Grammar-based techniques for creating ground-truthed sketch corpora

Scott MacLean[1], George Labahn, Edward Lank, Mirette Marzouk, David Tausky

{smaclean, glabahn, lank, msmarzouk, datausky}@uwaterloo.ca

David R. Cheriton School of Computer Science

University of Waterloo

200 University Avenue West

Waterloo, ON, Canada

N2L 3G1

a submission for the

---

[1]Corresponding author

Abstract:

Although publicly-available, ground-truthed corpora have proven useful for training, evaluating, and comparing recognition systems in many domains, the availability of such corpora for sketch recognizers, and math recognizers in particular, is currently quite poor. This paper presents a general approach to creating large, ground-truthed corpora for structured sketch domains such as mathematics. In the approach, random sketch templates are generated automatically using a grammar model of the sketch domain. These templates are transcribed manually, then automatically annotated with ground-truth. The annotation procedure uses the generated sketch templates to find a matching between transcribed and generated symbols. A large, ground-truthed corpus of handwritten mathematical expressions presented in the paper illustrates the utility of the approach.

1 Introduction

Many recognition domains have benefited from the creation of large, realistic corpora of ground-truthed input. Such corpora are valuable for training, evaluation, and regression testing of individual recognition systems. They also facilitate comparison between state-of-the-art recognizers. Accessible corpora enable the recognition contests which have proven useful for many fields. For example, the voice recognition, facial recognition, and OCR communities have – often with the help of third parties like NIST – created large, publicly-available corpora, standardized the problem statement for recognition, and held regular recognition competitions to evaluate state-of-the-art approaches. Such measures have enabled researchers in these fields to train sophisticated recognition models on real, representative data, and to objectively measure recognition accuracy, facilitating comparison between recognizers.

Unfortunately, the availability of ground-truthed corpora for sketch recognition domains is poor. Although math recognition, our chosen domain, has a history dating back at least four decades (e.g. [2, 11, 12]), we are not aware of any publicly-available corpus of hand-drawn math expressions. Instead, math recognition systems have been designed, trained, and evaluated largely on an ad-hoc basis, using performance metrics specific to a particular recognizer implementation (e.g. [4, 17, 6, 10]). Such variegated metrics prohibit any meaningful comparison between systems.

Creating large corpora for any hand-drawn sketch domain is challenging. First, capturing hand-drawn content is a lengthy process. Unlike in many scanned image domains, writing styles vary significantly across users, and it is difficult to use techniques like deformation or noise models to create realistic simulated data. (Bunke provides a useful survey of these gen-

erative models as they apply to handwriting [3].) Second, given a set of hand-drawn sketches, obtaining accurate ground-truth is difficult, essentially requiring manual interpretion of each diagram. Beyond the time required to interpret expressions, hand-drawn content can be ambiguous (see Figure 1), raising questions about the accuracy of the ground-truth created by a human reader. Third, in each hand drawn sketch individual symbols must be segmented, carefully matched with their ground truth, and the relationships between adjacent symbols must be inserted. This, also, is a manual task, requiring significant effort.

In this paper, we present two new techniques to help automate the construction of a large, hand-drawn, ground-truthed corpus of math expressions and validate our techniques by creating a publically available corpus of over 4500 expressions. One technique creates template expressions by randomly sampling the space of expressions generated by a grammar. The other automatically labels manual transcriptions of these expressions with ground-truth by finding a matching between generated and transcribed symbols. Although this paper focuses on handwritten mathematics, our techniques can be adapted for use in other sketch domains, provided their syntax can be described by a context-free grammar. The techniques are based on an analysis of the requirements and practical constraints imposed upon a corpus.

To be broadly useful, a sketch corpus must be accurately ground-truthed at both the syntactic and semantic levels, so that training and testing can be done reliably. It must also contain a large sample of sketches representative of the sketch domain and of many drawing styles, so that it is an accurate model of real-world inputs. Both of these properties are difficult to achieve in practice. Bias is inherent in any process of selecting sketches for inclusion in a corpus, limiting representativeness. Corpus designers may unintentionally introduce bias by selecting sketches based on their own knowledge of the sketch domain or of the tendencies of an existing recognizer. In complex domains, it may be impossible to anticipate all the possible forms that input may take, limiting domain coverage. In such domains, a single sketch may be reasonably interpreted in several ways, leading to ambiguous ground-truth. Such complexity further complicates training and testing processes.

Our corpus was created as a tool for training and testing the math recognition engine of MathBrush, our experimental pen-based system for interactive mathematics [9]. The system 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.

The rest of this paper is organized as follows. Section 2 explores the issues of bias and ambiguity mentioned above, and proposes a general methodology for creating useful cor-

pora. Section 3 develops a technique for deriving random expressions from a grammar while controlling bias. Section 4 presents and analyzes an algorithm which labels these manually-transcribed expressions with ground-truth. Finally, we conclude with some thoughts on how this work may be extended.

