Extracting Mathematical Expressions From Postscript Documents

Michael Yang
University of California, Irvine
School of Information and Computer Science
Irvine, CA 92697, USA
mlyang@ics.uci.edu

Richard Fateman
University of California, Berkeley
Department of Electrical Engineering and Computer Science
Berkeley, CA 94720, USA
fateman@cs.berkeley.edu

ABSTRACT

Full-text indexing of documents containing mathematics cannot be considered a complete success unless the mathematics symbolism is extracted and represented in a standardized form permitting both searching for formulas, and re-use of this information in (for example) computer algebra systems. Most documents produced in the past and subsequently digitally encoded, and even most of those potentially "born digital" in current journal production are—at best—encoded in a printer form such as Adobe Postscript [1], in which mathematics is not explicitly marked or easily identifiable. While one might look forward in the future to other document encodings such as MathML, the common journal or textbook product is essentially without semantic content: a jumble of odd characters. Sometimes it is just a jumble of black and white dots! In this paper we demonstrate an approach to decoding, to recognizing and extracting mathematical expressions, from a Postscript document. We can produce a syntactic representation of the extracted expressions which can then be used to generate various forms. For example, if we extract TEX or Presentation MathML, we can re-typeset the expression, but perhaps in a different size or font family. More significantly, if we start from this presentation information, we can hope to combine it with additional contextual processing of the surrounding text and meta-data associated with the document, to assign semantics, (e.g. content MathML), or provide versions in computer algebra system languages such as Maple or Mathematica. Finally, it is possible to use this material to present audio or braille versions of mathematics for the visually disabled. We have previously addressed some aspects of the higher level of processing (parsing TEX for example). In this paper we address the only first stage and concentrate on what may seem to be overly simple, but is in fact difficult to do precisely: extracting the mathematics parts from text.

ISSAC'04, July 4–7, 2004, Santander, Spain.

Categories and Subject Descriptors

H.3.7 [**Information Storage and Retrieval**]: Digital Libraries; I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation Algorithms

General Terms

Algorithms, Design, Documentation

Keywords

Optical character recognition, Postscript documents, mathematics, digital library, document image analysis

1. INTRODUCTION

It is common today to see complex documents on the internet posted as Postscript (PS) or Portable Document Format (PDF). Sometimes there is additional information that allows you to immediately extract the ASCII text for that document, and other material including *meta-data* such as title, author, creation date, etc. Sometimes (perhaps most of the time) there is inadequate meta-data, stored with the document. Short of printing the material on a page and having a human analyze it, even determining the sequence of words (for example in a multi-column document with footnotes and embedded figures) is not a trivial computational task. Nevertheless, extracting this information is important for indexing and search, as well as for access by visually handicapped readers. It is possible to approximate the solutions to these meta-data extraction problems for many conventional paper and journal layouts. This is demonstrated very effectively by NEC's ResearchIndex project (formerly CiteSeer, [10]). Consistent with their philosophy (and our belief), the web site provides substantial online links to associated descriptive papers.

An alternative representation used for some newer documents as HTML or (better) XML goes further towards a structured document in which presentation mathematics can be represented, but the use of this representation for new material, much less legacy, documents, is not common. Thus the Postscript (or merely scanned OCR) equivalent material is where we start.

The process of digesting material by NEC does not make special efforts toward understanding mathematical formulas. Either in-line or displayed, formulas are not "understood" in any sense: if they are extracted at all they appear as

jumbled characters in odd sizes and fonts. While a human can consider viewing the space on the page with these characters by reviewing the companion display form of a page, visually disabled readers or computer analysis cannot take this alternative path. Therefore it would be handy to have the formulas decoded into an indexable ASCII encoding of the mathematics suitable for search, re-use, or even insertion into a computation.

NEC's project is not unique: other organizations dealing with indexing of mathematics journals (for example JSTOR [6]) face similar unsolved problems with legacy documents, most of which are *not* solved by documents "born digital" yet stored as PS, PDF, or similar appearance-based encodings. A project headed by G. Michler [8] in Germany has proceeded to work on digitization of legacy math documents, and standards for communication have been set up based on best current practices by the International Mathematics Union [2].

