Optical Character Recognition and Parsing of Typeset Mathematics *

Richard J. Fateman
Taku Tokuyasu
Benjamin P. Berman
Nicholas Mitchell†
Computer Science Division, EECS Department
University of California at Berkeley

November 8, 1995

Abstract

There is a wealth of mathematical knowledge that could be potentially very useful in many computational applications, but is not available in electronic form. This knowledge comes in the form of mechanically typeset books and journals going back more than one hundred years. Besides these older sources, there are a great many current publications, filled with useful mathematical information, which are difficult if not impossible to obtain in electronic form. Our work intends to encode, for use by computer algebra systems, integral tables and other documents currently available in hardcopy only. Our strategy is to extract character information from these documents, which is then passed to higher-level parsing routines for further extraction of mathematical content (or any other useful two-dimensional semantic content). This information can then be output as, for example, a Lisp or TEX expression. We have also developed routines for rapid access to this information, specifically for finding matches with formulas in a table of integrals. This paper reviews our current efforts, and summarizes our results and the problems we have encountered.

*This work was supported in part by NSF Grants numbers CCR-9214963 and IRI-9411334, and by NSF Infrastructure Grant number CDA-8722788.

†Present Address: University of California, San Diego, Computer Science and Engineering Department, 9500 Gilman Drive, La Jolla, CA 92093-0114

1 Introduction

This is our problem: There exist many integral tables in printed form, such as the one compiled by Bierens de Haan [1]. Figure 1 is a sample entry from this table. It would be very useful to have this information easily accessible in electronic form. For example, during a session with one of the popular computer algebra systems (CAS), an integral formula might be necessary to continue a computation. The program could query the integral table, which would return with the answer in some appropriate format. Besides being fast, this approach gives a CAS access to a vast amount of information, including integrals which no system can evaluate algorithmically at present. Aside from having an army of human assistants typing in formulas (with the concomitant input errors), the problem of converting from print to an equivalent electronic form becomes one of developing methods to automatically recognize mathematical expressions from scanned images.[1]

This paper documents our present efforts to develop such a system. Our purpose in writing this paper is three-fold:

- to publicize the problems and prospects for automatic recognition of mathematical notation to both the mathematical and optical character recognition (OCR) communities;
- to present a two-dimensional parsing method, with particular attention paid to details which have been glossed over in the past;
- to describe this project within the overall context of creating a fast automatic database of mathematical formulas.

Because of its interesting structure, the mathematical formula context is a fruitful arena in which to experiment with OCR and two-dimensional (2-D) parsing methods, and we believe it will remain an important challenge for some time to come.

Recognition of mathematics has in fact been enjoying a resurgence of interest. For instance, the most recent ICDAR proceedings [3] contains at least five papers related to this area. Our application differs from these presentations in that we cannot assume ideal print quality, as appears to be standard in papers in this area. We are interested in recognizing specific texts of variable print quality, and hence are faced with problems that are particularly difficult to overcome.

We have also kept in mind how this work might fit into the overall framework of document processing and the possibility of using a more general formulation of the

[1] Academic Press has recently advertised that the integral table by Gradshteyn and Ryzhik [2] will soon be available on CD-ROM. While we have yet to see the CD in action, we believe our problem is of sufficient general interest to warrant further research, independent of whether the AP product has achieved full recognition of the printed text.

problem [4]. As our major interest lies in ways to make mathematical information accessible, a solution which is not completely satisfying from a theoretical point of view may nevertheless be sufficient, and, with the occasional call on human intervention for corrections, possibly faster.

Aside from a number of special problems such as novel notations introduced for a handful of formulas, or particularly idiosyncratic typesetting, the central problem we face at this point is dealing with noisy input data. In order to deal with this issue, we are presently developing a top-down approach in which the recognition and parsing modules interact more closely. That is, we can use structural information to help us guess about the characters, overcoming our (incorrect) assumption of perfect character recognition.

In this paper, we present our current system, which is geared towards handling the noise-free case. To preview our current situation, given a noise-free input image, we can indeed recognize all the characters and parse them into a Lisp expression appropriate for further computation or compact storage. We have also implemented a storage and retrieval method to access stored formulas by "approximate" matching.[2] This table and access method would normally be used by a computer algebra system, but could be used by a human user to "look up an integral." As the table has already been described in detail [5], we concentrate in this paper on the OCR and parsing algorithms.

The paper is organized as follows. In the next section, we describe a run through the system as it presently stands. Following that, two sections describe our current approach to OCR and 2-D parsing in more detail. OCR experts may wish to skim these sections for the topics that are of interest to them. We briefly describe the construction of the integral table, and then summarize our efforts.

2 Overview of Design

It is convenient to reduce the set of tasks to neat modules: scanning and segmentation of a page into "zones"; recognition of bits within zones into characters and lines; grouping of characters into token "boxes" (vertical grouping of lines to form symbols like "=", attaching the dot over the i to its body, horizontal grouping of letters to form words such as cos, sin, tan); parsing of these boxes to mathematics expressed in Lisp. Each of the modules works with neatly specified input-output relations so that the total recognition process is achieved by merely hooking together modules.

Such a modular decomposition generally requires a level of precursor perfection: namely, the design is based on the assumption that the inputs to each module are

[2] This technique works whether the formulas were originally obtained by optical character recognition (OCR) or by other methods such as direct data entry or derivation by computation.

correct. Only then can we expect the output to be correct.

We had hoped to rely on commercial OCR programs for the first level of correct input — the location and identification of characters and symbols. Our observations suggest, however, that commercial programs cannot handle mathematical formulas, and we therefore decided to write our own routines. We have implemented a set of traditional OCR pre-processing routines as well. For instance, we use a de-skewing algorithm to compensate for a rotated image, and a noise filter to remove spurious dots and other artifacts of the scanning process. We estimate that under the assumption of noise-free input, only 25% of the code we have written to date would be needed; perhaps this 25% would constitute 5% of a more fully bullet-proofed program. Such a program would still make mistakes on some inputs.[3]

We claim no particular originality in any of the given algorithms, for which Baird [6] and O'Gorman [7] provide keys to the literature.

In order to demonstrate the flow of our ideas, we have concentrated on developing a prototype to handle the perfectly-typeset case. This has been useful not only in allowing the sequence of modules to be tested, but has highlighted a number of issues which arise due to deviations from the ideal model. Our testing includes actual scanned formulas.

2.1 Idealized input

Mathematics is an enormously varied language. To be concrete, we initially consider algebraic expressions that might appear in the context of some computer algebra system. Here we use the syntax for Macsyma:

```
'INTEGRATE(A*X^2+B,X) = A*X^3/3+B*X
```

Macsyma (which is perfectly capable of computing the right-hand side of that equation, as are other commercial computer algebra systems) can render this expression in TEX form. Macsyma 418 provides this, explicitly:

```
$$ \int {a\,x^{2}+b}{\,dx}={{a\,x^{3}}\over{3}}
 +b\,x $$
```

which prints as

$$\int a\,x^2 + b\,dx = \frac{a\,x^3}{3} + b\,x$$

when run through appropriate software. One might expect that this printout constitutes "perfect" input.