2   Corpus creation

To be useful, a corpus must capture the relevant properties of the sketch domain and the natural variations on those properties in a complete and representative way, and it must be ground-truthed with very high accuracy. This section explores the difficulties associated with attaining these properties, and describes the approach we have taken to create a large ground-truthed corpus of hand-drawn mathematical expressions.

2.1   Complete and representative coverage

To be complete implies that a corpus contains all variations and combinations of syntax for a particular domain. To be representative implies that a corpus reflects the natural distribution of these syntactic elements in real-world usage. There is a natural conflict between these properties, limiting the extent to which they may co-exist in practice. A more representative corpus may lack examples of uncommon syntactic patterns, while a more complete corpus may be unrealistic.

The appropriate balance of these properties should be judged with respect to the domain properties used during recognition, rather than by superficial similarity to natural examples. For example, if a math recognizer determines whether or not two symbols form a superscript relationship by examining only bounding-box geometry, then it is unnecessary to ensure that a realistic distribution of symbol identities appears in superscript expressions in a corpus.

Because representativeness is relative to recognition features, rather than to human intuition, we propose to use an automatic approach to generating corpus expressions. Using a grammar-based model of handwritten mathematics, we generated random, syntactically-valid mathematical expressions as LaTeX strings (see Section 3). These template expressions are then converted to images and displayed to users for manual transcription.

2.2   Highly-accurate ground-truth

The benefits of recognizer training and the validity of recognition accuracy measurements are limited by the correctness of corpus ground-truth. However, in a complex sketch domain, such as handwritten math, it is not always possible to find a single, correct meaning for

an expression. For example, the expression shown in Figure 1 affords several reasonable interpretations: $ax + b, aX + b, a^x + b, axtb$, etc.

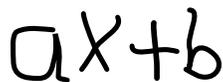$$aX+b$$

Figure 1: An ambiguous math expression.

Such examples suggest that it can be difficult, or even impossible, to provide perfect, unambiguous ground-truth. It may therefore be best to include several interpretations of a single expression as ground-truth, although it is not clear how they should be applied during recognizer testing or (especially) training. For our corpus, we have assumed that the writer's intentions represent objective truth. For example, if they intended to write $ax + b$, then what they wrote represents only that expression. This approach means that our corpus is self-consistent even if it is not fully general in ambiguous cases.

2.3   Mathematical expression corpus

Using the techniques described in this paper, we have created a publicly-available corpus of roughly 5000 hand-drawn mathematical expressions. In this section, we describe the expression transcription process, and the type of ground-truth provided with our corpus.

2.3.1   Transcription process

To ensure that the handwritten expressions in our corpus were realistic, we recruited 20 students to manually transcribe automatically-generated math expressions for one hour. Each student was reimbursed with a $10 coffee gift certificate.

In our study, automatically-generated mathematical expressions were displayed on-screen as images, and participants transcribed them using custom collection software running on Tablet PCs, as shown in Figure 2. Participants were instructed to write reasonably neatly, and to draw expressions and symbols as they would naturally, rather than copying the LATEX rendering.

Fifty-three manually-selected expressions from high-school and early University-level mathematics were also transcribed by each participant. These expressions provided a data set to "fall back on" in case the automatic processes failed to work well. Typical transcribed expressions are shown in Figure 3.

Any transcription that was incomplete or illegible to a human expert was discarded (e.g. the transcription shown in Figure 4). In all, 5119 transcriptions were collected from the
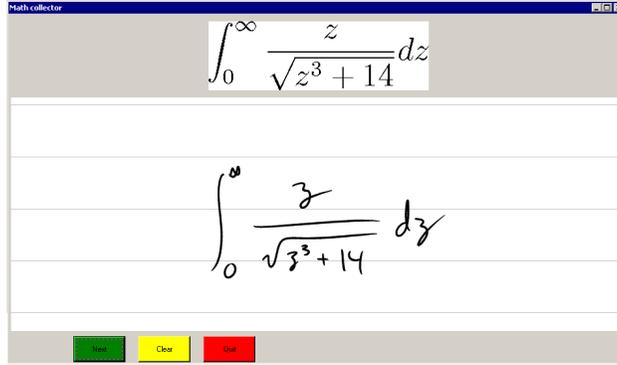
Figure 2: Our collection software in action.

Figure 3: Example expression transcriptions.

20 participants. Of these, 109 were blank and 355 were discarded, resulting in 4655 usable hand-drawn expressions. Comprising these expressions are 25963 symbols drawn and 21264 relationships between subexpressions.
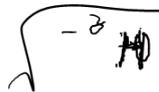

Figure 4: A discarded, illegible expression.

