Recognizing handwritten mathematics via fuzzy parsing

Scott MacLean
smaclean@uwaterloo.ca

George Labahn
glabahn@uwaterloo.ca

David R. Cheriton School of Computer Science, University of Waterloo
Technical Report CS-2010-13

June 12, 2010

Abstract

We present a new approach to multi-dimensional parsing using relational grammars and fuzzy sets. A fast, incremental parsing algorithm is developed, motivated by the two-dimensional structure of written mathematics. Our approach makes no hard decisions; recognized math expressions are reported to the user in ranked order. A flexible correction mechanism enables users to quickly choose the correct math expression in case of recognition errors or ambiguity.

1 Introduction

It is generally acknowledged that the primary methods by which people input mathematics on computers (e.g., LaTeX, Maple, Mathematica) are unintuitive and error-prone. A more natural method, at least on pen-based devices, is to use handwritten input. However, automatic recognition of handwritten mathematical expressions is a hard problem: even after forty years of research, no existing recognition system is able to accurately recognize a wide range of mathematical notation.

There are many similarities between math notation and other natural languages [4]. In particular, notations are not formally defined and can be ambiguous. For example, without contextual information, it is impossible to tell whether the notation $u(x + y)$ denotes a function application or a multiplication operation. Such semantic ambiguities, along with the syntactic ambiguities generally associated with handwriting recognition, make math notation a challenging recognition domain. These difficulties are compounded by the two-dimensional structures prevalent in handwritten math. Many well-known approaches for recognition and domain modeling (e.g., Markov models, grammars, dictionary lookup) cannot be directly applied to the more complicated structures found in math notation.

In this paper, we present a new formalism, called *fuzzy relational context-free grammars (fuzzy r-CFGs)*, for recognition problems in structured, ambiguous domains, and develop the requisite theory (Section 2) and algorithms (Sections 3 and 4) for practical two-dimensional

parsing. The definition of a fuzzy r-CFG incorporates not only the structure of the recognition domain, but also the uncertainty inherent in recognizing that structure. The grammar thus provides a formal model of the recognition process itself, in which the "reasonableness" of a particular interpretation of the user's handwriting is easily calculated. Such calculations permit a parser to make meaningful judgements about whether one interpretation is better than another.

Grammar-based approaches have been used to recognize mathematical notation since Blackwell and Anderson's original proposal [3]. They are occasionally used through hard-coded rule-based approaches, as in AlgoSketch [13] and its relatives, which use an approach similar to those of Zanibbi [24] and Rutherford [20]. More typically, grammars are used either as a verification step to confirm the validity of an expression recognized by some other means (e.g., [9, 21]), or, more interestingly in our case, as a flexible rule-based framework guiding the recognition process. Such approaches succeed, with varying degrees of generality and efficiency, at modeling two-dimensional syntax ([19, 2]), and may unify symbol and structural recognition problems ([22, 1]), or handle multiple ambiguous parses ([8, 25]). Our fuzzy r-CFG approach endeavours to provide a systematic framework for parsing ambiguous input by simultaneously modeling input, domain syntax, and relevant recognition processes.

Microsoft's Math Input Panel, included with its Windows 7 operating system, also provides a math recognition engine. However, the only extant publications on this recognizer are a patent application [5] and a non-technical talk [18], in which it was mentioned that the recognizer uses grammar-based techniques. With so few details available, it is difficult to evaluate the merits and deficiencies of Microsoft's recognizer in general.

In such an ambiguous domain as handwritten mathematics, it is unlikely that a recognizer will always correctly recognize users' writing. Using fuzzy r-CFGs, we propose instead to *preserve* the ambiguity inherent in the user's writing so that one may select the intended interpretation from a number of reasonable recognition results. Of course, more reasonable interpretations should be presented before less reasonable interpretations. We view recognition for complex, ambiguous domains as a co-operative process in which the user takes an active role to help resolve ambiguity, similarly to how one may need to restate or clarify ideas during a discussion between people.

This co-operative recognition process requires a feedback loop between the system and the user. If writing is recognized incorrectly, or affords more than one valid interpretation, then the user must be able to quickly and easily select the desired result. We wish to provide feedback to the user in real-time, and to allow corrections to be made at any time. Consequently, our parsing algorithms must be fast, incremental (i.e., supporting the insertion and removal of input elements as the user writes or erases), and adaptive to user feedback. Corrections made by the user must be maintained as he or she continues to write.

It is well-known that two-dimensional parsing is intractible in general. In order to develop efficient algorithms, we introduce two formal assumptions on the relations in our grammar. The *ordering assumption* (Sec. 3) defines the structure of physical layouts considered feasible for parsing, and the *monotone assumption* (Sec. 4) limits context-sensitivity so that our fuzzy set constructions are neatly decomposable. These assumptions limit both the number and the complexity of admissible parses so that parse results can be reported to the user in real-time.

This fuzzy r-CFG approach to parsing handwritten mathematics was developed for the

math recognition component of MathBrush, our pen-based system for interactive mathematics [11]. The system allows users to write mathematical expressions as they would using a pen and paper, and to edit and manipulate the expressions using computer algebra system operations that are invoked by pen-based interaction. As it is designed for real-time interaction rather than batch recognition, the MathBrush interface provides a useful platform for evaluating the utility of our parsing framework.

2 Fuzzy relational context-free grammars

Recognition may generally be seen as a process by which an observed, ambiguous, input is interpreted as a certain, structured expression. Fuzzy r-CFGs explicitly model this interpretation process as a fuzzy relation between concrete inputs and abstract expressions. The formalism therefore captures not only the idealized, abstract syntax of domain objects (as with a typical context-free grammar), but also the ambiguity inherent in the recognition process itself. In this section, we define fuzzy r-CFGs, give examples of their use, and describe fuzzy parsing in an abstract setting.

2.1 Definition

Recall that 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 *fuzzy relation* on X is a fuzzy set $(X \times X, \mu)$. The notions of set union, intersection, Cartesian product, etc. can be similarly extended to fuzzy sets. For details, refer to Zadeh [23]. To denote the grade of membership of a in a fuzzy set $\tilde{X}$, we will write $\tilde{X}(a)$ rather than referring explicitly to the name of the membership function, which is typically left unspecified. By $x \in \tilde{X} = (X, \mu)$, we mean $\mu(x) > 0$, and by $|\tilde{X}|$ we mean the number of elements having non-zero membership grade, rather than the sum of membership grades over $x \in X$.

Fuzzy relational context-free grammars are formally defined as follows:

Definition 1. *A* fuzzy relational context-free grammar G *is a tuple* $(\Sigma, N, S, T, R, r_\Sigma, P)$*, where:*

- Σ *is a set of terminal symbols;*
- N *is a set of nonterminal symbols;*
- $S \in N$ *is the* start symbol*;*
- T *is a set of observables;*
- R *is a set of fuzzy relations on* I*, where* I *is the set of interpretations, defined below;*
- r_Σ *is a fuzzy relation on* (T, Σ)*; and*
- P *is a set of productions, each of the form* $A_0 \overset{r}{\Rightarrow} A_1 A_2 \cdots A_k$*, where* $A_0 \in N, r \in R$*, and* $A_1, \ldots, A_k \in N \cup \Sigma$*.*

The form of an fuzzy r-CFG is generally similar to that of a traditional context-free grammar. We point out the differences below.

2.1.1 Observables

The set T of *observables* represents the set of all possible concrete inputs. Formally, T must be closed under union and intersection. In practice, for online recognition problems, an observable $t \in T$ is a set of ink strokes, each tracing out a particular curve in the (x, y) plane.

2.1.2 Set of interpretations

While typical context-free grammars deal with *strings*, we call the objects derivable from fuzzy r-CFGs *expressions*. Any terminal symbol $\alpha \in \Sigma$ is an expression. An expression e may also be formed by concatenating a number of expressions $e_1, \ldots, e_k$ by a relation $r \in R$. Such an *r-concatenation* is written $(e_1 r e_2 r \cdots r e_k)$.

The *representable set of* G is the set E of all expressions derivable from the nonterminals in N using productions in P. It may be constructed inductively as follows:

For each terminal $\alpha \in \Sigma$,

$$E_\alpha = \{\alpha\}.$$

For each production p of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$,

$$E_p = \{(e_1 r \cdots r e_k) : e_i \in E_{A_i}\}.$$

For each nonterminal A,

$$E_A = \bigcup_{p \in P \text{ having LHS } A} E_p;$$

and finally,

$$E = \bigcup_{A \in N} E_A.$$

The set of interpretations, then, is $I = T \times E$, where the observables may be *interpreted* as grammar expressions by the recognition process.

2.1.3 Relations

The set R contains fuzzy relations that give structure to expressions by modeling the relationships between subexpressions. These relations act on interpretations, allowing context-sensitive statements to be made about recognition in an otherwise context-free setting.

For example, consider Figure 1, which may be best interpreted as A^p or AP depending upon the case of the P symbol. Denote the A symbol by t_1 and the P symbol by t_2. Let $\nearrow \in R$ be a fuzzy relation for diagonal spatial relationships, and $\rightarrow$ be similar for horizontal adjacency relationships. Then we expect that $\nearrow\ ((t_1, A), (t_2, p)) > \nearrow\ ((t_1, A), (t_2, P))$ and $\rightarrow\ ((t_1, A), (t_2, P)) > \nearrow\ ((t_1, A), (t_2, P))$. Given explicit membership functions, we could evaluate whether or not these expectations are met.

Figure 1: An expression in which the optimal relation depends on symbol identity.

2.1.4 Terminal relation

The relation r_Σ models the relationship between observables and terminal symbols. In practice, it may be derived from the output of a symbol recognizer. For example, if a subset t' of the input observable was recognized as the letter b with, say, 60% confidence, then one could take $r_\Sigma((t', b)) = 0.6$.

2.1.5 Productions

The productions in P are similar to context-free grammar productions in that the left-hand element derives the sequence of right-hand elements. 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. Formally, given a production $A_0 \overset{r}{\Rightarrow} A_1 A_2 \cdots A_k$, if t_i denotes an observable interpretable as an expression e_i derivable from A_i (i.e., $e_i \in E_{A_i}$), then for $\bigcup_{i=1}^{k} t_i$ to be interpretable as $(e_1\, r \cdots r\, e_k)$ requires $((t_i, e_i), (t_{i+1}, e_{i+1})) \in r$ for $i = 1, \ldots, k-1$.

2.2 Examples

The following examples demonstrate how fuzzy r-CFG productions may be used to model the structure of mathematical writing. We use five binary spatial relations: $\nearrow$, $\rightarrow$, $\searrow$, $\downarrow$, $\odot$. The arrows indicate a general writing direction, while $\odot$ denotes containment (as in notations like $\sqrt{x}$, for instance).

