IJDAR
DOI 10.1007/s10032-012-0184-x

ORIGINAL PAPER

A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets

Scott MacLean · George Labahn

Received: 21 July 2010 / Revised: 29 February 2012 / Accepted: 3 March 2012

Abstract We present a new approach for parsing two-dimensional input using relational grammars and fuzzy sets. A fast, incremental parsing algorithm is developed, motivated by the two-dimensional structure of written mathematics. The approach reports all identifiable parses of the input. The parses are represented as a fuzzy set, in which the membership grade of a parse measures the similarity between it and the handwritten input. To identify and report parses efficiently, we adapt and apply existing techniques such as rectangular partitions and shared parse forests, and introduce new ideas such as relational classes and interchangeability. We also present a correction mechanism that allows users to navigate parse results and choose the correct interpretation in case of recognition errors or ambiguity. Such corrections are incorporated into subsequent incremental recognition results. Finally, we include two empirical evaluations of our recognizer. One uses a novel user-oriented correction count metric, while the other replicates the CROHME 2011 math recognition contest. Both evaluations demonstrate the effectiveness of our proposed approach.

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. Indeed, even after forty years of research, no existing recognition system is able to accurately recognize a wide range of mathematical notation [5,9].

There are many similarities between math notation and other natural languages [6]. 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.

The present work describes the math recognition system used in MathBrush, our pen-based system for interactive mathematics [17]. Using MathBrush, a user writes mathematical expressions as if they were using pen and paper. After entry, expressions may be edited and manipulated using computer algebra system (CAS) operations that are invoked by pen-based interaction. The output of CAS operations may be captured and further edited with the pen.

MathBrush is designed for real-time interaction, recognizing and reporting parse results as the user is writing. Its recognizer must therefore be fast enough to update parse results in real-time and flexible enough to support a mix of typeset and handwritten input. However, the ambiguity present in handwritten math makes it all but impossible to develop a recognition system with perfect or even near-perfect accuracy. To ensure that the system remains usable when recognition is imperfect, we believe that it is essential to capture multiple

S. MacLean (✉) · G. Labahn
Cheriton School of Computer Science, University of Waterloo,
200 University Ave. W., Waterloo, ON N2L 3G1, Canada
e-mail: smaclean@cs.uwaterloo.ca

G. Labahn
e-mail: glabahn@cs.uwaterloo.ca

Published online: 20 March 2012

parses simultaneously, and, by so doing, to allow the user to choose between alternatives in case of recognition errors. At the same time, the system must remain highly responsive, so efficiency is vital.

Motivated by these design considerations, this paper presents two main contributions in regard to recognizing handwritten math notation: fuzzy relational context-free grammars (fuzzy r-CFGs) and relational classes. Fuzzy r-CFGs specifically address recognition problems in structured, ambiguous domains. 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 and is presented in detail in Sect. 3.

Relational classes are a generalization and formalization of the syntactic symbol classes used by other authors. They represent families of similar-looking mathematical expressions and replace the typical assumption of independence between parse tree branches. Parses within a relational class are assumed to be interchangeable within a larger parse without affecting the parse confidence. This interchangeability reduces the number of expressions that must be considered when forming parse trees.

Together, these two ideas form the basis for a two-step parsing process that captures all recognizable parses of the input, while avoiding the exponential runtime typically associated with such an approach. In the first step, terminal symbols and potential subexpression structures are identified efficiently using a fuzzy r-CFG and associated parsing algorithms. This step generates a shared parse forest that simultaneously represents every recognizable parse tree. In the second step, particular parse trees are extracted from the shared parse forest in decreasing order of recognition confidence. Each tree represents the syntactic layout of a math expression and is rewritten to represent the expression's semantics. Trees are generated on demand as they are requested by the user through the interface of MathBrush. We use a modest set of relational classes to improve recognition accuracy while maintaining fast tree extraction. Although there may be exponentially many trees, extraction of a single tree remains fast, so that the user experiences no delay.

A secondary contribution of this paper is the correction count metric for evaluating recognition accuracy (Sect. 6). The metric provides an implementation-independent way to compare the effectiveness of recognizers that permit users to make corrections to their output in case of errors. Unlike the expression-level correctness rate that typically serves as a lowest common denominator for recognizer comparison, the correction count metric measures "how correct" a particular recognition result is without reference to the implementation details of any one recognizer.

Grammar-based approaches have been proposed for recognizing mathematical notation for decades—the next section surveys past and current related research. Section 3 presents fuzzy r-CFGs in an abstract setting, while Sect. 4 describes our two-step parsing process in detail. Section 5 provides some details of the symbol and relation classification systems used in MathBrush. Section 6 evaluates our math recognition system empirically on a publicly available data set. Finally, Sect. 7 offers some concluding remarks and future research directions.

2 Related work

There are a large number of existing grammar formalisms addressing the problem of two-dimensional parsing; Marriott, Meyer, and Wittenberg [29] provide an extensive survey. Generally, each formalism strikes a balance between the severity of the geometric constraints it imposes on a parser's input and the amount of time and space required to parse that input. Some formalisms, like general relational grammars and graph rewriting grammars, impose so little constraint that it is intractable to parse the languages they generate. Our work most closely resembles Tomita's 2D-CFGs [36] and the positional grammars developed by Costagliola and others [11].

Both of these formulations fit input elements into a rectangular grid-based structure. In the 2D-CFG case, each grid cell must be filled by a terminal, and adjacent grid regions of the same height (width) may be merged into horizontal (vertical) concatenations by applying grammar productions. However, this formalism is not rich enough to support mathematical notation. For example, consider fitting the symbols of the expression $\frac{a}{b}+1$ into a regular grid using one terminal per cell. The fraction requires three vertically adjacent cells, each of which has a neighbor to the right. But these three neighboring cells cannot be filled by the plus sign alone, and none are allowed to be empty. The expression can therefore not be represented by a 2D-CFG in a natural way.

The positional grammar formalism allows grid cells to be empty, avoiding the problem we just saw with 2D-CFGs. Positional grammar productions take the general form $A \Rightarrow A_1 r_1 A_2 \cdots r_{k-1} A_k$, where the A_i are terminals and/or nonterminals, and the r_i are relations describing spatial configurations. Crucially, given a current grid cell in the input and a grammar relation, the next cell in which parsing continues must be uniquely determined. Positional grammars, thus, require distinct language elements to be parsed by distinct paths through the input. Consider again the example $\frac{a}{b}+1$, which one might naturally parse by following the path $a \downarrow - \downarrow b \rightarrow + \rightarrow 1$, where the arrows indicate grammar relations. But then how would one parse $\frac{a}{b+1}$? After parsing the b, the $\rightarrow$ relation can direct the grammar to move horizontally, as required for the second case, or horizontally and upward, as required for the first case. But it cannot choose

one or the other depending on the circumstances. Again, the formalism is too weak for math notation.

Our work adopts aspects of both of these formalisms. We include multiple geometric relations similarly to positional grammars, but restrict the form of grammar productions, so that a single production may only use a single relation. This restriction facilitates efficient parsing via partitioning the input into horizontal and vertical concatenations—an aspect of Tomita's work that we share (along with Chou, as discussed below). Unlike 2D-CFGs, we do not require concatenated regions to have the same size and do not fit terminal symbols into a grid. And we omit the requirement of positional grammars that each relation direct the parser to a unique successor position. Instead, we use fuzzy sets to maintain multiple successor positions and associated confidence scores.

For math recognition in particular, grammar models are occasionally used through rule-based approaches, as in AlgoSketch [21] and its relatives, which take an approach similar to those of Zanibbi et al. [41] and Rutherford [33]. In Zanibbi's DRACULAE system, the input is processed in three passes based on tree rewriting. The first pass identifies baselines, which it organizes in a tree structure in which each edge represent geometric relationships. The second pass rewrites geometric structures as syntactic structures. (E.g., a horizontal line above one subexpression and below another would be rewritten as a fraction). The third pass rewrites syntactic structures as semantic parse trees suitable for computation. (E.g., the fraction would be rewritten as a division operation.)

Grammars may also be used as a verification step to confirm the validity of an expression recognized by some other means (e.g., [13,37]). Garain and Chaudhuri, for example, view an expression as a collection of horizontal baselines, which they call "levels". Each symbol is placed in a level as it is drawn. After the entire drawing is complete, it is segmented into horizontal and vertical "stripes," which are merged based on grammar rules and the levels. The resulting parse is validated against a grammar that generates TEX strings. If the parse is not valid, then alternatives are sought from the symbol recognizer.

Often, grammars are used as a flexible rule set to guide a recognition process based on traditional parsing techniques. Such use goes back as far as Anderson's original graphical rewriting system [2]. Already many of the features of modern grammar-based recognition systems are present in Anderson's work. In his approach, each grammar production is equipped with predicates that determine how a set of input elements should be partitioned for parsing. The predicates include rules applying to the minimum, maximum, and central x- and y-coordinates of expression and symbol bounding boxes, as well as threshold-based rules to ensure proper alignment of neighboring subexpressions. Productions not containing a terminal symbol on the RHS are restricted to contain only two non-terminals, as in Chomsky Normal Form. In such cases, Anderson includes a restriction similar to one later proposed by Liang et al. [22] (which we will encounter in Sect. 4) that input elements should be partitioned by drawing a straight line in the plane that splits the elements into two subsets and does not intersect any of them.

Another early grammar-based approach was proposed by Belaid and Haton [4]. In it, a symbol is chosen by the system as a starting point, and the symbols around it are divided into "zones" depending on context and grammar productions. The zones are then processed recursively in a top-down parsing scheme directed by operator identities. If the union of zones does not cover the desired subset of the input (i.e., the whole input for a complete parse), then the grammar is searched for rules that derive the starting point operator, and parsing proceeds in a bottom-up fashion.

Our ordering assumption, described in Sect. 4, which treats grammar productions as either vertical or horizontal concatenation, was first developed by Chou [10], who used a stochastic grammar to recognize synthetic binary images of math expressions. In Chou's model, each grammar production is assumed to be equally likely, terminal symbols are assigned probabilities based on Hamming distance between the input image and a template, and geometric relationships between symbols and subexpressions are determined by fixed, non-probabilistic rules about symbol size and baselines. Symbol ambiguity is permitted up to a predetermined probability threshold, and relation ambiguity is permitted only in the case of horizontal concatenation where baselines differ by one pixel or less. The most likely parse tree is obtained by a variant of the Cocke-Younger-Kasami (CYK) CFG parsing algorithm.

Winker et al. [39] later proposed an approach that provided for some ambiguity in geometric relationships, but not in terminal symbol identities. In their system, the space surrounding certain mathematical operators (e.g., fraction bar, summation sign, square root) is divided into a number of regions. Symbols lying in different regions may lead to different parses, and each of these possible parses is maintained and reported to the user. However, each time an ambiguity is detected, the parse is duplicated and both possibilities explored independently. This may lead to an exponential runtime, as well as an exponential number of parse results.

Miller and Viola [30], expanding on earlier work of Hull [16], proposed an approach in which a graph is dynamically created, the edges of which represent the application of grammar rules. They weight the graph edges by probability estimates of the parse results and use A$*$ to navigate to the most likely parse. The probability of a parse tree is the product of terminal symbol recognition probabilities and probabilities derived from geometric rules. Each production is assumed to be equally likely. Miller and Viola also introduced a convex

hull constraint. Instead of parsing all 2^n subsets of the input, this constraint restricts the parser to only those subsets that are geometrically convex, leading to a significant speedup.

To allow for ambiguous parse results, Miller and Viola divided the set of terminal symbols into four syntactic classes and allowed the parser to choose the class that best fits into the expression's geometry. This does permit some ambiguity between symbols, but if the top-ranked candidate within a class is incorrect, or if the syntactic class eventually settled upon by the parser is incorrect, there is no way to choose an alternative. The geometric rules used in the approach are not described, making it difficult to assess how much ambiguity in an expression's geometry they permit.

A CYK-based approach was developed by Álvaro et al. [1] for parsing typeset mathematics using 2-D stochastic CFGs. In their approach, the grammar is written in Chomsky normal form, and each production is associated with a geometric relation. Probability distributions over the relations were defined manually, and distributions over terminal symbols were derived from neural network outputs. These values are incorporated into a CYK parsing algorithm that outputs the most likely overall parse.

The four approaches just cited use probabilistic grammars to recognize math expressions. The main characteristic of probabilistic grammars is that one can assign probabilities to individual productions, and hence to derivation sequences and language elements. Each of the approaches applies a uniform distribution to grammar productions and treats terminal symbols and/or geometric relationships as probabilistic. In this context, the use of a uniform distribution over productions can be interpreted as making no assumptions about the composition of mathematical expressions aside from their formal properties, which are encoded in the grammar rules themselves. This lack of assumptions means that, at any given parsing step, the parser is not influenced by the production distribution and considers all its options to be equally likely. It also provides a neutral starting point from which training algorithms may learn more specific production probabilities. But even the uniform distribution cannot completely avoid bias during parsing, as stochastic CFGs are inherently biased toward parse trees of small height. Each derivation step reduces the probability of a parse, so, all else being equal, a more deeply nested tree will be considered to be less probable than a shallow tree. Approaches using stochastic grammars with uniform distributions and no training inherit this downside without benefiting from the ability to use a realistic distribution over productions.