2.3.2 Corpus ground-truth

In mathematical expressions, as with other structured sketch domains (e.g. electrical schematics, music), the meaning of a particular sketch is determined not only by the symbols it contains, but also by the relative positions of those symbols within the sketch. For example, two letters written side-by-side in mathematics (ab) mean something different than if the second letter is written as a superscript (a^b). To understand a sketch, one must therefore simultaneously understand its constituent symbols as well as the relationships between them.

In our corpus, ground-truth is provided for each sketch as an expression tree representing the mathematical expression drawn by the user. Each tree models the syntactic layout and the semantic meaning of a corpus expression. Each terminal symbol in a tree is associated with the strokes which render that symbol in the handwritten transcription. Our ground-truth therefore captures the entire structure of mathematical expressions: symbol identities,

two-dimensional relationships between symbols, subexpression nesting structures, and semantic interpretation. Figure 5 shows an schematic example of a ground-truth expression tree. In this figure, mathematical semantics are indicated by expression tree nodes, as are the spatial relationships between adjacent subexpressions (e.g. $\rightarrow$, $\nearrow$). Terminal nodes in the expression tree are matched with the corresponding strokes in the sketched expression.
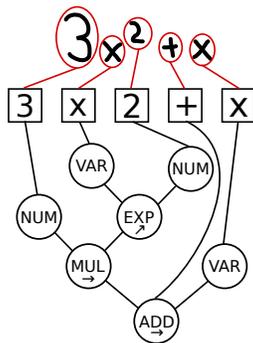

Figure 5: Schematic illustration of ground-truth for our mathematical corpus

3   Template expression generation

To select math expressions for transcription, we use an automated technique that generates a random derivation from a relational context-free grammar describing mathematical notation. This section defines relational context-free grammars, then shows how random derivations can be used to generate template expressions and how the biases of the generation process can be controlled.

3.1   Fuzzy relational context-free grammars

To model mathematical structure, we use a variant of the relational context-free grammars used by many authors ([16, 5, 15, 13], etc.) called a *fuzzy relational context-free grammar* (fuzzy r-CFG). In this scheme, grammar rules formalize the spatial patterns of symbol arrangements that construct mathematical meaning, and fuzzy sets provide a formalism for representing the degree to which a particular interpretation of a sketched expression is valid. Fuzziness is not used by our generation algorithm, but is used by the automatic ground-truthing technique described in the next section.

Definition 1. *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,*

6

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

Σ and N are similar to their namesakes in 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.

For our application, we take T to be the set of all possible sketches that a user can draw, r_Σ to be the output of a symbol recognizer, and R to be the set of relevant relationships between subexpressions (i.e. inline, superscript, subscript, inline vertical, and containment). The membership grade of a pairs of sketches in these relations represent recognition confidence in a particular spatial arrangement. These grades are combined with symbol recognition results to obtain confidence scores for parse trees.

The productions in P have the same fundamental structure as the productions of regular context-free grammars, except that the relation r appearing above the $\Rightarrow$ production symbol indicates a requirement that the relation r is satisfied by adjacent elements of the RHS.

For example, denoting nonterminals by [*], the production [ADD] $\overset{\rightarrow}{\Rightarrow}$ [ADD_LHS] + [ADD_RHS] models typical infix addition notation, while [SUP] $\overset{\nearrow}{\Rightarrow}$ [SUP_BASE][SUP_EXP] models exponentiation.

To model real-world meaning, nonterminal symbols may be associated with mathematical semantics. For example, expanding the [ADD] production above may imply that an addition operation is being generated. Other nonterminals may inherit the semantics of symbols expanded later. For example, the production [REL_OP] $\Rightarrow$ [LESS_THAN]|[GREATER_THAN_OP] acts as a placeholder aggregating similar operations together. The semantics of [REL_OP] may therefore differ depending upon which production is selected. In such a case, the semantics are inherited.

3.1.1 Derivations

Recall that a derivation sequence describes the expansion of nonterminal symbols as grammar productions are applied. For fuzzy r-CFGs, we extend these sequences to describe subexpression structure using parentheses.

For example, consider the toy grammar defined in Figure 6. `[ADD]` is the start symbol for this grammar.

$$\begin{aligned}
\texttt{[ADD]} &\overset{\rightarrow}{\Rightarrow} \texttt{[TERM]} + \texttt{[ADD]} &\textit{(addition)}\\
\texttt{[ADD]} &\Rightarrow \texttt{[TERM]}\\
\texttt{[TERM]} &\overset{\rightarrow}{\Rightarrow} \texttt{[VAR][TERM]} &\textit{(multiplication)}\\
\texttt{[TERM]} &\overset{\downarrow}{\Rightarrow} \texttt{[ADD]}\!-\!\texttt{[ADD]} &\textit{(fraction)}\\
\texttt{[TERM]} &\Rightarrow \texttt{[VAR]}\\
\texttt{[VAR]} &\Rightarrow a|b|\cdots &\textit{(variable)}
\end{aligned}$$

Figure 6: A simple fuzzy r-CFG describing some mathematical syntax.