[3] This suggests that in the non-ideal case, we consider alternative approaches or attempt to achieve high performance with modified goals, both of which are subjects of our present research.

We can produce even better "scan" results if we forego the use of paper. We can convert the TEX to PostScript, and the PostScript to a bitmap, without ever printing the bitmap. We can then feed the bitmap into our recognition program, being assured that

- There is no noise on the "page" caused by defects in the paper, or errors caused by over/under adhesion of ink (toner, etc).
- The characters are all fully-formed and correct to the nearest bit, to whatever detail we have specified (independent of any printer resolution).
- The scanning introduces no noise. In particular, the following problems do not occur: misalignment of paper, dirt on the page or in the scanning mechanism, over-scanning of margins, bindings, staple holes, etc.
- The fonts are all known, having been produced by the computer system in the first place. Computer scaling of fonts may have been done to produce various sizes, but this has been done systematically.
- The syntax is restricted to that which is known to the computer algebra system in the first place. (we cannot use the full generality of mathematics notation, even if it can in principle be typeset, because our notion of mathematics grammar is necessarily limited to those constructions we are aware of!). We are fairly comfortable with the subset of TEX used by CASs.

The above method is useful for developing databases of ideal characters [8]. Alternatively, we can use a printed source such as an integral table and scan it into the computer. Such images currently need to be manually cleansed, e.g. by separating merged characters, before presenting them to the OCR module.

2.2 Recognition

In the following discussion, we assume we are given perfect typeset input, with sufficient resolution so that no characters are touching or broken unexpectedly,[4] aligned perfectly on vertical and horizontal axes. While we allow some variation in the mapping of a given character to a grid of fixed resolution, the variations will be quite small.

Under these conditions, a character can be learned from exactly one example. Indeed, the "learning" of characters can be done entirely by reference to computable

[4] Although at low resolution, computer-typeset characters, especially italic characters, or characters that get very thin, like large parentheses, may become disconnected.

tables of bit-maps for the characters. Any *ad hoc* scheme that can accurately identify and distinguish each character from the others that might occur could be used.

One scheme (described further below) which we found rather accurate was to scale each character to a 5 by 5 grid and compute the gray-value in each of the 25 sections, and also keep track of absolute height and width. We recognize as distinct each size of character. Thus 9, 10, 11, and 12 point x would all be different and identified by size.

Running our simple recognizer on perfect data, we still have to address some problems. The two most prominent in our program are (a) the use of special criteria to recognize horizontal lines, and (b) picking up characters made up of multiple components (e.g. "i", "=" "!") . We use various heuristics to handle these issues, but have yet to develop a satisfactorily inexpensive and robust solution.

We note in passing that there is not a "single" mathematical language. The typeset characteristics of some glyphs varies substantially with historical and aesthetic criteria. One source of material (de Haan [1]) sets the "+" with a descender as low as the tail of a "p" and uses the form Cos for the mathematical cosine. The standard size of font used for TEX mathematics has a smaller "+" and uses cos for cosine. Accommodating such differences in one program requires different parameterizations.

2.3 Grouping

Given the perfection of the character recognition, as well as the regularity of the spacing, it becomes easier to group characters into tokens (perhaps using rather more fragile algorithms than would otherwise be advisable). Grouping digits 1, 2, 3 into the number "123", and the letters s, i, n, x into two tokens, "sin" and "x" becomes much easier. Note that a complete list of mathematical symbols (and perhaps their translation into other alphabets) is generally available. This makes it possible in principle, depending on tables of abbreviations in the recognizer, to typeset and re-recognize "Si" as either the product of "S" and "i" or perhaps as the name for a tabulated function (the Sine Integral).

2.4 Parsing

Knowing that we are limited to parsing expressions that can be produced by (say) Macsyma means we do not need to make sense of *ad hoc* notations or abbreviations, or even some conventional notations that do not fit into the current output repertoire of computer algebra systems. Examples of such excluded notations include certain conditional notations, reference markings, footnotes, and "comments." We are also guaranteed that there exists (at least) one valid Macsyma expression corresponding to the display we are parsing.

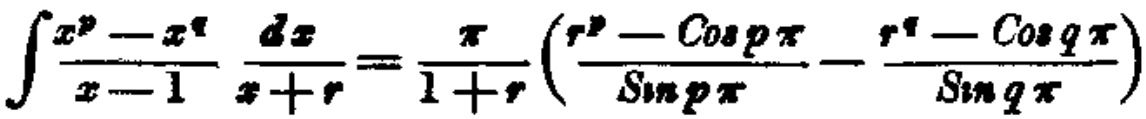

Figure 1: A moderately complicated equation.

2.5 Timing

To get an idea of the time this all takes, let us assume that prior to our looking at an equation, the page processing has identified a "zone" containing a single equation from [1], as in Figure 1. This was scanned at 200 dots per inch (dpi), and slightly cleaned up so that our naive OCR would work. Cleaning constituted breaking connected glyphs in *Sin* and *Cos*; we also removed the dots above the i's, but that could also have been handled in other ways.

In rewriting our code and data structures from an earlier 1994 program [12], we substantially sped up most parts of the process. With our current (October 1995) prototype, the time to change this bitmap, stored as a file, into a Lisp expression

```
(= ((Integral NIL NIL)
    (* (* (+ (^ x p) (* -1 (^ x q))) (^ (+ x -1) -1))
       (* 1 (^ (+ x r) -1)))
    x)
   (* (* pi (^ (+ 1 r) -1))
      (+ (* (+ (^ r p) (* -1 (Cos (* p pi))))
            (^ (Sin (* p pi)) -1))
         (* -1
            (* (+ (^ r q) (* -1 (Cos (* q pi))))
               (^ (Sin (* q pi)) -1))))))
```

was under 200 milliseconds (ms) of CPU time on an HP 9000/715 computer.

We started from a TIFF file originally produced offline by an Intel 486 PC-style computer and a Xerox scanning device. Reading this file and converting the data into a Lisp structure of run-length-encoded items took about 14 ms.[5] This produced a structure equivalent to a 663 by 77 bitmap. A Lisp program to separate this structure into 54 distinct connected components took 25 ms. Computing clusters of similar characters for ease in recognition took about 38ms. (In practice, this clustering would be done once in a training process, on a few pages of a reference text, to establish the characters. After these are identified by a human, the optical

[5] This time was averaged over 100 runs and includes Lisp garbage collection time

character recognition would be done on the full text.) In this case, a rescanning of the (same) text recognizing the characters takes about 80 ms.

Although these times benefited from a rather small character set — only those characters actually appearing in this expression were used — the time for running the same naive algorithm for recognition of a full-scale dictionary of glyphs including special characters would grow only linearly with the dictionary size. If the lookup times become a problem, a faster-than-linear lookup could be devised.