The multiplicative model used by these approaches assumes independence of individual symbols and subexpressions. This assumption is invalid (though understandable for the sake of runtime efficiency) in many domains, including math recognition. Consider, for example, the expression $ax^3 + bx^2 + c? + d$, and suppose the system is trying to determine the identity of the letter indicated by a question mark. Assuming independence, it may be more likely that the missing letter is classified as X or λ than the x that a human reader would expect. Miller and Viola [30] also point out the necessity of incorporating contextual information. Indeed, they give a convincing quantitative argument as motivation for their division of terminal symbols into syntactic classes which we discussed earlier. But it is unclear how these classes are incorporated into their geometry rules and how they fit into their probabilistic model, which assumes independence of all terminals.

As alluded to above, independence assumptions are known to degrade parser performance in natural language applications. A common technique used to improve performance is to use the geometric mean of probabilities rather than their product [28, §12.1]. This leads to the same type of scoring function we propose in the next section for use with fuzzy r-CFGs. Our approach is inspired by probabilistic parsing techniques, but is not, strictly speaking, a valid probabilistic method. While some approaches exist for relaxing the independence assumption in natural language processing (e.g., that of Caraballo and Charniak [7]), they use n-gram models, which may only be applied to mathematics when large corpora of realistic expressions are available.

One such corpus, owned by Microsoft, has been used by Shi, Li, and Soong [35] to develop a math recognizer based on hidden Markov models. In their method, an observed stroke sequence is used to compute likelihoods of stroke boundaries, symbol identities, and relations between symbols. In contrast to the work cited earlier, probability distributions for symbol and relation bigrams are determined empirically from the corpus data. A symbol graph representing alternative sequences of symbol identity assignment is created, and the optimal sequence is extracted and passed to a separate math structure analysis system. The approach of Shi et al. is sensitive to writing order—symbols must be written with consecutive strokes, and symbols adjacent in time must also be adjacent (in some model-specific sense) in the expression being written. These temporal restrictions are insufficient for our purposes. For example, a user may draw two parentheses separated by empty space and then fill in the space with a fraction. We wish to recognize such inputs, in which consecutive strokes are not directly connected by geometric relationships.

Even so, such probabilistic models are promising and should be extended and generalized once large realistic, corpora of handwritten math expressions are generally available. In the meantime, it is reasonable to model the ambiguity in math recognition using fuzzy sets rather than probability distributions. Recent work by Zhang et al. [42] added support for fuzzy geometric relations to FFES, a math recognition system using Zanibbi's DRACULAE recognizer, so that it could report multiple interpretations of the user's input. However, the number of interpretations can only be

controlled by adjusting thresholds. The ambiguous interpretations are pursued independently and reported to the user simultaneously, so recognition time is proportional to the amount of ambiguity in the drawing.

Fitzgerald et al. [12] proposed an approach to math recognition which models geometric relations as fuzzy constraints on the productions of an attribute grammar. In case of strong ambiguity in the identity of a symbol or geometric relation, the input is parsed repeatedly, choosing different identities each time in a best-first exploration of alternatives. Two-sided sigmoid functions of geometric features are used to represent the membership grades in fuzzy geometric relation. It is not clear if these membership grades combine into grades for entire parse trees, or if parses are simply reported as they arise from the best-first parsing process. On one hand, the system is efficient in the sense that only strong ambiguities cause it to investigate alternative parses, but on the other, such alternatives must be parsed independently and serially, reducing efficiency. What constitutes a strong ambiguity is hard-coded, so the correct parse may fail to be reported if the membership grade of a particular geometric relation falls below a threshold. Like the work of Fitzgerald et al, our method of tree extraction—presented in Sect. 4.3—employs a best-first search strategy. But, it can report as many trees as the user requests, making available all parse trees with non-zero recognition confidence.

Our work avoids many of the difficulties identified in this section. It permits ambiguity of terminal symbols as well as geometric relations. It does not limit the number of alternative parses and allows users to select between them, but it does not construct all of the (potentially exponentially many) interpretations of the user's writing before producing output. By using fuzzy sets, we need not be constrained by probabilistic independence and can use context-sensitive scoring functions when combining subexpressions into larger expressions. By generating interpretations on demand as the user requests them, only as much work is done as is necessary to produce the correct interpretation.

3 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.

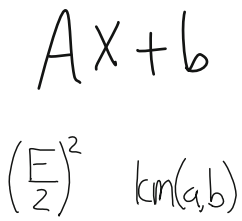

Fig. 1 Ambiguous mathematical expressions

3.1 Review of fuzzy sets

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 [40]. 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 $x \in X$.

3.2 Fuzzy r-CFG intuition

Before defining fuzzy r-CFGs formally, we motivate the definition through an example. Consider the top expression in Fig. 1: reasonable interpretations include $Ax + b$, $Ax + 6$, $A^x + b$, $A^x tb$, etc. There are three important points to note. The identity of each terminal symbol is ambiguous. (Generally, even which strokes ought to be combined to form a distinct symbol may be ambiguous, as in the bottom pair of expressions in the figure. Does the first contain a binomial term or a fraction? Is the second km or lcm?) The geometric relationships between symbols and subexpressions determine the mathematical semantics with which those symbols should be interpreted (e.g., Ax vs. A^x). And, given a particular choice of symbol identities and spatial relationships, all remaining ambiguity is semantic. In cases of semantic ambiguity (i.e., the same notation representing multiple distinct expressions, as in $u(x + y)$), it is not possible for a syntactic parser to reliably obtain the correct parse.

The rules governing the assembly of mathematical expressions from their component parts are known with certainty. Uncertainty only arises because the identities of and relationships between handwritten symbols are ambiguous. This ambiguity is a property of each particular drawing, not of the formal structure of math expressions. As such, it is not necessary to assign probabilities or fuzzy membership grades to each production in our grammar, though such an extension could certainly be made to incorporate prior domain knowledge. Instead, we model formally only the ambiguity arising from symbol and geometric relation classification processes.

This represents a significant difference between our model and traditional fuzzy grammars. Typically, fuzzy grammars are a straightforward modification of probabilistic grammars in which each non-terminal is considered to be a fuzzy set, and its productions are each assigned a membership grade [20]. Such grammars generate languages in which each string has a particular grade of membership, just as each string in a stochastic language is derived with a particular probability.

In our approach, the language itself is not fuzzy. It consists of exactly those math expressions derivable from the grammar's start symbol. Fuzziness represents the degree to which a particular drawing is compatible with a particular math expression in the grammar's language. It is therefore not a property of an expression, but of the input, and it provides a way to measure the similarity between a handwritten input and one of its potential interpretations as a math expression.

3.3 Fuzzy r-CFG definition

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

3.3.1 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.

3.3.2 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 size $|e|$ of an expression counts the number of terminal symbols that appear in it. For example, the size of A^x is $|A \nearrow x| = 2$ and the size of $A^x + b$ is $|(A \nearrow x) \rightarrow + \rightarrow b| = 4$. In these expressions, the parentheses indicate subexpression grouping, not terminal symbols. The arrows indicate spatial relationships corresponding to general writing direction; they are described in detail below.

The *representable set of* G is the set E of all expressions derivable from the non-terminals 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 $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$,

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

For each non-terminal 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 is $I = T \times E$. A pair in I corresponds to the interpretation, through recognition, of a particular observable as a particular expression. We make I fuzzy by assigning membership grades to each interpretation. As described in the motivation section, the membership grade $I((t, e))$ measures the degree to which the observable t is compatible with the expression e, as measured by the fuzzy relations in the grammar. Note that the (crisp) language generated by G is E_S, where S is the start symbol.

The language structure generated by a fuzzy r-CFG is similar to one we used in previous work on sketch corpus creation [24]. In that formulation, a grammar was used as a generative language model for producing random mathematical expressions. Context-sensitive probabilities were assigned to production rules, similarly to stochastic CFG applications. More importantly, recognition processes were not involved, so the relations acted only on expressions, not on interpretations. This distinction reflects the grammar's purpose in each case: as a language model for the generation application and as a model of sketch interpretation for the recognition application that we address in this paper.

3.3.3 Relations

The set R contains fuzzy relations that give structure to expressions by modeling the relationships between subexpressions. These relations act on interpretations, allowing

Fig. 2 An expression in which the optimal relation depends on symbol identity

context-sensitive statements to be made about recognition in an otherwise context-free setting.

The necessity of context-sensitive evaluation was mentioned in Sect. 2 and is further illustrated by Fig. 2. The expression shown in the figure 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

$$\rightarrow ((t_1, A), (t_2, P)) > \nearrow ((t_1, A), (t_2, P)).$$

That is, the membership grade of a pair of interpretations in a spatial relation should vary, depending on the expressions (i.e., the context) involved. Given explicit membership functions, we could evaluate whether or not these expectations are met.

3.3.4 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 $r_\Sigma((t', b)) = 0.6$.

3.3.5 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., $e_i \in E_{A_i}$ and $(t_i, e_i) \in I$), 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 $i = 1, \ldots, k-1$.

3.4 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 non-terminals and $+$ is a terminal. Then the production `[ADD]` $\overset{\rightarrow}{\Rightarrow}$ `[EXPR]` $+$ `[EXPR]` models the syntax for infix addition: two expressions joined by the addition symbol, written from left to right.
2. The production `[SUP]` $\overset{\nearrow}{\Rightarrow}$ `[EXPR][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:

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

   `[INTEGRAL]` $\overset{\rightarrow}{\Rightarrow}$ `[ILIMITS][EXPR]`d`[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.

3.5 Semantic expression trees and textual output

Fuzzy r-CFG productions model the two-dimensional syntax of mathematical expressions. To represent mathematical semantics, each production is also associated with rules for generating textual output and math expression trees with semantic information. For example, consider again the production `[ADD]` $\overset{\rightarrow}{\Rightarrow}$ `[EXPR]` $+$ `[EXPR]`. A rule to generate a MathML string would be written in our grammar format as `<mrow>%1<mo>+</mo>%3</mrow>`, where the "`%`n" notation indicates that the string representation of the nth RHS element should be inserted at that point. A rule to generate a semantic expression tree would be written `ADD(%1,%3)`. This rule would generate a tree with the root node labeled with addition semantics ("`ADD`") and two subtrees. Similarly to the string case, the `%`n notation indicates that the tree representation of the nth RHS element should be used as a subtree. Hence, the first child tree corresponds to the left-hand operand of the addition expression, and the second child tree corresponds to the right-hand operand.

3.6 The fuzzy set of interpretations

Because of ambiguity, there are usually several expressions that are reasonable interpretations of a particular input observable $t \in T$ (e.g., A^p and AP are both reasonable interpretations of Fig. 2). 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 set I of interpretations mentioned above; i.e., $I_t = \{e : (t, e) \in I\}$. It remains to specify the membership function $I_t(e) = I(t, e)$ concretely.

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 non-terminals. This normal form is easily realized by, for each $\alpha \in \Sigma$, introducing a new non-terminal 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

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

There is room for experimentation when choosing the membership function of $I_{t_1,\ldots,t_k}^p$. The typical combination rule in fuzzy systems is to take the minimum of all relevant membership grades. In the present context, this rule has the disadvantage that all parses sharing a low-scoring symbol or relation are assigned the same score. Zhang et al. [42] found that using multiplication when combining membership grades helped to prevent ties. We, therefore, compute the membership grade of an r-concatenation $e = e_1 r \cdots r e_k$ in $I_{t_1,\ldots,t_k}^p$ as

$$I_{t_1,\ldots,t_k}^p(e) = \left(\prod_{i=1}^{k} I_{t_i}^{A_i}(e_i)^{2|e_i|-1} \prod_{i=1}^{k-1} r((t_i, e_i), (t_{i+1}, e_{i+1}))\right)^{1/(2|e|-1)} \tag{3}$$

As an expression always contains exactly one more terminal than relation, this function assigns as an expression's membership grade the geometric mean of the grades of all of its components. Geometric averaging preserves the tie-breaking properties of multiplication while normalizing for expression size.

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 if 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.

4 Parsing fuzzy r-CFGs

There are two particular properties of fuzzy r-CFGs that make parsing difficult. Like other relational grammars, the languages they generate are multi-dimensional. This prevents the straightforward application of common parsing techniques like Earley's algorithm, which assume a simply ordered sequence of input tokens. Multi-dimensionality also complicates the task of deciding which subsets of the input may contain a valid parse. Furthermore, because our input is ambiguous, we require a parser to report all recognizable parse trees. Since there may be exponentially many trees, some effort is required to ensure a reasonable running time. This section presents two formal assumptions on the structure of the relations of fuzzy r-CFGs and describes how to construct an efficient fuzzy r-CFG parser using those assumptions. Detailed algorithms are omitted but may be found in a technical report [27].