Using this grammar, the structure of the expression $\frac{ab+b}{c}$ is completely and uniquely captured by the following derivation sequence:

$$\begin{aligned}
\texttt{[ADD]} &\Rightarrow \texttt{[TERM]} \Rightarrow (\texttt{[ADD]} \downarrow - \downarrow \texttt{[ADD]})\\
&\Rightarrow ((\texttt{[TERM]} \rightarrow + \rightarrow \texttt{[ADD]}) \downarrow - \downarrow \texttt{[ADD]})\\
&\Rightarrow (((\texttt{[VAR]} \rightarrow \texttt{[TERM]}) \rightarrow + \rightarrow \texttt{[ADD]}) \downarrow - \downarrow \texttt{[ADD]})\\
&\Rightarrow^* (((\texttt{[VAR]} \rightarrow \texttt{[VAR]}) \rightarrow + \rightarrow \texttt{[VAR]}) \downarrow - \downarrow \texttt{[VAR]})\\
&\Rightarrow (((a \rightarrow b) \rightarrow + \rightarrow b) \downarrow - \downarrow c)
\end{aligned}$$

Note that the parentheses indicate subexpression grouping. We call the final, fully-parenthesized step in a derivation sequence a *derivation string*. Derivation strings completely and uniquely capture the idealized two-dimensional structure of a mathematical expression.

3.2   Random derivations

The essential idea of generating a random derivation is quite basic: given a current nonterminal, choose a grammar production having that nonterminal on its LHS 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 control 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.

8

3.2.1 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 guarantees a maximum expression depth, since eventually $p \geq 1$, but still allows the expression length and complexity to vary considerably.

3.2.2 Managing semantic distribution

The structure of a grammar can bias the distribution of generated expressions. To see why, consider the grammar given in Figure 6. 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 are not aware of any report on the large-scale distribution of semantics in mathematical writing, so we do not attempt to model the distribution of expression types encountered in practice. Instead, we distribute uniformly over all expression types supported by our grammar, as follows. For each nonterminal symbol N, the algorithm constructs a list of all mathematical expression types derivable from N. Then, whenever it expands an instance of N, it selects an expression type uniformly at random, rather than a production.

To repeat the example above, suppose the algorithm is expanding an instance of [ADD]. Rather than choosing randomly between the two productions with LHS [ADD], it randomly selects one of the expression types derivable from [ADD] (i.e. one of "addition", "multiplication", "fraction", or "variable") and generates the intermediate derivation steps all at once. For example, if "fraction" is selected, then the derivation sequence

$$[\text{ADD}] \Rightarrow [\text{TERM}] \Rightarrow ([\text{ADD}] \downarrow - \downarrow [\text{ADD}])$$

is generated in a single step.

Using this approach, it is possible to approximate any desired distribution over mathematical semantics. One could, for instance, bias expression generation toward fractions to obtain a number of deeply nested fractions when designing a testing corpus. Similarly, one could emperically approximate the frequency with which various mathematical operators occur and "plug in" those values to obtain a more realistic distribution of expression types. However, basing generation on measured frequencies re-introduces problems related to unintentional bias of corpus designers, as the frequencies may be a function of the mathematical domain that was sampled. Furthermore, to truly obtain a randomized, but realistic-looking, corpus would require a more sophisticated domain model. There is a significant difference between simply choosing, say, integration ten percent of the time, compared with choosing integration in appropriate contexts, with appropriate variable names, integrands, limits, etc.

3.2.3   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. Although the grammar contains a preponderance of ascender symbols (since every capital letter is an ascender), this approach of random class selection yields a uniform distribution over symbol shapes rather than symbols identities.

3.3   Output

Each production in our grammar is equipped with a string generator which converts the internal grammar representation into a string. We used these generators to produce LaTeX strings representing each generated expression. These strings were then converted to images for display in our transcription program by using standard tools. We also produced a derivation string for each generated expression. Derivation strings are used by the automatic ground-truthing algorithm described in the next section.

Figure 7 shows two examples of expressions generated by the above process.

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

Figure 7: Generated expressions

4   Automatic ground-truthing

Using a program to automatically provide ground-truth labels would significantly reduce the amount of manual labour required to construct a large corpus. However, using a particular recognition system R to provide ground-truth labels may limit the capabilities of a new recognition system trained on that ground truth, because the available training data is restricted by the capabilities of R. Our approach to automatic ground-truthing avoids this situation.

We wish to avoid making any assumptions that could impact the validity of training and testing procedures using our corpus. Therefore, to label a transcribed expression with ground-truth, we use only symbol and relation classifiers that have no knowledge of mathematical semantics, and were trained in isolation from our corpus data. Using the derivation string associated with the transcription, the labeling problem is simply to match each terminal symbol in the string to the corresponding ink strokes in the transcription. This section describes our labeling algorithm and evaluates its accuracy with respect to three different metrics. Each metric is applicable for different uses of corpus data.