2. WHY VIEW MATHEMATICAL DOCUMENTS ONLINE?

The argument should be obvious to mathematical researchers today. Historically, the research community used conventional paper journal or proceedings publication as a means of propagating findings, establishing priorities of discovery, and (through refereeing), establishing a scholarly reputation for authors. Today it appears that for many purposes and for many scientific communities it is more effective to have papers freely available in the journal/library online repository, in an author's individual repository, a non-refereed collection such as ArXiv.org where authors submit preprints, or allow papers to be indexed/copied over into an archive of free papers such as NEC ResearchIndex.

A recent study [7] by Lawrence shows that the mean number of citations to an article that is freely available online is 7.03 as opposed to an offline article which has a mean number of citations of only 2.74, 2.6 times less (aside: in fact, we found these statistics in a paper that was freely available online!). Lawrence's research suggests that the ease in accessing an online document as well as the availability increases the readership of the article. An increase in the effectiveness of "data mining" of scientific articles can provide a useful service to society [14].

The advantages for access, review, cross-reference, and essentially all uses of published research benefit from on-line internet availability [12]. The major downside appears to be the loss of revenue for publishers. Secondarily, there is (perhaps) a need to find a substitute for peer-review, some process to certify the "correctness" of articles, essential when publications are considered evidence for academic career advancement[1].

Given this background, any improvement in technology for encoding accuracy stands to benefit all related disciplines. At Berkeley, our digital library research activity could use these encodings to improve the accuracy of the semantic layer in Multivalent Documents (MVD) [13]. Mathematical expressions could be represented in some intermediate form: in some cases TEX is sufficient, but we prefer a more semantically rich encoding. Options include the Lisp form we initially produced (2001), or XML, MathML, OpenMath, or a command-string in a computer algebra system language. Interpreters, or MVD "behaviors," can be written to convert the richer intermediate representations to those that are more superficial (say, for printing). An alternative would be possible as a separate PDF layer for mathematics, which we believe would have similar advantages to the MVD representation, but would depend on proprietary technology. Our best current hope for clear semantics is a translation into an unambiguous utterance in a computer algebra system from which the other forms can be derived. Not all published mathematics can be managed in this way.

[1]See for example the topic "Re-inventing Scholarly Information Dissemination and Use" at http://elib.berkeley.edu

3. IMPROVING MATH ACCESS

To increase the ease of access and availability of a paper online, the paper needs to be indexable and searchable. With plain text and HTML documents, this is easily done by most commercial search engines such as Google or Excite. However, complex documents such as those formatted in Postscript, where the document is essentially stored as a program whose execution on a stack-based interpreter results in a page image, provide much more of a challenge than text. NEC's ResearchIndex bridges much of this gap by processing citations it finds in Postscript/PDF documents and linking papers to each other through these citations, making it easier to find and retrieve related journal and conference papers. It is also of note that Google not only searches through webpages, but is able to search through Postscript and PDF documents on those webpages as well.

In a research context, being able to recognize and process mathematical expressions from a scholarly document can, at least in principle, try to identify similar mathematics in distinct domains; thus it might be possible to identify previously unrecognized relations between threads in (say) chemistry, commutative algebra, and biology. We say "in principle" because we consider it unlikely that investigators would, without prior knowledge of a field stumble on substantially identical notation. Nevertheless, there might be new connections to find, for subsequent searchers.

If mathematics were contained on their own math semantics layer in a MVD [13], researchers could cut and paste mathematical expressions into a computer algebra system saving time over hand-typing in the expressions, while also avoiding the introduction of errors. Currently our MVD math layer looks like Lisp, but existing programs can re-render this so as to use on MathML or OpenMath.

We illustrate with a simple example. Searching for citations for G. N. Watson's *A Treatise on the Theory of Bessel Functions,* ResearchIndex returns 73 citations. Here is one citation context[2] , only slightly edited by inserting newlines.

```
.....0.002 0.0025 200 400 600 800 1000 k Figure 3.
Left: The standard Airy distribution.
Right: Observed frequencies of core sizes k 2
[20; 1000] in 50,000 random maps of size 2,000,
showing the bimodal character of the distribution.
variety of integral or power
series representations including (see [1, 45]) 1)
 Ai(z) 1 2 Z 1 1 e i(zt t 3 =3) dt = 1 3 2=3 1 X
n=0 3 1=3 z n ( n 1) 3) n sin 2(n 1) 3 :
```