4.1 Representing I_t as a shared parse forest

In Eq. 2, each set $I_{t_1,\ldots,t_k}^p$ contains up to $\prod_i \left|I_{t_i}^{A_i}\right|$ expressions. It is therefore infeasible to identify or construct each expression individually before reporting parse results to the user. This problem is similar to that of parsing ambiguous languages, in which the same input may be represented by many parse trees. Indeed, the language of math expressions is ambiguous even in the absence of syntactic fuzziness due to the semantic ambiguities identified earlier. Parsing ambiguous languages is a well-studied problem; we adopt the shared parse forest approach of Lang [18], in which all recognizable parses are simultaneously represented by an AND-OR tree.

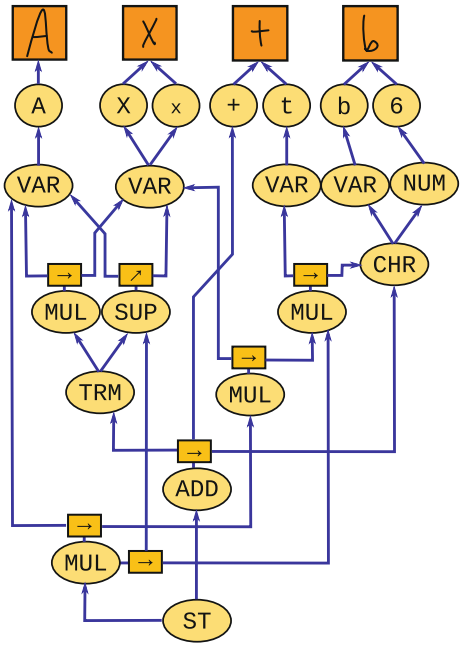

Fig. 3 Shared parse forest for figure

For example, consider again the expression shown in Fig. 1, along with the following toy grammar:

$$
\begin{aligned}
&\texttt{[ST]} \Rightarrow \texttt{[ADD]} \mid \texttt{[TRM]}\\
&\texttt{[ADD]} \overset{\rightarrow}{\Rightarrow} \texttt{[TRM]} + \texttt{[ST]}\\
&\texttt{[TRM]} \Rightarrow \texttt{[MUL]} \mid \texttt{[SUP]} \mid \texttt{[CHR]}\\
&\texttt{[MUL]} \overset{\rightarrow}{\Rightarrow} \texttt{[SUP][TRM]} \mid \texttt{[CHR][TRM]}\\
&\texttt{[SUP]} \overset{\nearrow}{\Rightarrow} \texttt{[CHR][ST]}\\
&\texttt{[CHR]} \Rightarrow \texttt{[VAR]} \mid \texttt{[NUM]}\\
&\texttt{[VAR]} \Rightarrow a \mid b \mid \cdots \mid z\\
&\texttt{[NUM]} \Rightarrow 0 \mid 1 \mid \cdots \mid 9
\end{aligned}
$$

Figure 3 depicts a shared parse forest representing some possible interpretations of Fig. 1. In the figure, the boxed arrows are AND nodes. Those arrows indicate the relation that links derived subexpressions. The ovals are OR nodes representing derivations of non-terminals on particular subsets of the input. The circles relate subsets of the input with terminal symbols from the grammar. Simple productions of the form `[CHR]` $\Rightarrow$ `[VAR]`, for example, have been omitted for clarity. Any tree rooted at the `[ST]` node that has exactly one path to each input element is a valid parse tree. This shared parse forest captures, for example, the expressions $Ax + b$, $AX + 6$, $A^x + 6$, $A^X tb$, etc. If an expression is incomplete (e.g., $(x + y$ without the closing parenthesis), then no parse will exist for the correct interpretation. However, other parses using different interpretations of the input may exist (e.g., $lx + y$ or $C_x ty$).

Parsing a fuzzy r-CFG may be divided into two steps: forest construction, in which a shared parse forest is created that represents all recognizable parses of the input, and tree extraction, in which individual parse trees are extracted from the forest in decreasing order of membership grade. We describe each of these steps in turn.

4.2 Shared parse forest construction

Because the symbols appearing in a two-dimensional math expression cannot be simply ordered, one would naively have to parse every subset of the input in order to obtain all possible parses. Similarly, recall from Eq. 2 that I_t^p is constructed from a union taken over all partitions of t. It is intractable to take this union literally. To develop a practical parsing algorithm, we introduce constraints on partitions so as to limit how many must be considered. The constraints are based on the two-dimensional structure of mathematical notation.

4.2.1 The ordering assumption and rectangular sets

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 $\operatorname{ord} \downarrow = <_y$.

Informally, we assume that each geometric relation $r \in R$ is embedded in either $<_x$ or $<_y$. Thus, we may treat any grammar production as generating either horizontal or vertical concatenations of subexpressions, making the partition-selection problem much simpler.

More formally, 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 $\max_{\operatorname{ord} r} t_1 \geq_{\operatorname{ord} r} \min_{\operatorname{ord} r} t_2$.

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 Fig. 1, to parse $A^x + b$ in the obvious way requires that the A begins before the x, and the $+$ begins after the x but 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 `[FRAC]` $\overset{\downarrow}{\Rightarrow}$ `[EXPR]`—`[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. [22] 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

$$t' = \left\{a \in t : \min_x t' \leq a \leq \max_x t'\right\} \cap \left\{a \in t : \min_y t' \leq a \leq \max_y t'\right\}.$$

Call a partition $t_1 \cup \cdots \cup t_k$ of a rectangular set t *rectangular* if every t_i is rectangular in t.

From the definition of $<_x$ and $<_y$, a set t' that is rectangular in t must include all input elements in t whose left edge lies between the left-most left edge of an element in t' and the right-most left edge *and* whose top edge lies between the top- and bottom-most top edges of elements of 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

$$\left((t_j, e_j), (t_{j+1}, e_{j+1})\right), \ldots, ((t_{i-1}, e_{i-1}), (t_i, e_i)) \in r.$$

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 $\min_d t_j \leq_d a' \leq_d \max_d t_j$ 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 contiguous substrings in one-dimensional string parsing. This definition could be generalized to arbitrary dimension, giving "hypercube sets" of input elements.

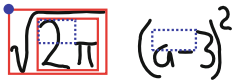

Fig. 4 Expressions with overlapping symbol bounding boxes

Following Liang et al, notice that any rectangular set $u \subseteq t$ can be constructed by choosing any four input elements in t and taking them to be represent the left, right, top, and bottom boundaries of the set. There are therefore $\mathcal{O}\left(|t|^4\right)$ rectangular subsets of t. If we instead naively parsed every subset of the input, there would of course be $2^{|t|}$ subsets to process. The ordering assumption, thus, yields a substantial reduction in the number of subsets that must be considered for parsing.

Liang et al. define a rectangular hull of a set of input elements to be their geometric bounding box, and they parse only those sets whose rectangular hulls have a null intersection. I.e., no bounding box of a subset selected for parsing can intersect that of any other parsed set. This formulation causes problems for containment notations like square roots, as well as for somewhat crowded or messy handwriting styles, which our rectangular set formulation avoids. For example, consider the square root expression on the left of Fig. 4. The rectangular hulls of the root symbol and its argument are shown as solid boxes and are the (union of) geometric bounding boxes of the symbols. Note that the rectangular hull of the square root symbol intersects (in fact contains) that of its contents. The argument cannot be separated from the operator into non-intersecting hulls.

When considering input subsets as rectangular sets, we do not use the natural geometric bounding box of the strokes, as Liang et al. do. Instead, we take only the minimal x- and y-coordinates of each stroke and consider the bounding box (or rectangular hull) of those points. The "boundary" of the root symbol in Fig. 4 is thus the single point at the top-left of its bounding box, and the boundary of the rectangular set representing the argument 2π is shown as a dotted box. By using minimal coordinates instead of complete bounding boxes, the rectangular set boundaries do not intersect.

Similarly, in the expression on the right of Fig. 4, the rectangular hull of opening parenthesis intersects the hull of the a symbol and that of the closing parenthesis intersects the hulls of both the 3 and the exponent 2. As before, the expression cannot be partitioned such that the required subsets' hulls are non-intersecting. But the boundary of the rectangular set representing $a-3$ (indicated by a dotted box) extends only to the left edge of the 3 symbol. The boundaries of the parentheses and the exponent are, as for the root sign, single points at the top-left corner of their bounding boxes. This small change—taking the left- and top-most coordinates of strokes as their representative points, "spaces out" overlapping writing and facilitates non-intersecting partitions.

Figure 5 illustrates schematically the recursive rectangular partitioning of an expression, following the expected parse of $\sum_{i=1}^{n-1} \frac{i^2}{n-i}$. The whole expression is a rectangular set. The central dotted vertical line indicates a rectangular partition

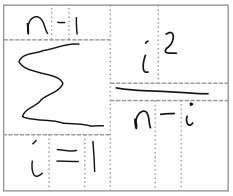

Fig. 5 Recursive rectangular partitions in an expression

into the sum symbol (with limits) and the summand. In the summand, for example, the two horizontal dashed lines indicate a rectangular partition into the numerator, fraction bar, and denominator, and the numerator and denominator are further partitioned into rectangular sets, each containing a single symbol. Note that resulting boxes in the figure are meant to emphasize the hierarchical structure of the partition. They do not indicate the geometric bounding boxes of the rectangular sets.

4.2.2 Parsing algorithm

Using the restriction to rectangular partitions derived above, it is straightforward to adapt the CYK bottom-up parsing algorithm so that is generates shared parse forests for fuzzy r-CFGs. Detailed algorithms are given in a technical report [27]. Experiments with the CYK approach showed that, while all rectangular sets must be enumerated and parsed, relatively few actually contribute to valid parse trees. A similar observation was made by Grune and Jacobs [15] in the case of ambiguous languages. The algorithm's runtime, while predictable, is always the worst-case time.

Instead of CYK, we, therefore, adapted to fuzzy r-CFGs, a tabular variant of Unger's method for CFG parsing [38]. In this approach, we assume that the grammar is in the normal form described in Sect. 3.6. At a high level, the algorithm parses a production p on an input subset t as follows:

1. If p is a terminal production, $A_0 \Rightarrow \alpha$, then check if $(t, \alpha) \in r_\Sigma$. If so, add the parse to table entry (t, α); otherwise parsing fails.
2. Otherwise, p is of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$. For every rectangular partition $t_1 \cup \cdots \cup t_k$ of t, parse each non-terminal A_i on t_i. If any of the subparses fail, then fail on the current partition. Otherwise, add the partition to table entry (t, A_0). If parsing fails on every partition, then parsing fails on t.

One drawback of this algorithm is that its runtime is exponential in the size of the RHS of a grammar production, since Case 2 iterates over $\binom{|t|}{k-1}$ partitions. This bound is obtained by sorting the input by $<_{\text{ord}\,r}$ and choosing $k-1$ split points to induce a rectangular partition. In the worst case, then (i.e., when parses exist on every partition), our algorithm must consider every grammar production on every rectangular set, giving a complexity of $\mathcal{O}\left(n^{3+k}pk\right)$, where n is the number of elements in the input observable, p is the number of grammar productions, and k is the number of RHS tokens in the largest production. The extra factor of k arises from writing up to k partition subsets into a parse table entry in Case 2. Importantly, k can be controlled by the designer of a grammar as a tradeoff between RHS size and number of grammar productions. For example, if the grammar is written in Chomsky Normal Form (CNF), then the complexity is $\mathcal{O}\left(n^5 p\right)$. Note that the general bound is asymptotically tight to the worst-case size of the parse forest, since there may be $\mathcal{O}\left(n^4 p\right)$ table entries (counting each production as a distinct entry), each of which may link to $\mathcal{O}\left(n^{k-1}k\right)$ other table entries.

Instead of writing our grammar in CNF, we allow arbitrary production lengths and use the following three optimizations to reduce the number of partitions that must be considered. The first two are typical when using Unger's method [15]; the third is specific to fuzzy r-CFG parsing.

1. *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. We, therefore, "seed" the parse table with symbol recognition results and limit the enumeration of rectangular partitions to those for which the appropriate terminal symbols are already present in the parse table. Such seeding also facilitates recognition of typeset symbols, which are simply inserted into the parse table with their known identities prior to invocation of the parsing algorithm.
2. *Minimum non-terminal length.* If there are no empty productions in the grammar, then each non-terminal must expand 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 non-terminals $A_1 \cdots A_k$. These quantities further constrain which partitions are feasible.
   For example, consider the top expression in Fig. 1. Suppose we are parsing the production `[ADD]` $\overset{\rightarrow}{\Rightarrow}$ `[TRM]` + `[ST]` 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.
3. *Spatial relation test.* Just because an input subset can be partitioned into rectangular sets does not mean that

those sets satisfy the geometric relations specified by a grammar production. Unfortunately, the grammar relations act on expressions as well as observables, so they cannot be tested during parsing because expressions are not explicitly constructed. As there may be exponentially many expressions, they *cannot* be constructed, and therefore, we cannot evaluate the grammar's spatial relations, which vary with expression identity. 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

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

These relations will be specified more concretely in Sect. 4.3.2 below. If $\hat{r}(t_i, t_{i+1}) = 0$ for any pair of adjacent sets in a rectangular partition, the partition need not be parsed.

