was 55.1%, while for scenario 2, it dropped to 46.5%. This reject rate will be discussed in more detail in Section 5.3.

The most appropriate measure of annotation accuracy depends on one's application. If annotated data is used to train spatial relation classifiers on bounding box information, it is appropriate to count similar, but not identical, bounding-box annotations as partially correct because they may still provide useful training data. On the other hand, if one uses automatically-generated ground-truth to test the accuracy of an isolated symbol recognition system, then full expression or exact bounding-box accuracy may be a more appropriate measurement.

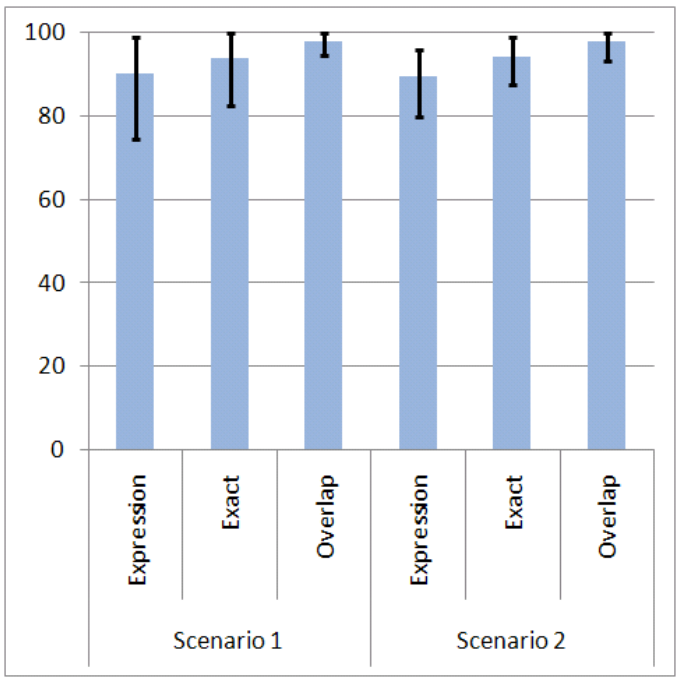

Figure 5.4: Automatic annotation accuracy

5.3 Discussion

Our automatic ground-truthing method has a high accuracy rate but suffers from a high reject rate. It is important to note that the accuracy of this annotation technique remained high through all experiments. While it would be preferable to maintain accuracy with a lower reject rate, the present algorithm is far more useful than it would be if both accuracy and reject rates were lower. Accurately ground-truthing even half of a large corpus automatically would obviate a significant amount of manual annotation.

It is not clear how to further reduce the rejection rate of the ground-truthing algorithm. In scenario 1, described in section 5.2, a number of samples were rejected due to symbol classification errors. The errors arose because the generic models in the symbol classifier differed from particular users' handwriting styles. By pre-training on 20% of samples, the rejection rate was reduced from 55.1% to 46.5%, as is shown in the results of scenario 2. However, experiments pre-training the symbol recognizer on a larger proportion of samples did not demonstrate significant improvement.

One approach to lowering the reject rate is to incorporate manual intervention directly into the ground-truthing application. In cases currently rejected due to low recognition scores, an operator could manually annotate the problem symbols and let the automatic procedure handle the rest. We expect such an approach would handle the majority of the cases rejected by the current system while requiring only a fraction of the time required for full manual annotation. Other approaches

to reducing the reject rate may also be possible. Our corpus includes the entire set of expressions, and other researchers are free to experiment with algorithms to perform symbol identification and subexpression relation assignment.

It is interesting to note that in order to verify our accuracy claims, we manually ground-truthed the entire corpus. Unfortunately, with any new technique, some measure of its effectiveness is necessary. In this case, to test how well we could ground-truth data, we needed ground-truthed data against which to compare our algorithm's output. However, we now have the confidence that, should we expand our corpus, we can automate the ground-truthing and still expect high accuracy in the data we generate.