The next collection of steps include

- Checking for possible "lexical" groupings like adjacent digits constituting a number, or $S\ i\ n$ constituting a reserved word, as well as disambiguating the twelve horizontal lines in this equation which constitute various pieces (minus, divide-bar, parts of the equal-sign, etc.).[6] (50 ms)

- Arranging the tokens into boxes to provide information for the parser regarding super/subscript vertical and horizontal concatenation information. (20 ms)

- Parsing: a mostly-conventional programming-language style parse, but with some special features. For example, the parser must disambiguate the meaning of "horizontal space": it must decide ax is `(* a x)` but $\sin x$ is `(sin x)`. (3 ms)

In total, the post-character-recognition stages combined took about 70ms on this example.

3 Optical Character Recognition

In this and subsequent sections we provide details on our project. In this section, we describe the OCR module in more detail.

It may not be immediately evident why OCR should be much of a problem for the task at hand. There are quite impressive (rapid, accurate, versatile, inexpensive) systems of hardware and software for scanning of simple text. Rates of thousands of words per minute, accuracy of 99% or higher, and even automated page-feeding, are available in systems with a retail cost of under $5,000 today. Within this budget one can buy an excellent page scanner, a high-speed personal computer, and one of several commercial OCR software programs.

[6] As a heuristic to aid further processing, we extend the divide-bars so that they will be more likely to span the width of the numerator and denominator. This accommodates our parser's assumption that the left-most character of a quotient is the divide bar.

However, at present it seems that most commercial programs cannot handle mathematical text. In our attempts to use commercial software on perfectly-formed characters arranged in equations, we found the rate of recognition fell dramatically to 10% or less.[7] Commercial systems have apparently traded off general performance in recognizing characters at arbitrary locations for higher accuracy in recognizing words within a line. Many heuristics that work well on straight text and its multi-column or tabular variations do not help for mathematical text, which is characterized by a more complex layout in two-dimensions. For example, an instance of the word "log" in the numerator of a fraction can be completely unrecognized, even though the word presents no difficulty in a line of text. Additional features of mathematical text include occasional use of special characters of varying sizes and different character bigram frequencies arising from a specialized lexicon. The difficulty is compounded by the fact that many of the source documents in which we are interested are older texts which are not cleanly printed.

Thus we have found ourselves writing our own character recognition software. Other specialized tasks (besides equations) may yield to the same general approach. Joined with some graph-type recognition, certain types of chemical and physical diagrams could be tackled as well (see e.g., O'Gorman and Kasturi [7] for discussions of related topics).

3.1 Objectives

Our objective for OCR is to start from a scanned binary representation of the mathematical expression (a "bitmap") and end up with a set of pairs: (glyphs, location). A glyph is a representation of a character that includes font information as well as the alphanumeric value.

The OCR software generally available focuses primarily on one-dimensional text, or perhaps blocks of text in which the characters can be placed in a linear sequential order. The usual location information is conveyed through the sequence of the characters in a one-dimensional stream of ASCII characters in a file. A limited amount of spacing, columnar, and global skew information may be also included in this type of representation.[8] By contrast, for mathematics, standard features such as the baseline and line spacing become subsumed under a more general two-dimensional framework for the layout of information.

[7] They are however, remarkably capable in *other* domains. A useful general reference is the paper by Bokser [9], which discusses in some detail the Calera scheme; other competing commercial systems include Caere and Xerox Imaging Systems.

[8] While representations such as the XDOC format from Xerox may in principle be more powerful, the associated OCR mechanisms do not seem to support recognition of images such as complex fractions which require their full generality.

Our problem is perhaps made easier by the fact that we are, at least at present, unconcerned about achieving "omnifont" recognition. As one target of our interest is large volumes of integral tables, these are large enough that it makes sense to learn a *particular* set of characters and conventions for each book. Both size and font information provide useful hints to the parser, as the use of large letters, small letters, bold-face, italics, and different fonts can provide additional semantics. We have found references (e.g. de Haan [1]) using different fonts and sizing for numbers depending upon their use as (for example) page numbers, equation labels, reference numbers, base-line coefficients, exponents, or footnotes. The variations are nevertheless small enough that we can reasonably envision learning them all.

3.2 General methods

A variety of methods have been developed for OCR, based on character morphology [10], character "skeletons" [10, 11], feature vectors [9], Hausdorff distances [12, 13], etc. Each method works well in certain circumstances. Further progress in this area depends in part on coming up with systematic ways of testing these various methods. For instance, databases of English text with their "truthed" equivalents and models of noise [14] have been developed to address this issue.

Probably the most fully-formed and described theory for the recognition of equations is one that emphasizes the impact of noise, in work by P. Chou [15]. This paper describes a 2-D parser which runs on a bit map produced by `eqn` and then peppered with noise. Although there is certainly a strong prospect of further achievement along these lines (i.e., using a stochastic context-free grammar, and treating recognition as inversion of a signal processing algorithm in the presence of noise), the cost of this approach for our problem is formidable. We also note that the production rules of the grammar must map from the grammar's "start" symbol down to a rectangular array of pixels. In spite of its apparent generality, the algorithm described is actually far more restrictive in some ways than we can afford. For example two factors can be horizontally concatenated only if each has the same point size and the baselines differ by at most one pixel.

We back off from this admirably general long-term approach to explore what we actually must do to get OCR to work on examples of interest in the short term. The method we describe here for recognizing characters is a simple one based on "property vectors" (as suggested, e.g., in [9]). This straightforward approach works well for our present purposes. As we better understand how to deal with non-ideal input, the role of the recognition module may also change. In particular, the solution of some problems will probably require contextual information above the single-character level.

3.2.1 Property vectors

Low-level character recognition engines typically rely on building a database of character templates for recognition. These templates could be bitmaps of the characters themselves, as used for instance by the Hausdorff distance method [12]. In the property vector approach, each character is represented by a vector of features, which in our case is constructed as follows: Surround the character by its bounding box, divide the box into subdivisions (e.g. a 5 by 5 rectangular grid), and then count the percentage of black pixels in each subdivision. The property vector is made up of this set of gray-levels, plus two more data items, the height-to-width ratio and the absolute height in pixels of the bounding box (each of these is placed in eight bits for compactness).

Property vectors live in a high-dimensional space (in this case, 27 dimensions). This approach relies on the intuition that different characters lie "far apart" from each other in this space. Humans, for instance, can read even defocussed characters with some reliability (although context plays a role in disambiguating blurred words). We use a Euclidean metric to define the distance between characters.

The use of property vectors involves two stages, training and recognition. To train the database, we choose a set of characters of a given type font and size, which is then represented (either via hardcopy or completely electronically) at a given scan resolution, e.g. 300 dpi. Our algorithm sequentially picks off connected segments of black pixels from the image and presents them to a human for identification. The database consists of the resulting array of property vectors and list of corresponding character names.

Using an alphabet of TEX's "computer modern" characters, e.g.,

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0
! @ # $ % & * ( ) - = + [ ] { } ‘ , . ′ < > / ? | \ ˜ ˆ

we can get a feel for the effectiveness of this classification scheme by constructing a table of the (Euclidean) distances between each pair of the characters. For twelve-point characters at a scan resolution of 300x300 dpi, the average distance between characters was 1,838,857 units, and the ten shortest distances are given below.