[2]a paper by Cyril Banderier and others, "Random Maps, Coalescing Saddles, Singularity Analysis, and Airy Phenomena," *Random Structures and Algorithms*, 19 3-4, 194–246 (2001)

```
Equipped with this de nition, we present the main
character of the paper, a probability distribution
closely related to the Airy function. De nition 1.
The standard ....
```

Clearly, there are some problems here with the context, and especially the representation of the mathematical expressions[3]. The user is not given very good information from this citation. It would be far more useful and convenient to the reader if this output could display the mathematical formula in the document in a readily useful form. We see two possible solutions, each first requiring that we find the place on the page where recognition has problems:

1. Take a snapshot of the mathematical expressions (perhaps as a GIF image file) from the document at that location. Intersperse plain text and the snapshots of the mathematical expressions for context. The solution also works for line art, graphs, half-tone photos, and any other unsolved parts of the page image. Reference to the image is not helpful for visually disabled, or for computer indexing.
2. Parse the mathematical expressions in the document and return a pre-typeset version in some commonly used format such as TEX or MathML.

The first solution presents only small technical difficulties, as seen for example by the LATEX to HTML translators that intersperse small images in a field of text. But it reduces many mathematics articles to essentially a sequence of images, and none of the math content would be indexed except those key names that might incidentally also appear in text. In fact, given the nature of OCR, it is our experience that short intervening text segments are not recognized accurately because conventional OCR programs are mislead by what appears to be changes in base-line and character size, and become far less accurate. In a worst case, the document might be nothing more than a picture. More realistically we would hope that some approximation of the meta-data of article title, author, and some words in the abstract would be recovered even from the most notation-heavy document. Titles consisting solely of mathematical symbols are unusual; one hopes that the recognized words provide some key content.

We prefer the latter solution when possible: it provides a much more flexible representation where we can apply some semantic analysis to the returned expressions and even pass it to a computer algebra system for processing. *If we cannot make sense of particular formulas, the first approach could always be a fallback. In either case, it is necessary to detect the mathematics in the document first.*

In this particular case, extraction of the document image shows two formulas in the middle of the citation:

$$\mathrm{A}i(z) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{i(zt+t^3/3)}dt$$

$$= \frac{1}{\pi 3^{2/3}}\sum_{n=0}^{\infty}(3^{1/3}z)^n\frac{\Gamma((n+1)/3)}{n!}\sin\frac{2(n+1)\pi}{3}.$$

[3]there is also a problem with ligatures, since "definition" is rendered as "de nition". The numbers along the top are from a graph. The document seems also to have changed since being indexed, since the reference to Watson's treatise is now citation 50, rather than 45!

Our goal is to encode these two forms in an internal format that could be transformed to XML, MathML, or through the intermediary of TEX shown below, the rendering above.

```
$${\mbox Ai}(z) = {1\over{2 \pi}}
    \int _{-\infty}^{+\infty} e^{i(zt+t^3/3)}dt$$
$${}= {1 \over {\pi 3^{2/3}}}\sum_{n=0}^\infty
    (3^{1/3}z)^n
    {{\Gamma((n+1)/3)} \over {n!}}
    \sin {{2(n+1)\pi}\over 3}$$
```

4. WHAT TO DO WITH MATHEMATICS ONCE WE'VE DETECTED IT

An obvious target is a typesetting language such as TEX in which the formulas are translated into the text strings that, if processed through TEX would have the same appearance as the formula. Figuring out the exact notation that was used by an author to produce a TEX formula is not possible in general, since there are many distinct utterances which map to the same output. But for many formulas there is a reasonably simple form that can be put together by merely observing the positional relations of symbols. A similar approach might be to produce a MathML presentation form. MathML is now "understood" by several browsers to the point of making visual displays of appropriate size and position.

An important step further would be to find a correct semantic encoding, say an expression in a computer algebra system language like Maple or Mathematica, or perhaps an encoding in *content* MathML.