1. Suppose that `[ADD]` and `[EXPR]` are nonterminals and + is a terminal. Then the production $\texttt{[ADD]} \overset{\rightarrow}{\Rightarrow} \texttt{[EXPR]} + \texttt{[EXPR]}$ models the syntax for infix addition: two expressions joined by the addition symbol, written from left to right.

2. The production $\texttt{[SUP]} \overset{\nearrow}{\Rightarrow} \texttt{[EXPR]}\texttt{[EXPR]}$ models superscript syntax. Interpreted as exponentiation, the first RHS token represents the base of the exponentiation, and the second represents the exponent. The tokens are connected by the $\nearrow$ relation, reflecting the expected spatial relationship between subexpressions.

3. The following pair of productions models the syntax of definite integration:

$$\texttt{[ILIMITS]} \overset{\downarrow}{\Rightarrow} \texttt{[EXPR]} \int \texttt{[EXPR]}$$

$$\texttt{[INTEGRAL]} \overset{\rightarrow}{\Rightarrow} \texttt{[ILIMITS]}\texttt{[EXPR]}\, d\, \texttt{[VAR]}$$

   Definite integrals are drawn using two writing directions. The limits of integration and integration sign itself are written in a vertical stack, while the integration sign, integrand, and variable of integration are written from left to right.

2.3 Parsing with fuzzy r-CFGs

Because of ambiguity, there are usually several expressions which are reasonable interpretations of a particular input observable $t \in T$ (e.g., A^p and AP are both reasonable interpretations of Figure 1). We represent all of the interpretations of t as a fuzzy set I_t of expressions. The membership function of I_t gives the extent to which the structure of an expression matches the structure of t, as measured by r_Σ and the other grammar relations. This set can be thought of as a "slice" of the fuzzy recognition relation discussed above. More specifically, calling the recognition relation $\mathbf{R}$, define $I_t = \{e : (t, e) \in \mathbf{R}\}$, and $I_t(e) = \mathbf{R}(t, e)$.

For cleaner notation, assume that the grammar productions are in a normal form such that each production is either of the form $A_0 \Rightarrow \alpha$, where $\alpha \in \Sigma$ is a terminal symbol, or of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$, where all of the A_i are nonterminals. This normal form is easily realized by, for each $\alpha \in \Sigma$, introducing a new nonterminal X_α, replacing all instances of α in existing productions by X_α, and adding the production $X_\alpha \Rightarrow \alpha$.

The set I_t of interpretations of t is then constructed as follows. For every terminal production p of the form $A_0 \Rightarrow \alpha$, define a fuzzy set $I_t^p = \{\alpha\}$, where $I_t^p(\alpha) = r_\Sigma((t, \alpha))$ and $I_t^p(\beta) = 0$ for $\beta \neq \alpha$.

For every production p of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$, define a fuzzy set (taking the union over all partitions of t)

$$I_t^p = \bigcup_{t_1 \cup \cdots \cup t_k = t} I_{t_1,\ldots,t_k}^p, \tag{1}$$

where

$$I_{t_1,\ldots,t_k}^p = \left\{(e_1 r \cdots r e_k) : e_i \in I_{t_i}^{A_i},\ ((t_i, e_i), (t_{i+1}, e_{i+1})) \in r, i = 1, \ldots, k-1\right\}. \tag{2}$$

There is room for experimentation when choosing the membership function of $I_{t_1,\ldots,t_k}^p$. Zhang et al [25] found that using multiplication when computing membership grades helped to prevent ties. We might therefore compute the membership grade $I_{t_1,\ldots,t_k}^p(e_1 r \cdots r e_k)$ by

$$\sqrt{\left(\prod_{i=1}^{k} I_{t_i}^{A_i}(e_i)\right)^{\frac{1}{k}} \left(\prod_{i=1}^{k-1} r\left((t_i, e_i), (t_{i+1}, e_{i+1})\right)\right)^{\frac{1}{k-1}}}. \tag{3}$$

Geometric averaging preserves the tie-breaking properties of multiplication while normalizing for expression size. An alternative approach is to treat expression concatenation as a type of fuzzy Cartesian product and to put

$$I_{t_1,\ldots,t_k}^p(e_1 r \cdots r e_k) = \min\left(\min_i \left\{I_{t_i}^{A_i}(e_i)\right\}, \min_i \left\{r\left((t_i, e_i), (t_{i+1}, e_{i+1})\right)\right\}\right). \tag{4}$$

We compare the effects of each of these choices in Section 6.

Given all of the I_t^p, the fuzzy set of interpretations for a particular non-terminal $A \in N$ is

$$I_t^A = \bigcup_{p \text{ having LHS } A} I_t^p,$$

and the fuzzy set of interpretations of an observable t is $I_t = I_t^S$, where S is the start symbol.

An expression e is in I_t iff t is interpretable as e and e is derivable from the start symbol S. The recognition problem is therefore equivalent to the extraction of elements of I_t (t being the user's input) in decreasing order of membership grade.

2.4 Representing I_t efficiently

In Equation 2, each set $I^p_{t_1,\ldots,t_k}$ contains $\mathcal{O}\left(\prod_i \left|I^{A_i}_{t_i}\right|\right)$ expressions. We therefore cannot hope to explicitly construct the sets of interpretations. Instead, we use a compact graph-based representation that completely captures the structure of the all feasible interpretations without constructing them. From this structure, we construct particular interpretations and report them to the user one at a time.

Treating $N \times T$ as a set of nodes, define $B(A,t)$ for $A \in N, t \in T$, the set of *branches at* (A,t) by

$$B(A,t) = \left\{(p;(t_1,\ldots,t_k)) : \left|I^p_{t_1,\ldots,t_k}\right| > 0\right\}.$$

Each set of branches $B(A,t)$ represents I^A_t. Each branch $(p;(t_1,\ldots,t_k)) \in B(A,t)$ represents $I^p_{t_1,\ldots,t_k}$. If p is $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$, then to obtain an expression $(e_1 r \cdots r e_k) \in I^p_{t_1,\ldots,t_k}$ from the corresponding branch requires the recursive extraction of each e_i from the branch set $B(A_i,t_i)$. But in terms of space requirements, $B(A,t)$ contains at most one entry per production and partition, thus avoiding the combinatorial explosion associated with a naive representation of I^A_t. All $\mathcal{O}\left(\prod_i \left|I^{A_i}_{t_i}\right|\right)$ potential parses mentioned above are represented implicitly by a single branch. Each branch stores pointers to k branch sets $B(A_i,t_i)$, which each hold implicitly-represented subexpressions.

This representation of I_t motivates a two-stage parsing process. The first stage, parsing, examines the input and uses grammar productions and relations to construct this branching structure B. However, no parsed expressions are explicitly created. The second stage, extraction, follows links in B to extract particular parsed expressions. The next two sections describe these algorithms in detail. But first, we demonstrate these ideas more concretely through an example.

2.4.1 Example

Treating the elements of $B(A_0,t)$ as k graph edges from (A_0,t) to $(A_1,t_1),\ldots,(A_k,t_k)$, B can be thought of as a graph in which all parse trees on t are overlaid on the same set of vertices. For example, consider the expression shown in Figure 2 along with the following toy grammar:

$$\begin{aligned}
\texttt{[EXPR]} &\Rightarrow \texttt{[ADD]} \mid \texttt{[MUL]} \mid \texttt{[VAR]} \\
\texttt{[ADD]} &\overset{\rightarrow}{\Rightarrow} \texttt{[VAR]} + \texttt{[ADD]} \mid \texttt{[VAR]} \\
\texttt{[MUL]} &\overset{\rightarrow}{\Rightarrow} \texttt{[VAR]}\,\texttt{[MUL]} \mid \texttt{[VAR]} \\
\texttt{[VAR]} &\Rightarrow a \mid b \mid \cdots \mid z
\end{aligned}$$

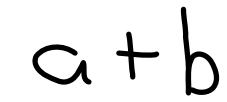

Figure 2: Example handwritten expression.

Suppose that, after applying r_Σ and the $\rightarrow$ relation, there are only two parses of this expression, corresponding to the derivations of $a + b$,

$$\begin{aligned} \texttt{[EXPR]} &\Rightarrow \texttt{[ADD]} \Rightarrow (\texttt{[VAR]} \rightarrow + \rightarrow \texttt{[ADD]}) \\ &\Rightarrow (a \rightarrow + \rightarrow \texttt{[VAR]}) \\ &\Rightarrow (a \rightarrow + \rightarrow b)\,, \end{aligned}$$

and of atb,

$$\begin{aligned} \texttt{[EXPR]} &\Rightarrow \texttt{[MUL]} \Rightarrow (\texttt{[VAR]} \rightarrow \texttt{[MUL]}) \\ &\Rightarrow (a \rightarrow (\texttt{[VAR]} \rightarrow \texttt{[MUL]})) \\ &\Rightarrow (a \rightarrow (t \rightarrow \texttt{[VAR]})) \\ &\Rightarrow (a \rightarrow (t \rightarrow b))\,. \end{aligned}$$

A graphical representation of the branching structure that represents these derivations is shown in Figure 3. In the figure, the arrows indicate derivation. The square boxes at the top represent parts of the observable input. The ovals represent nonterminals on a particular subset of the input (the sets $B(A, t)$), and the small boxes indicate subexpressions that must be taken together as a group. Each group must satisfy the grammar relation indicated inside the box. A valid parse corresponds to a selection of arrows that forms a branching path from the root to all of the leaf nodes. In this case, exactly two such branchings are possible, corresponding to the above derivations of the expressions $a + b$ and atb.

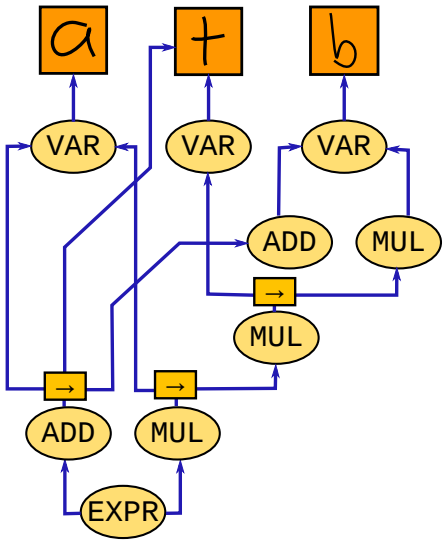

Figure 3: Parse graph corresponding to the expression in Figure 2.

3 Practical algorithms for parsing fuzzy r-CFGs

We now demonstrate how to efficiently construct the graphical representation B of the fuzzy set I_t of interpretations for an input observable t. Recall from Equation 2 that I_t^p is

constructed from a union taken over all partitions of t. It is intractible to take this union literally, so we will first develop precise constraints on which partitions should be considered feasible. These constraints are based on the two-dimensional structure of mathematical notation.