Distance	Characters
48792	O o
49616	X x
52008	V v

```
 60352       W w
 61208       i l
 61936       I i
 68968       I l
 82696       Z z
 95408       n u
154096       C c
```

Thus, while some characters are relatively close together, there is still ample room to distinguish between such ideally-scanned characters. Also, the characters which are close together tend to be different cases of a given character, and hence conflating them would not necessarily be a big issue.

The closeness of an "i" with an "l" brings up an important limitation of the present approach. Since we are extracting a set of connected black pixels from the image, we at present deal effectively only with single-component characters. The property vector for "i" above refers only to the lower segment, without the upper dot. The closeness of "i" to "l" is thus quite understandable. Since a human is already involved at the identification stage, he or she could also "edit" the bounding box for such multiple-component characters, which are not too great in number. Alternatively, a heuristic such as looking for a dot within a fixed space above the ambiguous segment [16] can be used, at the cost of introducing a possibly document-specific parameter.

Character recognition proceeds along similar lines to that in the training stage. We form the property vector for an input set of pixels and calculate its distance from the property vectors in the database. If the minimum value is below a threshold value, we declare the glyph recognized, and return its label and the location of its bounding box (i.e., the coordinates of its lower left corner and height and width). If the minimum distance to a learned character is above threshold, we again ask a human for assistance. In principle, a program could return a set of glyphs with probabilities, requiring subsequent steps to choose a particular glyph. Although such processing is clearly a possibility, it appears to substantially increase the complexity of these further steps.

3.3 Improving OCR

The generalization of the above method to mathematical text is straightforward. Indeed, given an ideal image of a mathematical expression, we not only can successfully recognize all the symbols but can also parse it into the corresponding abstract mathematical expression (as Lisp data).

A number of issues arise with respect to our method of character recognition, and also of parsing (to be described below). As mentioned previously, we are presently

limited to single-component characters, although we can work around this. We also find that the property vector method is surprisingly sensitive to changes in character font and size. While the versions of a given character in different fonts do seem to form a cluster, in the sense that an italic A is closer to a Times-Roman A than any other Times-Roman character, the shape of the cluster in property-vector space is by no means "spherical." This would limit the usefulness of a threshold-based criterion for recognition as a general method. Databases at different point sizes are also necessary in order to achieve good recognition rates, even in the ideal case. While the thrust of much commercial development has been towards omnifont recognition, the potential for proliferation of separate templates in a database is less worrisome for us, since we will typically be dealing with a few mathematical texts at a time, for which a thorough database can be built up.

The most serious problem we have encountered is due to noise. A system based solely on recognizing single characters tends to perform very badly on images with merged or broken characters, as might arise from setting the scanning threshold to be too low or too high. Such problems are exacerbated by the potential for ambiguity, e.g. between a merged "rn" and an "m", or a "dx" and a "clx." Their resolution appears to depend on the acquisition of further contextual information, such as the point size of the main font in use, extraction of a baseline, the meaning of various regions in the document (such as page number or table heading), etc. We at present utilize this kind of information only *after* the OCR stage, during parsing, but in fact it may be advantageous to implement this in a pre-processing stage. Such a pre-processing stage, as we are presently developing it, involves tasks such as de-skewing the page, removing noise such as small spots or extended black regions along bindings and margins as might appear in copier output, and recursively segmenting the image into regions separated by vertical and horizontal white space. Related work has been done recently by, e.g. Okamoto [17]. The goal is to segment the image as far as possible before presenting a boxed region to the OCR engine. In combination with, for instance, special recognition of horizontal lines, we could extract information such "Inside this box is the numerator of a fraction consisting of twelve-point-baseline symbols." This might be extracted before the OCR engine looks within the box. Our recursive segmentation routine appears capable of providing us with such information, though our current program cannot yet proceed within a square-root sign (since this has unbroken horizontal and vertical extents).[9]

In the case of integral tables, we can take advantage of further formatting information such as the fact that each entry begins with an entry number, followed by an integral sign, that a symbol such as "dx" will almost always appear before the = sign

[9] It may be simplest to remove the horizontal segment of the square-root sign and then re-proceed with segmentation.

and will yield the variable of integration, etc. Further possibilities include returning a heuristic certainty factor along with identification information from the OCR engine, and extending identification to word-level strings such as "*sin*" instead of requiring character-level recognition.

3.3.1 Are Typeset Results Correct?

We have been asked if we take for granted that the formulas in the tables of integrals are error-free. Quite the contrary. Substitution of numbers for parameters and numerical integration can form a partial confirmation [18], but this too may be inadequate—the prospect for errors with regard to validity of domains is also substantial, and a particular weak point with computer algebra systems today. Only with *in*definite integral formulas is there a prospect of checking by differentiating the result. Even this tends to present difficulties because results may appear as drastically different forms whose equivalence may defy simple proof. In fact, even confirmed "equivalent" algebraic results may have different domains, and consequently be non-equivalent.

4 Parsing of Two-dimensional Mathematics

As discussed in the previous section, the job of the optical character recognition stage is to return the locations of glyphs (characters) on a two-dimensional page. In this section, we describe our efforts to interpret them as algebraic expressions. For example, given the location and identities that we see as $3x^2$, we wish to return an expression `(times 3 (power x 2))`. Success in this task provides, among other things:

1. Technology for parsing of mathematical data such as integral tables, mathematical or scientific journals, etc.

2. Demonstration of a "niche" 2-D parsing technique that could be a model for other parsing tasks. These might include parsing of stylized tables, tabular music, chemical formulas. and other tasks.

For our purposes here we assume that the OCR phase feeds us the correct glyphs and positions from a scanned page. A more complete system must devote some effort to correcting glyph errors. We leave this for future work.

4.1 Prior work in 2-D Parsing

Two-dimensional mathematical expression parsing has been addressed at several times; in the late 1960's [19, 20, 21] and more recently [15]. The 60's research can be characterized as a mixture of, on the one hand, pragmatic work, and on the other, rigorous but impractical syntactic treatment of the subject.

For example, Anderson's parser [19] handled a very large domain of mathematical expressions. However, at the time of his work (1969), his recognizer appears to have been tested to its limits with equations having about eight symbols. In our application, even a modest-size formula, omitting the citation of references, will tend to have 30 or more characters (see Figure 2).

While it may seem that Anderson's work solves a problem rather close to ours, in fact there is a significant difference. Due to the interactive nature of the handwritten input which Anderson used, the recognition program could insist that the user re-write an ambiguous or unrecognizable character.

The character of the noise in our problem is different: Anderson's character-based noise was corrected in the input recognition module. On the other hand, Anderson had more variable positioning and size of the characters, depending upon the human doing the writing.

We will usually not be able to go back to the user and ask for another handwriting sample, but we can expect size and spacing to follow rules. Furthermore, some of our expressions continue for several lines, and in some cases span pages.

