Josef B. Baker; Alan P. Sexton; Volker Sorge
Towards Reverse Engineering of PDF Documents

In: Petr Sojka and Thierry Bouche (eds.): Towards a Digital Mathematics Library. Bertinoro, Italy, July 20-21st, 2011. Masaryk University Press, Brno, Czech Republic, 2011. pp. 65--75.

Persistent URL: http://dml.cz/dmlcz/702603

Towards Reverse Engineering of PDF Documents

Josef B. Baker, Alan P. Sexton, and Volker Sorge

School of Computer Science, University of Birmingham
Email: {j.baker|a.p.sexton|v.sorge}@cs.bham.ac.uk
URL: http://www.cs.bham.ac.uk/~{jbb|aps|vxs}

Abstract. We present a progress report on our ongoing project of reverse engineering scientific PDF documents. The aim is to obtain mathematical markup that can be used as source for regenerating a document that resembles the original as closely as possible. This source can then be a basis for further document processing. Our current tool uses specialised PDF extraction together with image analysis to produce near perfect input for parsing mathematical formula. Applying a linear grammar and specific drivers for each output format to this input, we can produce an accurate reproduction of formulae when presented with their coordinates. In this paper we will show how this information can be exploited to discover the locations of both inline and display formulae, and also to perform rudimentary layout analysis of the whole document, identifying structures such as headings and paragraphs.

1 Introduction

Converting PDF files into alternative formats can offer users the ability to do more than just view or print a document. Indeed, there exist a number of software tools that enable their conversion into formats such as ASCII or Word, along with the copy-to-clipboard function available with the majority of PDF viewers. However, all the currently existing tools focus on the extraction of regular text from documents and none are capable of faithfully extracting and translating non-textual components, such as the document's format and styling, mathematical formulae or tables. We are working on a system that allows faithful reverse engineering of entire PDF documents, with a particular emphasis on converting mathematical content into markup languages like LaTeX or MathML.

In previous work we have focused on the reconstruction of mathematical formulae in PDF documents and their parsing into LaTeX and MathML using formal grammars. While this yielded good results and enabled reproduction of formulae very close to the original, the main drawback of the technique was that formulae had to be manually identified and clipped from PDF documents.

In this paper we report on a significant extension to our previous work by automatically identifying formulae through the analysis of symbols, fonts and their spatial relationships within each page. Furthermore, we show how this allows us to extract both text and mathematical content and we demonstrate how this information can be used to perform layout analysis of a page. This

Petr Sojka, Thierry Bouche (editors): DML 2011, Towards a Digital Mathematics Library, pp. 65–75.
 ISBN 978-80-210-5542-1

paper is a progress report on the current state of the overall project. An evaluation of the effectiveness of our approach is presented in [3], which compares our results to those of the Infty mathematical document analysis system [9].

In Sec. 2 we review the key points of our previous approaches to formula recognition from PDF documents as described in [2]. Sec. 3 then explains how this process is extended to perform whole page analysis, including layout analysis and recognition of inline mathematics. We discuss advantages and disadvantages of our approach as well as future improvements in Sec. 4 and conclude in Sec. 5.

2 Previous Work

In [2] we demonstrated an approach to formula recognition from PDF documents, which bypassed the standard but troublesome OCR stage by replacing it with PDF analysis to produce high quality results. Here we summarise this approach.

The PDF specification is very large and complex, covering 9 different versions. Some PDF files even store their contents in a raster image format and contain no more usable information than the images themselves. We therefore focus on a subset of PDF, which encode symbol information in an analysable form and covers a large amount of published scientific and mathematical material.

The files that we can parse use either Type 1 or Type 2 fonts, with their respective font encoding and width objects contained in the PDF file. We also require a valid PDF file, as many have corrupt reference tables, missing or erroneous objects. This generally limits our software to files generated from LaTeX, but not exclusively, as we have also had success working with those generated from `Troff`, `OpenOffice`.

We begin by rendering a PDF file to a TIFF image, from which a user is able to select specific areas of mathematics, an example of which is shown in Fig. 1. For each of these clips, the co-ordinates and dimensions are calculated, along with those of every connected component contained in that area. The information is then saved together with other meta data about the image, such as the name and page number of the file it was selected from. This meta data is then used by a PDF extractor in order to find the correct page within the file to process.