3.1 The ordering assumption and rectangular sets

Informally, we consider the situation in which each grammar relation r contains relatively few elements, so that only a small number of partitions of t can pass the relation membership tests in Equations 3 and 4, as follows.

Define two total orders on observables: $<_x$ orders observables by minimum x-coordinate from left to right, and $<_y$ orders observables by minimum y-coordinate from top to bottom. (We take the y axis to be oriented downward.) Associate each relation $r \in R$ with one of these orders, denoted $\operatorname{ord} r$. $\operatorname{ord} r$ is the dominant writing direction used for a particular relation. For math recognition, we use $\operatorname{ord} \rightarrow = \operatorname{ord} \nearrow = \operatorname{ord} \searrow = \operatorname{ord} \odot = <_x$, and $\operatorname{ord} \downarrow = <_y$.

Denote by $\min_d t$ the element $a \in t$ such that $a <_d b$ for all $b \in t$ aside from a, and define $\max_d t$ similarly.

Assumption 1 (Ordering)**.** *Let t_1, t_2 be observables, and let e_1, e_2 be representable expressions. We assume that $r((t_1, e_1), (t_2, e_2)) = 0$ whenever $\max_{\operatorname{ord} r} t_1 \geq_{\operatorname{ord} r} \min_{\operatorname{ord} r} t_2$.*

Roughly speaking, the ordering assumption says that, for a parse to exist on $t_1 \cup t_2$, the last symbol of t_1 must begin before the first symbol of t_2 along the dominant writing direction of the expression being parsed. For example, in Figure 2, to parse $a + b$ in the obvious way requires that the a begins before the $+$ and the $+$ begins before the b when the symbols are considered from left to right (i.e., ordered by $<_x$).

Similarly, we could formulate a production for fractions as $\texttt{[FRAC]} \overset{\downarrow}{\Rightarrow} \texttt{[EXPR]}\text{—}\texttt{[EXPR]}$. Then to parse a fraction would require that the bottom symbol of the numerator began before the fraction bar, and the fraction bar began before the top symbol of the denominator, when considered from top to bottom (i.e., ordered by $<_y$).

Liang et al [14] proposed *rectangular hulls* as a subset-selection constraint for two-dimensional parsing. A very similar constraint that we call *rectangular sets* is implied by the ordering assumption.

Definition 2 (Rectangular set/partition)**.** *Call a subset t' of t* rectangular in t *if it satisfies $t' = \{a \in t : \min_x t' \leq a \leq \max_x t'\} \cap \{a \in t : \min_y t' \leq a \leq \max_y t'\}$. Call a partition $t_1 \cup \cdots \cup t_k$ of a rectangular set t* rectangular *if every t_i is rectangular in t.*

Proposition 1. *Let $t \in T$ be an observable, and let p be a production of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$. Under the ordering assumption, if $(e_1 r \cdots r e_k) \in I^p_{t_1,\ldots,t_k}$, then the partition $t_1 \cup \cdots \cup t_k$ of t is rectangular.*

Proof. Let $d = \operatorname{ord} r$, and choose any t_i. We must show that

$$t_i = \left\{a \in t : \min_x t_i \leq_x a \leq_x \max_x t_i\right\} \cap \left\{a \in t : \min_y t_i \leq_y a \leq_y \max_y t_i\right\}.$$

It is clear that t_i is a subset of the RHS, so suppose that there is some $a' \in t$ in the RHS put into $t_j \neq t_i$ by the partition of t. If $j < i$, then $((t_j, e_j), (t_{j+1}, e_{j+1})), \ldots, ((t_{i-1}, e_{i-1}), (t_i, e_i)) \in r$. By the assumption, $\max_d t_j <_d \min_d t_{j+1} \leq \max_d t_{j+1} <_d \cdots <_d \min_d t_i$. But we know $\min_d t_j \leq_d a' \leq_d \max_d t_j$ (since $a' \in t_j$) and $\min_d t_i \leq_d a' \leq_d \max_d t_i$ (since a' is in the RHS), so $\min_d t_i \leq_d a' \leq_d \max_d t_j$, a contradiction. A similar contradiction can be obtained in the case where $j > i$. □

Rectangular sets are the natural two-dimensional generalization of consecutive characters in one-dimensional string parsing. This definition could be generalized to arbitrary dimension, giving "hypercube sets" of input elements.

Following Liang et al, notice that any rectangular set $u \subseteq t$ can be constructed by choosing any four input elements in t and including in u all input elements in their rectangular hull. There are therefore $\mathcal{O}(|t|^4)$ rectangular subsets of t. The ordering assumption thus yields a substantial reduction in the number of subsets that must be considered for parsing.

Unlike the formulation of rectangular hulls used by Liang et al, we do not prohibit region bounding boxes from cutting through other symbols. This provision allows for square roots and other containment notations, and is more accommodating of somewhat cramped or sloppy writing in which symbol bounding boxes overlap, as in Figure 4.

Figure 4: Expressions with overlapping symbol bounding boxes.

3.2 Parsing algorithm

Using the restriction to rectangular partitions derived above, it is straightforward to develop a bottom-up parsing algorithm that constructs B in provably polynomial time. An example of such an algorithm is given in Appendix A. However, that algorithm requires the explicit enumeration of every rectangular subset of the input, so it is not suitable for incremental recognition. Whenever the user adds or removes a stroke, many of the subsets must be re-computed before the parse graph may be updated. This precomputation step is too slow for the real-time reporting of parse results that is required for interactive systems like MathBrush. To eliminate the brute-force enumeration of all rectangular subsets, we developed a top-down parsing algorithm for fuzzy r-CFGs.

Assume now that the grammar is in the normal form described in Section 2.3. A straightforward recursive-descent approach to parsing a production p on an observable t might work as follows:

1. If p is a terminal production, $A_0 \Rightarrow \alpha$, then check if $(t, \alpha) \in r_\Sigma$. If so, the parse succeeds; otherwise it fails.

2. Otherwise, p is of the form $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$. For every rectangular partition $t_1 \cup \cdots \cup t_k$ of t, parse each nonterminal A_i on t_i. If any of the sub-parses fail, then fail on the current partition. If parsing fails on every partition, then fail.

Parsing a nonterminal A amounts to parsing each production p such that A is the LHS element of p. Note that each t_i in case 2 has $|t_i| < |t|$, guaranteeing termination.

Case 2 iterates over $\mathcal{O}\left(\binom{n}{k-1}\right)$ partitions, where $n = |t|$. This bound is obtained by sorting the input by $<_{\text{ord}\, r}$ and choosing $k-1$ split points to induce a rectangular partition. A tradeoff therefore exists between the number k of RHS elements of a grammar production and the number of partitions that must be considered during parsing. If the grammar is in Chomsky Normal Form (CNF), then only $n-1$ partitions need to be considered. But a production with k RHS tokens expands to $k-1$ distinct productions in an equivalent CNF grammar.

Instead of using CNF, we allow arbitrary production lengths and use the following three optimizations to reduce the number of partitions that must be considered. Note that we no longer require the normal form used by the recursive-descent algorithm: productions may freely combine nonterminal and terminal symbols.

1. *Spatial relation test.* The grammar relations act on expressions as well as observables. As such, they cannot be tested during parsing because expressions are not explictly constructed. Still, we can speed up parsing by testing whether relations are satisfied which approximate the grammar relations.

   Namely, for each relation $r \in R$, we test the relation

   $$\hat{r}(t_1, t_2) = \begin{cases} 1 & \text{if } \exists e_1, e_2 \text{ s.t. } ((t_1, e_1), (t_2, e_2)) \in r \\ 0 & \text{otherwise} \end{cases}.$$

   Many relations used in practice are somewhat independent of the expressions e_1, e_2, or use information derived solely from the observables t_1, t_2 (e.g., bounding box measurements). So, it is feasible to approximate the relations $\hat{r}$ quite well.

2. *Terminal symbol milestones.* The terminal symbol relation r_Σ may be used to guide partitioning. Suppose p is $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_{i-1} \alpha A_{i+1} \cdots A_k$, where α is a terminal symbol. Then r_Σ must contain (t_i, α) for a parse to exist on a partition $t_1 \cup \cdots \cup t_k$. That is, given a partition, any subset corresponding to a terminal symbol in the grammar production must be recognizable as that terminal symbol.

3. *Minimum nonterminal length.* If there are no empty productions in the grammar, then each nonterminal always expands to at least one terminal symbol. Moreover, given a minimum number of strokes required to recognize a particular symbol (e.g., at least two strokes may be required to form an F), one can compute the minimal number of input elements required to parse any sequence of nonterminals $A_1 \cdots A_k$, denoted $minlen(A_1 \cdots A_k)$. This function further constrains the partitions which are feasible.

   For example, consider the example expression shown in Figure 2. Suppose we are parsing the production `[ADD]` $\overset{\rightarrow}{\Rightarrow}$ `[VAR]` + `[ADD]` and we know that the + symbol must be drawn with exactly two strokes. Then `[ADD]` cannot be parsed on fewer than 4 input strokes, and the input must be partitioned into $t_1 \cup t_2 \cup t_3$ such that $|t_1| \geq 1, |t_2| = 2, |t_3| \geq 1$. Furthermore, from the previous optimization, t_2 must be chosen so that $(t_2, +) \in r_\Sigma$. In this particular case, only one partition is feasible.

Parsing a production $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$ on an observable t proceeds by two mutually recursive procedures.

The first procedure, PARSE-NT-SEQ, parses a sequence $A_1 \cdots A_k$ of nonterminals on an observable t, as follows:

1. If $k = 1$, then parse the nonterminal A_1 on t. Fail if this sub-parse fails.

2. Otherwise, for every rectangular partition of t into two subsets, t_1 and t_2, such that $|t_1| \geq minlen(A_1)$ and $|t_2| \geq minlen(A_2 \cdots A_k)$, parse A_1 on t_1, and recursively parse the nonterminal sequence $A_2 \cdots A_k$ on t_2. Fail if either parse fails.

The second procedure, PARSE-SEQ, parses a general sequence $A_1 \cdots A_k$ of nonterminals and terminals on t, as follows. Let $d = \operatorname{ord} r$, and let i be minimal such that A_i is a terminal symbol. Then each rectangular $t' \subseteq t$ such that $(t', A_i) \in r_\Sigma$ induces a rectangular partition of t into t_1, t', t_2, where $\max_d t_1 <_d \min_d t'$ and $\max_d t' <_d \min_d t_2$. For each of these partitions satisfying $|t_1| \geq minlen(A_1 \cdots A_{i-1})$ and $|t_2| \geq minlen(A_{i+1} \cdots A_k)$, parse $A_1 \cdots A_{i-1}$ on t_1, and parse $A_{i+1} \cdots A_k$ on t_2. Fail if either sub-parse fails.