A much faster implementation of a parser for a subset of Anderson's language was implemented by Lewis [22], in an impressive demonstration of a handwriting-driven conformal mapping program. However, the language we need to parse is more complicated than that used by Lewis, or the more tentative study by Martin [23].

Chou's work [15] by contrast views the recognition of equations as a particular domain within an overall framework, which regards recognition of 2-D images as fundamentally a signal processing task to decode a structural (syntactic) signal corrupted by noise [4]. This is a more elegant, but perhaps less pragmatic approach, especially with full 2-D structure. We hope our own programs may assist in providing a realistic level of performance in terms of speed and ability to handle specific complex texts.

4.2 A motivating example

Figure 2 is an expression from a table of definite integrals [1]. It is extracted from table 18, which contains selected definite integrals with limits from 0 to ∞.

The original page was scanned at 400 dots per inch. The reproduction from Postscript is at 300 dots per inch, enlarging it by 25%. Note that the subexpression x^p is "one character" as are the letters "ng" in *Tang* (the name for tangent). Also

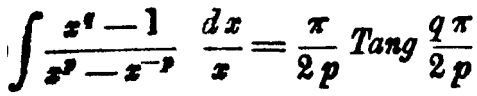

Figure 2: formula 18.6, page 45 from de Haan.

note that, contrary to current practice, the names of trigonometric functions are in italics. This particular formula for a definite integral can in fact be found by computer algebra systems, although the neat formula given is much smaller than that obtained from (for example) Mathematica 2.2.

$$\frac{\pi \left(-\frac{1}{2}-\frac{q}{2p}\right)\left(1+\frac{q}{2p}\right)\tan(\frac{\pi q}{2p})}{2p\left(-1-\frac{q}{2p}\right)\left(\frac{1}{2}+\frac{q}{2p}\right)}.$$

Interestingly, the computer algebra system's result as well as the table's is incorrect over some possible parameter values. (Try $p = 1, q = 2$.)[10]

In any case, our parsed and partly simplified version (ready for insertion into a table of integrals) would look something like this:

```
(integral-rule x 0 infinity     ;;variable, limits
  (times                        ;; the integrand
     (plus (power x q) -1)
     (power (plus (power x p) -1)
                  (times -1
                         (power x (times -1 p))))
            -1)
     (power x -1))
  (times 1/2 pi (power p -1)   ;;result
         (tan (times 1/2 q pi (power p -1))))
  nil)                          ;;no conditions
```

We could also generate a TEX version that would be something like this:

```
$$\int_0^\infty {{x^q -1}\over{x^p - x^{-p}}}
                {{dx} \over x}
 = {\pi \over {2 p}}
   \tan {{q \pi} \over {2 p}} $$
```

[10]Other computer algebra systems: Macsyma 417.100, appears to do somewhat better because before giving an answer, it asks a series of questions such as "Is $(q+p)/(2p)$ an integer?" and can conclude that the integral is not convergent sometimes.

and after processing would appear this way:

$$\int_0^\infty \frac{x^q - 1}{x^p - x^{-p}} \frac{dx}{x} = \frac{\pi}{2p} \tan \frac{q\pi}{2p}$$

4.3 Simplifying our task

Rather than viewing the equation recognition process as a parsing problem of a completely arbitrary 2-D analog of a context-free grammar, it seems to pay to restrict ourselves to a subset of the context-free languages that is easier to parse, and yet sufficiently descriptive to handle most, if not all, of the cases of interest. This is quite analogous to the usual programming language case, where a programming language may be modified to reside in some easily-parsed subset of the context-free languages.

Our current program is not based on an automatic parser-generator, but on a somewhat mechanically generated variant of a "recursive descent" parser, and hence a grammar that maps into such a parser is particularly desirable.

A heuristic suggested at least as long ago as 1968 [19] claims that for the usual mathematical notation, a parse proceeding left-to-right along a baseline can output ordered expressions: for example, we will see the horizontal divide bars before the numerator or denominators; likewise we will see the integral (or summation or product) sign before the limits or integrand of the integral. Furthermore, this method works well with a recursive style of parsing, because each sub-expression is contained in a sub-rectangle offset in some direction from the known main baseline.

Without too much loss in generality, we can consider the typical use of TEX as a generator for typeset equations; more restricted is the subset of TEX that is generated as output from computer algebra systems. We believe this set is strictly left-to-right parseable, *if* we include a few heuristics. The (only) exceptions we have observed are the cases where upper or lower limits extend further to the left of the integral, summation or product ($\int$, $\sum$ or $\prod$) symbols, and where a divide-bar is not quite as wide as it should be. Our heuristic cure is to attach a "spur" extending to the left, to artificially widen such symbols. Nevertheless, there are still some prospects that may confuse:

$$e^{\int f dx} \quad \text{vs} \quad \int_e f dx$$

but the first of these expressions is more likely to be typeset as

$$\exp(\int f dx).$$

4.3.1 Center or base lines

The uniformity of type-setting is probably the most important simplifying factor for us. Looking at Figure 2 we see that the integral is roughly centered above and be-

low an imaginary horizontal line drawn through the expression. The line divides the integral sign in half, then follows along the two horizontal divide-bars, splits the equals-sign, and so on. We can either use this center line as a key, or following the actual typesetting conventions, slip down about a half-character-height to the "baseline" to align the expressions to follow. Font description metrics provide measurements with respect to the baseline, indicating that letters such as "o, c, e" with curved bottoms drop slightly below the baseline; numerous characters have substantial descenders, not only "g, j, p, q, y" but also "(){}[]/,;_". Some characters are raised up—that is, they have baselines below their lowest extent, including "-" and "=". Oddly, the character "+" in one source [1] is so large that it must be treated as having a descender, but not (for example) in output from TEX.

A significant exception to the left-to-right method occurs in the case of a multiple-line formula. In this case we intend to (but have not yet implemented) breaking the typical page into horizontal strips of the equations and continuations. The relative spacing is larger between separate formulas than between continuation lines. The second and subsequent lines of an integral formula (these lines do not have the integral sign on them), are placed to the right of the main line, aligning the major operator (usually a +). We can use additional checking to confirm the multiple-line formulas: a line that does not begin with a formula number and integral sign, but begins with an operator + or − is a continuation.

4.3.2 Non-ambiguity

It seems obvious that table entries should have been initially designed to be easy to interpret unambiguously; success is not always achieved, however. The mind-set of the table-compilers may include precedence rules, notations, abbreviations, semantics for some functions, and other context that differ from modern conventions. Furthermore, we have encountered some nasty ploys to save space on the page, some requiring understanding of English directives, or the interpretation of ellipses (...). These will probably not be handled entirely automatically. Note that disambiguation rules used for one table of integrals may not serve for others. Some kinds of expressions are disambiguated by size, and others by context: e^{ry} versus $e^{r}y$. Adjacency can mean function application as in $\sin x$, multiplication ax; but adjacency of dx signals "integration with respect to x". Spaces also are seen separating the formula number, the formula, and the reference citation in the equation "decorations".