4.1   Algorithm

Given a derivation string D and a corresponding transcription, our goal is to find a function f mapping each terminal symbol in D to the set of ink strokes representing that symbol. This task is not trivial because, although we know which symbols should appear in the transcription, we do not know exactly where they are. There may be multiple instances of some symbols (as in $x^3 + 3x^2 + 2x + 3$, for example), and handwriting is ambiguous, so each group of strokes may be recognizable as several different symbols.

More formally, a derivation string D is either a single terminal symbol, $D = \alpha$, or it contains a number of smaller derivation strings concatenated by a relation, $D = (D_1 r \ldots r D_k)$. Let $S(\alpha, t) \in [0, 1]$ be the symbol recognition score for terminal α on a group t of ink strokes. Similarly, for each relation r, let $r(t_1, t_2) \in [0, 1]$ be the classifier confidence that relation r applies between groups t_1, t_2 of ink strokes.

Fix a derivation string D, and let $f(\alpha)$ be a function matching each terminal symbol α appearing in D to a group of ink strokes in the transcription. Define the goodness of the matching f on derivation string D, $\omega(f, D)$ inductively, as follows:

$$\omega(f, D) = \begin{cases} S(\alpha, f(\alpha)) & \text{for } D = \alpha \\ \left(\prod_{i=1}^{k} \omega(f, D_i) \right) \left(\prod_{i=1}^{k-1} r(f(D_i), f(D_{i+1})) \right) & \text{for } d = (D_1 r \ldots r D_k) \end{cases}$$

Using these definitions, we wish to find $f^* = \text{argmax}_f \, \omega(f, D)$. Note that we only need to consider matchings f that map each terminal α apearing in D to a group g of strokes such that $S(\alpha, g) > 0$.

Unfortunately, this formulation is not convenient for computation. If the symbol recognizer reports n_i potential instances for each terminal α_i appearing in D, then there are $\mathcal{O}(\prod_i n_i)$ feasible matchings f. Furthermore, consider a natural recursive solution in which we find f for a derivation string $(D_1 r \ldots r D_k)$ by dividing the transcription into k subexpressions and recursing. To be matched, the ith subexpression must contain exactly the terminal symbols appearing in D_i, so in the worst case, there are $\mathcal{O}(\prod_i n_i)$ subdivisions to recurse on! Clearly, we cannot afford to enumerate all the possibilities.

Instead, we use the following simplification: rather than measuring the relation $r(f(D_i), f(D_{i+1}))$ between subexpressions, we measure $r(f(D_i), f(\alpha_h))$, where α_h is the first terminal symbol to appear in D_{i+1}. This replacement avoids enumerating subdivisions explicitly and facilitates a straightforward best-first search algorithm.

Using this simplification, we cannot guarantee that the ground-truthing algorithm will find the optimal matching f^*. But, intuitively, if there is a relation between two subexpressions, we expect this relation to hold between the first subexpression and the first symbol of the second subexpression as well. For example, in e^{x+2}, the subexpression $x + 2$ and its leading symbol x are both in superscript positions relative to the e. In the fraction $\frac{a}{b+c}$, the $b + c$ subexpression and its leading symbol b are both below the fraction bar. Note that, depending on the behaviour of the symbol and relational classifiers, the optimal matching may be incorrect, or matchings may not exist at all (see Figure 10 and associated discussion.)

The algorithm matches the terminals in the derivation string from left to right, keeping track of where subexpressions start and end. Suppose that the terminal symbols in the derivation string are $\alpha_1, \alpha_2, \ldots, \alpha_n$, and that a matching f has been obtained for $\alpha_1, \ldots, \alpha_{m-1}$ with running score z. The ink used by the matched symbols is marked "used". All other ink is currently "unused". The ink to measure the relation into α_m against is called I_{prev}. To find the rest of the matching, the algorithm proceeds as follows:

if $m = n + 1$ **then**

   return success

Let r be the incoming relation to α_m

Order all "unused" occurrences of α_m in the input ink in decreasing order of confidence

for each possible occurrence I_{curr} of α_m **do**

$f(\alpha_m) \leftarrow I_{curr}$

$c \leftarrow S(\alpha_m, I_{curr}) \cdot r(I_{prev}, I_{curr})$

Mark all strokes in I_{curr} "used"

if α_m is the last symbol of a subexpression **then**

    Let α_{m-p} be the first symbol of the subexpression

    Let $I_{sub} = f(\alpha_{m-p}) \cup \cdots \cup f(\alpha_m)$

    Recurse at α_{m+1} with score zc and $I_{prev} = I_{sub}$

else

    Recurse at α_{m+1} with score zc and $I_{prev} = I_{curr}$

if recursion returned success **then**

    return success

else

    // no match was found; backtrack and try a different assignment for α_m

    Mark all strokes in I_{curr} "unused"

end for

return failure

When $m = 1$ (i.e. when the algorithm is matching against the first symbol), we sort all occurences of α_1 by the sum of spatial relationship scores from all other possible symbols, in increasing order. The first candidate under this ordering is unlikely to be a good choice to follow any symbol in the derivation string, and is therefore likely to be the first symbol. Loosely speaking, this choice corresponds to the top-left symbol in the drawing. It 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.