The extractor makes two passes over the file, the first of which is to collate the required content streams, which hold instructions for placing and displaying characters, lines and images, along with a number of other resources, including font dictionaries and character names. The second pass of the file sequentially processes each of these instructions to identify and extract each symbol, its respective font name, size and base point along with all lines and their coordinates.

The information produced by this process, whilst sufficient for the analysis of one dimensional text, is not accurate enough for the recognition of the two

dimensional relationships that occur with mathematical formulae. Therefore the connected components obtained during image analysis are registered to the characters and lines, resulting in precise spatial information.

The next stage is to parse this input, with the ultimate aim of producing a version of the formula in an output such as LaTeX. This is achieved by using a parser based on a linear grammar, a heavily modified version of that described by Anderson [1], in which spatial relationships between symbols in a given formula were analysed. His grammar, whilst very efficient, was quite restrictive and lacked the flexibility to cope with the different styles of typesetting that are common today. Therefore we removed many of the spatial relationship restrictions and extended the grammar to include accents, under bars, over bars, braces and multiline formulae and also to analyse symbol fonts, sizes and alignment.

This analysis produces a string representing the formula, an example of which is shown in Listing 1.1, the linearised version of Fig. 1.

Parse trees are then generated from the linearised string and used as intermediate representations for subsequent translation into mathematical markup. Different types of parse trees are generated, from simple parse trees that hold only the basic information on the relationships between symbols in a formula, to more complex parse trees that incorporated information on font, character sizes, and relative spacing between symbols.

Finally, output specific drivers are used to translate these parse trees into mathematical markup. Two main drivers have been created: One producing LaTeX code that faithfully reproduces the original formula taking spatial information into account and sometimes inserting this information explicitly into the produced code. A second is aimed at generating LaTeX that closely resembles code that could have been written by a human mathematician. Whilst the latter does not necessarily reproduce *exactly* the same formula as in the original document, it has the advantage that its LaTeX lends itself more to a semantic evaluation as well as cleaning up potential layout mistakes introduced by the author. The idea to use more than one driver to implement these goals is primarily to have a clear separation that enables an easy parameterisation of our software tool depending on the target application.

The resultant LaTeX code produced for Fig. 1 is shown in Listing 1.2. Observe that the translation is a straightforward translation into standard LaTeX without assuming any third party packages in the LaTeX environment (e.g., we use array environments as opposed to anything more sophisticated, such as, for example, those provided by some of the `amsmath` packages). Note also that the `cmbxa` prefix command is used to set its argument into a specialised font defined in the preamble of the resulting LaTeX file. While this treatment has the drawback that the produced LaTeX is less intuitive, it has the advantage that we can reproduce any specialist font that is actually used in the document. As a consequence, we can deal not only with documents that have been produced within a standard LaTeX environment, but also with those that have been produced by other

tools or where standard fonts have been replaced — a common practice among publishers.

In order to regain more intuitive LaTeX that is closer to what a user would actually write, one could introduce font mappings to specialist commands, as well as use specialist environments for matrices or multiline formulae, etc. However, this will require a more elaborate level of analysis, which is currently not implemented, but might be added in the future.

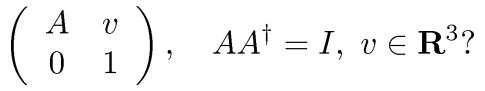

Fig. 1. Clipped image of a formula

```
matrix(<parenleftbigg, CMEX10, 9.963>)(row(col(<A, CMMI10,
    9.963>)col(<v, CMMI10, 9.963>))row(col(<zero, CMR10, 9.963>)
    col(<one, CMR10, 9.963>)))(<parenrightbigg, CMEX10, 9.963>)
    w3 <comma, CMMI10, 9.963> w4 sup <A A, CMMI10, 9.963>)(<
    dagger, CMSY7, 6.974>) w3 <equal, CMR10, 9.963> w2 <I comma,
     CMMI10, 9.963> w4 <v, CMMI10, 9.963> w2 <element, CMSY10,
    9.963> w2 sup(<R, CMBX10, 9.963>)(<three, CMR7, 6.974>) w1 <
    question, CMR10, 9.963>
```