4.4 Problems and some solutions

As discussed by Anderson [19], the distinguishing feature of 2-D parsing (compared to linear parsing) is the need to keep track of additional relationships between the

terminals and non-terminals of the language. Any particular parse may be under several geometric inequality **constraints** specifying the locations of bounding boxes and centers for one expression relative to other. These can be tracked via attributes in a syntax-directed scheme, or additional components in the terminal and non-terminal objects handled in a recursive-descent parser, which is what we've done.

The problems and solutions can be grouped into those which are likely to be common over most of the equation-processing texts, and those which are corpus-specific.

4.4.1 Those with general solutions

In the expression

$$\frac{1+e^x}{e^y}$$

notice that the horizontal line breaks the space into two regions and that these two regions must not interact. That is, we should not be able to think that the y has an exponent of e, because one lies below the line, the other lies above the line. Thus, the parse of the numerator is under the constraint "the y-coordinate must be greater than some value". More generally, a parser may be under n spatial constraints if it is parsing an expression that is nested n subexpressions deep. In our case we are able to make do with the simplification that a sub-parse must begin in a specified rectangular region and can continue directly to the right of that region. Consider the simple expression e^x. In a parse starting with the e, we have to be able to determine how far away to look for a possible exponent (the "starting" rectangle). We also must consider the continuation of the exponent as in $e^{2n+5}3^x$. However, in analyzing the exponent subexpression $2n+5$, we have to, at some point, decide when we are done. If we miscalculate we might end up with the expression $e^{2n+5x}3$.

4.4.2 Corpus-based considerations

Segmenting the page We must segregate descriptive text, titles, line-numbers, footnotes, as well as page decorations such as footers, headers, line-numbers, etc. Some of these may be sufficiently complicated as to recommend pre-processing by hand. In our initial tests we have generally used a "rubber-band rectangle" technique to visually separate out equations from a scanned page. The cropped rectangle is then processed further.

Special fonts and abbreviations Some texts may use additional type fonts (Gothic, Black Forest, an occasional Hebrew letter); some use l for log, sn for sin, or $Tang$

for tan. de Haan [1] for example has a particular notation that changes the meaning of the solidus (/) in exponents via the "Notation de Kramp" such that $c^{a/b}$ is $c(c+b)(c+2b)\cdots(c+(a-1)b)$ and thus $1^{a/1}$ is our usual $a!$. Many of these types of problems have solutions which can be parameterized on a book-specific basis. Papers appearing in a particular journal, having been typeset historically by the same mechanical setting process will mostly likely be consistent in their spacing conventions for such details as the relationship of an exponent and a base. Therefore, it is possible to catalog all the conventions that are specific to a book (or perhaps to a book designer and publisher). Then this dictionary will be another input to the parsing algorithm. Although we now have separate dictionaries for TEX, de Haan [1], and Gradshteyn [2], not all such conventions can be predicted; an author is relatively free to invent new conventions for notation, including temporary abbreviations, and draw upon additional character sets.

4.5 Outline of our parsing algorithm

4.5.1 Input

A page of glyphs and locations is presented to us as a set of 5-tuples. Each 5-tuple represents (glyph-name, x-origin, y-origin, width, height). The glyph-name may be a compound object itself including font and point size, although for exposition here we assume it is a simple atom.

The x-y origin data represents the lower left corner of the bounding box for the character with (0,0) representing the page's left bottom margin. The width and height are specified in pixels, currently at 400 dots per inch resolution. The ordering of the scan is generally irrelevant. Some characters may be delivered in pieces. Thus an "=" will be provided as two short "hline" (horizontal lines).

The example from Figure 2 is delivered (simplified for exposition) as

```
(integral 0 1 51 120)
(x 52 7 30 29)
(hline 52 54 207 12)
(x 82 79 27 27)
(p 83 22 19 26)
(q 113 93 18 26)
(hline 115 20 52 8)
(hline 145 92 52 7)
(x 177 12 27 28)
(hline 203 39 29 5)
(1 209 83 20 40)
(p 232 25 23 26)
```

```
(d 298 85 30 38)
(hline 300 59 62 7)
(x 312 14 29 27)
(x 333 85 27 27)
(hline 375 57 54 7) ; bottom of =
(hline 375 68 54 7)
(2 446 15 26 40)
(hline 448 61 61 6)
(pi 463 86 34 26)
(p 480 1 32 43)
(T 528 54 36 42)
(a 557 54 24 26)
(n 583 54 26 27)
(g 605 41 29 39)
(q 652 73 25 41)
(2 653 17 24 39)
(hline 658 62 62 5)
(pi 688 87 32 27)
(p 686 2 32 42))
```

4.5.2 Output

For each equation successfully parsed, we expect a Lisp-like parenthesized expression (as shown in Section 4.2) and a bounding box. The expression will not in general be entirely simplified algebraically. We expect substantial further processing will be necessary to make it suitable for storage in a computer algebra system as a pattern-replacement pair to be retrieved on request.

4.5.3 Processing

For our processing needs, we can divide the parsing task into four phases, some of which can be interspersed. The first phase handles sorting and segmentation, next we recognize tokens like tan or numbers, following this we order the 2-D collection into a structure that is linear, and finally we use a nearly-conventional parser on this structure.

Segmentation This phase separates equations from each other and from the page decorations. We can use local clues such as recognizing that each equation has an integral sign on the left and possibly a continuation line or lines. For continuation

lines we generate must "attach" the continuation to the right side of the previous line. Distinct equations are generally separable by noticing sufficient vertical white-space.

At our current stage of testing, we can take a page of scanned equations and segment it into formulas like that of Figure 2. In this figure we have cropped (by hand) the markings which relate to the citations.

Lexical analysis Within each segment, we collect adjacent digits and names to form numbers and symbols such as sin, cos, tan, log, exp, or in our Figure 2, *Tang*. Printed formulas do not contain multiple-character names (other than those in the usual list of functions, and those normally in Roman font), so other adjacent letters or numbers such as $2p$ or dx are *not* joined in the lexical stage.

We can distinguish horizontal lines in five categories: quotient bar, minus sign, top of square (or nth) root, and part of $=$, $\geq$, $\leq$. We could also collect comma + dot as ";", recognize "i", etc., if the OCR phase has not already done so. Other items may be single-letter names, if alphabetic; other symbols such as parentheses, are their own categories.