So, to parse a production, we just check whether its RHS contains terminals or not, and invoke the appropriate procedure. To clarify the outline of the algorithm, we have omitted the relational tests described in optimization 1. These details are included in the pseudocode of Algorithms 1 through 3.

Algorithm 1 PARSE-NT-SEQ: Top-down parser for nonterminal sequences

Require: A sequence $A_1 \cdots A_k$ of nonterminal symbols, a set t of input elements, a grammar relation r, a leading set t_L, and a trailing set t_R.
  $B' \leftarrow \phi$
  $d \leftarrow \operatorname{ord} r$
  if $k = 1$ **then**
    // Verify that t fits in via $\hat{r}$ with the leading and trailing sets.
    if $((t_L, t) \in \hat{r}$ **or** $t_L = \phi)$ **and** $((t, t_R) \in \hat{r}$ **or** $t_R = \phi)$ **then**
      return PARSE(A_1, t)
  for $L = minlen(A_1), \ldots, |t| - minlen(A_2 \cdots A_k)$ **do**
    Let $t_1 \subseteq t$ be the first L elements of t ordered by $<_d$.
    if $(t_L, t_1) \in \hat{r}$ **or** $t_L = \phi$ **then**
      $B_1 \leftarrow$ PARSE(A_1, t_1)
      if $B_1 \neq \phi$ **then**
        $B_2 \leftarrow$ PARSE-NT-SEQ$(A_2 \cdots A_k, t \setminus t_1, r, t_1, t_R)$
        $B' \leftarrow B' \cup \{b_1, b_2 : b_1 \in B_1, \ldots, b_2 \in B_2\}$
  return B'

Algorithm 2 PARSE-SEQ: Top-down parser for general production sequences

Require: A sequence $A_1 \cdots A_k$ of grammar symbols from $N \cup \Sigma$, a set t of input elements, a grammar relation r, and a leading set t_L.
  Let i be minimal such that A_i is a terminal symbol
  $B' \leftarrow \phi$
  $d \leftarrow \operatorname{ord} r$
  for every $t' \subseteq t$ such that $(t', A_i) \in r_\Sigma$ **do**
    $t_1 \leftarrow \{a \in t : a <_d \min_d t'\}$
    $t_2 \leftarrow \{a \in t : \max_d t' <_d a\}$
    if $minlen(A_1 \cdots A_{i-1}) \leq |t_1|$ **and** $minlen(A_{i+1} \cdots A_k) \leq |t_2|$ **then**
      $B_1 \leftarrow$ PARSE-NT-SEQ$(A_1 \cdots A_{i-1}, t_1, r, t_L, t')$
      if at least one of $A_{i+1}, \ldots, A_k$ is a terminal symbol **then**
        $B_2 \leftarrow$ PARSE-SEQ$(A_{i+1} \cdots A_k, t_2, r, t')$
      else
        $B_2 \leftarrow$ PARSE-NT-SEQ$(A_{k+1} \cdots A_k, t_2, r, t', \phi)$
      $B' \leftarrow B' \cup \{b_1, t', b_2 : b_1 \in B_1, b_2 \in B_2\}$
  return B'

Algorithm 3 PARSE: Top-down parser entry point

Require: A nonterminal A_0 and a set t of input elements.
  if $B(A_0, t)$ is not marked "parsed" **then**
    for each production p of the form $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$ **do**
      if at least one of $A_1, \ldots, A_k$ is a terminal symbol **then**
        $B' \leftarrow$ PARSE-SEQ$(A_1 \cdots A_k, t, r, \phi)$
      else
        $B' \leftarrow$ PARSE-NT-SEQ$(A_1 \cdots A_k, t, r, \phi, \phi)$
    Mark $B(A_0, t)$ "parsed"
    $B(A_0, t) \leftarrow \{(p; x) : x \in B'\}$
  return B'

These functions comprise the two-dimensional parsing algorithm that we use in practice for MathBrush. Each production $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$ is parsed recursively by parsing $A_2 \cdots A_k$ for each valid subset choice for A_1, aborting a recursive branch when no valid subset choices exist. This approach, along with the recursive-descent algorithm, may be seen as a depth-first search of the space of valid partitions. But, the optimizations introduce constraints on the search space, significantly speeding up the parsing process. The technique is fast enough that recognition results may be updated and displayed in real-time as the user is writing.

3.3 Incremental parsing

The above parsing algorithms may be used incrementally without any significant changes. Suppose that parsing is complete for some observable $t = \{a_1, \ldots, a_n\}$. We must handle two cases: the addition of a new observable a_{n+1} to t if the user draws a new stroke, and the removal of an observable a_i from t if the user erases a stroke.

In the case where a new observable a_{n+1} is added, every existing entry $B(A,t)$ of the parse graph still remains valid. We may simply invoke the parser on the new input $\{a_1,\ldots,a_{n+1}\}$ and all existing parse results will be re-used.

In the case where an existing observable a_i is removed, the situation is slightly more complicated. Any entry $B(A,t)$ such that t includes a_i becomes invalid and must be removed from the parse graph. When the parser is invoked on $\{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n\}$, any existing parse results that do not include the stroke a_i will be re-used.

This approach to incremental parsing works particularly well when subsets of the input are represented by bit vectors. Each input element is represented by a bit, where the first stroke drawn corresponds to the lowest-order bit and the most recent stroke to the highest-order bit. If a bit is set, then the corresponding input element is included in the subset, otherwise it is not.

Using this representation, when the first stroke is drawn, the parser might create an entry $B(A,\mathtt{1})$ in the parse graph. After a second stroke is drawn, the bit vectors representing subsets will contain two bits. But since the low-order bit corresponds to the first stroke, accessing $B(A,\mathtt{01})$ is the same as accessing the entry $B(A,\mathtt{1})$ that was created when the input was just one stroke.

4 Extracting interpretations

Recall that recognition in the context of fuzzy r-CFGs corresponds to extracting elements from I_t, the fuzzy set of interpretations of the user's writing. We wish to report these interpretations to the user in decreasing order of membership grade so that more reasonable interpretations are seen before less reasonable ones. In this section, we develop a straightforward technique for extracting interpretations in that order.

4.1 The monotone assumption

Denote the element of a fuzzy set X with nth-highest membership grade by $[n]X$, so that the highest-ranked interpretation of the user's input is $[0]I_t$, the second-highest is $[1]I_t$, and so on.

Consider the problem of finding $[0]I^p_{t_1,\ldots,t_k}$, the highest-ranked parse of a production p using the partition $t_1\cup\cdots\cup t_k$ of t. Every expression $e\in I^p_{t_1,\ldots,t_k}$ is an r-concatenation of the form $(e_1 r\cdots r e_k)$, where each e_i is $[m_i]I^{A_i}_{t_i}$ for some m_i. The grade of membership in $I^p_{t_1,\ldots,t_k}$ of each such expression is a function of the subexpression membership grades $I^{A_i}_{t_i}(e_i)$, and the relation memberships $r((t_i,e_i),(t_{i+1},e_{i+1}))$. Since we have no information about how the relation's membership function varies with the e_i (hence with the m_i), the only way to determine $[0]I^p_t$ is to iterate over all possible m_i, evaluate the relation membership functions, combine grades into a membership grade in $I^p_{t_1,\ldots,t_k}$, and take the highest-graded expression overall.

This exhaustive algorithm is impractical. To develop a tractable algorithm, we introduce the *monotone* assumption on the grammar relations, as follows.

Assumption 2 (Monotone)**.** *Let t_1,t_2 be observables, $e_1\in I_{t_1}, e_2\in I_{t_2}$, and $r\in R$. We*

assume that, for any expressions $\hat{e}_1, \hat{e}_2$ such that $I_{t_1}(\hat{e}_1) \leq I_{t_1}(e_1)$ and $I_{t_2}(\hat{e}_2) \leq I_{t_2}(e_2)$, we have $r\left((t_1, \hat{e}_1), (t_2, \hat{e}_2)\right) \leq r\left((t_1, e_1), (t_2, e_2)\right)$.

Furthermore, let p be a production $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$, and for $i = 1, \ldots, k$, let $t_i \in T$ and $e_i \in I_{t_i}^{A_i}$. We assume that if $I_{t_i}^{A_i}(\hat{e}_i) \leq I_{t_i}^{A_i}(e_i)$ and $r\left((t_i, \hat{e}_i), (t_{i+1}, \hat{e}_{i+1})\right) \leq r\left((t_i, e_i), (t_{i+1}, e_{i+1})\right)$ for all i, then $I_{t_1,\ldots,t_k}^{p}(\hat{e}_1 r \cdots r \hat{e}_k) \leq I_{t_1,\ldots,t_k}(e_1 r \cdots r e_k)$.

That is, we assume that the membership function of each relation r is monotonically increasing with the membership grades of its expression arguments, and assume that the membership function of each $I_{t_1,\ldots,t_k}^{p}$ is monotonically increasing with all of its constituent parts.

4.2 Extracting elements of I_t

Using the monotone assumption, we develop an algorithm for extracting elements of I_t in decreasing order of membership grade. The basic idea of our algorithm is to first extract the most highly-ranked expression, $[0]\, I_t$, and then to recursively maintain an expanding "frontier" of potential next-best expressions, from which subsequent expressions are obtained.

First, note that $[n]I_t = [n]I_t^S$ for S the start symbol. So it suffices to consider extracting $[n]I_t^{A_0}$ for a nonterminal A_0.

To do so, number the grammar productions having LHS A_0 as $p_1, \ldots, p_N$. It is clear that $[0]I_t^{A_0} = \text{argmax}_i [0]I_t^{p_i}$. Now suppose we have determined $[j]I_t^{A_0}$ up to $j = n$, yielding a partial set of expressions $I = \left\{[j]I_t^{A_0} : j = 0, \ldots, n\right\}$. Then $[n+1]\, I_t^{A_0}$ is given by the following result.

Proposition 2. *Let $I = \bigcup_i \left\{[j]\, I_t^{p_i} : 0 \leq j \leq n_i\right\}$. That is, the $n_i + 1$ most highly-ranked elements of each $I_t^{p_i}$ have already been extracted from $I_t^{A_0}$. Then $[n+1]I_t^{A_0} = \text{argmax}_i [n_i + 1]I_t^{p_i}$.*

Here, the indices n_i represent the expanding frontier of potential next-best expressions. The actual next-best expression is just the best expression from the frontier. After extraction, the frontier must be updated by incrementing the appropriate n_i.