4.2.3 Incremental parsing

The parsing algorithm above 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 of the parse table 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 reused. Note that, although existing parses remain valid, they will not necessarily be reused. For example, suppose the expression $a^3 + b$ is changed to $a^3 + 2b$ by the insertion of a new stroke representing the number 2. Then the parse of $a^3 + b$ on the original three strokes is still a valid interpretation of those strokes. But those strokes no longer form a rectangular set with respect to the new set of input strokes, so they will not be considered as a group when the new input is parsed. However, the existing parses of a^3 and $+$ remain individually valid, and they need only be combined with the new parse of $2b$ to form the entire expression.

In the case where an existing observable a_i is removed, the situation is slightly more complicated. Any parse table entry for an input subset t that includes a_i becomes invalid and must be removed from the table. When the parser is invoked on the revised input set $\{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\}$, any existing parse results that do not include the stroke a_i will be reused.

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 $(\texttt{1}, A)$ in the parse table. 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 $(\texttt{01}, A)$ is the same as accessing the entry $(\texttt{1}, A)$ that was created when the input was just one stroke.

Incremental parsing is useful for two main reasons. It permits a parser to process in the background as the user is writing, helping to avoid long delays after the user has finished writing the expression. It also allows for reporting of intermediate parse results so that the user can tell whether the parser is going off-track and make any necessary corrections to the output. The tree extraction technique described in the next subsection extracts only complete parse trees (and thus cannot report partial results, like $a+?$, in which the right-hand operand has not yet been written). However, it is straightforward to adapt a grammar so as to support partial expressions: one simply creates new non-terminals that derive prefixes or suffixes of complete expressions.

4.3 Parse tree extraction

The links between entries in the parse table implicitly specify a shared parse forest. It remains to extract individual parse trees from this forest in decreasing order of membership grade in the fuzzy set I_t of interpretations of the user's input t. To do so, we must explicitly evaluate the membership grades of particular parse trees and compare them to determine which is highest, second highest, and so on. Given the lack of constraint on the grammar relations and the large number of parse trees in the forest, this may naively be a very time-consuming task.

Consider, for example, the problem of determining the highest-ranked parse tree in the forest shown in Fig. 3. Call the input subsets t_1, t_2, t_3, t_4 from left to right. We need to check whether $\rightarrow\ ((t_1, A), (t_2, x)) >\nearrow\ ((t_1, A), (t_2, x))$ to decide whether the first two symbols form a multiplication or exponentiation. But those geometric relations may give different results for different expression choices, so we also need to check whether $\rightarrow\ ((t_1, A), (t_2, X)) >\nearrow ((t_1, A), (t_2, X))$. It could be the case that one relationship has a higher grade when t_2 is interpreted as x, and the other is higher when it is interpreted as X. Indeed, such behavior is desirable to ensure correct recognition of ambiguous cases as illustrated in Fig. 2. We further need to combine those geometric relation grades with the relevant terminal relation grades, $r_\Sigma((t_1, A)), r_\Sigma((t_2, x)), r_\Sigma((t_2, X))$, to find

membership grades in $I_{t_1 \cup t_2}$, and thereby determine the best-first ordering of subexpressions.

Because we have specified no constraints on the grammar relations, it is possible for two expressions e_1, e_2 to have very low membership grades in I_{t_1} and I_{t_2}, but for the pair of interpretations $(t_1, e_1), (t_2, e_2)$ to have sufficiently high membership in a relation r that $e_1 r e_2$ is the highest-graded interpretation in $I_{t_1 \cup t_2}$. To determine the highest-graded parse tree overall, one must examine every single parse tree represented in the forest. In particular, at the parse forest node representing the set $I^p_{t_1,\ldots,t_k}$ of parses for production $p = A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$ on the partition $t_1 \cup \cdots \cup t_k$ of t, a total of $\prod_{i=1}^{k} \left| I^{A_i}_{t_i} \right|$ trees would need to be constructed and graded. This quantity is exactly $\left| I^p_{t_1,\ldots,t_k} \right|$, which is still less than the total number $\left| I^{A_0}_t \right|$ of parses of A_0 on t. At a node higher up the parse forest referencing parses of A_0 on t, every tree in $I^{A_0}_t$ would need to be extracted and combined with sibling trees. The number of trees that must be examined by this naive approach thus increases combinatorially as parse tree depth increases. Such an approach is obviously not feasible if we wish to report parses to the user in real-time as (s)he writes.

4.3.1 Relational classes

Recall from Sect. 3 that E is the set of all expressions generated by the productions of a given fuzzy r-CFG. We define several *relational classes* $c_1, \ldots, c_m$, and assign every expression in E to at least one class. Thus, $E = c_1 \cup \cdots \cup c_m$, but the c_i need not form a partition of E.

These classes play a role in our system similar to that of syntactic or symbol classes in the approaches of Miller and Viola [30], Zanibbi et al. [41], and Rutherford [33]. We include five relational classes for archetypal symbol shapes, which we extend by a special class, *box*, which represents all multi-symbol expressions. Details are provided in the next section.

To facilitate efficient tree extraction, we assume that the grammar relations depend only on expressions' relational classes, not on their precise structures. In the example above, one might expect that

$$\rightarrow ((t_1, A), (t_2, x)) \not\equiv \rightarrow ((t_1, A), (t_2, X)),$$

as x is a centered symbol and X is an ascender. One might also expect that

$$\rightarrow ((t_1 \cup t_2), A^x), (t_3, +)) = \rightarrow ((t_1 \cup t_2, Ax), (t_3, +)).$$

That is, the extent to which the first two symbols are judged to be horizontally adjacent with the plus sign does not depend on whether one interprets them as a multiplication or an exponentiation.

Our selection of relational classes captures the intuition that symbol identities have a greater effect on subexpression layout than subexpression identities have on the layout of larger expressions. If such an intuition seems unreasonable, one is free to introduce relational classes corresponding to subexpression structures, so that, for example, $\rightarrow ((t_1 \cup t_2, e), (t_3, +))$ would vary depending on whether e was A^x or Ax. One is not restricted, as in existing approaches, to syntactic classes that represent collections of individual symbols. The relational class approach, thus, provides control over the tradeoff between context sensitivity in grammar relations and execution time.

Formally, denote by C the set of relational classes and by $cl(e)$ the set of classes to which an expression e belongs. Each grammar relation may be viewed as a union of class-specific relations:

$$r = \bigcup_{c_1, c_2 \in C} r_{c_1,c_2},$$

where $r_{c_1,c_2}((t_1, e_1), (t_2, e_2)) = 0$ if $c_1 \notin cl(e_1)$ or $c_2 \notin cl(e_2)$. Then, by the usual rules of fuzzy sets,

$$r((t_1, e_1), (t_2, e_2)) = \max_{\substack{c_1 \in cl(e_1) \\ c_2 \in cl(e_2)}} r_{c_1,c_2}((t_1, e_1), (t_2, e_2)). \quad (4)$$

Using the formulation above, we constrain the grammar relations by the following assumption:

Assumption 2 (*Interchangeability*) Let t_1, t_2 be observables, let $e_1, e_2, \hat{e}_1, \hat{e}_2$ be representable expressions, and let $r \in R$ be a relation. We assume for relational classes c_1, c_2 that

$$r_{c_1,c_2}((t_1, e_1), (t_2, e_2)) = r_{c_1,c_2}\left((t_1, \hat{e}_1), (t_2, \hat{e}_2)\right)$$

whenever $c_1 \in cl(e_1), cl(\hat{e}_1)$ and $c_2 \in cl(e_2), cl(\hat{e}_2)$.

Each production p, and hence each non-terminal symbol A, may be associated with sets $cl(p), cl(A)$ of relational classes in the natural way, so that, if A can derive an expression e via production p, then $cl(e) \subseteq cl(p) \subseteq cl(A)$.

When constructing the parse table, each time a potential parse of some production p is identified on an input subset t, the relational classes associated with p are noted in the parse table. Thus, when extracting trees, the potential classes of expressions that might be obtained from any node of the shared parse forest are known.

This formulation is similar in some ways to the scoring rules developed by Rhee and Kim [32] to represent all parse trees with identical structure, but differing symbol identities by a single tree with indeterminate terminal symbols. Working in a cost-minimization framework, they defined the cost of a spatial relationship between two indeterminate symbols as the minimum relationship cost between determinate symbols, taken over all recognized possibilities for the indeterminate symbols' identities. This definition is similar in form

to our calculation of a relation membership grade as the maximum class-specific membership grade, taken over relational classes. In our language, Rhee and Kim assign each terminal symbol to its own relational class.

4.3.2 Spatial relation test in parse forest construction

Recall from Sect. 4.2.2 that, during parse forest construction, we test whether a partition of the input is feasible for parsing by evaluating spatial relations that approximate the fuzzy grammar relations. Now we may define these approximate relations more specifically, using relational classes.

A pair of observables (t_1, t_2) should have membership grade 1 in an approximate relation $\hat{r}$ if there exist expressions e_1, e_2 such that the pair of interpretations $((t_1, e_1), (t_2, e_2))$ has non-zero membership in the corresponding grammar relation r. Using relational classes, this is the same as asking whether

$$\max_{c_1, c_2 \in C} r_{c_1, c_2}\left((t_1, e_1), (t_2, e_2)\right) > 0.$$

That is, do relational classes exist such that the interpretations will have non-zero membership in the grammar relation, regardless of what the expressions are (since they are assumed to be interchangeable)? If so, we take $\hat{r}(t_1, t_2) = 1$; if not, $\hat{r}(t_1, t_2) = 0$.

Note that these $\hat{r}$ relations are used only in the forest construction stage. In the tree extraction algorithm given in the next section, explicit expressions are constructed, so the true grammar relations are used based on each particular expression's relational classes.

4.3.3 Efficient tree extraction using relational classes

Consider again the problem of determining the most highly-graded parse of an input t. The most highly-graded parse of a non-terminal A on t is just the most highly-graded parse, taken over all productions with LHS A. Similarly, the most highly-graded parse of a production p on t is the most highly-graded parse, taken over all partitions of t on which potential parses of p were found. These partitions are noted in the parse forest by our variant of Unger's method.

To find the most highly-graded parse of p on a particular partition $t_1 \cup \cdots \cup t_k$ of t, suppose that p is of the form $A_0 \overset{r}{\Rightarrow} A_1 \cdots A_k$. (If p is of the form $A_0 \Rightarrow \alpha$ for a terminal $\alpha \in \Sigma$, then the problem is trivial.) If any table entry (t_i, A_i) contains only one relational class, then the most highly-graded parse must include the most highly-graded interpretation in $I_{t_i}^{A_i}$, because under the interchangeability assumption, choosing a lower-graded interpretation cannot increase the grade of membership in r, hence it must decrease the overall membership grade in I_t^p. In such a case, we can simply recursively extract the most highly-graded parse of A_i on t_i and paste it into a parse tree.

More generally, if a table entry (t_i, A_i) contains several classes, then the most highly-ranked interpretation of each class must be considered. Thus, the membership grade in I_t must be evaluated for at most $\prod_i |C_i|$ interpretations, where C_i is the set of relational classes that were noted at table entry (t_i, A_i) during parsing. Thus, when interpretations within a given table entry share relational classes (as they usually will for a reasonable selection of classes), significantly fewer interpretations need to be extracted and graded than in the naive approach to tree extraction discussed at the outset of this subsection.

Note that, because grammar relations are binary, two table entries (t_i, A_i) and (t_j, A_j) become independent if there is some ℓ between i and j such that table entry (t_ℓ, A_ℓ) contains only one relational class. In such cases, the relational classes at cells (t_i, A_i) and (t_j, A_j) need not be varied with respect to each other, reducing the number of evaluations needed.

To find the most highly-graded parse of A_i on t_i that has relational class c, one need to only know which productions of A_i generate expressions of that class and recursively find the most highly-graded parse among those productions. Notice that $|t_i| < |t|$, so no infinite recursion is possible.

To extract the top-ranked parse of the entire input, therefore, one visits every parse table cell in bottom-up order, determining the highest-ranked local parse at each one. Considering different productions with the same LHS as different parse table cells, there are at most $n^4 p$ cells, where n is the number of input elements and p is the number of grammar productions. The forest construction algorithm identifies at most $\binom{n}{k-1}$ distinct rectangular partitions on which to parse p, where k is the number of right-hand tokens in the largest production. Using relational classes, $\prod_i |C_i|$ interpretations must be considered per partition, as described above. This is bounded naively by c^k, where $c = |C|$ is the total number of classes. Note that, since the approach proceeds bottom-up, the top-ranked interpretations from partition subsets are available in constant time, as they have already been computed and stored. Still, it takes time $\mathcal{O}(k)$ to extract them, combine them into a larger interpretation and compute its membership grade. Thus, obtaining the top-ranked parse tree takes worst-case time $\mathcal{O}\left(n^{3+k} p c^k k\right)$. This is the same complexity as the forest construction algorithm with an additional factor of c^k arising from the relational classes. Importantly, both c and k are determined by the grammar designer, allowing a tradeoff between flexibility and speed. If all expressions are considered to be in the same relational class (i.e., if relational membership functions are context-insensitive), then $c = 1$, and this extra factor disappears.