5. DETECTING MATH IN POSTSCRIPT

Our technique is to use as much information as we have about the glyphs, fonts, and positions to find mathematics in a Postscript document using the heuristics described in Fateman [4]. In a TEX document, the regular font for typesetting text paragraphs is often Times-Roman while mathematics is typeset in a combination of Computer Modern Roman font (CMR10 below) and Computer Modern Math Italic font (CMMI10 below). Unfortunately, the FontInfo and FontName fields, which provide us with these details, may not survive the processing; they are optional in a Postscript font dictionary. However, at the very least we can detect when a new font is set and, further, we can assign a number to each new unique font that has been defined, and use this information to make a guess at when we are typesetting mathematics.

To give an example of an optimal case, suppose we have the following typeset expression $f(x) = y$ which we process into the following[4]:

```
f, type: text, font: CMMI10, bbox: (8555.47 ...)
(, type: 4,    font: CMR10,  bbox: (222.168 ...)
x, type: text, font: CMMI10, bbox: (555.474 ...)
), type: 4,    font: CMR10,  bbox: (1245.06 ...)
=, type: 1,    font: CMR10,  bbox: (333.36 ...)
y, type: text, font: CMMI10, bbox: (339.084 ...)
```

Each row begins with the character or string which was detected in the Postscript document. Here we have used

[4]There are actually 4 numbers after `bbox:` which specify the bounding box. We omit details due to space constraints.

the fact that TEX will typeset a function in the following manner to detect math:

1. The function name is in Computer Modern Math Italic.
2. The opening and closing parenthesis are typeset in Computer Modern Roman.
3. The simple arguments of the function are typeset in Computer Modern Math Italic.

In a tougher version of the same information we may have the following:

```
f, type: text, font: 1,  bbox: (8555.47 ...)
(, type: 4,    font: 2,  bbox: (222.168 ...)
x, type: text, font: 1,  bbox: (555.474 ...)
), type: 4,    font: 2,  bbox: (1245.06 ...)
=, type: 1,    font: 2,  bbox: (333.36 ...)
y, type: text, font: 1,  bbox: (339.084 ...)
```

Here, the font names are not present and we will need to make guesses based on the font changes and fontnumbers as to which expressions are mathematics and which are plain text.

6. IMPLEMENTATION

Imagine we have found a document on the internet that is of some interest, perhaps following in the footsteps of NEC. Or that we have found the document *BECAUSE* of NEC's citation, but we wish to index it better.

Our initial processing of the Postscript document is done by a modified version of Prescript [11], a program which modifies operators of the Ghostscript interpreter to output various information about strings and bounding box information about the typeset expressions. In our program, we are especially concerned with a small subset of Postscript operators. A short explanation of what each operator (that we are concerned with) does in Postscript followed what we augment the operator to provide follows:

- `show, kshow, widthshow, ashow, awidthshow` - Operators that paint the glyphs on the canvas. These are used to capture information such as the bounding box and the baseline.
- `showpage` - Prints a page to the output device. Used to determine when we have a new page.
- `setfont` - Changes the current font being used (may also define a new font in the process). Used to figure out font names (i.e. the fontname field), when the current font changes, and the bounding box for various characters of the font to aid in computation of clumping (discussed later). For a good representative mix of different bounding boxes, the following letters are used: X, a, t, x, p, d, l, m, n.
- `rlineto, lineto, fill, stroke` - The first two operators, `rlineto` and `lineto`, define lines to be drawn. `fill` fills in a current region with the current color. The last operator, `stroke` performs the actual drawing. These operators are captured to find divide bars which are rendered as filled rectangles using these operators.
- `moveto` - Changes the current point on the canvas.
- `scale` - Scales the coordinates that the document is defined in.
- `translate` - Translates the origin horizontally and/or vertically.