In the worst case, the algorithm will eventually try all possible matchings. It is therefore guaranteed to obtain a matching f if one exists. Heuristically, the algorithm's best-first search strategy should find a good matching quickly, and we found this to be the case in practice. If no matchings exist, then the algorithm will reject the input as unmatchable after exhausting all possible search paths. The number of search paths varies depending on the input, the derivation string, and the symbol and relational classification results. The theoretical worst case occurs when every symbol appears as a recognition candidate for every subset of strokes and every relation score is non-zero. If there are n strokes in the input and k terminal symbols in the derivation string, then the number of search paths is roughly bounded by k^n. (This bound simply counts the number of assignments of the n strokes to the k symbols.) Note, however, that this theoretical worst case is virtually impossible in

practice. When symbol and relation classification are accurate enough, then the algorithm is linear in k, as it matches each terminal symbol without backtracking.

4.2 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:

1. Running the algorithm on every transcription.
2. Pre-training the symbol recognizer for each user on approximately 20% of their input data, selected randomly, and then running the algorithm on the remaining 80%.

Scenario 2 was introduced because the our model-based symbol recognizer often possessed no models similar to how participants drew certain symbols, causing many transcriptions to be rejected in Scenario 1. This scenario was intended to isolate the ground-truthing algorithm from symbol recognition errors as much as possible.

The symbol recognizer first identifies groups of ink strokes which may correspond to distinct symbols, then generates a fixed number of symbol candidates for each group. These groups may overlap. In each scenario, we varied the number of candidates that were generated for each group to gauge the effect on rejection rate and accuracy.

4.2.1 Measures of accuracy

A natural way to measure the accuracy of our ground-truthing algorithm is to measure the similarity between automatically- and manually- ground-truthed symbol bounding boxes. We evaluated our ground-truthing algorithm under the following three accuracy metrics.

1. *Full expression:* In this measurement, each annotated expression is given a score of 0 or 1. If every symbol in the automatically-annotated expression was matched to the same group of strokes as its manually-annotated counterpart, then the score is 1; otherwise it is 0.
2. *Symbol-based:* In this measurement, each symbol is given a score of 0 or 1. If the automatically-annotated symbol was matched to the same group of strokes as its manually-annotated counterpart, then the score is 1; otherwise it is 0.
3. *Bounding-box overlap:* In this measurement, each symbol is assigned a score between 0 and 1, given by the similarity of the bounding box of the automatically-annotated

14

symbol to that of its manually-generated counterpart. Box similarity is computed as the proportion of the larger box covered by the smaller 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.

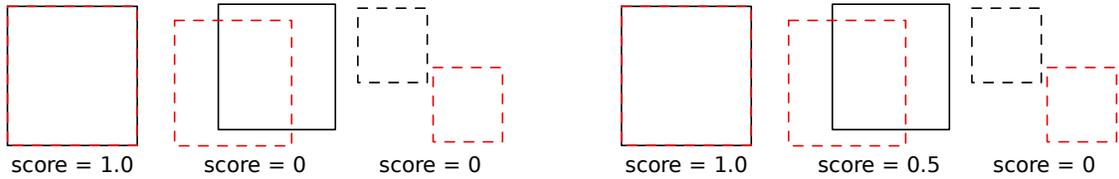Figure 8 illustrates the difference between exact and overlap bounding-box accuracy.

Figure 8: Exact (left) and bounding-box overlap (right) accuracy.