The above process may be generalized to solve the problem of finding the mth-most highly-graded parse, given that all of the $m-1$ more-highly-graded parses are known. To find

the mth parse of a non-terminal A, we maintain a priority queue of known parses. When the $(m-1)$st parse is extracted, it was obtained as, say, the jth parse from a particular production p. Thus, to find the mth parse, we extract the $(j+1)$st parse from p, add it to the queue, and pop the most highly-ranked parse out of the queue. A similar process may be used to extract the mth-most highly-graded parse from a particular production (selecting over partitions), and from a particular relational class (selecting over productions).

A slightly more complicated process is required to find the nth-most highly-graded parse of a production $A_0 \stackrel{r}{\Rightarrow} A_1 r \cdots r A_k$ on a particular partition $t_1 \cup \cdots \cup t_k$ of t. Again, we maintain a priority queue of known parses. During extraction of the most highly-ranked parse, each interpretation whose membership grade in I_t was evaluated is added to the queue, and only one—the most highly ranked—is removed. Inductively, then, the $m-1$st parse was an r-concatenation of the j_ith parses of each A_i on t_i of class c_i, for $i = 1, \ldots, k$. Denote this parse by $(j_1, \ldots, j_k)$, where the integer at each position indicates the ranking of the subexpression at that position in the concatenation. By the interchangeability assumption, the next most highly-ranked parse using the same selection of relational classes must be one of $(j_1+1, j_2, \ldots, j_k)$, $(j_1, j_2+1, j_3, \ldots, j_k)$, $\ldots$, $(j_1, \ldots, j_{k-1}, j_k+1)$. Therefore, the membership grades of all k of these expressions are evaluated, and the expressions are added to the priority queue. The mth-most highly-ranked parse overall is simply popped off of the queue and its provenance noted, as in the previous case.

In this case, tree extraction is much faster than for the top-ranked tree. That case requires the top-ranked tree to be determined for each cell in the parse table, whereas, when finding the mth-most highly-ranked parse, only those table cells and relational classes explicitly used to form the $(m-1)$st most highly-ranked parse must be considered. There are $\mathcal{O}(np)$ such table cells. ($\mathcal{O}(n)$ for the number of nodes in the parse tree which have more than one child, and $\mathcal{O}(p)$ to account for trivial productions of the form $A \Rightarrow B$.) At the $\mathcal{O}(n)$ table cells arising from non-trivial productions, at most k new candidate interpretations are created and evaluated, each requiring the extraction of only one subinterpretation. Extraction, thus, has worst-case cost $\tilde{\mathcal{O}}(np+nk)$, where the so-called "soft-O" notation $\tilde{\mathcal{O}}(*)$ ignores the logarithmic factors arising from priority queue operations.

4.4 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.

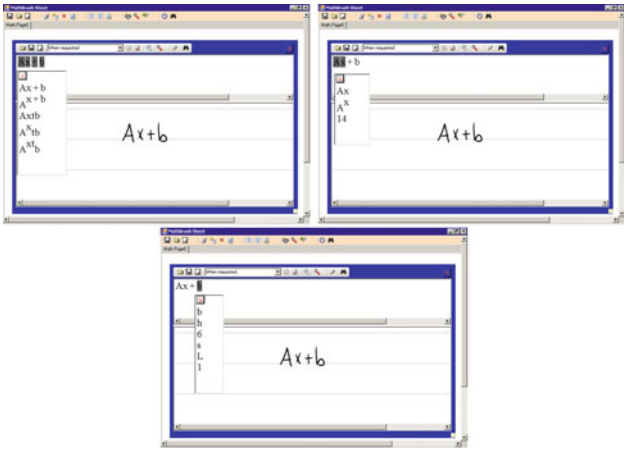

Fig. 6 Interface for displaying and selecting alternative interpretations. Different subexpressions have been selected for alternatives; clockwise from *top-left*: the whole expression, the first two symbols, and the last symbol

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 non-terminal.

Both of these lock types are useful in different circumstances. For example, suppose a user draws the top expression in Fig. 1, meaning the expression $A^X + b$, but the first result returned by the recognizer is $A^x t b$. In our system, the user can correct the result to an addition, $A^x + b$. This applies a semantic lock to the entire input, requiring all derived expressions to be additions. The user could then correct the x to X, applying an expression lock to the corresponding ink subset. The graphical interface used in MathBrush for corrections is shown in Fig. 6.

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 non-terminal A_L by a semantic lock, then we take $I_t^{A'}$ to be empty for all non-terminals $A' \neq A_L$, except those for which a derivation sequence $A' \Rightarrow^* A_L$ exists, in which case we take $I_t^{A'} = I_t^{A_L}$.

5 Computing the grammar relations

The fuzzy r-CFG formulation requires the calculation of two types of fuzzy relations: the terminal symbol relation r_Σ and the geometric relations in R. This section outlines the symbol

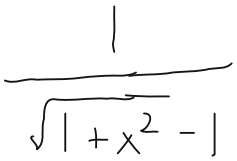

Fig. 7 Multiple strokes may be combined into one

and relation classification subsystems from which our math recognizer obtains membership grades for those relations.

5.1 Terminal symbol relation

The symbol classification subsystem receives strokes as input one at a time as they are drawn by the user. Strokes may be merged if they tend in the same direction and their ends are sufficiently close together. For example, in Fig. 7, the square root sign and fraction bar are each drawn with two strokes that would be merged together.

The input strokes are maintained in a list. The first element of the list is the first stroke to be received as input. A stroke's successor in the list is the stroke that is nearest to it that has not already appeared in the list. There is no reliance on temporal information, so writing or editing order has minimal impact on classification results. Special processing is invoked for strokes that appear to be dots: they are placed after the stroke with which a combination of horizontal and vertical distance measurements is minimized.

Next, groups of strokes that may correspond to distinct terminal symbols are identified. Two types of candidate groups are identified by extracting contiguous sublists from the list of strokes. First, *proximity groups* are extracted and scored based on a combination of stroke proximity and bounding box alignment. Then *stacked groups* are constructed by identifying collections of proximity groups arranged in vertical stacks; they capture such symbols as $\leq, \pm, \equiv$, etc.

Model-based symbol recognition is performed on each candidate group in a two-step process. Feature-based matching is used first as a pruning step: features such as first and last point, width, height, and arc length are extracted from the candidate group. They are compared with corresponding features of model symbols, and the numerical differences are tallied in a weighted sum.

Those models with a sufficiently small feature difference are then compared with the candidate stroke group using a fast variant of elastic matching distance [26]. Based on comparison of stroke-based features, the input strokes may be reordered and/or individually reversed so as to obtain the smallest match distance. The elastic matching distances are inverted so that small distances correspond to large scores and then weighted by the appropriate group score (proximity group score for most symbols, stacked group score for stack-based symbols, as described above).

Many symbols share similar-looking strokes or subsymbols. For example, the symbol E "contains" the symbol F. Because symbols with more strokes are drawn with more variability, they typically are scored more poorly by our recognizer. We, therefore, augment the scores of larger symbols by a weighted sum of the scores of similar-looking smaller symbols. The weighting coefficients are automatically determined ahead of time by measuring symbol-to-symbol similarity among the models in the symbol database using the elastic matching algorithm.

Finally, the augmented scores are normalized so that the highest-scoring symbol has a score of one. These final scores give the membership grades for the fuzzy relation r_Σ.

5.2 Geometric relations

The geometric relation membership functions are based on bounding box geometry. In our discussion in Sect. 4 of rectangular sets, we used a definition of bounding boxes based on the orderings $<_x$ and $<_y$. In this section, we are concerned with actual expression geometry, so we consider the natural definition of a bounding box. That is, a stroke's bounding box is the smallest axis-aligned rectangle that completely covers the stroke.

The membership function for the containment relation only considers the amount of overlap between the bounding boxes of t_1 and t_2. We define the *overlap proportion*, $ol(t_1, t_2)$, as the area of the bounding boxes' intersection divided by the area of the smaller box. The membership function for the containment relation is $\odot\left((t_1, e_1), (t_2, e_2)\right) = ol(t_1, t_2)$.

The membership functions for the other geometric relations incorporate the distance and angle between the bounding boxes of the observables. They are all of the form

$$\begin{aligned} &r\left((t_1, e_1), (t_2, e_2)\right) \\ &= \theta\left((t_1, e_1), (t_2, e_2)\right) \times d\left(t_1, t_2\right) \times \left(1 - \frac{1}{2} ol\left(t_1, t_2\right)\right), \end{aligned}$$

where θ is a scoring function based on angle and d is a distance-based penalty function.

The penalty function d ensures that observables satisfying the relations are within a reasonable distance of one another. To compute d, the size of each of t_1 and t_2 is calculated as the average of bounding box width and height. Next, a distance threshold t is obtained as half the average of the two sizes, clamped to the range $[1/6, 1/3]$ (measured in inches). If the distance Δ between the bounding boxes is less than t, then $d = 1$. Otherwise, d decreases linearly to zero at $\Delta = 3t$.

The points between which the angle is measured to compute θ are selected based on the relation r and the relational class of the expression e_2. The x coordinate is chosen centrally, but the choice of y coordinate varies. For the $\downarrow$ relation, it is chosen centrally. Table 1 summarizes how it is chosen

Table 1 Relational classes and measurement point selection

Rel. class	Ex. symbol(s)	y coordinate
Box	(none)	$\frac{top(t_2)+bottom(t_2)}{2}$
Default	A, +, (, $\int$, $\sum$, $\surd$, —,.	
Baseline	x,σ	$top(t_2) + height(t_2)/10$
Descender	q,ρ	$top(t_2) + height(t_2)/20$
Half-ascender	i,t	$top(t_2) + height(t_2)/3$
j	j	$top(t_2) + height(t_2)/4$

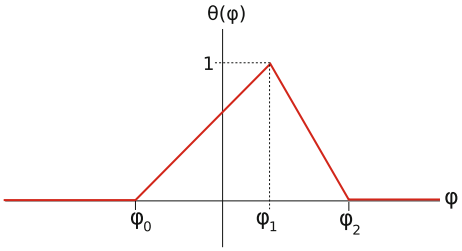

Fig. 8 θ function component of relation membership grade

Table 2 Angle thresholds for geometric relation membership functions

Relation	φ_0	φ_1	φ_2
→	−90	0	90
↗	−90	−37.5	0
↘	−20	35	160
↓	0	90	180

for the other relations (→ , ↗ , and ↘), as well as listing the relational classes used in our system. These choices, as well as the selection of threshold values below, were made through experimentation on a small dataset unrelated to those used in the evaluation section.

Given those measurement points, we measure the angle φ between them relative to the x axis. The angle-based function θ is triangular with three parameters $\varphi_0, \varphi_1, \varphi_2$, as follows:

$$\theta(\varphi) = \begin{cases} 0 & \text{if } \varphi < \varphi_0 \\ \frac{\varphi-\varphi_0}{\varphi_1-\varphi_0} & \text{if } \varphi_0 \le \varphi \le \varphi_1 \\ \frac{\varphi_2-\varphi}{\varphi_2-\varphi_1} & \text{if } \varphi_1 \le \varphi \le \varphi_2 \\ 0 & \text{if } \varphi > \varphi_2. \end{cases}$$

Figure 8 indicates the functions' behavior schematically. Table 2 lists the thresholds for each relation, in degrees. As mentioned above, these values were selected based on manual examination of a small dataset. Note that the y-axis increases downward.

6 Evaluation

We evaluated the accuracy of our math recognizer experimentally using two publicly available databases of handdrawn math expressions. For the first evaluation, we used an expression corpus developed by our research group at the University of Waterloo [24,25]. For the second, comparative evaluation, we obtained the data set and marking scripts used at the recent CROHME 2011 math recognition competition [31]. By using the CROHME data, we may compare our system against the four recognizers which participated in the competition, as well as a baseline recognizer developed by one of the competition organizers.

6.1 Evaluation on the Waterloo corpus

Many of the roughly 4,500 legible expressions in our corpus include mathematical notations not currently supported by our parser (e.g., set notation, multi-symbol variable names). Our test set thus contained 3,672 expressions from 20 writers, of which 53 expressions were common to all writers, and the remainder were randomly generated for each writer. Since this corpus was also intended to provide examples of individual symbols and the geometric relationships between symbols, it contains a large number of expressions with only one or two symbols. We used all of the expressions containing three or fewer symbols as training data for our symbol recognizer, as described in the methodology section below. In all, our grammar contains 49 non-terminals and 104 terminal symbols organized into 176 productions. The grammar is given in its entirety as Appendix A.

6.1.1 Correction count metric

Devising objective metrics for evaluating the real-world accuracy of a recognition system is difficult. Several authors have proposed accuracy metrics (e.g., [8,14,19,33,41]) based on implementation details specific to their particular recognizers. It is therefore difficult to directly compare their evaluation results to one another, or to apply their methodologies to our evaluation.