Listing 1.1. Linearised version of clip

```
\[\left(\begin{array}{cc} A & v \\ 0 & 1 \end{array}\right)
    , \quad AA ^{ \dagger } = I , \quad v \in \
    cmbxa{R} ^{ 3 } ? \]
```

Listing 1.2. Output LaTeX code

Further drivers consist of a module producing Presentation MathML, as well as one that generates input for Festival, a speech synthesis tool. Most of the drivers focus upon the reconstruction of mathematics for presentation. However, we have also made some initial steps towards supporting a semantic interpretation of the parsed mathematical content [4], by constructing tools for semantic ground truthing of mathematical documents.

3 Layout Analysis

The main change over our previous work is that we now analyse and translate entire documents automatically rather than just single, manually clipped formulae. As a consequence we need to analyse the layout of each page of the document in order to reproduce it as faithfully as possible. This requires both changes to the extraction process and new drivers to perform the layout analysis. The former is realised by adding a pre-processing step in the extraction

process that identifies single lines on a page. The latter consists of two steps: separating mathematics and regular text in single lines and attempting to reassemble specific print areas from consecutive lines. This information can then be exploited during the translation of extracted content into a final output format.

3.1 Linewise Extraction of PDF Content

In order to extract character information for the whole document, the input PDF file is initially burst into single page PDFs, which are all rendered to TIFF images. For the purpose of connected component to symbol registration, each image and its respective PDF file are then treated as standard clips. From this we attempt the first stage of layout analysis, where we try to identify any columns and lines comprising the page. Projection profile cutting is used for this task though horizontal cuts, i.e. those between lines, are only made if the white space between symbols exceeds a certain threshold. This is found by ordering the connected components by their top y coordinate and calculating the median white space between each pair of sequential components, when the value is greater than zero.

The result of this process is a number of files, each representing a line, containing a list of symbols and their attributes. Each line is then linearised to produce its string representation as discussed in Sec. 2. In addition, we pass the bounding box information for each line, which can be used in the subsequent analysis steps.

Consider the example given in Fig. 2, a page from a freely available book on function theory [7], where the left hand side is an image of the original PDF page (observe that for the example it is not necessary to explicitly read individual characters). This page will be broken down into 26 lines and for each of the identified lines a representation will be computed. For instance the representation for the second line would be of the form

```
894 1057 248 58 <P r o o f period , CMBX10 , 9.963>
```

where the first four integers represent the bounding box information in the form of the x, y coordinate of the line on the page plus height and width of the line. Given this line-by-line information, the main layout analysis proceeds in two steps. First, lines are separated into text lines and display style mathematics, which are then grouped together into paragraphs and further classified.

One interpretation error can be observed in the fourth line of the multi-line math expression in Fig. 2. Here the author has forgotten a closing parenthesis in the superscript expression of the last B. In our current implementation fences are explicitly opened with LaTeX `left` and closed with `right` commands. While our implementation keeps track of matching fences and if necessary adds dummy `left` or `right` commands if there is a mismatch, this is done for the entire formula and not for subgroups inside the formula (e.g., like a superscript). As a consequence the opening parenthesis in the superscript is closed only after the entire expression, which leads to the misinterpretation

Spanning Line
Flushleft Line
Multi Line Math
Flushleft Line
Single Equation
Unindented Paragraph
Single Line Math
Unindented Paragraph
Single Line Math
Unindented Paragraph
Flushleft Line
Unindented Paragraph

Proof.

$$\begin{aligned}\|[\exp(n(B-I))-B^n]x\| &= \|e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}(B^k-B^n)x\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|(B^k-B^n)x\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|(B^{|k-n|}-I)x\| \\ &= e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|(B-I)(I+B+\cdots+B^{(|k-n|-1)}x\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\|(B-I)x\|.\end{aligned}$$

So to prove (11.22) it is enough establish the inequality

$$e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\le\sqrt{n}. \tag{11.23}$$