For example, from the expression $\frac{a+b^c}{de}$, the modified Postscript interpreter will return:

```
#S(command :type scale :x 0.009 :y -0.009)
#S(command :type translate :x 8000 :y -80000.0)
#S(command :type moveto :x 0 :y 0)
#S(command :type moveto :x 6401 :y 6783)
#S(command :type setfont :x 1 :y (1134 1000))
#S(box :font CMMI8 :fontnumber (1)
  :baseline 6782.99 :intlist (97 )
  :y1 6777.81 :x1 6873.02 :y0 6436.77 :x0 6444.42)
#S(command :type setfont :x 2 :y (1106 1000))
#S(box :font CMR8 :fontnumber (2)
  :baseline 6782.99 :intlist (43 )
  :y1 6888.86 :x1 7579.29 :y0 6222.22 :x0 6953.21)
#S(command :type setfont :x 1)
#S(box :font CMMI8 :fontnumber (1)
  :baseline 6782.99 :intlist (98 )
  :y1 6777.83 :x1 8000.05 :y0 6172.82 :x0 7666.69)
#S(command :type moveto :x 8036 :y 6470)
#S(command :type setfont :x 3 :y (1230 1000))
#S(box :font CMMI6 :fontnumber (3)
  :baseline 6470.0 :intlist (99 )
  :y1 6444.42 :x1 8444.42 :y0 6212.32 :x0 8110.43)
#S(command :type moveto :x 6444.44 :y 7000.0)
#S(command :type rlineto :x 2061 :y 0)
#S(command :type rlineto :x 0 :y -45)
#S(command :type rlineto :x -2061 :y 0)
#S(command :type fill :x 6444.44 :y 6955.02)
#S(command :type moveto :x 6971 :y 7764)
#S(command :type setfont :x 1)
#S(box :font CMMI8 :fontnumber (1)
  :baseline 7764.0 :intlist (100 101 )
  :y1 7777.83 :x1 7889.76 :y0 7112.01 :x0 7000.0)
```

From this output, initial processing will proceed through these steps:

1. **Basic word assembly**. In Postscript, strings are sometimes broken up into fragments so they can be typeset better. To handle this, we use the same heuristic [3] as `pstotext` and `prescript` to assemble the string fragments into words, that is,

   > ...strings separated by a distance of less than 0.3 times the minimum of the average character widths in the two strings are considered to be part of the same word.

   We make one modification to their algorithm in that we require that the fonts of the two string fragments being combined have the same font.

2. **Determine types**. Determine the type of all objects (from the "Pass one: initial separation" heuristics discussed in Fateman [5]). That is, we are looking for items that may be part of mathematical expressions. These include:

- open parenthesis/brackets that are followed immediately by close parenthesis/brackets, i.e. "()" or "[]"
- mathematical characters such as +, =, and Greek symbols
- characters that are in italics or bold (e.g. Times-Italic, Times-Bold, CMMI family of fonts)
- numeric digits in Roman typefaces (e.g. Times-Roman, CMR family of fonts)
- words that occur commonly in mathematics such as cos, sin, or tan; constants in physics such as J, K, W, or E that are typeset in Times-Roman
- punctuation such as periods, commas, colons, and semicolons

3. **Assemble structures**. Pattern match certain groups of commands to find glyphs that are created by multiple commands, for example: square roots, integral signs, and (in our example) horizontal lines:

```
#S(command :type moveto :x 6444.44 :y 7000.0)
#S(command :type rlineto :x 2061 :y 0)
#S(command :type rlineto :x 0 :y -45)
#S(command :type rlineto :x -2061 :y 0)
#S(command :type fill :x 6444.44 :y 6955.02)
```

Although our program faces some of the difficulties of optical character recognition (OCR) programs (more properly "document image analysis"), some components are much easier. Consider the task of determining the meaning of horizontal lines; during the initial processing of an OCR program, the horizontal lines could be one of many different things such as a divide bar, a subtraction sign, or, perhaps, part on an equal sign. In a Postscript document, however, the latter two are encoded in a string and have been separated out during the preprocess phase. Therefore such tasks as determining the meaning of a horizontal line are easier.

The result of our processing is as follows:

```
a, type: text, font: CMMI8, bbox: (6444.42 ...)
+, type: 1, font: CMR8, bbox: (6953.21 ...)
b, type: text, font: CMMI8, bbox: (7666.69 ...)
c, type: text, font: CMMI6, bbox: (8110.43 ...)
hline, type: hline, font: nil, bbox: (6354.48 ...)
de, type: text, font: CMMI8, bbox: (7000.0 ...)
```

The processed data is then stable-sorted vertically, then horizontally via the bounding box information of each item. The result is a list of elements that are ordered by their left-most horizontal position, with ties favoring text that is higher up in the page. This aids in the clumping part of the program where we create box structures of the recognized math. The sorted version of the example expression is below:

```
hline, type: hline, font: nil, bbox: (6354.48 ...)
a, type: text, font: CMMI8, bbox: (6444.42 ...)
+, type: 1, font: CMR8, bbox: (6953.21 ...)
de, type: text, font: CMMI8, bbox: (7000.0 ...)
b, type: text, font: CMMI8, bbox: (7666.69 ...)
c, type: text, font: CMMI6, bbox: (8110.43 ...)
1, type: text, font: CMR12, bbox: (25698.9 ...)
```