Chapter 6

Conclusions

This thesis has presented some tools that we have used to generate a large, accurately ground-truthed corpus of hand-drawn mathematical expressions. The corpus is publicly available, and opens new possibilities for applying statistical techniques to math recognition and evaluating recognition accuracy. In this chapter, we describe some ways that the present work could be extended and refined, then summarize the highlights of the thesis.

6.1 Future work

We believe that the general corpus collection method outlined in Chapter 3 ensures broad, representative coverage while saving researcher time whenever possible through automated processes. Extensions to this work therefore fall into two categories: extensions to the template expression generation technique presented in Chapter 4, and improvements to the math expression ground-truthing algorithm from Chapter 5.

6.1.1 Template expression generation

The algorithm presented in Chapter 4 worked very well for generating syntactically-correct mathematical expressions from a relational context-free grammar. We can see two possible directions for immediate extension:

1. Construct grammars and template expressions for different types of sketches, and
2. Make the semantic content of generated expressions more "realistic".

We discuss each of these extensions briefly.

New sketch types

Grammar-based generation and recognition approaches are amenable to any sketch domain which can be described by a formal grammar. These may include course-of-action diagrams, flowcharts, electrical schematics, chemistry formulas, etc. So long as the information content can be described in a structured way, a grammar can be used to formalize that structure.

Although our template generation technique was demonstrated by generating mathematical expressions, it is more broadly applicable and could be used as the starting point for corpus creation in many other sketch domains.

Realistic semantics

The examples of generated expressions in Section 4.6 show that, while the math expressions constructed using our automatic approach are syntactically correct, they do not look "natural" to transcribers or observers. Variable and operation selection occurs at random, so variable names do not match one another within expressions, operations are combined in ways not arising in common mathematical use, and subexpressions which are commonly short (such as limits of summation) often contain abnormally many symbols.

These artifacts of randomized generation weaken any arguments we make that generated expressions are representative of the sketch domain. The argument would be more solid if our technique incorporated some logic making generated expressions more "natural-looking". There are several simple tools one could use to achieve this:

1. Assign prior probabilities to variables, operators, and expression types to reflect their natural rate of occurence.
2. After variable or operator selection, re-weight the distribution so as to reflect the natural co-occurence rate with other variables and operators.

These techniques would cause the distribution of automatically-generated expressions to more closely resemble that of human-generated expressions, rather than that uniform distribution over symbol shapes and operations that we currently emulate. Whether such a resemblance is necessary or even desirable is an open question.

6.1.2 Ground-truthing mathematical sketches

As described in Section 5.3, the simple ground-truthing algorithm we devised was very accurate, but rejected roughly half of its inputs as unrecognizable. Clearly, we would like to reduce this rejection rate while maintaining high accuracy. Such a reduction will likely require more sophisticated techniques than those we have

developed here, but two simple extensions may prove useful: to incorporate subexpression grouping information present in the grammar derivations, and to perform targeted symbol recognizer pre-training.

Subexpression group information

Our ground-truthing algorithm discards all subexpression nesting information and attempts to match a two-dimensional grammar derivation as a linear string. This approach is simple, but often fails, especially in larger expressions with more complicated two-dimensional structure. It may be possible to reduce the rejection rate by keeping this nesting information and constructing a more accurate programmatic model of the expression's spatial layout.

Targeted recognizer pre-training

Scenario 2 in Section 5.2.1 involved randomized pre-training of a symbol recognizer on each user's handwriting prior to running the ground-truthing algorithm. The goal of this process was to isolate the errors and rejections made during ground-truthing to any errors and rejections made by the symbol recognizer. However, randomized pre-training is not the most effective way to achieve this isolation.

The symbol recognizer we used includes a database of known symbol types and shapes. Input symbols are compared to database symbols to determine matches, which are returned in ranked order. Some symbols drawn by study participants are similar to symbols already present in the database. Pre-training on such symbols accomplishes nothing. Ideally, pre-training should be performed only on symbols drawn by participants which are different from any model symbols in the database.