Consider the space of all sequences $\mathbf{a}=\{a_0,a_1,\dots\}$ with finite norm relative to scalar product

$$(\mathbf{a},\mathbf{b}):=e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}a_k\overline{b_k}.$$

The Cauchy-Schwarz inequality applied to $\mathbf{a}$ with $a_k=|k-n|$ and $\mathbf{b}$ with $b_k\equiv 1$ gives

$$e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\le\sqrt{e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}(k-n)^2}\cdot\sqrt{e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}}.$$

The second square root is one, and we recognize the sum under the first square root as the variance of the Poisson distribution with parameter n, and we know that this variance is n. QED

11.7 Convergence of semigroups.

We are going to be interested in the following type of result. We would like to know that if A_n is a sequence of operators generating equibounded one parameter semi-groups $\exp tA_n$ and $A_n\to A$ where A generates an equibounded semi-group $\exp tA$ then the semi-groups converge, i.e. $\exp tA_n\to\exp tA$. We will prove such a result for the case of contractions. But before we can even formulate the result, we have to deal with the fact that each A_n comes equipped with its own domain of definition, $D(A_n)$. We do not want to make the overly

Proof.

$$\begin{aligned}\|[\exp(n(B-I))-B^n]x\| &= \|e^{-n}\textstyle\sum_{k=0}^{\infty}\frac{n^k}{k!}\left(B^k-B^n\right)x\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|\left(B^k-B^n\right)x\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|\left(B^{|k-n|}-I\right)x\| \\ &= e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}\|(B-I)\left(I+B+\cdots+B^{(|k-n|-1)x}\right\| \\ &\le e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\|(B-I)x\|.\end{aligned}$$

So to prove (11.22) it is enough establish the inequality

$$e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\le\sqrt{n}. \tag{11.23}$$

Consider the space of all sequences $\mathbf{a}=\{a_0,a_1,...\}$ with finite norm relative to scalar product

$$(\mathbf{a},\mathbf{b}):=e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}a_k\overline{b_k}.$$

The Cauchy-Schwarz inequality applied to $\mathbf{a}$ with $a_k=|k-n|$ and $\mathbf{b}$ with $b_k\equiv 1$ gives

$$e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}|k-n|\le\sqrt{e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}(k-n)^2}\cdot\sqrt{e^{-n}\sum_{k=0}^{\infty}\frac{n^k}{k!}}.$$

The second square root is one, and we recognize the sum under the first square root as the variance of the Poisson distribution with parameter n, and we know that this variance is n. QED

1 1 . 7 Convergence of semigroups.

We are going to be interested in the following type of result. We would like to know that if A_n is a sequence of operators generating equibounded one pa − rameter semi-groups $\exp tA_n$ and $A_n\to A$ where A generates an equibounded semi-group $\exp tA$ then the semi-groups converge, i.e. $\exp tA_n\to\exp tA$. We will prove such a result for the case of contractions. But before we can even formulate the result, we have to deal with the fact that each A_n comes equipped with its own domain of definition, $D(A_n)$. We do not want to make the overly

Fig. 2. Page 317 from [7]. Original on the left and rendered LaTeX output on the right.

that can be observed in Fig. 2. Having analysed this error, we plan to modify our fence balancing algorithm to respect formula subgroups.

3.2 Analysis of Lines

All linearised lines of a page are then parsed using a LALR parser, resulting in a collection of parse trees. These parse trees are an intermediate representation, one for each line, containing structural information that can be exploited for the next steps in the layout analysis as well as in subsequent translation into output formats.

Each line is analysed separately and classified by whether it is primarily a text line or a math line. The single elements in a line are translated by linearly assembling consecutive words, identifying sequences of mathematical expressions and assembling them into single inline math formulae. A line is then treated as a text line if it

(a) contains only a sequence of words,
(b) if it contains at least two consecutive words and the number of inline math expressions is not larger than the number of words,
(c) contains more than three consecutive words regardless of the number of inline math expressions.

Everything else will be treated as display style mathematics.

In our example in Fig. 2 we get 8 math lines, whereas all others are recognised as text lines, possibly containing inline mathematics.