From the sorted data, we apply the clumping heuristics based on the Math/OCR programs used by Fateman [5]:

> We use a computation based only on bounding boxes. It is a kind of iterative smearing where all bounding boxes horizontally within a chosen x-smear distance are merged repeatedly until the process ceases to make any changes. Then a similar y direction merging is done. Setting these parameters is tricky. The x-smear should be larger than the inter-character space but smaller than the inter-word space. The y-smear is especially tricky since it must be considerably smaller than the inter-line space yet large enough to grab all the parts of a built-up fraction or even a multiple-line display.

From the clumping heuristic, we obtain the following built-up expression, as a Lisp data structure:

```
(vbox (hbox a + (superbox b c)) hline de)
```

One of the problems we had with building up expressions is that some programs that generate Postscript files will modify the user coordinate space of the document. We cannot assume that the units in the Postscript document in question are the standard point (1/72 inch). This can be remedied by seeing what changes are made to the current transformation matrix in the graphics state of the Postscript document. Calls to operators such as `translate`, `rotate`, `scale`, and `concat` need to be tracked and accounted for in the calculations for building up expressions (e.g. how many "units to the right" should I look for the next lexeme).

As a check, the built up expression can be run through a simple recursive printer to generate the final typesetting commands to typeset the expression:

```
$a + b ^ c \over de $
```

which is typeset into: $\frac{a+b^c}{de}$, our original expression!

It may seem that in this paper we have avoided discussing the "interesting" part, which is the parsing of 2-D mathematical expressions. In fact this ground is well-worn. Its successes and limitations, as well as our current implementation are discussed at length in our earlier paper [5]. The success of this program is substantially improved if there are no errors in identification of glyphs, font sizes and positions. This is an unwarranted assumption in the "OCR" world but it is precisely the situation for our current program. Processing Postscript bypasses the error-prone scanning of printed paper, (or worse, handwritten input of mathematics). We feel compelled to note once again that mathematical notations do not represent a solution as much as a starting point. Even a skilled mathematician, applying considerable effort and patience, may fail to understand a mathematical paper.

7. FURTHER WORK TO BE DONE

The current prototype of the system works for Postscript documents generated by LaTeX and `dvips`. There are many other programs that generate Postscript document (for example, Acrobat Distiller or `pdf2ps`) and additional work may be needed so that the system can best process Postscript documents created by these other programs.

The current program also requires the fontname fields of the fonts used in the Postscript documents to be present (as given in the "optimal" case document). Additional heuristics (or programs written in Postscript) must be devised in the case where fonts are not given.

In principle there is no difficulty in finding test sets.

For example, there are hundreds of thousands of possible test cases available from digital library collections such as the NEC Research Index. We have corresponded with the NEC project scientists, and hope to be able to augment their indexing capabilities. We have found that, since PDF can be produced by a large number of programs, and our tools are mostly oriented toward documents originally in TEX, they need more tuning (To determine from a Postscript document "what kind of source text generated you?") before we attempt a large test. There is also a rather substantial problem, if we have a large benchmark, of determining "is this result correct?" Without the original TEX, we have no automated "ground-truth" detection. Our own limited tests starting from TEX and attempting to reconstruct the original TEX suggest that the concept is reasonable but that high quality requires grueling engineering attention to detail, and incremental augmentation of the reconstruction phase [5].

Interestingly, NEC is processing new documents that are available in Postscript by essentially printing them out (virtually) and re-recognizing them with "OCR" or document understanding systems. The advantage to this is that the OCR programs have been refined (really, hacked endlessly) over more than a decade to try to deduce meta-data and layout information (paragraph ordering on multi-column pages, titles, page numbers, etc.) in a far more sophisticated fashion than is plausible *ab initio* on "raw" Postscript, using a simple program like `pstotext`. Thus NEC can build upon the years of engineering heuristic development in the OCR business instead of reduplicating it on the raw Postscript.