These three measurements are defined in increasing order of permissiveness of match. Full expression accuracy requires every symbol in a transcription to be annotated exactly 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.
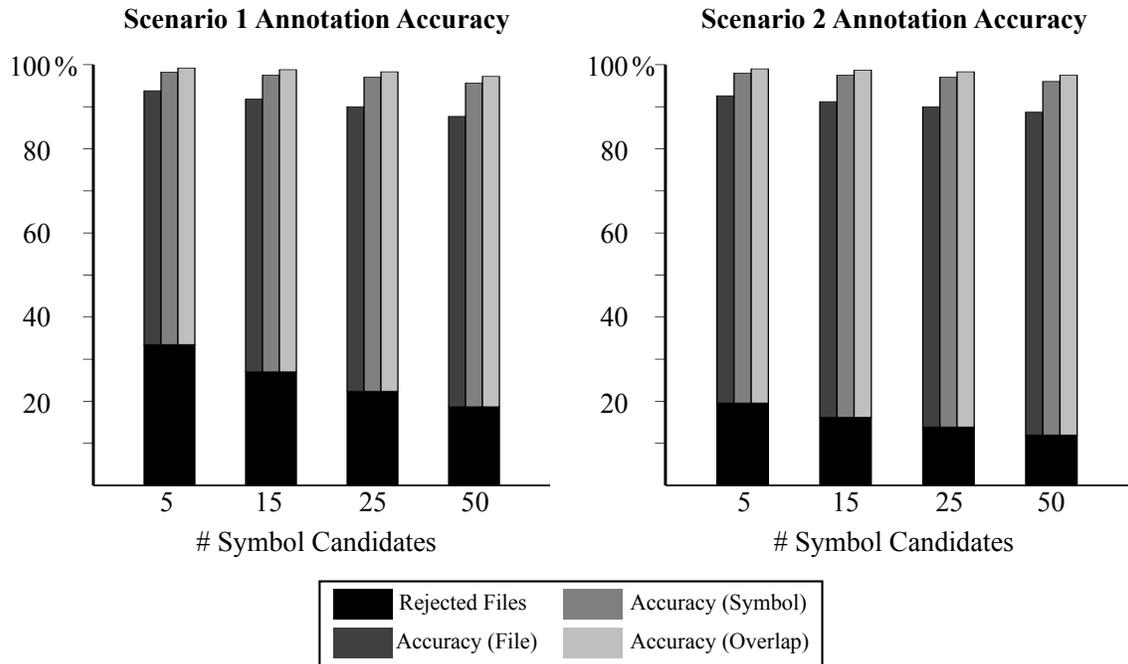

Figure 9: Automatic annotation accuracy

The ground-truthing algorithm produces highly accurate ground-truth for hand-drawn input when compared to manual ground-truth, as shown by Figure 9. In these graphs, the

wide black bars indicate the proportion of transcriptions which were rejected by the ground-truthing algorithm. The narrower bars indicate the labelling accuracy on the non-rejected data under each of the measurement schemes described above. The x-axis values indicate how many recognition candidates were returned from the symbol recognizer for each group of strokes identified as a potential symbol.

The algorithm rejects fewer transcriptions as the number of available symbol candidates increases. This behaviour is expected, as it becomes more likely that all of the required terminal symbols will be present in the symbol recognizer's output when more candidates are reported. Providing the algorithm with more candidates also reduces its accuracy, but only slightly. For example, in Scenario 1, the rejection rate dropped from about 34% to 19%, while the symbol-level accuracy dropped from 97% to 95%, and overlap accuracy only fell from 99% to 97%.

The main difference between Scenarios 1 and 2 is the rejection rate. Pre-training the symbol recognizer significantly increased the number of transcriptions that the algorithm was able to label, indicating that symbol recognition quality was a main cause of rejection in Scenario 1. The rejection rate for Scenario 2 was about 20% when 8 symbol candidates were used, almost 15% lower than the similar experiment in Scenario 1. This reduction was present in all cases, but became less marked as the number of symbol candidates increased.

The type and degree of accuracy required from ground-truth varies with the intended use of a corpus. 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. If one uses ground-truthed corpus expressions to test the accuracy of an isolated symbol recognition system, then full expression or exact bounding-box accuracy may be a more appropriate measurement.

4.3   Discussion

There are two sources of errors for the labeling algorithm: rejection, and inaccurate labeling. Inaccuracy is a more serious error than rejection – since we would like to use the algorithm's output as ground-truth, it is imperative that the output is as accurate as possible. We prefer to have no labeling at all than an incorrect labeling.

Rejection is only possible when symbol recognition fails to identify all of the required symbols, or the relational classifiers assign zero confidence to the required symbol arrangements. Figure 10 shows examples of each of these cases. The transcription on the left was rejected because the symbol recognizer failed to identify the "J" symbol. The transcription

on the right was rejected because of the simplification used in the algorithm. The relational classifiers did not detect a relation between the "+" and the numerator "x" in the denominator of the top-level fraction. In our experiments, symbol recognition was the dominant cause of rejection.

$$-\left[\int_{(z)}^{r}\int y\,dJ\,d\int\right] \qquad \frac{x}{a+\dfrac{x}{b-\dfrac{x}{c}}}$$

Figure 10: Inputs rejected by the labeling algorithm.

Incorrect labelings occured almost exclusively on transcriptions containing symbols drawn with a different number of strokes than the symbol recognizer expected.In such a situation, the symbol recognizer will never output the mismatched symbol as a candidate for the correct group of strokes. For example, in Figure 11, the right parenthesis is split into two strokes. The manual ground-truth groups these strokes together as the parenthesis, but the labeling algorithm simply picked the single stroke with the higher recognition score. Cases like this one illustrate the complexity of judging whether ground-truth is "correct", and point out the utility of the overlap accuracy metric, which counts this labeling as about 95% correct.

$$\mu^{M_m}(x)$$

Figure 11: An incorrect labeling resulting from a divided symbol.