The bounding boxes for collected objects are adjusted as needed. For example the *Tang* token in our sample input now looks like `(Tang 528 41 106 55)`

To distinguish a minus sign from a quotient bar, we construct a square bisected by the horizontal line. If there is a character in a box intersecting the upper half, we check to see if it is also a horizontal line (an = sign), a $>$ (an $\geq$) etc. If none of these situations hold and there is also a character in the lower half, then it is a quotient bar. (In practice we must shrink these boxes slightly because italic characters' bounding boxes tend to overlap, and hence the tail of the p may in fact be literally below the $-$ in $-p$. After identifying the quotient bar, we then extend it slightly to the left to make sure it is to the left of any character in the numerator or denominator. We must also account for noticing the TOP "-" of an "=" first.) A square-root bar (when detached) is identified by the presence of the check-mark piece at its lower left. If there are no characters above or below, it is a minus sign. There are other cases involving letters that are underlined or overlined, but frankly, figuring out

$$\frac{\underline{a}}{b} \equiv \frac{a}{\overline{b}}$$

which in TEX is

```
$ {{ \underline a} \over { b} } \equiv
 {a \over {\overline b} }$
```

requires more global processing than we are willing to use. We do not expect to be able to un-TEX everything. Nevertheless, these methods correctly recognize all the

quotient and minus signs in the sample of Figure 2 as well as the equal sign. Note that the widths of the minus signs range from 0.05 to 0.23 inches; the quotient bars are from 0.15 to 0.51 inches. At this stage the result is the following collection of tokens:

```
(integral 0 1 51 120)
 (x 52 7 30 29)
 (quotient 46 54 216 12) ;widened
 (x 82 79 27 27)
 (p 83 22 19 26)
 (q 113 93 18 26)
 (- 115 20 52 8)
 (- 145 92 52 7)
 (x 177 12 27 28)
 (- 203 39 29 5)
 (1 209 83 20 40)
 (p 232 25 23 26)
 (d 298 85 30 38)
 (quotient 294 59 71 7)
 (x 312 14 29 27)
 (x 333 85 27 27)
 (= 375 57 54 18) ;joined vertically
 (2 446 15 26 40)
 (quotient 442 61 70 6)
 (pi 463 86 34 26)
 (p 480 1 32 43)
 (Tang 528 41 106 55) ;joined horizontally
 (q 652 73 25 41)
 (2 653 17 24 39)
 (quotient 652 62 71 5)
 (pi 688 87 32 27)
 (p 686 2 32 42)
```

Linearization In this phase we build horizontal and vertical hierarchical structures that encode the adjacency detected amongst the tokens in a non-numerical format.

Heuristics for what constitutes "adjacent" and what constitutes "superscript" are somewhat corpus-dependent and must be in part based on baseline measurements for characters. Notice that in the form x^p the exponent is nearly adjacent to the base, while the form p^x provides a much larger disparity. This disparity is negligible if the

base-lines of the characters rather than their bounding boxes are used.

The structure we produce here consists of encoders such as `vbox`, a box encoding vertical information, `expbox`, encoding the relationship of exponents to bases, and `hbox`, a catch-all for horizontal encoding. Our full grammar may need more precise encoders such as right-vertical box (`rvbox`) and centered-vertical box (`cvbox`) to account for the two ways of type-setting an integral's limits, but we may not need these subtleties generally. The output of this phase for the integral solution represented in Figure 2 would be:

```
(hbox
   (vbox integral nil nil)
   (vbox quotient
         (hbox (expbox x q) - 1)
         (hbox (expbox x p) - (expbox x (hbox - p))))
   (vbox quotient
         (hbox d x)
         x)
   =
   (vbox quotient
         pi
         (hbox 2 p))
   Tang
   (vbox quotient
         (hbox q pi)
         (hbox 2 p)))
```

Because the integral box is capable of encoding information vertically we stick it in a `vbox`. Technically, the exponent "2" in x^2 should be in an `hbox` because it could very well be x^{y+2}; however we abbreviate `(hbox x)` as `x`.

The process of parsing is one of "recursive descent" where each parse of a subexpression is confined to a region of the surface. For example, presuming that the leftmost character of the region (x_0, y_0) = bottom-left to (x_1, y_1) = top-right is an integral sign, one can divide the remaining region into three: above, below and to the right of the $\int$. If there is an upper limit, then it must begin in the rectangle above the $\int$ and below the upper limit y_1. The limit can continue to the right of its start-region. Similarly for the lower limit. If neither of these regions produce an expression, then the whole region to the right of the $\int$ must be parsed as the integrand, dx, $=$, and the answer. The key in each case is that given a prospective rectangle for an expression, the leftmost glyph governs the meaning of the expression.

The case of a fraction is guided by the horizontal quotient bar which is the leftmost character. Our program also limits the horizontal extent of the numerator and the

denominator by the size of the quotient bar. In the case of a superscript, for those symbols where a superscript is plausible (we do not expect, and therefore do not look for superscripts, everywhere. Certainly not on a $+$, for example.) we key off the baseline of the main expression. Other expressions like $x + a$ or $-x$ are treated by proceeding from left to right along (approximately) the baseline.

Final phase Linearization is followed by an essentially conventional parser. The encoding of the previous phase has removed from consideration the absolute location information of characters, and linearly-encoded as `vbox`s, deviation from the baseline. This phase handles the more sophisticated interpretation necessary to determine when adjacency implies multiplication, when it implies function application and when it implies some kind of operator (like d followed by x).

4.6 Summary: Parsing

It is too early to judge how effective this parser will fare in the face of noisy input and highly variable expressions. In particular, for dealing with noise, building recovery into a recursive descent parser is one possible approach. We may find it advantageous to consider Chou's techniques [4] as a fall-back position based on stochastic parsing and probabilistic recognition of equations. We believe this will be by comparison much slower.

5 Table lookup

The preceding sections described how we extract information from a printed page of integrals. We now briefly turn to the question of how to make the information usable to others, once it is parsed. As mentioned earlier, we conceive of our system as responding to human or machine queries for the value of a definite or indefinite integral. For convenience, such a query is expressed by presenting an expression for matching, in a Lisp data form. Any contemporary computer algebra system (CAS) could provide an expression in a form easily converted to such a query. The system returns with an answer or answers, possibly with conditions on its region of validity. Additional information might also be available, such as the original source of the entry and a bitmap of the source image.

A major consideration in the design of the system is its speed. If it is fast relative to a CAS, for instance, then there is little penalty to query the table first before trying what could be an elaborate algebraic calculation. Access times for a prototype containing 685 integrals averaged a few milliseconds on a workstation computer. This

compares quite favorably with commercial CAS's such as Mathematica, Maple, and Macsyma, for which computations can be easily of the order of a second or more. We give a brief and somewhat simplified account of the structure of the system below. Details of the system can be found in Einwohner [5].

5.1 Table design

An entry in the table consists of information including the integrand, the limits of integration, regions of validity of the answer, and the source of the entry. The "answer" is typically a closed form result, a reduction formula, or a program to evaluate the integral. A unique acquisition number is associated with each entry, plus a list of keys which represents the search path by which the entry can be reached.

Given an input integral, the system begins its search for the matching entries by constructing a sequence of keys from the integrand. The sequence is drived from descending down the algebraic tree of operators in the integrand. Thus $log(sin(x))$ would be log-sin-identity.

In essence, each prefix of the sequence is then hashed to find a set of acquisition numbers for integrals with the same encoding. By taking the intersection of these lists, we are left with the acquisition numbers of the entries which match the input integral, up to parameter bindings.