Recently, though, some authors have proposed some recognizer-independent evaluation schemes that rely only on the output of a recognizer and treat its implementation as a black box. Awal et al. [3] proposed an approach in which the accuracy of symbol segmentation, symbol recognition, relation, and expression recognition are reported separately. A positive feature of this approach is that it makes clear, in the case of incorrect expression recognition, which recognizer subsystem failed, and a version of it was used for the CROHME competition (no relation accuracy measurements were used). Sain et al. [34], following the intuition of Garain and Chaudhuri [14] that errors far from the dominant

baseline are less important than those on or near the main baseline, proposed a scheme called EMERS. In it, the edit distance between parse trees representing recognized and ground-truth expressions measures the accuracy of a recognition result. Edits are weighted so that those more deeply nested in the parse tree have less cost.

These approaches are both valuable in that they are easily implemented, permit objective comparisons between recognizers, and provide valuable feedback to recognizer developers. But they both measure the amount by which a recognized expression deviates from its ground-truth and do not consider whether the ground-truth was achievable by the recognizer at all. If a recognizer consistently fails to recognize a particular structure or symbol, the deviation from ground-truth may be small, even though the recognizer is effectively useless for that recognition task.

Our recognizer was designed 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. So as well as asking "Is the recognition correct?", we want to ask not "How many symbols were wrong?" or "How many transformation steps give the correct answer", but "Is the desired result attainable?", and "How much effort must the user expend to get it?" To a user, it is not necessarily the case that an error on the main baseline (say, recognizing $a + b$ as atb) is more incorrect than one on a secondary baseline (say, recognizing $n^{2^{1-\varepsilon}}$ as $n^{2l^{-\varepsilon}}$).

To answer these questions, we count the number of corrections that a user would need to make to a recognition result in order to obtain the correct parse. If an input is recognized correctly, then it requires zero corrections. Similarly, if it is recognized "almost correctly", it requires fewer corrections than if the recognition is quite poor. This 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 math recognizer, for instance, count the number of required corrections and obtain comparable measurements. Our evaluation scheme, thus, provides an abstract way to compare the performance of recognition systems without direct reference to their implementation details.

Liu et al. [23] also devised a user-based correction cost model for evaluating a diagram recognition system. They measured the physical time required to correct the errors in recognized diagrams. This approach would also be useful for evaluating our system, but the size of the expression corpus makes it impractical to manually test and correct the recognition results.

Instead, we have automated 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. It returns the number of corrections required to obtain the correct expression.

Algorithm 1 outlines this process. Our recognizer uses a "context" to refer to any node in the shared parse forest. So, as the algorithm descends into an expression tree, it can request alternatives for any particular subexpression and having any particular semantics. For example, if an expression intended to be $a + c$ was instead recognized as $a + C$, the algorithm would detect the correct top-level structure and the correct expression on the left side of the addition sign. On the right side, it would request alternatives "in context"; that is, using only the ink related to the c symbol and being feasible for the right side of an addition expression, as determined by the grammar.

Algorithm 1 $cc(g, e)$: Count the number of corrections required to match recognition results to ground-truth.

Require: A recognizer R, a ground-truth expression g, and the first recognition alternative e from R.
  if $e = g$ **then**
    return 0
  $errorhere \leftarrow 0$ // Indicates whether an error appears in the top level of the expression tree
  while e has different top-level semantics from g or uses a different partition of the input **do**
    $errorhere \leftarrow 1$
    $e \leftarrow$ the next alternative from R in this context
    if e is null (R provides no more alternatives) **then**
      return ∞
  for each pair of subexpressions e_i, g_i of e, g **do**
    $n_i \leftarrow$ recurse on R, g_i, e_i
    if $n_i = \infty$ **then**
      return ∞
  return $errorhere + \sum n_i$

The correction count produced by this algorithm is akin to a tree edit distance, except that it is constrained so that the edits must be obtainable through the recognizer's output. They cannot be arbitrary terminal or subtree substitutions.

6.1.2 Methodology

Although the correction count metric provides accuracy scores for both correct and incorrect recognition results, there are some types of recognition errors that it cannot account for.

For example, if an expression is recognized correctly except for one symbol, for which the correct alternative is not available, then the correction count will be ∞, even though the expression is "nearly correct".

To reduce the number of these types of results, we tested the recognizer in two scenarios. In the first, called *default*, we divided the 3,672 corpus transcriptions into training and testing sets. The training set contained all 1,536 transcriptions having fewer than four symbols. The remaining 2,136 transcriptions formed the testing set. These transcriptions contained between four and 23 symbols each, with an average of seven. All of the symbols were extracted from the training set and used to augment our pre-existing symbol database. The pre-existing database contained samples of each symbol written by one to three writers and is unrelated to the evaluation data set. It was used to ensure baseline coverage over all symbol types.

The second scenario, called *perfect*, evaluated the quality of expression parsing in isolation from symbol recognition. In it, the same testing set of 2,136 transcriptions was used, but the terminal symbol relation was evaluated so as to exactly match ground truth. There were no alternatives for the parser to choose between, and no ambiguities in stroke grouping. This scenario, thus, represents a "best possible world" for the parser and gives an upper bound on its real-world performance.

Each transcription used for testing may be placed into one of the following four categories:

1. *Correct*: No corrections were required. The top-ranked interpretation was correct.
2. *Attainable*: The correct interpretation was obtained from the recognizer after one or more corrections.
3. *Incorrect*: The correct interpretation was not obtained from the recognizer.
4. *Infeasible*: The symbol recognizer failed to provide the correct symbol candidates to the parser, preventing correct recognition.

Note that the "incorrect" and "infeasible" categories both refer to incorrectly recognized transcriptions. Using two categories allows us to distinguish between those transcriptions for which the correct expression structure was not identified at all ("incorrect") and those for which the correct structure may have been identified, but correct recognition was impossible because one or more symbols did not have the correct candidate available for the parser to choose ("infeasible"). Our symbol classifier distinguishes between over 100 symbols, but is significantly less accurate than the relation classifier. As such, it is useful to divide incorrectly recognized transcriptions into two categories, so as to know which of the two subsystems caused the failure.

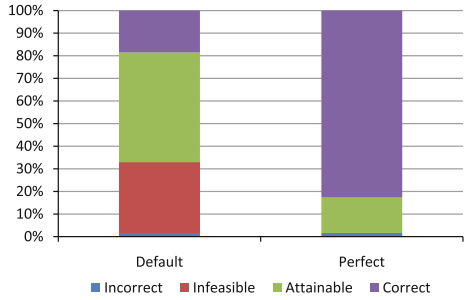

Fig. 9 Categorization of transcriptions during evaluation

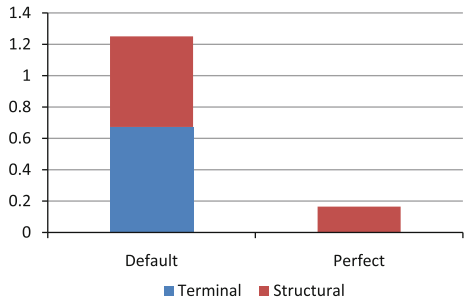

Fig. 10 Average correction count during evaluation

6.1.3 Accuracy evaluation

Figure 9 summarizes the categorization of the testing set in both scenarios. Figure 10 shows the correction count in each scenario, averaged over correct and attainable transcriptions. The correction count is divided into terminal corrections (i.e., corrections that only change terminal symbol identity) and structural corrections (i.e., all others).

Overall, about 19 % of the transcriptions were recognized correctly in the default scenario, compared with about 83 % in the perfect scenario. 67 % of the transcriptions were attainable in the default scenario with about 1.2 corrections on average, while over 98 % were attainable in the perfect scenario, with about 0.2 corrections on average. If we consider only feasible transcriptions (those for which the correct symbol identities were detected, but not necessarily top-ranked, by the symbol classifier), then 27 % of the transcriptions were recognized correctly, and 98 % were attainable in the default scenario. (The figures for the perfect scenario are unchanged, as all transcriptions were feasible in that case.) This attainability rate is very close to the one attained in the perfect scenario, but the correct rate is still quite low, reflecting the difficulty of simultaneously resolving all symbol classification ambiguity perfectly.

When an expression was attainable, it rarely required more than a few corrections. The top graph in Fig. 11 shows how many expressions required various numbers of corrections in the default scenario. The bottom graph shows the same frequency measurement for the perfect scenario. This indicates

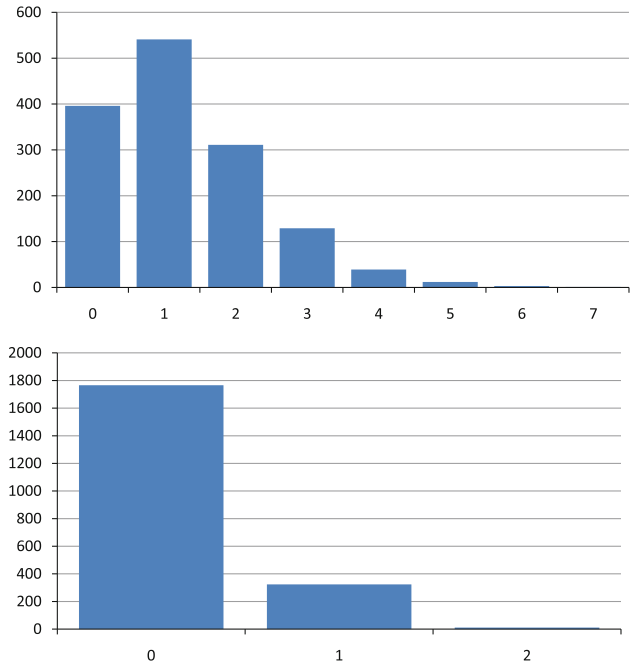

Fig. 11 Expression frequency for correction counts in the default (*top*) and perfect (*bottom*) scenarios

that it would usually be fairly simple and quick for a user to manually correct recognition errors and obtain the correct results.

Aside from symbol classification errors, there were two main reasons why transcriptions were classified as incorrect.

1. *Violation of the ordering assumption.* Although the ordering assumption of Sect. 4 was motivated by the general block-based structure of math notation, it was occasionally violated by transcriptions in our corpus. In the left expression of Fig. 12, the right parenthesis extends to the left beyond the left edge of the "2" symbol. When ordered by $<_x$, the parenthesis therefore comes first. Assuming correct symbol recognition, the parser therefore sees, from left to right, the subsequence (9.)2, preventing correct recognition of the parenthesized number. Violations of the ordering assumptions were also sometimes caused by more reasonable-looking expressions, as in the right expression of the figure. Here, the C symbol begins to the right of the y symbol. Again, this causes the C to be considered by the parser to come strictly after the y and prevents the lower range of the summation from being considered during the parser's recursive process of input subdivision.
2. *Geometric relation failure.* When symbol or subexpression bounding boxes are arranged such that the distance or angle between them falls outside of the ranges described in Sect. 5, then the arrangement is assigned a membership grade of zero. Such arrangements are never attainable, which sometimes caused errors. Again, these errors were often caused by messy writing and incorrect transcriptions (e.g., the left expression in Fig. 13 was intended to contain the subexpressions qY and jX, but the writing appears to read qy and jx), but were also caused by reasonable transcriptions (e.g., in the right expression, the subexpression $\sum \Pi - \int Ddy$ was intended to be a superscript of b, but was not recognized as such due to geometric relation failure).

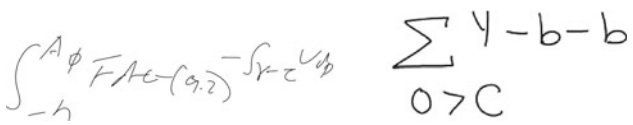

Fig. 12 Expressions violating the ordering assumption

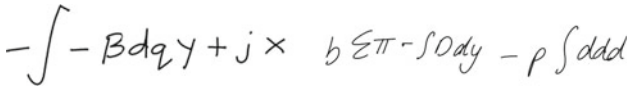

Fig. 13 Expressions for which relation classification failed

6.1.4 Performance evaluation

The worst-case runtime estimates given in Sect. 4 are quite pessimistic. In actual use, the time required to parse an input and report results depends on many factors that are difficult to quantify generally, including the number of distinct rectangular subsets of the input, the number of subset pairs which have non-zero membership grade in relations (and in how many relations they have such scores), the amount of ambiguity in the symbols used and in the placement of the strokes making them up, and so on. It is therefore interesting to investigate how parse table size varies with input size and to, thus, obtain empirical estimates of the parser's behavior in realistic cases.

We first consider the number of rectangular subsets of the input actually used by the parser. (Recall that, in the worst case, there are $\binom{n}{4} = \mathcal{O}(n^4)$ distinct rectangular subsets.) The graphs in Fig. 14 show log–log plots of the number of subsets considered with respect to the number of strokes appearing in the input. The default scenario is shown on the top, and the perfect scenario on the bottom. Based on this data, the parser explored on the order of $n^{2.9}$ subsets, on average, in the default scenario, and $n^{2.2}$ subsets in the perfect scenario. This second figure mirrors findings by Liang et al. [22] in their work on rectangular hulls.