Expressions may be extracted in ranked order from a set I_t^p for a production p just as above, except that the production-wise sets I_t^p replace the nonterminal-wise sets $I_t^{A_0}$, and are replaced by the partition-wise sets $I_{t_1,\ldots,t_k}^{p}$.

However, the idea must be generalized somewhat to extract expressions from the sets $I_{t_1,\ldots,t_k}^{p}$. In what follows, $t_1, \ldots, t_k$ are observables and p is a production $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$. We represent an r-concatenation $\left([n_1]\, I_{t_1}^{A_1} r \cdots r\, [n_k]\, I_{t_k}^{A_k}\right)$ by a vector $(n_1, \ldots, n_k) \in \mathbb{Z}_+^k$ (that is, a vector of k non-negative integers representing subexpression ranks).

This vector representation gives a geometric interpretation of an expression. Under the monotone assumption, expressions are ordered by component-wise comparison of their corresponding vectors.

Lemma 1. *Let $e = (n_1, \ldots, n_k)$ and $\hat{e} = (\hat{n}_1, \ldots, \hat{n}_k)$ be any expressions. If $(\hat{n}_1, \ldots, \hat{n}_k) \leq (n_1, \ldots, n_k)$, then $I_{t_1,\ldots,t_k}^{p}(\hat{e}) \leq I_{t_1,\ldots,t_k}^{p}(e)$, where $x \leq y$ for vectors $x = (x_1, \ldots, x_k), y = (y_1, \ldots, y_k)$ if $x_i \leq y_i$ for all i.*

Proof. By definition, the membership grade in $I_{t_i}^{A_i}$ of $[n_i]\, I_{t_i}^{A_i}$ is at least that of $[\hat{n}_i]\, I_{t_i}^{A_i}$ for every i. So, by the monotone assumption, we have $r\left(\left(t_i, [\hat{n}_i]\, I_{t_i}^{A_i}\right), \left(t_{i+1}, [\hat{n}_{i+1}]\, I_{t_{i+1}}^{A_{i+1}}\right)\right) \leq r\left(\left(t_i, [n_i]\, I_{t_i}^{A_i}\right), \left(t_{i+1}, [n_{i+1}]\, I_{t_{i+1}}^{A_{i+1}}\right)\right)$ for all i. Hence, $I^p_{t_1,\ldots,t_k}(\hat{e}) \leq I^p_{t_1,\ldots,t_k}(e)$. □

So, the monotone assumption implies that we cannot skip expressions by jumping over neighbouring points. The following definition makes this precise.

Definition 3. *Let $e = [n] I_t^p = (m_1, \ldots, m_k)$ for some indices m_i. We define the* successor set of e *to be*

$$succ(e) = \left\{x \in \mathbb{Z}_+^k : x > e, \|x - e\|_1 = 1\right\}.$$

Thus, $succ(e)$ contains all of the expressions obtainable from $I^p_{t_1,\ldots,t_k}$ by incrementing exactly one of the extraction indices m_i of $e = \left([m_1]\, I_{t_1}^{A_1} r \cdots r\, [m_k]\, I_{t_k}^{A_k}\right)$. The frontier of potential next-best expressions is given by the union of the successor sets of every expression already extracted, denoted S below. The following result shows inductively how to extract elements of $I^p_{t_1,\ldots,t_k}$.

Proposition 3. *The following statements hold.*

1. $[0]\, I^p_{t_1,\ldots,t_k} = e_0 = (0, 0, \ldots, 0)$.
2. *Suppose we have determined $[j]\, I^p_{t_1,\ldots,t_k}$ up to $j = n$, yielding a partial set of expressions I. Let $S = \bigcup_{e \in I} succ(e)$. Then*
   (a) $I = \bigcup_{e \in S} \left\{x \in \mathbb{Z}_+^k : x < e\right\}$, *and*
   (b) $[n+1]\, I^p_{t_1,\ldots,t_k} = e_{n+1} = \operatorname{argmax}_{e \in S} I^p_{t_1,\ldots,t_k}(e)$.

Proof. 1. Suppose to the contrary that $[0]\, I^p_{t_1,\ldots,t_k} = e' = (m_1, \ldots, m_k)$ with $I^p_{t_1,\ldots,t_k}(e') > I^p_{t_1,\ldots,t_k}(e_0)$. This immediately contradicts Lemma 1 since $(m_1, \ldots, m_k) \geq (0, \ldots, 0)$.

2. Both statements are true after extracting only $e_0 = (0, \ldots, 0)$. Proceeding inductively, we first prove 2a. Let $x < e$ for some $e \in S$. By Lemma 1, we have $I^p_{t_1,\ldots,t_k}(x) \leq I^p_{t_1,\ldots,t_k}(e)$. So, subject to the condition that, in case of ties, we choose the expression whose vector is closest to the origin, x must have already been extracted and thus is already in I.

   Now we prove 2b by showing that $[n+1]\, I^p_{t_1,\ldots,t_k}$ is in S. Suppose to the contrary that $[n+1]\, I^p_{t_1,\ldots,t_k} = e' = (m_1, \ldots, m_k)$ with $e' \notin S$ and $I^p_{t_1,\ldots,t_k}(e') > I^p_{t_1,\ldots,t_k}(e_{n+1})$.

   Since $e' \notin I$, we have by 2a that there is some $x \in I$ with $e' > x$. But S contains all expressions having minimal distance to I, so there must also be some $y \in S$ with $e' > y$. Then by Lemma 1, $I^p_{t_1,\ldots,t_k}(y) \geq I^p_{t_1,\ldots,t_k}(e')$. So we may safely choose an expression from S to be $[n+1]\, I^p_{t_1,\ldots,t_k}$, namely e_{n+1}.

□

These results demonstrate how to extract interpretations of the user's writing from I_t by decreasing grade of membership. Appendix B gives a detailed pseudocode implementation using priority queues.

4.3 Handling user corrections

As mentioned in the introduction, we wish to provide to users a simple correction mechanism so that they may select their desired interpretation in case of recognition errors or multiple ambiguous interpretations. Our recognizer facilitates such corrections by allowing *locks* to be set on any subset of the input.

Two types of locks are supported:

1. *Expression locks*, which fix the interpretation of a particular subset of the input to be a specified expression, and

2. *Semantic locks*, which force interpretations of a particular subset of the input to be derived from a specified nonterminal.

Both of these lock types are useful in different circumstances. For example, if $x + a$ was recognized as $X + a$, then an expression lock may be applied to the portion of the input recognized as the X symbol, forcing it to be interpreted as a lower-case x instead. If $x + a$ was recognized as xta, then a semantic lock may be applied to the entire input, forcing an addition expression to be derived. If recognition gave Xta, then the lock types may be combined.

Consider extracting an expression tree from input t. If t is locked to an expression e by an expression lock, then we consider I_t to contain only one element, e, with membership grade 1. If t is locked to a nonterminal A_L by a semantic lock, then we take $I_t^{A'}$ to be empty for all nonterminals $A' \neq A_L$, except those for which a derivation sequence $A' \Rightarrow^* A_L$ exists, in which case we take $I_t^{A'} = I_t^{A_L}$.

MathBrush allows users to navigate ranked interpretations on any subset of their input, as shown in Figure 5. In case recognition is not accurate, the user may select the correct interpretation and an appropriate lock is applied to the parse graph. The corrections are maintained in subsequent parse results as the user continues to write. In this way, correction is simple and relatively painless – certainly easier than erasing the expression and rewriting it from scratch.

4.4 Weakening the monotone assumption

The monotone assumption essentially prohibits context-sensitivity in the grammar relations in the sense that any interactions between subexpressions are ignored. As such, it is similar to assuming statistical independence of subexpressions, a common practice in stochastic parsing. In some cases, such assumptions are invalid – consider again Figure 1, which was interpreted best as AP or A^p depending on the case of the P.

Intuitively, the identities of terminal symbols should have the largest effect on the degree to which grammar relations apply, and we expect relation membership grades to stabilize as expression size increases. For example, consider how terminal identity changes the mathematical semantics in the Ax (or A^X) subexpression of Figure 6, but, regardless or whether the writer intended Ax or A^X, the subexpressions Ax (say), $+$, and b of Figure 6 are certainly linked by the $\rightarrow$ relation.

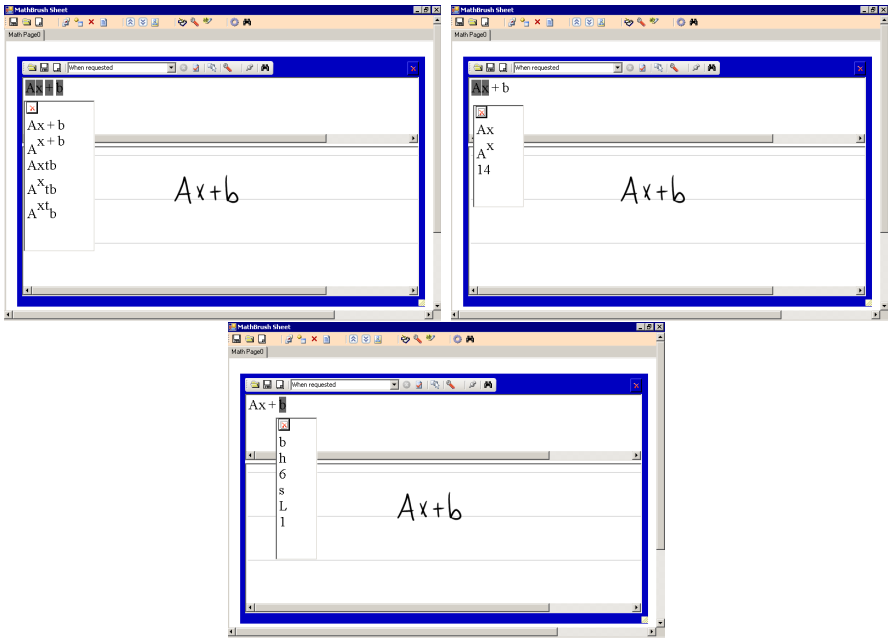

Figure 5: Interface for displaying and selecting alternative interpretations.

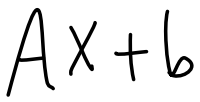

Figure 6: An ambiguous mathematical expression.

In practice, terminal symbols are not subject to the monotone assumption in our parser. Instead, we use context-sensitive relations for terminals that vary depending on expected symbol shapes (e.g., baseline, ascender, descender, etc.), as follows. The relations are described briefly in Section 5.