The OCR of mathematical formulas is unfortunately not seen as a commercially important task and thus the OCR programs do not solve *our* problem particularly well. In fact, reducing the problem to OCR of math is a reduction to a previous well known and partially solved problem. As noted, the decoding of Postscript at least gives us 100% of the characters and locations, presuming that we can figure out the font situation.

8. CONCLUSIONS

Searching for some semantic meaning to Postscript math is a challenging task, and like the similar task of optical character recognition (OCR), one that we will not solve perfectly[5]. Of course humans have difficulty reading (or writing) correct mathematics, and there is nothing to prevent one from typesetting nonsense — even ill-formed nonsense — in TEX.

Even though our problem statement seems more constrained in scope than full OCR: we are given the glyphs, positions, and (probable) fonts, there still numerous difficulties that prevent us from truly "understanding" mathematics on a page.

There is an underlying question here of how math documents should be encoded, and one that is central to the computer algebra systems building community. The increased interest in web-based documentation, tutorials, and advanced notebook-like interfaces for interactive CAS, all point to the important relationship that must be recognized between mathematics, documents and computation. This is in part recognized by Design Science, a company which has received (Dec, 2003) NSF funding to make web-based math accessible for visually disabled. (http://www.dessci.com/en/). It appears that they hope to do this by building MathML tools. Unfortunately a survey they conducted using Google, in September, 2003, on a few terms like "factoring polynomials" shows that out of hundreds of pages retrieved, not one uses MathML. Rather they use HTML, PDF, etc. [9] Thus tools like ours, translating from documents to math, may be critical to making math accessible. We have shown an approach in principle, but an industrial strength engineering solution is needed if it is to be widely applied.

It is not likely that people will go back and encode, by hand, legacy web mathematics pages. For new documents, the role of the author and editor of a mathematical journal article could be expanded beyond that of merely supplying and proof-reading camera-ready or computer-ready TEX. In the future we hope that authors, who have become used to typesetting in the publication process, will be encouraged to include MathML or another semantic encoding of their writing (instead of, or in addition to, TEX or other presentation style source). An outright requirement of such an encoding is unreasonable, since the formal inclusion of new results into the existing formal structure of (say) OpenMath is a substantial and potentially tedious and distracting effort. Once such an encoding is provided, we hope that the publication process will maintain this encoding instead of obscuring it through the production of page proofs and printed pages devoid of their digital origins. The value of such a preservation would seem obvious, but apparently has not been realized by those publishers (including ACM) who publish their online journals as page images, removing the TEX, MathML, or other source encoding.

9. ACKNOWLEDGMENTS

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Michael Yang was supported in part by an NSF undergraduate research grant during the summer of 2002. Thanks to referees of an earlier draft and to N. Soiffer and R. Miner for comments.

10. APPENDIX: USING THE PROGRAMS

This section gives a quick overview on how to use the Postscript mathematical expression locating/extraction system. We have tested the system using the following programs on a Sun/ Solaris platform:

- Aladdin Ghostscript 6.0
- Allegro Common Lisp 6.0
- LATEXWeb2C 7.3.1
- `dvips 5.86`

10.1 Preprocess

The processing begins with a Postscript document. For experiments it is convenient to create a document starting with TEX so the result can be compared with the original document. In the Preprocess step, a sample TEX document is converted to a `dvi` file via LATEX or a related program and

[5]Of course, it usually doesn't pay to reduce a problem to a harder one: as just mentioned, if the OCR of printed math were to be perfected, the reduction for our problem would work!

then to Postscript via `dvips`. The commands that we ran were:

```
latex file.tex
dvips -Ppdf file.dvi
```

which produced a Postscript file called `file.ps`. The Postscript document is run through a modified version of Ghostscript, a common Postscript interpreter available on many different platforms. The modifications to Ghostscript are achieved by simply adding a prolog (`mlypre.ps`) and epilog (`mlypre2.ps`) to the main Postscript document as it is run through the interpreter. The modifications generate output that includes the bounding boxes, `currentpoint`s of each string (which we are taking as the baselines for most text), the font (if it exists), and the string itself, which in our case we change to integer representation of each of the individual characters. This transformation is done to address issues of portability in the next phase. In addition, commands such as `rlineto`, `moveto`, and `stroke` are also outputted for further analysis.