To determine which symbols should be pre-trained on, the recognizer must be used and its results compared with manually-generated ground-truth information. Any recognition mismatches can be collected and used to generate a set of symbols to use for pre-training. In this way, maximum isolation can be achieved to distinguish between recognition errors and ground-truthing errors, allowing more realistic rejection and error rates to be determined for the ground-truthing algorithm.

6.1.3 Recognition accuracy evaluation

The mathematical corpus we created will be used to evaluate recognizer accuracy. Objective accuracy metrics are therefore another avenue for investigation. This is an area which has not received much attention in the past, possibly due to the lack of standardized testing corpora.

Among the papers describing accuracy cited in Chapter 2, there is substantial variation in the size of the testing suite, the number of writers, the types of expressions, and the number of recognized symbols. Furthermore, there are differences in

the criteria used to determine correctness. Is an expression correct if all the symbols are recognized correctly? If the system infers the correct expression despite misclassified symbols? If a recognizer produces multiple interpretations, must the first result be correct? If not, how many incorrect interpretations are tolerable?

There are many open questions which must be resolved to create a standardized, objective evaluation framework. It is likely that the answers to these questions will differ based on application domain and use cases. Therefore, a general framework allowing recognition systems to be compared and evaluated under differing metrics would be ideal. In this way, application designers could select the technology most appropriate to their own problem domains, and researchers could focus their work on improving specific weaknesses in existing technology.

6.2 Conclusions

In this thesis, we presented a number of useful tools for constructing large, accurately ground-truthed corpora of hand-drawn sketches. These tools were illustrated through the creation of a corpus of mathematical expressions, but some of the techniques used are broadly applicable to corpus creation in general.

Chapter 3 gave a general framework for corpus creation. It proposed to automatically generate a large number of template sketches, transcribe these sketches by hand, and ground-truth them manually unless viable automatic procedures existed. This process prevents designer bias from affecting expression selection, ensures that the expressions are drawn in a way that reflects natural use, and saves researcher time as much as possible through automated ground-truthing. The process was exemplified through a collection project which created a corpus of roughly 5000 hand-drawn math expressions.

Next, Chapter 4 detailed our novel technique for automatically generating template expressions from a relational context-free grammar. The technique extracts a random derivation from a grammar modeling the sketch domain, and it manages expression length and nonterminal selection so as to produce expressions which are easily transcribed by human users, but still provide broad domain coverage.

Finally, Chapter 5 presented an algorithm for ground-truthing hand-drawn mathematical expressions. The algorithm is naive but highly accurate, although it has a very high rejection rate of about 50%. Ground-truthing accuracy was determined both with and without symbol recognizer pre-training, and the accuracy and reject rate of the algorithm were discussed.

These tools have proven to be useful in the creation of our mathematical corpus, and we believe that similar techniques can be used in many sketch recognition domains. It is our hope that other researchers will use the corpus we have collected in their own research.

References

[1] S.K. Bandyopadhyay, *Flowchart recognition - a syntactic approach*, Int. J. Electron. **67** (1989), no. 2, 179–185.

[2] Joost van Beusekom, Faisal Shafait, and Thomas M. Breuel, *Automated ocr ground truth generation*, Document Analysis Systems, 2008. DAS '08. The Eighth IAPR International Workshop on, Sept. 2008, pp. 111–117.

[3] Frederick W. Blackwell and Robert H. Anderson, *An on-line symbolic mathematics system using hand-printed two-dimensional notation*, Proceedings of the 1969 24th national conference (New York, NY, USA), ACM, 1969, pp. 551–557.

[4] D. Blostein and A. Grbavec, *Recognition of mathematical notation*, 1996.

[5] D. Blostein and L. Haken, *The lime music editor: a diagram editor involving complex translations*, Softw. Pract. Exper. **24** (1994), no. 3, 289–306.