Let $t_1, \ldots, t_k$ be observables and p be a production $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$. Suppose that, when extracting $[0]\, I^p_{t_1,\ldots,t_k}$, it happens that the subexpressions $[0]\, I^{A_i}_{t_i}$ are lone terminal expressions for each i in some index set $I = \{i_1, \ldots, i_m\}$. Then, instead of using the approach indicated by Lemma 3, we explicitly consider all possible terminal combinations by constructing the partial set of expressions

$$\left\{ \left([m_1]\, I^{A_1}_{t_1} r \cdots r\, [m_k]\, I^{A_k}_{t_k} \right) : 0 \leq m_i < M_i \right\},$$

where $M_i = \left| I^{A_i}_{t_i} \right|$ for $i \in I$, and $M_i = 1$ otherwise. Considering all of these expressions takes advantage of any sensitivity to symbol identities that exists in the relation membership functions.

In such a case, it may take time up to $\mathcal{O}\left(\prod_i \left| I^{A_i}_{t_i} \right|\right)$ to obtain the highest-ranked expression in $I^p_{t_1,\ldots,t_k}$. But the number of symbol candidates assigned a non-zero membership grade in r_Σ for each subset of the input is easily capped at a small constant, and the symbols obtainable from $I^{A_i}_{t_i}$ are restricted by the fact that they must be derivable from A_i, so this exponential runtime is not too significant.

5 Relation membership functions

Our relation membership functions are based on bounding-box geometry. Since each symbol has a characteristic general shape with respect to a writing baseline, the relative positions of subexpressions depends on which symbols comprise them. The membership functions take this fact into account.

Following Rutherford [20], we assign *relational classes* to an expression which describe its expected bounding-box profile relative to the baseline. We extend Rutherford's classes with three special classes, AGGREGATE, SYMBOL, and BOX, representing any expression, any terminal symbol, and any multi-symbol expression, respectively. We also allow each terminal symbol to act as its own relational class.

In order to compute the grade of membership of $((t_1, e_1), (t_2, e_2))$ in a relation r, aside from the containment relation $\odot$, the membership function selects a particular point in the bounding box of each t_i as its centroid, depending on the class of e_i. For example, the centroid is chosen relatively higher on descender symbols and relatively lower on ascender symbols. It is chosen centrally for those classes for which no detailed information is available, such as AGGREGATE. Then, the angle and distance between centroids is measured. From these measurements, the membership function computes two scores by measuring the deviation of the measured values from an "ideal" measurement estimated from training data. The membership grade is given by the product of these scores. To measure the containment relation we compute the amount of overlap between the two bounding boxes.

For any expression, the applicable relational classes can be ordered from most- to least-specific. For example, the terminal expression y could be considered as a "y" symbol, then as a descender symbol, then as a generic terminal expression, then as a generic expression. The expression x^2 could be considered as a multi-symbol expression, then as a generic expression.

Because of the large number of combinations, some pairs of relational classes may not have been trained on sufficient data to provide a reasonable degree of confidence in the results. In such cases, the membership function automatically falls back to a less-specific selection of classes to avoid assigning artificially low membership grades to reasonable-looking expressions.

Some symbols may have more than one possible bounding-box profile. For example, the z symbol may be written as a small symbol centered on the baseline, or as a descender. In such cases, the membership function returns the maximum grade taken over all class assignments.

6 Evaluation

We evaluated the accuracy of our math recognizer experimentally using a ground-truthed corpus of hand-drawn math expressions described in previous work [16, 15]. Of the roughly 4500 legible expressions in the corpus, many included mathematical notations not currently supported by the grammar used by our parser for recognition (e.g., ellipses, set notation, multi-symbol variable names). Our test set thus contained 3610 expressions from 20 writers, of which 53 expressions were common to all writers, and the remainder were unique to each writer.

Devising objective metrics for evaluating the real-world accuracy of a recognition system

is difficult. Several authors have proposed accuracy metrics which are quite specific to the particular math recognizer being measured (e.g. [24, 7, 12, 10, 20]). Our recognizer was developed for the MathBrush pen-based mathematics system, and is intended for real-time, interactive use by human writers. As such, we believe that a user-oriented accuracy model provides the best way to assess its performance.

In particular, our goal is to measure the amount of effort a user must expend to get the system to do what they want. This is similar to Brooks' proposal that the "ratio of function to conceptual complexity is the ultimate test of system design" [6]. For example, suppose a user draws Figure 6, meaning the expression $A^X + b$, but the first result returned by the recognizer is $A^x tb$. In our system, the user can correct the result to an addition, $A^x + b$, and then correct the x to X. We can therefore count two corrections, or two "units of effort". An alternative measurement is to count the total number of results that the user sees while making those corrections. So if the second candidate after x was y, and the third was X, then that single correction would count as two units.

This scheme has the side-effect of measuring recognition accuracy. If an input is recognized correctly, then it counts as zero units of effort. Similarly, if it is recognized "almost correctly", it counts as fewer units than if the recognition is quite poor. Furthermore, the effort-based metric is generally applicable to any recognition system, though it clearly is intended to be used with systems providing some correction or feedback mechanism. One could similarly navigate the recognition alternatives provided by Microsoft's recognizer, for instance, count the relevant operations, and obtain comparable measurements. Our evaluation scheme thus provides an abstract way to compare the performance of varied recognition systems without direct reference to their implementation details.

To automate the evaluation process, we developed a testing program that simulates a user interacting with the recognizer. The program passes ink representing a math expression to the recognizer. Upon receiving the recognition results, the program compares them to the ground-truth associated with the ink. If the highest-ranked result is not correct, then the testing system makes corrections to the recognition results, as a user would, to obtain the correct interpretation. That is, the system browses through lists of alternative interpretations for subexpressions or symbols, searching for corrections matching the symbols and structures in the ground-truth.

This approach immediately yields four possible classifications for each test input:

1. *Attainable*: The correct interpretation was found by the testing system.
2. *Recognizable*: Interpretations existed, but the correct interpretation was not found.
3. *Unrecognizable*: The input did not generate any interpretations (it was rejected by the recognizer).
4. *Infeasible*: The symbol recognizer failed to provide the correct symbol candidates to the parser, preventing correct recognition.

In case of attainability, the amount of effort required to attain the correct interpretation must be measured. To do so, we recorded, for each input, the number of *views* and *corrections* made before the correct interpretation was found. Views counted how many alternatives had

to be browsed through (or "viewed") before the correct interpretation was found. Corrections counted how many corrections had to be made. The corrections figure was further subdivided into symbol-level corrections (e.g., replace X by x), and structural corrections (e.g., replace $A^x tb$ by $A^x + b$).

Our symbol recognizer uses a simple elastic matching variant [17] trained on the handwriting of a single writer. As such, its performance is mediocre on many of the corpus examples, since different writers draw symbols in different ways. Many inputs were thus classified as infeasible.

For the present work, we are primarily interested in evaluating the quality of our recognizer's structural analysis, not its symbol classification. To this end, we introduced a special testing mode called "*Perfect*", which isolates the parser from symbol recognition errors. In this mode, the parser receives the correct identity of each symbol from a dummy symbol recognizer. There are no alternatives to choose between, and no ambiguities in stroke grouping. The mode represents a "best possible world" for the parser and gives an upper bound on its real-world performance.

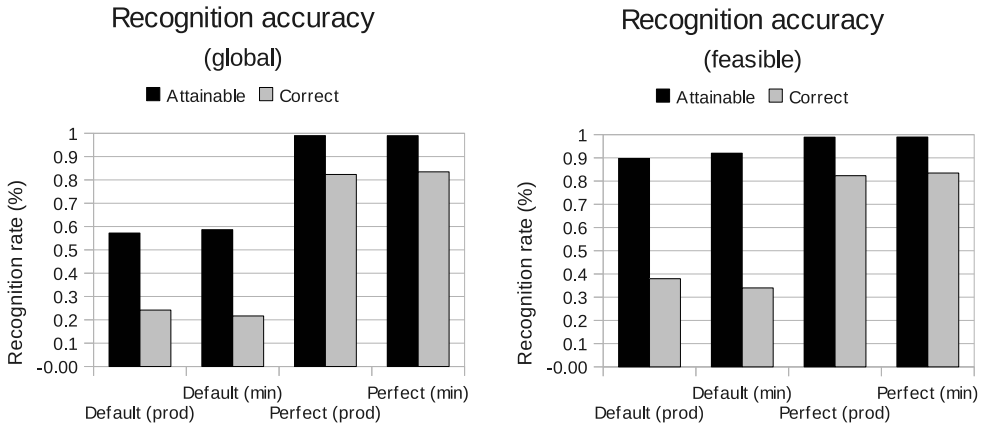

Figure 7: Recognition accuracy in aggregate.

Figure 7 shows two graphs of recognition accuracy aggregated over the entire expression corpus. The graphs show two recognition rates: "Attainable" indicates the proportion of input expressions for which the correct recognition result was obtained, while "Correct" indicates the proportion of expressions for which the top-ranked interpretation matched the ground-truth expression. These rates are indicated for both the product-based (Eqn. 3) and min-based (Eqn. 4) membership functions.

The graph marked "global" considers all available input expressions. The graph marked "feasible" considers only those expressions for which it was possible for the parser to find the correct interpretation (i.e., those for which the correct symbol identities were available to the parser).

The graphs shown in Figure 8 indicate the average number of corrections and views required to obtain the desired interpretation in each of these scenarios. As the number of corrections is irrelevant when the desired expression is unattainable, these graphs only count corrections on attainable expressions.

From the graphs, some conclusions may be drawn:

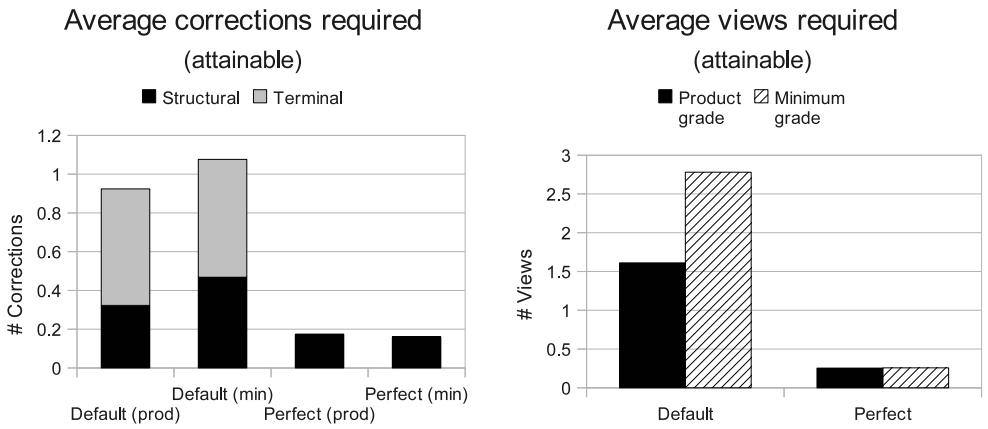

Figure 8: Number of corrections required, in aggregate.