The following command does the initial processing:

```
gs -q -dNODISPLAY -soutfile=output mlypre.ps
  psfile mlypre2.ps quit.ps
```

where *output* is the destination file and *psfile* is the source Postscript document that needs to be examined.

The output generated is a text file that constitutes definitions of a series of Lisp lists with structures (i.e. `box` and `command`) as its elements. `box`s are structures for strings that have been found and `command`s are structures for Postscript commands such as `moveto` or `rlineto`. The variable `pages` tells how many pages were processed from the Postscript document. Each list is named `page[N]` where `N` is a integer such that $0 < N <=$ `pages`.

10.2 Process

The data from the preprocess phase is then processed by the Lisp programs `pass1` and `pass2` which implement phase 1 and phase 2, respectively, of three phase algorithm described by Fateman [4]. This can be done by loading `util.cl` into Lisp and calling `(process page[N])`. These programs are rudimentary in various respects, having been written to recognize formulas occurring in integral tables. The programs do not effectively handle matrices, for example.

To view the actual bounding boxes and baselines on the Postscript document in question, `ps-draw` can be run on the output of `(process page[N])` to generate the necessary Postscript commands to draw the boxes. We currently do not have an automated way of displaying the boxes in the page, but the commands can be cut and pasted into the Postscript file directly on the page in question, just before the `showpage` operator (n.b. most of the time, Postscript documents have common Postscript commands aliased at the beginning of the document for efficiency issues. Therefore, you may not see an explicit `showpage` command; for example, `latex` uses `eop`.

10.3 Typesetting

To output TEX commands from the built-up expression generated by `process`, run the Lisp program `typeset`. Ideally this would reproduce the typesetting command for the formula, but at best one can expect is one of the (many) possible forms that will produce the formula or an approximation of it.

11. REFERENCES

[1] Adobe Systems Incorporated, *Postscript Language Reference, Third Edition.* Reading, Massachusetts: Addison-Wesley Publishing Company, 1999.

[2] CEIC/IMU. (2002, April). [Online]. Available: http://www.ceic.math.ca/ceic_docs/best_practices/Best-Practices.pdf

[3] Digital Equipment Corporation. pstotext. [Online]. Available: http://www.research.compaq.com/SRC/virtualpaper/pstotext.html

[4] R. Fateman, "How to find mathematics on a scanned page," in *Proc. SPIE Vol. 3967, Document Recognition and Retrieval VII.* Daniel P. Lopresti; Jiangying Zhou (eds), December 1999, pp. 98–109.

[5] R. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell, "Optical character recognition and parsing of typeset mathematics," *Journal of Visual Communication and Image Representation*, vol. 7, no. 1, pp. 2–15, March 1996.

[6] JSTOR. [Online]. Available: http://www.jstor.org/about/bibliography.html

[7] S. Lawrence, "Online or invisible," *Nature*, vol. 411, no. 6837, p. 521, 2001.

[8] G. Michler, "How to build a prototype for a distributed digital mathematics archive library," *Workshop: Retro-Digitalization of mathematical journals and automatic formula recognition*, 2001. [Online]. Available: http://www.emis.de/proceedings/MKM2001/michler.pdf

[9] R. Miner and P. Topping, "Math on the web: A status report september, 2003 focus: Interactive math," *Design Science, Inc.* [Online]. Available: http://www.dessci.com/en/reference/webmath/status/status_Sep_03.htm

[10] NEC Research Institute. Researchindex. [Online]. Available: http://www.neci.nec.com/~lawrence/researchindex.html

[11] C. Neville-Manning and T. Reed. (1996, June) A Postscript to plain text converter. [Online]. Available: http://www.nzdl.org/html/prescript.html

[12] A. Odlyzko, "The future of scientific communication," in *Access to Publicly Financed Research: The Global Research Village III.* P. Wouters and P. Schroeder (eds), 2000, pp. 273–278.

[13] T. A. Phelps and R. Wilensky, "Multivalent documents: Anywhere, anytime, any type, every way user-improvable digital documents," *Communications of the ACM*, June 2000.

[14] University of California, Berkeley. Center for Information Technology Research In the Interest of Society (CITRIS). [Online]. Available: http://www.citris.berkeley.edu/