Using an off-line, image-based symbol recognizer could help to avoid this problem. While we currently focus on creating corpora for online mathematics, our labeling technique is also applicable to the off-line domain. Given accurate ground truth for scanned hand-written mathematical expressions, symbols could be segmented from the background, recognized using an ICR engine that returns an n-best list, and then matched using our algorithm.

A more significant theoretical source of inaccuracy is when the derivation string does not match the actual structure of the transcription. This may happen in cases where there are multiple ways of writing an expression, and the transcriber chose a way different from the one appearing in the automatically-generated derivation string (e.g. if a template expression $\frac{1}{2}$ was transcribed horizontally as 1/2). In these cases, if the labeler finds a matching, then the ground-truth will be incorrect because it does not reflect the actual spatial relationships

appearing in the transcription. In our study, this type of inaccuracy was very rare, probably because mathematics has a fairly standardized syntax. In more flexible domains, though, it could be a more significant problem.

5 Conclusions and future work

The sketch recognition community currently lacks publicly-available, ground-truthed corpora for training and testing recognition systems. In this paper, we advocate the use of automated techniques to aid in the creation of such corpora. We presented a grammar-based approach for generating sketch templates, which may then be manually transcribed at minimal cost to construct large sketch corpora. We also presented an automatic ground-truthing method working in conjunction with our generation technique. These techniques were applied during the creation of our ground-truthed corpus of hand-drawn mathematical expressions. The corpus is available at http://www.cs.uwaterloo.ca/scg/mathbrush/mathdata.shtml.

We are not aware of any work on grammar-based algorithms to aid in the generation and labeling of large on-line sketch corpora. However, 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 (e.g. [7], [14]), or match real inputs to separate ground-truth created by hand, potentially with mistakes in both matching and ground-truthing (e.g. [1]). Occasionally aspects of both approaches are combined as in [8].

Our techniques for generating and labeling corpus data have much in common with both approaches, but also have some important differences. Synthetic data is generated as in the first approach, but this data is intended to be transcribed by human users. Real input is matched to separate ground-truth, but the ground-truth is automatically generated and is free from errors. By decoupling expression generation from ground-truth generation, we are free to experiment with algorithms for each task separately.

While the template generation algorithm presented in Section 3 worked well for generating syntactically-correct mathematical expressions, it is difficult to argue that these expressions are truly representative of the expressions commonly used by working mathematicians or students. To generate more realistic expressions, we intend to undertake a study of the notations and normative rules commonly used in mathematical writing. Using this information, we plan to augment our grammar with probability distributions to generate more realistic corpora focusing on particular mathematical domains.

The ground-truthing algorithm presented in Section 4 was very accurate, but always

rejected over 10% of its inputs. We wish to reduce this rejection rate further. One way to do so is by modifying our collection protocol. Pre-training the recognizer proved to be an effective way to reduce the rejection rate, but since we trained on randomly-selected samples, the reduction was likely far from optimal. In future studies, we will ask participants to transcribe a number of specially-designed expressions intended to capture their writing style for each relevant symbol, and then set these transcriptions aside specifically for pre-training.

Another way to reduce the rejection rate is to change the labeling algorithm. The simplification we used to obtain a tractable labeling algorithm was a significant, but not primary, cause for rejection. One way to mitigate this problem is to use relational classifiers specifically trained to detect relations between a subexpression and the first symbol in the next subexpression. Another way is to include more of the two-dimensional structure of handwritten math in the labeling algorithm. We intend to investigate both of these possibilities, while still keeping the labeler tractable.

Acknowledgements

References

[1] 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.

[2] 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.

[3] Horst Bunke, *Recognition of cursive roman handwriting - past, present and future*, ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition (Washington, DC, USA), IEEE Computer Society, 2003, p. 448.

[4] Kam-Fai Chan and Dit-Yan Yeung, *Error detection, error correction and performance evaluation in on-line mathematical expression recognition*, in On-Line Mathematical Expression Recognition,âĂİ Pattern Recognition, 1999.

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

[6] Utpal Garain and B. Chaudhuri, *A corpus for ocr research on mathematical expressions*, Int. J. Doc. Anal. Recognit. **7** (2005), no. 4, 241–259.

[7] 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.

[8] 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.

[9] 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).

[10] 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.

[11] 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.

[12] 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.

[13] Joan Mas, Joaquim A. Jorge, Gemma Sánchez, and Josep Lladós, *Representing and parsing sketched symbols using adjacency grammars and a grid-directed parser*, GREC, 2007, pp. 169–180.

[14] 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.

[15] 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.

[16] 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.

[17] R. Zanibbi, D. Blostein, and J.R. Cordy, *Recognizing mathematical expressions using tree transformation*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **24** (2002), no. 11, 1455–1467.