1. In the global scenario, as expected, the recognition rate for the *Perfect* scenario (99%) was higher than for the *Default* scenario (57%).

2. When correct symbol recognition results were available, the parser was able to obtain the desired interpretation 90% of the time in the *Default* scenario.

3. There is a large gap between the attainable and perfect recognition rates.

4. When the desired interpretation was attainable, but was not the first interpretation reported, it was obtained after fewer than one correction on average, and after viewing only about 1.6 individual recognition results (with the product-based membership grade). Intuitively, we may interpret this as meaning, "when the desired interpretation was not the first interpretation reported, it was usually easy to find."

When given only a single, correct, subdivision of the input into terminal symbols, our parser was very accurate. In these cases, it attained the desired interpretation in over 99% of the test inputs. The desired interpretation was the first interpretation reported for roughly 80% of the test inputs. But, when multiple grouping possibilities exist, and symbols may take on multiple identities, as in the *Default* scenario, "Correct" accuracy falls to only about 40%, even if we omit the cases where symbol recognition failed to provide the correct candidates. However, even in these cases with both symbol- and structure-level ambiguity, the desired interpretation was attainable over 90% of the time after only about one correction, on average. Our experiments therefore indicate that, at least for our recognizer, the introduction of ambiguous symbol grouping possibilities and multiple recognition candidates is a major complicating factor in recognizing handwritten math.

The parser's accuracy was similar between the product- and min-based membership grades. However, somewhat more corrections were required with the min-based grade, particularly in the *Default* scenario. This is reasonable since several distinct parses may share a common partition of some subset of the input that has a low membership grade in a relation. All these interpretations may therefore have the same overall membership grade in I_t, so we cannot expect the desired interpretation to always be reported first. In contrast,

Figure 9: Ground-truth can be ambiguous or misleading.

the product-based membership grade preserves ordering across such ties, so we expect more reasonable interpretations to be reported earlier than when using the min-based grade. We use the product-based grade in practice.

Our corpus of hand-drawn math expressions contains some ambiguities in its ground-truth. The ground-truth represents the expression that we asked someone to transcribe. But, their transcription may not precisely match the expression. For example, the transcription on the left of Figure 9 was intended to represent the expression on the right, but the structure is not quite the same. The accuracy figures should therefore be taken as approximations.

To identify the most severe sources of errors in our parser, we analyzed the 36 unattainable inputs from the *Perfect* scenario. These errors can be attributed to two main causes: violation of the ordering assumption (14 inputs), and membership grade zero in grammar relations (22 inputs). Figure 10 shows some example inputs in which the ordering assumption was violated. Figure 11 shows some inputs for which relation classification failed to identify the desired relations.

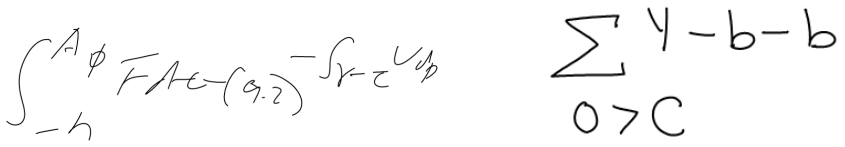

Figure 10: Expressions violating the ordering assumption.

Figure 11: Expressions for which relation classification failed.

It is possible for each of these errors to be caused by mistakes in transcription or by messy writing. For example, in the left-hand expression of Figure 10, the right parenthesis extends to the left beyond the start of the 2 symbol, violating the ordering assumption. The left-hand expression of Figure 11 was intended to contain the subexpressions q_Y and j_X, not qy and jx, as the writer appears to have drawn, leading to (perhaps appropriate) failure of the relational membership functions.

Roughly one third of the unattainable expressions were due to these types of errors or sloppiness in transcription. However, reasonable-looking expressions also led to these errors, as evinced by the right-hand expressions in Figures 10 and 11. In Figure 10, the C symbol

begins to the right of the y symbol. This arrangement violates the ordering assumption and thus prevents the expression from being recognized as a summation. In Figure 11, the subexpression $\sum \Pi - \int Ddy$ was intended to be a superscript of b. Because the exponent is so wide, relational classification failed. These examples point to deficiencies in our approach, for which we propose some solutions in the next section.

7 Conclusions and future work

We have described the motivation, general principles, and core algorithms for our parser for handwritten mathematics. We introduced a new fuzzy r-CFG formalism designed particularly for parsing ambiguous, non-linear input. This formalism captures both the structures and the ambiguities inherent in mathematical writing in particular, and it explicitly models recognition processes like symbol and relation classification. We believe that this formalism is applicable to any structured domain exhibiting the types of syntactic and semantic ambiguities found in natural languages.

The formalization of the recognition domain, including its ambiguity, is an important aspect of our work. By stating working assumptions using this formalism, we were able not only to derive flexible and efficient algorithms, but also to precisely characterize the circumstances in which they are correct, and how our practical algorithms deviate from the theoretical interpretation of fuzzy parsing. We are able to distinguish between recognition errors caused by deficiencies in our grammar model and those caused by an overly-restrictive assumption. Indeed, the experiments in the previous section demonstrated that the fuzzy r-CFG model is useful for handling recognition ambiguities, even though the assumptions we made to obtain tractable algorithms were occasionally violated. An obvious goal for future work is therefore to weaken those assumptions and so broaden the parser's generality.

To facilitate efficient parsing, we introduced the ordering assumption and demonstrated how it leads naturally to the rectangular hulls proposed by Liang et al. While this assumption theoretically allows most mathematical notations to be adequately described, there are many practical examples of mathematical writing which, while easily readable by humans, cannot be parsed by our current algorithm because of the ordering assumption. We intend to investigate how this assumption may be relaxed to more flexibly model expression geometry while still affording efficient parsing algorithms.

To report parse results in ranked order of confidence, we introduced the monotone assumption on the grammar relations, prohibiting context-sensitivity. Although we have extended our basic algorithm to support context sensitivity in the case of relations involving terminal symbols, the extension is somewhat ad-hoc and uses a brute force method. It cannot be generalized to larger subexpressions without significantly degrading the performance of our parser. We therefore plan to investigate less strict assumptions on the behaviour of grammar relations which will allow a greater degree of context-sensitivity without a significant performance penalty.

The fuzzy membership grade functions defined in Section 2, while reasonable, cannot naturally account for some types of information that might be useful during recognition. (For example, measured subexpression co-occurence frequencies.). We plan to investigate the application of Bayesian probability theory to the two-dimensional parsing problem. Be-

cause the terminal symbols are ambiguous in handwritten input, one cannot simply extend existing stochastic CFG parsing algorithms to two dimensions. Many of the ideas underlying our algorithms should be similar between the fuzzy and Bayesian approaches, and a formal probabilistic model would provide an alternative mathematical framework in which to derive confidence scores for interpretations. More generally, we are looking into efficient computational methods for combining variegated information (e.g., probabilistic, possibilistic, rule-based, etc.) during parsing.

Finally, our algorithms must be made more scalable. While they do address the complexity issues arising in our parsing scheme, they take a fairly brute-force approach within the constraints allowed by the simplifying assumptions. As such, there is a considerable decrease in parsing speed as the input becomes large. We plan to investigate ways in which efficiency can be improved while keeping available all relevant recognition results.

Weakening our working assumptions is a challenging and interesting problem. The techniques described in this paper have led to a useful and extensible parser for handwritten mathematics, and the fuzzy r-CFG formalism is flexible enough that many extensions and variants of parsing techniques may be "plugged in" to the same general architecture. We may freely vary the algorithms for building and manipulating the parse graphs using different working assumptions, and take the combination most suitable for the particular recognition domain of interest.

References

[1] Ahmad-Montaser Awal, Harold Mouchere, and Christian Viard-Gaudin, *Towards handwritten mathematical expression recognition*, Document Analysis and Recognition, International Conference on (Los Alamitos, CA, USA), IEEE Computer Society, 2009, pp. 1046–1050.

[2] Josef B. Baker, Alan P. Sexton, and Volker Sorge, *Faithful mathematical formula recognition from pdf documents*, Proc., Ninth IAPR Int'l. Workshop on Document Analysis Systems, 2010, pp. 485–492.

[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] Dorothea Blostein, *Math-literate computers*, Calculemus/MKM (Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, eds.), Lecture Notes in Computer Science, vol. 5625, Springer, 2009, pp. 2–13.

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

[6] Frederick P. Brooks, *The mythical man-month*, Addison-Wesley, 1975.

[7] 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, Pattern Recognition, 1999.

[8] J.A. Fitzgerald, F. Geiselbrechtinger, and T. Kechadi, *Mathpad: A fuzzy logic-based recognition system for handwritten mathematics*, Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on, vol. 2, Sept. 2007, pp. 694–698.

[9] 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.

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

[11] 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), 599–606.

[12] 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.

[13] Chuanjun Li, Timothy S. Miller, Robert C. Zeleznik, and Joseph J. LaViola Jr., *Algosketch: Algorithm sketching and interactive computation*, Proc., Sketch-Based Interfaces and Modeling, 2008.

[14] Percy Liang, Mukund Narasimhan, Michael Shilman, and Paul Viola, *Efficient geometric algorithms for parsing in two dimensions*, ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition (Washington, DC, USA), IEEE Computer Society, 2005, pp. 1172–1177.

[15] S. MacLean, G. Labahn, E. Lank, M. Marzouk, and D. Tausky, *Grammar-based techniques for creating ground-truthed sketch corpora*, Int'l. J. Document Analysis and Recognition. (2010).

[16] Scott MacLean, *Tools for the efficient generation of hand-drawn corpora based on context-free grammars*, Third Int'l. Workshop on Pen-Based Math. Comp., http://www.orcca.on.ca/conferences/cicm09/workshops/PenMath/ programme-hand.html, 2009.

[17] Scott MacLean and George Labahn, *Elastic matching in linear time and constant space*, Proc., Ninth IAPR Int'l. Workshop on Document Analysis Systems, 2010, (Short paper), pp. 551–554.

[18] Marko Panic, *Math handwriting recognition in windows 7 and its benefits*, Proc., Calculemus, 2009, (Invited talk), pp. 29–30.

[19] 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.

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

[21] Seiichi Toyota, Seiichi Uchida, and Masakazu Suzuki, *Structural analysis of mathematical formulae with verification based on formula description grammar*, Document Analysis Systems VII, 2006, pp. 153–163.

[22] R. Yamamoto, S. Sako, T. Nishimoto, and S. Sagayama, *On-line recognition of handwritten mathematical expression based on stroke-based stochastic context-free grammar*, The Tenth International Workshop on Frontiers in Handwriting Recognition, 2006.