[6] Radakovic Bogdan, Goran Predovic, and Bodin Dresevic, *Geometric parsing of mathematical expressions*, U.S. Patent Application #20080253657, Filed 2007, Microsoft Corporation.

[7] Kam-Fai Chan and Dit-Yan Yeung, *An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions*, Pattern Recognition **33** (1998), 375–384.

[8] ———, *Error detection, error correction and performance evaluation in on-line mathematical expression recognition*, in On-Line Mathematical Expression Recognition, Pattern Recognition, 1999.

[9] ———, *Mathematical expression recognition: A survey*, International Journal on Document Analysis and Recognition **3** (1999), 3–15.

[10] Hyunil Choi, Sung-Jung Cho, and J.H. Kim, *Generation of handwritten characters with bayesian network based on-line handwriting recognizers*, Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on, Aug. 2003, pp. 995–999.

[11] P.A. Chou, *Recognition of equations using a two-dimensional stochastic context-free grammar*, SPIE Visual Communications and Image Processing IV **1199** (1989), 852–863.

[12] Gennaro Costagliola, Masaru Tomita, and Shi-Kuo Chang, *A generalized parser for 2-d languages*, Visual Languages, 1991, pp. 98–104.

[13] J. Dolinsky and H. Takagi, *Synthesizing handwritten characters using naturalness learning*, Computational Cybernetics, 2007. ICCC 2007. IEEE International Conference on, Oct. 2007, pp. 101–106.

[14] B. Edwards and V. Chandran, *Machine recognition of hand-drawn circuit diagrams*, Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on, vol. 6, 2000, pp. 3618–3621 vol.6.

[15] Martin Field, Sam Gordon, Eric Peterson, Raquel Robinson, Tom Stahovich, and Christine Alvarado, *The effect of task on classification accuracy: Using gesture recognition techniques in free-sketch recognition*, Sixth Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM), ACM, 2009, p. nn.

[16] Julian Fierrez, Javier Ortega-Garcia, Doroteo Torre Toledano, and Joaquin Gonzalez-Rodriguez, *Biosec baseline corpus: A multimodal biometric database*, Pattern Recognition **40** (2007), no. 4, 1389 – 1392.

[17] U. Garain and B.B. Chaudhuri, *Recognition of online handwritten mathematical expressions*, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **34** (2004), no. 6, 2366–2376.

[18] P. Heroux, E. Barbu, S. Adam, and E. Trupin, *Automatic ground-truth generation for document image analysis and understanding*, Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on, vol. 1, Sept. 2007, pp. 476–480.

[19] G.E. Kopec and P.A. Chou, *Document image decoding using markov source models*, Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on, vol. 5, Apr 1993, pp. 85–88 vol.5.

[20] Anand Kumar, A. Balasubramanian, Anoop Namboodiri, and C.V. Jawahar, *Model-Based Annotation of Online Handwritten Datasets*, Tenth International Workshop on Frontiers in Handwriting Recognition (Guy Lorette, ed.), Université de Rennes 1, Suvisoft, Oct. 2006.

[21] G. Labahn, E. Lank, S. MacLean, M. Marzouk, and D. Tausky, *Mathbrush: A system for doing math on pen-based devices*, The Eighth IAPR Workshop on Document Analysis Systems (DAS) (2008).

[22] _____, *Mathbrush: A system for doing math on pen-based devices*, Proc. of the Eighth IAPR Workshop on Document Analysis Systems, 2008.

[23] G. Labahn, E. Lank, M. Marzouk, A. Bunt, S. MacLean, and D. Tausky, *Mathbrush: A case study for interactive pen-based mathematics*, Fifth Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) (2008).

[24] E. Lank, J. Thorley, S. Chen, and D. Blostein, *On-line recognition of uml diagrams*, Document Analysis and Recognition, International Conference on **0** (2001), 0356.