In order to speed this process, each key is assigned an estimate of its "rarity", and the search is carried out rarity-ordered, with the key of highest rarity being hashed first. For instance, `log` is considered rarer than `power-x` (i.e., the variable of integration raised to some power), and hence would be hashed first. We thus start out with a list of acquisition numbers as short as possible. We also find out as early as possible if we should terminate the search (or try a transformation of the integral and try again). In the case where there are matching entries in the table, the search terminates when one of the special keywords `const` or `identity`, which correspond to a constant parameter or the variable of integration, is encountered.

At the end of the search process, we have a list of acquisition numbers of entries which match the input integrand, up to assignment of parameters. The parameters in each table entry may be:

1. the same as those in the input, in which case we have a perfect match;

2. more general than the input parameters (e.g., variables versus constants), in which case we set up binding equations to specify their values;

3. more specific than the input parameters, in which case we return an additional *proviso* indicating this fact; or

4. inconsistent with the input, in which case we throw it away.

For the entries which are not thrown out, the "answers" are ordered according to the above list and returned with the parameters bound to the appropriate values. Having specified the form of the output from the table, this information is now available, e.g., for use by a CAS.

6 Summary

In this paper, we describe the status of a system for enabling rapid electronic access to mathematical information presently locked away in print form. We envision scanning in the pages of printed tables of integrals, from which we extract character and higher-level information for eventual conversion into, e.g., a Lisp or TEX expression. We can currently accomplish this task on various types of simplified, noise-free input. The most important problem which we now face is how to deal with realistic noisy input. We are developing a top-down approach which we expect will significantly improve our ability to deal with this issue. We have also developed an efficient automated method for storage and retrieval of integrals, whose access time is only weakly dependent on table size. We expect this system will be useful in considerably enhancing the speed and power of computer algebra systems, especially in symbolic definite integration.

We entered into this exploration initially with the view that we would use commercial OCR technology and merely fit a "back-end" of a mathematics parser and table-lookup mechanism. We found that domain-specific assumptions, and optimizations for those domains, made the use of commercial OCR software infeasible. Our hope is that the OCR techniques we have developed will be more flexible to solve niche character recognition problems with characteristics that differ substantially from the "omnifont text" basis of commercial OCR, and that our programs may be used by others.

This work is not complete. We expect to proceed further to develop improved general OCR off-line character recognition capability for the specific domain of mathematical expressions, review our success with a more complete top-down approach to extraction of information from documents, and as one benchmark application, provide "world class" on-line information about integrals and related formulas.

7 Acknowledgments

We acknowledge useful discussions on this and related topics with Alex Bui, Ted Einwohner, Brian Evans, Dan Halpern, Gary Kopec, and Paul Tero. For access to these programs, contact R. Fateman (fateman@cs.berkeley.edu).

References

[1] D. Bierens de Haan. *Nouvelles Tables d'Inteǵrales Defínies* edition of 1867 corrected, with an English Translation of the Introduction by J.F. Ritt. G. E. Stechert & Co. NY, 1939.

[2] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series, and Products*, Academic Press, NY, 1980.

[3] *Third International Conference on Document Analysis and Recognition - ICDAR '95*, Montreal, Canada, August 14–16, 1995, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[4] P. Chou and G. Kopec. A stochastic attribute grammar model of document production and its use in document recognition. *First International Workshop on Principles of Document Processing*, Washington, DC, Oct. 21–23, 1992.

[5] T.H. Einwohner and Richard J. Fateman, Searching techniques for integral tables, in A.H.M. Levelt, ed., *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation - ISSAC '95*, July 10-12, 1995, Montreal, Canada, ACM Press, NY, 1995.

[6] H.S. Baird, H. Bunke, K. Yamamoto, eds., *Structured Document Image Analysis*, Springer-Verlag, Berlin, 1992.

[7] Lawrence O'Gorman and Rangachar Kasturi, eds., *Document Image Analysis*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[8] Frank Jenkins and Junichi Kanai, The use of synthesized images to evaluate the performance of optical character recognition devices and algorithms, in Luc M. Vincent, Theo Pavlidis, eds., *Document Recognition*, Proceedings of SPIE; v. 2181, Bellingham, WA, 1994.

[9] M. Bokser. Omnidocument technologies, *Proceedings of the IEEE,* July 1992, vol.80, (no.7) 1066—78.

[10] Paul D. Gader, Edward R. Dougherty, and Jean C. Serra, eds., *Image algebra and morphological image processing III*, Proceedings of SPIE; v. 1769, Bellingham, WA, 1992.

[11] Simon Kahan, Theo Pavlidis, and Henry S. Baird, "On the Recognition of Printed Characters of Any Font and Size," IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-9, No. 2, March, 1987.

[12] B. P. Berman and R. J. Fateman. Optical Character Recognition for Typset Mathematics, *Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation - ISSAC '94*, July, 1994, Oxford, UK, ACM Press, NY, 1994.

[13] D.P. Huttenlocher, Gregory A. Klandermand, William J. Rucklidge. Comparing images using the Hausdorff distance, IEEE Transactions on Pattern Analysis and Machine Intelligence, **15**, 1993, 850–63.

[14] Henry S. Baird, Document Image Defect Models, in H.S. Baird, H. Bunke, K. Yamamoto, eds., *Structured document image analysis*, Berlin, Springer-Verlag, 1992.

[15] P. Chou, Recognition of equations using a two-dimensional stochastic context-free grammar, *Proceedings, SPIE Conference on Visual Communications and Image Processing IV, Philadelphia, PA*, 1989, pp. 852–863.

[16] Steven C. Bagley and Gary E. Kopec, Editing Images of Text, *Commun. ACM*, Dec. 1994, vol.37, (no.12):63-72.

[17] M. Okamoto and A. Miyazawa. An Experimental Implementation of a Document Recognition System for Papers Containing Mathematical Expressions, in H.S. Baird, H. Bunke, K. Yamamoto, eds., *Structured document image analysis*, Berlin, Springer-Verlag, 1992.

[18] Melvin Klerer and Fred Grossman, Error rates in tables of indefinite integrals, *Industrial Math.*, **18**, 1968, 31–62. See also, by the same authors, *A new table of indefinite integrals; computer processed*, Dover, New York, 1971.

[19] R. H. Anderson. *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics*, Ph.D dissertation Harvard Univ.,Cambridge, MA, Jan. 1968. Also (shorter version) in M. Klerer and J. Reinfelds (eds). *Interactive Systems for Experimental Applied Mathematics*, Acad. Press, 1968. 436–459.

[20] W. A. Martin. *Computer Input/Output of Mathematical Expressions*, Ph.D dissertation, M.I.T. EECS Dep't, 1967.

[21] Mark B. Wells and James B. Morris, eds., *Proceedings of a Symposium on Two-Dimensional Man-Machine Communication*, SIGPLAN Notices 7, No. 10, October, 1972 (ACM).

[22] H. R. Lewis, Two Applications of Hand-Printed Two-Dimensional Computer Input, Senior Honors thesis, Committee on Applied Mathematics, Harvard College. (also, Proj. TACT Rpt. 2), 1968.

[23] W. A. Martin, *A Fast Parsing Scheme for Hand-Printed Mthematical Expressions*, MIT AI Project Memo 145, Project MAC Memo 360, October, 1967.