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

[24] 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.

[25] Ling Zhang, D. Blostein, and R. Zanibbi, *Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions*, Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on, 2005, pp. 972–976 Vol. 2.

A Bottom-up fuzzy r-CFG parsing

Using the restriction to rectangular partitions derived in Section 3, we develop a straightforward bottom-up parsing algorithm for fuzzy r-CFGs that runs in provably polynomial time. This algorithm examines an input observable t and constructs the parse graph B described in Section 2.

Algorithm 4 generalizes to two dimensions the well-known CYK algorithm for CFG parsing. It uses dynamic programming to parse each rectangular subset of the input from smallest to largest. As in the CYK algorithm, the grammar is assumed to be in Chomsky Normal Form. That is, each production is either of the form $A_0 \Rightarrow \alpha$ for some $\alpha \in \Sigma$, or of the form $A_0 \stackrel{r}{\Rightarrow} A_1 A_2$, where $r \in R$ and $B, C \in N$.

Algorithm 4 Bottom-up fuzzy r-CFG parser

Require: An input observable t.
  for every rectangular subset $t' \subset t$ of size $|t'| = 1, 2, \ldots, |t|$ **do**
    for each production p of the form $A \Rightarrow \alpha$ **do**
      if $(t', \alpha) \in r_\Sigma$ **then**
        $B(A, t') \leftarrow B(A, t') \cup \{(p; (t'))\}$
    for each production p of the form $A_0 \overset{r}{\Rightarrow} A_1 A_2$ **do**
      $d \leftarrow \operatorname{ord} r$
      for $x \in (t' \setminus \{\min_d t'\})$ **do**
        $t_1 \leftarrow \{a \in t' : a <_d x\}$
        $t_2 \leftarrow \{a \in t' : a \geq_d x\}$
        if $B(A_1, t_1) \neq \phi$ and $B(A_2, t_2) \neq \phi$ **then**
          $B(A, t') \leftarrow B(A, t') \cup \{(p; (t_1, t_2))\}$

Any rectangular subset of an observable t may be constructed by following Algorithm 5. We can thus pre-compute all rectangular subsets of t so that they are accessible to Algorithm 4 in constant time. This precomputation step requires $\mathcal{O}(|t|^5 \log |t|)$ operations using standard sorting techniques. The algorithm proper requires $\mathcal{O}(|P||t|)$ operations per rectangular subset, for a total runtime of $\mathcal{O}(|t|^5 (|P| + \log |t|))$.

Algorithm 5 Rectangular subset extraction.

Let d be one of x, y and let d' be the other.
List the elements of t in increasing order under $<_d$.
Extract a contiguous subsequence. (We now have a set $t' \subseteq t$ satisfying $t' = \{a \in t : \min_d t' \leq_d a \leq_d \max_d t'\}$.)
Re-order the remaining elements into increasing order under $<_{d'}$.
Extract a contiguous subsequence. (The subsequence elements comprise a rectangular subset of t.)

B Interpretation extraction algorithm

Given an observable t, we wish to extract interpretations of t from the parse graph B in decreasing order of "reasonableness". We view this problem as the explicit construction of the fuzzy set of interpretations I_t, one member at a time, from its compact, implicit representation in B. It is sufficient to describe how to obtain the most reasonable interpretation, $[0] I_t$, and how to obtain $[n+1] I_t$ given $[n] I_t$.

We divide the problem into two parts: extracting interpretations from a particular branch $(p; (t_1, \ldots, t_k))$, and extracting interpretations from a branch set $B(A, t)$. These two parts are implemented as mutually-recursive procedures invoked according to the structure of the grammar productions.

Algorithms 6 and 8 implement the first part, while Algorithms 7 and 9 implement the second part. The algorithms essentially translate the consequences of the monotone assumption of Section 4 into parse graph operations. Note that each algorithm uses data structures

local to the point in B from which it is extracting expressions. One can think of these algorithms as being associated directly with each node and branch in the parse graph. The process is initialized by calling BEST-NONTERMINAL-EXPRESSION(S, t) with S the start symbol and t the entire input. Similarly, NEXT-NONTERMINAL-EXPRESSION(S, t) may be called repeatedly to iterate over all parses of t.

Furthermore, two distinct modes of extraction are supported. In the EXHAUSTIVE mode, expressions are extracted exactly as suggested by the monotone assumption. However, this leads to an overwhelming number of expressions being available to the user. A second mode, called SEMANTICS, is more restrictive, and is the default mode for expressions larger than a single terminal symbol.

In SEMANTICS mode, all reported expressions must either be derived from different productions (and thus represent different parse structures) different mathematical semantics), or, if derived from the same production, must partition the input differently into subexpressions. This restriction effectively constrains the number of alternatives available, while still allowing all possible parse results to be obtained by examining the alternatives for different subexpressions. It is easily implemented by extracting only one expression per branch.

Algorithm 6 BEST-LINK-EXPRESSION: Extract the most highly-ranked expression from a branch.

Require: A branch $(p; (t_1, \ldots, t_k))$
  $cache \leftarrow \{\}$ // Initialize priority queue local to the branch
  if p is a terminal production $A_0 \stackrel{r}{\Rightarrow} \alpha$ **then**
    $e^* \leftarrow \alpha$
  else
    (p is of the form $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$)
    for $i = 1, \ldots, k$ **do**
      $e_i \leftarrow$ BEST-NONTERMINAL-EXPRESSION(A_i, t_i)
  $e^* \leftarrow (e_1 r \cdots r e_k)$
  $[0]\, I^p_{t_1 \cup \cdots \cup t_k} \leftarrow e^*$
  return e^*

Algorithm 7 NEXT-LINK-EXPRESSION: Extract the next most highly-ranked expression from a branch.

Require: A branch $(p; (t_1, \ldots, t_k))$
  if extraction mode is SEMANTICS **then**
    return NONE
  if p is a terminal production $A_0 \stackrel{r}{\Rightarrow} \alpha$ **then**
    return NONE
  (p is of the form $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$)
  Suppose n expressions have already been extracted from this branch.
  Let $(e_1 r \cdots r e_k) = [n]\, I^p_{t_1 \cup \cdots \cup t_k}$ be the last expression extracted here
  Let m_i be such that $e_i = [m_i]\, I^{A_i}_{t_i}$ for $i = 1, \ldots, l$
  for $i = 1, \ldots, k$ **do**
    if $m_i = |I_{t_i}|$ **then**
      $\hat{e}_i \leftarrow$ NEXT-NONTERMINAL-EXPRESSION(A_i, t_i)
    else
      $\hat{e}_i \leftarrow [m_i + 1]\, I^{A_i}_{t_i}$
    Add $e = (e_1 r \cdots r e_{i-1} r \hat{e}_i r e_{i+1} r \cdots r e_k)$ to *cache* with priority $I^p_t(e)$
  if *cache* is empty **then**
    return NONE
  Pop e^* from *cache*
  $[n+1]\, I^p_{t_1 \cup \cdots \cup t_k} \leftarrow e^*$
  return e^*

Algorithm 8 BEST-NONTERMINAL-EXPRESSION: Extract the most highly-ranked expression derivable from a nonterminal.

Require: A nonterminal A and an observable t.
  cache $\leftarrow \{\}$ // Initialize priority queue local to (A, t)
  for every branch $(p; x) \in B(A, t)$ **do**
    Add $e =$ BEST-LINK-EXPRESSION$(p; x)$ to *cache* with priority $I^p_t(e)$
  Pop e^* from *cache*
  $[0]\, I^A_t \leftarrow e^*$
  return e^*

Algorithm 9 NEXT-NONTERMINAL-EXPRESSION: Extract the next most highly-ranked expression derivable from a nonterminal.

Require: A nonterminal A and an observable t.
  Suppose n expressions have already been extracted from $B(A, t)$.
  Let $e = [n]\, I_t^A$ be the last expression extracted here
  Let $(p; (t_1, \ldots, t_k))$ be the branch from which e was extracted
  Add $\hat{e} =$ NEXT-LINK-EXPRESSION $(p; (t_1, \ldots, t_k))$ to *cache* with priority $I_t^p(e)$
  if *cache* is empty **then**
    return NONE
  Pop e^* from *cache*
  $[n+1]\, I_t^A \leftarrow e^*$
  return e^*

It is difficult to precisely quantify the complexity of these algorithms. However, we can characterize them in terms of the size of B. The initialization step, yielding $[0]I_t$, follows every possible branch in $B(S, t)$ for S the start symbol and t the entire input, visiting each node once. Extracting $[n+1]I_t$ entails a visit to only one branch per nonterminal visited, but to k nonterminals per branch $(p; (t_1, \ldots, t_k))$ visited, where k is the number of RHS tokens in the production p. Note, though, that each such token corresponds to a subexpression in the parse. The amount of work performed in this case is therefore directly proportional to the number of nodes in a parse tree representing $[n]I_t$.

In practice, we must report mathematical expressions to the user not as abstract parse structures, but in some human-readable format. In our system, expressions take two final forms: mathematical expression trees, and strings. To generate these human-understandable formats, each grammar production p is associated with a tree generator and a string generator. The tree generator produces an expression tree that describes how subexpressions 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 generators are defined in an external text file. For example, the following defines the grammar production for addition:

```
ADD_OP :R: [ADD_TERM] + [REL_TERM]
          {ADD(%1 'EXPR_LHS', %3 'EXPR_RHS')}
          `%1 + %3`
```

In this example, on the first line, `ADD_OP` is the name of the production's LHS nonterminal, `ADD_TERM` and `REL_TERM` are two other nonterminals, `+` is the name of a terminal symbol, and `R` is the textual relation code for the $\rightarrow$ 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` that has 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 `ADD_TERM`. The second child is labeled `EXPR_RHS` (the right hand operand) and is linked to the tree output by the generator for `REL_TERM`. 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 ADD_TERM and the right hand operand REL_TERM.

Mathematical semantics are indicated by the root labels produced by tree generators. The production above therefore has semantic type ADD. We call a nonterminal providing a semantic type *labelled*. 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.)

```
[REL_TERM] :: [ADD_OP] | [SUB_OP] | [ADD_TERM]
```

The nonterminal symbol REL_TERM represents a collection of expression types with the same level of precedence – in this case, addition, subtraction, and an isolated addition term. Each of the three nonterminals that the symbol REL_TERM can produce have distinct semantic types. REL_TERM itself does not have a fixed semantic type. Rather, it inherits the expression tree (and hence the semantic type) given by the tree generator for the nonterminal it produces. Unlabelled nonterminals like REL_TERM can therefore represent different semantic types in different contexts. An unlabelled nonterminal may derive other unlabelled nonterminals, so the semantic types it can assume are not always immediately apparent from its productions.