The number of parse table cells depends primarily on the grammar (which is fixed throughout this evaluation), the number of rectangular subsets, and the number of pairs of subsets that are members of the fuzzy geometric relations. Figure 15 shows log–log plots of the number of parse table cells with respect to the number of input strokes. As before,

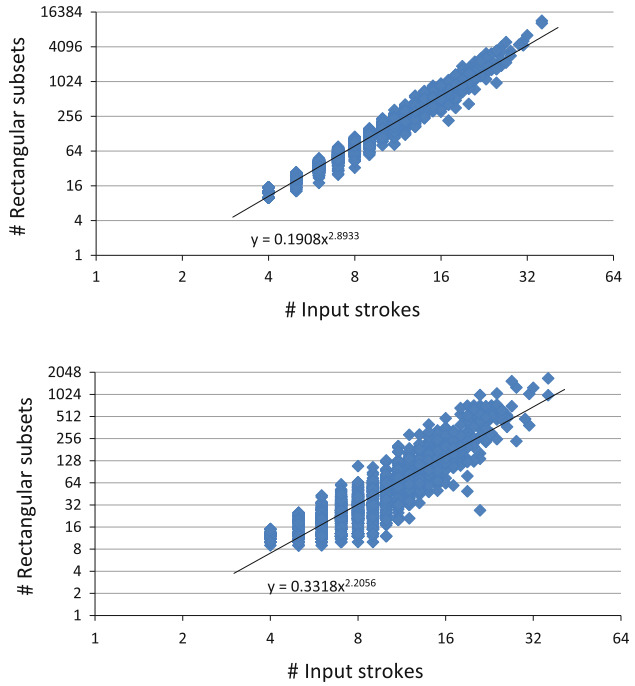

Fig. 14 log–log plots of number of rectangular subsets w.r.t. input size

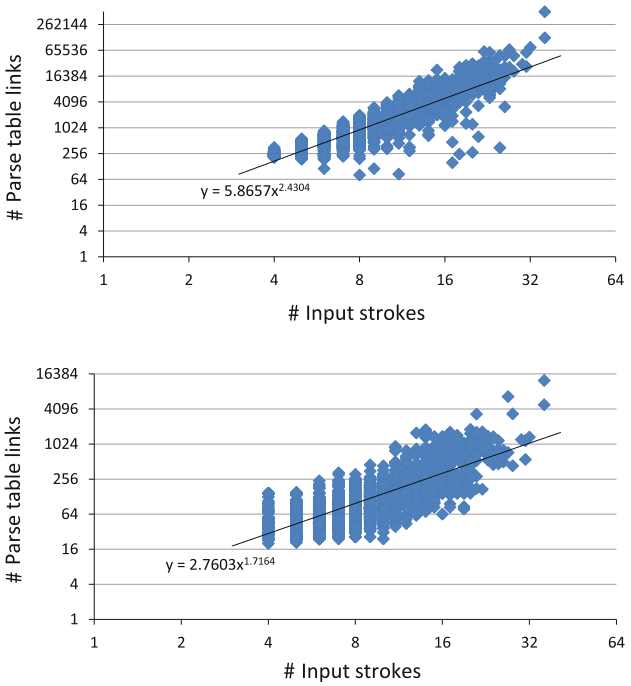

Fig. 16 log–log plots of number of parse table links w.r.t. input size

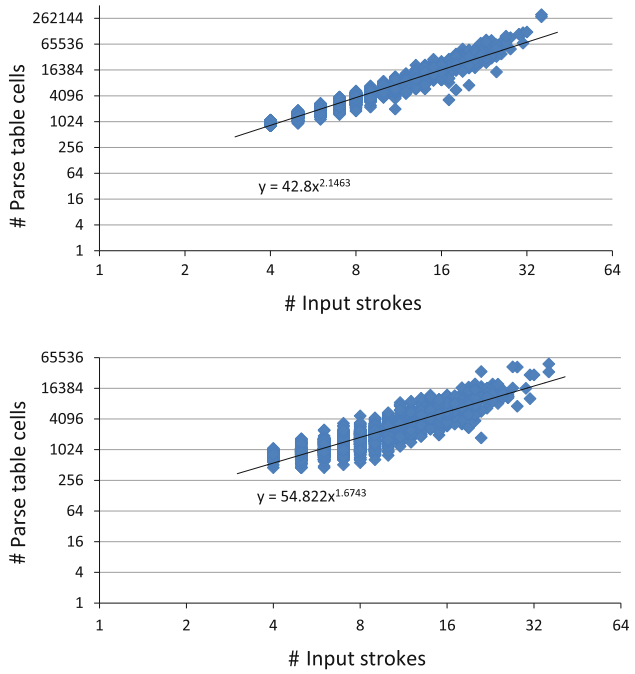

Fig. 15 log–log plots of number of parse table cells w.r.t. input size

the top graph corresponds to the default scenario, while the bottom graph corresponds to the perfect scenario. In the default scenario, the parse table contained on the order of $n^{2.1}$ cells on average, while it contained $n^{1.7}$ in the perfect scenario. It is interesting that the exponents are lower when counting parse cells than when counting subsets, indicating that not all of the rectangular subsets explored are involved in any parses. This is possible because the forest construction algorithm from Sect. 4 parses a production $A_0 \stackrel{r}{\Rightarrow} A_1 \cdots A_k$ on a partition $t_1 \cup \cdots \cup t_k$ by first parsing A_1 on t_1, then A_2 on t_2, and so on, until it reaches the end of the production, or a subparse fails. So some of the subsets t_i may be explored without contributing to any useful parse results. It is possible that a heuristic more sophisticated than the rectangular set restriction would reduce the amount of such wasted effort. At the same time, the small size of the parse table relative to the number of subsets available for parsing shows that our parser does a good job of ignoring unproductive subsets since they are identified early in the parsing process.

Finally, we consider the number of links between cells in the parse table (represented by sets of arrows emanating from the AND nodes in Fig. 3). The number of alternative interpretations the tree extraction algorithm must consider is directly linked to the number of inter-cell parse table links. The top graph in Fig. 16 indicates how link count scales with input size in the default scenario, and the bottom graph shows the same measurements for the perfect scenario. As before, we use this data to estimate that, on average, the parse table contains on the order of $n^{2.4}$ links in the default scenario and $n^{1.7}$ in the perfect scenario. These figures indicate that the parse graph is typically quite sparsely linked. That is, the parser identifies only a small number of ways to parse a given production on a given input subset. Intuitively, this makes sense: to parse a production like `[EXPR] + [EXPR]`, one must have a symbol that looks like a plus sign, surrounded

Table 3 Evaluation on CROHME 2011 corpus

Recognizer	Stroke reco.	Symbol seg.	Symbol reco.	Expression reco.
Part 1 of corpus				
MathBrush	71.73	84.09	85.99	32.04
Rochester Institute of Technology (USA)	55.91	60.01	87.24	7.18
Sabanci University (TR)	20.90	26.66	81.22	1.66
Universitat Politècnica de València (ES)	78.73	88.07	92.22	29.28
Institute for Language and Speech Processing (GR)	48.26	67.75	86.30	0.00
IRCCyN-IVC (FR)	78.57	87.56	91.67	40.88
Part 2 of corpus				
MathBrush	66.82	80.26	86.11	20.11
Rochester Institute of Technology (USA)	51.58	56.50	91.29	2.59
Sabanci University (TR)	19.36	24.42	84.45	1.15
Universitat Politècnica de València (ES)	78.38	87.82	92.56	19.83
Institute for Language and Speech Processing (GR)	52.28	78.77	78.67	0.00
IRCCyN-IVC (FR)	70.79	84.23	87.16	22.41

by expressions, all plausibly horizontally adjacent. For most written addition expressions, there will be few reasonable ways of breaking up the input to satisfy these constraints. That the parse table links reflect this fact illustrates the efficacy of the terminal symbol milestone and spatial relation test heuristics built into our parser.

6.2 Evaluation on the CROHME 2011 corpus

A recent math recognition competition compared the accuracy of five recognizers on the CROHME 2011 corpus [31]. To compare these five recognizers to our own, we acquired the corpus data and associated marking scripts and evaluated the MathBrush recognizer on the competition data.

The data are divided into two parts, each of which includes training and testing data. The first part includes a relatively small selection of mathematical notation, while the second includes a larger selection. For details, refer to the competition paper [31]. According to competition organizers, the recognizers were typically not trained exclusively on the training data. We, therefore, trained our symbol recognizer on the same data as in the previous evaluation, augmented by all of the symbols appearing in the appropriate part of the CROHME training data. For recognition, we used the grammars provided in the competition documentation by converting them into the file format recognized by our parser.

The accuracy of our recognizer was measured using a perl script provided by the competition organizers. In this evaluation, only the top-ranked parse was considered. There are four accuracy measurements. Stroke reco. indicates the percentage of input strokes that were correctly recognized and placed in the parse tree. Symbol seg. indicates the percentage of symbols for which the correct strokes were properly grouped together. Symbol reco. indicates the percentage of symbol recognized correctly, out of those correctly grouped. Finally, expression reco. indicates the percentage of expressions for which the top-ranked parse tree was exactly correct (corresponding to a "correct" result in our classification scheme for the previous evaluation).

The results are shown in Table 3, which also reproduces for comparison purposes the updated competition results appearing on the CROHME website.[1] Note that the IRCCyN-IVC recognizer was developed by one of the competition organizers and did not directly participate in the competition. Our MathBrush parser obtained higher expression recognition rates than the competition participants, though they were still lower than those of the IRCCyN-IVC recognizer. However, our symbol segmentation and, in particular, symbol recognition rates were not as high as those of some competition participants. These lower-level subsystems would therefore be promising areas on which to focus in future work, so as to improve our overall recognition rates.

7 Conclusions and future work

We have described a recognition system for handwritten mathematical notation, which reports all recognizable interpretations of users' writing and allows users to repair incorrect symbols or subexpressions, while maintaining a runtime

[1] http://www.isical.ac.in/~crohme2011/result.html.

that is appropriate for real-time use. To build this system, we introduced a new fuzzy r-CFG formalism designed 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 ambiguities found in natural languages.

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 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 extended the idea of syntactic classes to one of relational classes. We plan to expand our selection of relational classes from representing symbol shapes to subexpression shapes, so that our relation membership functions can account for superscript and subscript geometry, for example, rather than treating all multi-symbol expressions as equivalent. However, parse speed decreases significantly when the parser must choose between many classes. We, therefore, plan to investigate more targeted assumptions on the behavior of grammar relations which will allow a greater degree of context sensitivity without a significant performance penalty. The rule-based approach used in this paper does not scale well to large sets of relational classes, for which empirical approaches are better-suited. At present, we lack sufficient empirical data to condition the class-specific relation membership functions, but the creation of new handwritten corpora should provide useful training data.

The fuzzy membership grade functions defined in Sect. 3, while reasonable, cannot naturally account for some types of information that might be useful during recognition. (For example, subexpression co-occurence frequencies, the distribution of subexpressions within larger expressions, etc.) We plan to investigate the application of Bayesian probability theory and formal statistical methods to the two-dimensional parsing problem in a way that avoids dependence on writing order. As well as the examples mentioned above, there are several existing components of our system, such as the relation classifier, for which it seems likely that appropriately conditioned statistical information would be useful. 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 per-strokes efficiency can be improved while keeping available all relevant recognition results.

Appendix A: Grammar specification

Below are the productions used in our math recognition grammar. Note that, although matrices are included in the grammar so that they may be part of mathematical expressions, the [MATRIX-INTERNAL] non-terminal causes the parser to invoke a specialized matrix recognition engine instead of parsing the corresponding rectangular set in the usual way. This special system is necessary to ensure that the matrices are well-formed, as it is impossible in a context-free language to specify in a general way that each row contains the same number of entries. The matrix analysis engine will be the subject of a future report.