[25] E.H. Lank, *A retargetable framework for interactive diagram recognition*, Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on, Aug. 2003, pp. 185–189 vol.1.

[26] Joseph J. Laviola, Jr., *Mathematical sketching: a new approach to creating and exploring dynamic illustrations*, Ph.D. thesis, Providence, RI, USA, 2005, Adviser-Dam, Andries Van.

[27] Joseph J. LaViola, Jr., *An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations*, Third Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) (New York, NY, USA), ACM, 2006, pp. 157–164.

[28] Joseph J. LaViola, Jr. and Robert C. Zeleznik, *Mathpad2: a system for the creation and exploration of mathematical sketches*, SIGGRAPH '04: ACM SIGGRAPH 2004 Papers (New York, NY, USA), ACM, 2004, pp. 432–440.

[29] Scott MacLean, *Matching techniques for mathematical symbol recognition*, Tech. report, Symbolic Computation Group - University of Waterloo, 2007.

[30] William A. Martin, *Computer input/output of mathematical expressions*, SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation (New York, NY, USA), ACM, 1971, pp. 78–89.

[31] Reiner Marzinkewitsch, *Operating computer algebra systems by handprinted input*, ISSAC '91: Proceedings of the 1991 international symposium on Symbolic and algebraic computation (New York, NY, USA), ACM, 1991, pp. 411–413.

[32] O. Okun and M. Pietikainen, *Automatic ground-truth generation for skew-tolerance evaluation of document layout analysis methods*, Pattern Recognition, 2000. Proceedings. 15th International Conference on, vol. 4, 2000, pp. 376–379 vol.4.

[33] P.J. Phillips, Hyeonjoon Moon, P. Rauss, and S.A. Rizvi, *The feret evaluation methodology for face-recognition algorithms*, Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on, Jun 1997, pp. 137–143.

[34] V. Popovici, J. Thiran, E. Bailly-Bailliere, S. Bengio, F. Bimbot, M. Hamouz, J. Kittler, J. Mariethoz, J. Matas, K. Messer, B. Ruiz, and F. Poiree, *The BANCA Database and Evaluation Protocol*, 4th International Conference on Audio- and Video-Based Biometric Person Authentication, Guildford, UK (Berlin), Lecture Notes in Computer Science, vol. 2688, SPIE, 2003, pp. 625–638.

[35] D. Prusa and V. Hlavac, *Mathematical formulae recognition using 2d grammars*, Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on, vol. 2, Sept. 2007, pp. 849–853.

[36] I. Rutherford, *Structural analysis for pen-based math input systems*, Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2005.

[37] Jana Schwarz and Václav Matousek, *Creation of a corpus of training sentences based on automated dialogue analysis*, TSD '01: Proceedings of the 4th International Conference on Text, Speech and Dialogue (London, UK), Springer-Verlag, 2001, pp. 418–426.

[38] Steve Smithies, Kevin Novins, and James Arvo, *A handwriting-based equation editor*, Graphics Interface, 1999, pp. 84–91.

[39] K. Wittenburg, L. Weitzman, and J. Talley, *Unification-based grammars and tabular parsing for graphical languages*, Journal of Visual Languages and Computing **2** (1991), 347–370.

[40] L.A. Zadeh, *Fuzzy sets*, Information Control **8** (1965), 338–353.

[41] R. Zanibbi, D. Blostein, and J.R. Cordy, *Baseline structure analysis of handwritten mathematics notation*, Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on, 2001, pp. 768–773.

[42] _____, *Recognizing mathematical expressions using tree transformation*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **24** (2002), no. 11, 1455–1467.

[43] Robert Zeleznik, Timothy Miller, Chuanjun Li, and Joseph J. Laviola, Jr., *Mathpaper: Mathematical sketching with fluid support for interactive computation*, SG '08: Proceedings of the 9th international symposium on Smart Graphics (Berlin, Heidelberg), Springer-Verlag, 2008, pp. 20–32.