[EXPR] ⇒ [REL-EXPR] | [REL-TERM]

[REL-EXPR] $\overrightarrow{\Rightarrow}$ [REL-TERM][REL-OP][EXPR]

[REL-OP] ⇒ =|≠|<|>|≤|≥

[REL-TERM] ⇒ [ADD-EXPR] | [ADD-LEAD-TERM]

[ADD-EXPR] $\overrightarrow{\Rightarrow}$ [ADD-LEAD-TERM][ADD-OP][REL-TERM]

[ADD-OP] ⇒ + | − | ±

[ADD-LEAD-TERM] ⇒ [ADD-TERM] | [NEG]

[NEG] $\overrightarrow{\Rightarrow}$ −[ADD-TERM]

[ADD-TERM] ⇒ [MULT] | [NUM] | [MULT-TRAIL]

[MULT] $\overrightarrow{\Rightarrow}$ [MULT-LEAD-TERM][MULT-RHS]

[MULT-LEAD-TERM] ⇒ [MULT-TERM] | [NUM]

[MULT-RHS] ⇒ [MULT-REC] | [MULT-TRAIL]

[MULT-REC] $\overrightarrow{\Rightarrow}$ [MULT-TERM][MULT-RHS]

[MULT-TRAIL] ⇒ [MULT-TERM] | [INTEGRAL] | [SUM] | [LIMIT]

[MULT-TERM] ⇒ [VAR-EXPR] | [FRAC] | [SUP] | [ROOT]

[VAR-EXPR] ⇒ [VAR-TERM] | [FUNC] | [FENCED] | [MATRIX]

[VAR-TERM] ⇒ [LETTER] | [SUBSCRIPT]

[SUBSCRIPT] $\searrow$ [LETTER][SUB-BASE]

[SUB-BASE] ⇒ [NUM] | [VAR-TERM]

[FENCED] ⇒ [PARENS] | [BRACKETS] | [BRACES]

[PARENS] $\overrightarrow{\Rightarrow}$ ([REL-TERM])

[BRACKETS] $\overrightarrow{\Rightarrow}$ [[REL-TERM]]

$$
\begin{aligned}
\texttt{[BRACES]} &\overset{\rightrightarrows}{\Rightarrow} \{\texttt{[REL-TERM]}\}\\
\texttt{[MATRIX]} &\overset{\rightrightarrows}{\Rightarrow} [\texttt{[MATRIX-INTERNAL]}]\\
\texttt{[FUNC]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[FUNC-LHS][PARENS]}\\
&\quad \mid \texttt{[FUNC-SPECNAME][ADD-TERM]}\\
\texttt{[FUNC-SPECNAME]} &\overset{\rightrightarrows}{\Rightarrow} \sin \mid \cos \mid \tan \mid \exp \mid \log \mid \ln \mid \operatorname{erf}\\
\texttt{[FUNC-LHS]} &\Rightarrow \texttt{[FUNC-NAME]} \mid \texttt{[FUNC-SUP]}\\
\texttt{[FUNC-NAME]} &\Rightarrow \texttt{[VAR-TERM]} \mid \texttt{[FUNC-SPECNAME]}\\
\texttt{[FUNC-SUP]} &\overset{\nearrow}{\Rightarrow} \texttt{[FUNC-NAME][REL-TERM]}\\
\texttt{[FRAC]} &\overset{\downarrow}{\Rightarrow} \texttt{[REL-TERM]}\frac{}{}\texttt{[REL-TERM]}\\
\texttt{[SUP]} &\overset{\nearrow}{\Rightarrow} \texttt{[SUP-BASE][REL-TERM]}\\
\texttt{[SUP-BASE]} &\Rightarrow \texttt{[VAR-EXPR]} \mid \texttt{[ROOT]}\\
\texttt{[ROOT]} &\overset{\circlearrowright}{\Rightarrow} \sqrt{}\texttt{[REL-TERM]}\\
\texttt{[SUM]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[SUM-LIMITS][REL-TERM]}\\
\texttt{[SUM-LIMITS]} &\Rightarrow \sum\\
\texttt{[SUM-LIMITS]} &\overset{\downarrow}{\Rightarrow} \sum \texttt{[EXPR]}\\
\texttt{[SUM-LIMITS]} &\overset{\downarrow}{\Rightarrow} \sum \texttt{[REL-TERM]} \sum \texttt{[EXPR]}\\
\texttt{[INTEGRAL]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[INT-LIMITS][REL-TERM]}\\
&\quad d\,\texttt{[VAR-TERM]}\\
\texttt{[INT-LIMITS]} &\Rightarrow \int\\
\texttt{[INT-LIMITS]} &\overset{\downarrow}{\Rightarrow} \int \texttt{[REL-TERM]}\\
\texttt{[INT-LIMITS]} &\overset{\downarrow}{\Rightarrow} \texttt{[REL-TERM]} \int\\
\texttt{[INT-LIMITS]} &\Rightarrow \texttt{[INT-UPRGT-LIMITS]}\\
\texttt{[INT-UPRGT-LIMITS]} &\overset{\nearrow}{\Rightarrow} \int \texttt{[REL-TERM]}\\
\texttt{[INT-LIMITS]} &\overset{\searrow}{\Rightarrow} \int \texttt{[REL-TERM]}\\
\texttt{[INT-LIMITS]} &\overset{\downarrow}{\Rightarrow} \texttt{[REL-TERM]} \int \texttt{[REL-TERM]}\\
\texttt{[INT-LIMITS]} &\overset{\searrow}{\Rightarrow} \texttt{[INT-UPRGT-LIMITS][REL-TERM]}\\
\texttt{[LIMIT]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[LIM-LHS][REL-TERM]}\\
\texttt{[LIM-LHS]} &\overset{\downarrow}{\Rightarrow} \texttt{[LIM-TEXT][LIM-APPROACH]}\\
\texttt{[LIM-TEXT]} &\overset{\rightrightarrows}{\Rightarrow} \lim\\
\texttt{[LIM-APPROACH]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[VAR-TERM]} \rightarrow \texttt{[REL-TERM]}\\
\texttt{[NUM]} &\Rightarrow \texttt{[INT]} \mid \texttt{[FLOAT]} \mid \infty\\
\texttt{[INT]} &\Rightarrow \texttt{[DIGIT]}\\
\texttt{[INT]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[DIGIT][INT]}\\
\texttt{[DIGIT]} &\Rightarrow 0 \mid 1 \mid \cdots \mid 9\\
\texttt{[FLOAT]} &\overset{\rightrightarrows}{\Rightarrow} \texttt{[FLOAT-LEAD][INT]}\\
\texttt{[FLOAT-LEAD]} &\overset{\searrow}{\Rightarrow} \texttt{[INT]}\cdot\\
\texttt{[LETTER]} &\Rightarrow a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z\\
&\quad \mid \alpha \mid \beta \mid \gamma \mid \delta \mid \epsilon \mid \pi \mid \sigma \mid \theta \mid \lambda \mid \mu \mid \phi\\
&\quad \mid \psi \mid \rho \mid \tau \mid \xi \mid \zeta \mid \Delta \mid \Gamma \mid \Omega \mid \Pi \mid \Sigma
\end{aligned}
$$

References

1. Álvaro, F., Sánchez, J.-A., Benedí, J.-M.: Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In: Proceedings of the International Conference on Document Analysis and Recognition, pp. 1225–1229 (2011)
2. Anderson, R.H.: Syntax-directed Recognition of Hand-printed Two-dimensional Mathematics, Ph.D. thesis, Harvard University (1968)
3. Awal, A.-M., Mouchère, H., Viard-Gaudin, C.: The problem of handwritten mathematical expression recognition evaluation. In: Proceedings of the International Conference on Frontiers in Handwriting Recognition, pp. 646–651 (2010)
4. Belaid, A., Haton, J.-P.: A syntactic approach for handwritten mathematical formula recognition. IEEE Trans. Pattern Anal. Mach. Intell. **6**(1), 105–111 (1984)
5. Blostein, D., Grbavec, A. : Recognition of mathematical notation. In: Wang, P.S.P., Bunke, H. (eds.) Handbook on Optical Character Recognition in Document Analysis, pp. 557–582. World Scientific, Singapore (1996)
6. Blostein, D. : Math-literate computers, Calculemus/MKM. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Lecture Notes in Computer Science, vol. 5625, pp. 2–13. Springer, Berlin (2009)
7. Caraballo, S.A., Charniak, E.: New figures of merit for best-first probabilistic chart parsing. Comput. Linguist. **24**(2), 275–298 (1998)
8. Chan, K.-F., Yeung, D.-Y.: Error detection, error correction and performance evaluation in on-line mathematical expression recognition. In: On-Line Mathematical Expression Recognition, Pattern Recognition (1999)
9. Chan, K.-F., Yeung, D.-Y.: Mathematical expression recognition: a survey. Int. J. Doc. Anal. Recogn. **3**, 3–15 (1999)
10. Chou, P.A.: Recognition of equations using a two-dimensional stochastic context-free grammar. In: Proceedings of the SPIE, Visual Communication and Image Processing IV, vol. 1199, pp. 852–863 (1989)
11. Costagliola, G., De Lucia, A., Orefice, S., Tortora, G. : Positional grammars: a formalism for lr-like parsing of visual languages. In: Marriott, K., Meyer, B. (eds.) Visual Language Theory, pp. 171–192. Springer, New York (1998)
12. Fitzgerald, J.A., Geiselbrechtinger, F., Kechadi, T.: Mathpad: A fuzzy logic-based recognition system for handwritten mathematics, Document Analysis and Recognition. ICDAR 2007. Ninth International Conference on, vol. 2, Sept. 2007, pp. 694–698 (2007)
13. Garain, U., Chaudhuri, B.B.: Recognition of online handwritten mathematical expressions. Syst. Man. Cybern. Part B: Cybern. IEEE Trans. **34**(6), 2366–2376 (2004)
14. Garain, U., Chaudhuri, B.: A corpus for ocr research on mathematical expressions. Int. J. Doc. Anal. Recognit. **7**(4), 241–259 (2005)
15. Grune, D., Jacobs, C.J.H.: Parsing Techniques: A Practical Guide, 2 edn. Springer, Berlin (2008)
16. Hull, J.: Recognition of mathematics using a two-dimensional trainable context-free grammar, Master's thesis, Massachusetts Institute of Technology (1996)
17. Labahn, G., Lank, E., MacLean, S., Marzouk, M., Tausky, D.: Mathbrush: a system for doing math on pen-based devices. In: The Eighth IAPR Workshop on Document Analysis Systems (DAS), pp. 599–606 (2008)
18. Bernard, L.: Towards a Uniform Formal Framework for Parsing, Current Issues in Parsing Technology. pp. 153–171. Kluwer, Dordrecht (1991)

19. Laviola, Jr. J.J.: Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations, Ph.D. thesis, Brown University (2005)
20. Lee, E.T., Zadeh, L.A. : Note on fuzzy languages. In: Klir, G.J., Yuan, B. (eds.) Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems, pp. 69–82. World Scientific, Singapore (1996)
21. Li, C., Miller, T.S., Zeleznik, R.C., LaViola Jr. J.J.: Algosketch: Algorithm sketching and interactive computation. In: Proceedings of Sketch-Based Interfaces and Modeling (2008)
22. Liang, P., Narasimhan, M., Shilman, M., Viola, P.: 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, pp. 1172–1177 (2005)
23. Liu, W., Zhang, L., Tang, L., Dori, D.: Cost evaluation of interactively correcting recognized engineering drawings, Lecture Notes in Computer Science, no. 1941, pp. 329–334 (2000)
24. MacLean, S., Labahn, G., Lank, E., Marzouk, M., Tausky, D.: Grammar-based techniques for creating ground-truthed sketch corpora. Int. J. Doc. Anal. Recogn. **14**, 65–74 (2011)
25. MacLean, S.: Tools for the efficient generation of hand-drawn corpora based on context-free grammars. In: Third International Workshop on Pen-Based Mathematics Computing, http://www.orcca.on.ca/conferences/cicm09/workshops/PenMath/programme-hand.html (2009)
26. MacLean, S., Labahn, G.: Elastic matching in linear time and constant space. In: Proceedings of Ninth IAPR International Workshop on Document Analysis Systems, (Short paper), pp. 551–554 (2010)
27. MacLean, S., Labahn, G.: Recognizing handwritten mathematics via fuzzy parsing, Technical Report CS-2010-13, School of Computer Science, University of Waterloo (2010)
28. Manning, C.D., Schuetze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge (1999)
29. Marriott, K., Meyer, B., Wittenburg, K.B. : A survey of visual language specification and recognition. In: Marriott, K., Meyer, B. (eds.) Visual Language Theory, pp. 5–85. Springer, New York (1998)
30. Miller, E.G., Viola, P.A.: Ambiguity and constraint in mathematical expression recognition. In: Proceedings of the Fifteenth National/tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, pp. 784–791 (1998)
31. Mouchère, H., Viard-Gaudin, C., Garain, U., Kim, D.H., Kim, J.H.: Crohme2011: Competition on recognition of online handwritten mathematical expressions. In: Proceedings of the 11th International Conference on Document Analysis and Recognition (2011)
32. Rhee, T.H., Kim, J.H.: Efficient search strategy in structural analysis for handwritten mathematical expression recognition. Pattern Recogn. **42**(12), 3192–3201 (2009)
33. Rutherford, I.: Structural analysis for pen-based math input systems, Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo (2005)
34. Sain, K., Dasgupta, A., Garain, U.: Emers: a tree matching-based performance evaluation of mathematical expression recognition systems. IJDAR **14**(1), 75–85 (2011)
35. Shi, Y., Li, H., Soong, F.K.: A unified framework for symbol segmentation and recognition of handwritten mathematical expressions. In: Ninth International Conference on Document Analysis and Recognition, pp. 854–858 (2007)
36. Tomita, M.: Parsing 2-dimensional languages. In: Tomita, M. (ed.) Current Issues in Parsing Technology, pp. 277–289. Kluwer, Dordrecht (1991)
37. Toyota, S., Uchida, S., Suzuki, M.: Structural analysis of mathematical formulae with verification based on formula description grammar. Doc. Anal. Syst. **VII**:153–163 (2006)
38. Unger, S.H.: A global parser for context-free phrase structure grammars. Commun. ACM **11**, 240–247 (1968)
39. Winkler, H.-J., Fahrner, H., Lang, M.: A soft-decision approach for structural analysis of handwritten mathematical expressions. In: Proceedings of International Conference on Acoustics, Speech and Signal Processing, pp. 2459–2462 (1995)
40. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**, 338–353 (1965)
41. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognizing mathematical expressions using tree transformation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(11), 1455–1467 (2002)
42. Zhang, L., Blostein, D., Zanibbi, R.: Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In: Proceedings of Eighth International Conference on Document Analysis and Recognition, vol. 2, pp. 972–976 (2005)