3.3 Assembling Vertical Areas

In a next step we then combine consecutive lines as much as possible to assemble meaningful multi-line areas. Here we also exploit the bounding box information of each line by comparing it with the overall dimension of the print area of the page. That latter can be easily computed by combining all bounding boxes of all lines. This additional structural information can be further exploited for setting content by the output drivers.

Consecutive display-style math lines are combined into single multi-line math expressions. We thereby distinguish four different types of math expressions:

Single Line Math A single display style math expression. Both previous and next lines (if either exist) have to be text lines.

Multi Line Math A contiguous sequence of display style math lines.

Single Equation A single line display style math expression where a tag has been identified that might function as a label for the formula. Tags are identified if (1) a math line starts within a small threshold of the left margin or ends within a small threshold of the right margin, but not both, and (2) there is a discernible distance between the leftmost (or rightmost) expression and the other expressions of that line. An identified tag can be subsequently exploited by any translation driver.

Multi Line Equation A contiguous sequence of display style math lines where some lines have been identified as equation lines in the above sense.

Similar to math lines we also combine consecutive text lines into paragraphs, where paragraphs are separated if

(a) there is a change of font size,
(b) the vertical space between lines is larger than the arithmetic median of vertical space between all consecutive text lines identified on the page,
(c) the horizontal orientation of lines changes,
(d) if a line has a left indentation or the previous line ends prematurely.

We again distinguish a number of different types of single lines and paragraphs, depending on their spatial relationship to the text margins:

Spanning Line A single line starting at the left margin and ending at the right margin.

Flushleft Line A single line starting at the left margin but ending observably before the right margin.

Flushright Line A single line ending at the right margin but starting observably after the left margin.

Centred Line A line that both starts and ends observably after the left margin and before the right margin, respectively. It does not have to be fully centred around the horizontal centre of the text area.

Indented Paragraph A paragraph of consecutive lines, where the first line is a flushright line, while all other lines are spanning lines, with the exception of the last line, that can be a flushleft line.

Unindented Paragraph A paragraph of consecutive lines, where all lines are spanning lines, with the exception of the last line, that can be a flushleft line.

Centred Paragraph A paragraph consisting of consecutive centred lines. This paragraph can be both ragged left and ragged right.

Observe that for all the above we allow for a certain fuzziness, i.e., a line only has to match within a small threshold of the left or right margin for classification.

For our example in Fig. 2 the result of the layout analysis is given in the middle column. We can see that the topmost line is recognised as a spanning line, simply because it starts and ends with the margins of the text area in spite of the significant white space in the line. Also note that the fifth area is recognised as Single Equation with a right hand tag (11.23). On the other hand the third area is classified as Multi Line Math although its fourth line is within the right margin of the text area and could be considered an Equation. This is due to the fact that there is no rightmost expression that has significant distance to the other expressions.

3.4 Translation into Markup

Once the layout analysis is complete, specific drivers are employed to translate the content into actual markup. Currently we have two drivers, one for MathML

and one for LaTeX markup. The most developed driver is a LaTeX driver, which attempts to set the text components as faithfully as possible according to the classification derived in the layout analysis. For the translation of formulae we make use of the already developed mechanisms described in Sec. 2. In addition the contained font and spacing information is exploited to set characters and words in the correct font and size as well as to include additional space if necessary.

The result of the LaTeX driver for our example is given in the right column of Fig. 2. While the actual output is already close to the original input document, there are still a number of discrepancies. We will therefore use this example when we discuss some of the shortcomings of the translation in the next section.

4 Discussion

We have evaluated our current approach of combined layout analysis and formula translation quantitatively against a small ground truth set of articles similar to the one presented in [8]. These results are presented in [3] together with a comparison to the results of the Infty system [9] on the same data, which uses a conventional OCR approach to extract content and layout. The paper also presents a qualitative comparison of our results with Infty's.

In this section we will now concentrate on a qualitative discussion exclusively for the results of our procedure and in particular point out some of the shortcomings we have identified and that we intend to address in the future.

The advantages of using projection profile cutting to find lines are that of speed and efficiency, and it also works well on many standard layouts including those with multiple columns. However the presence of figures, tables and vertically overlapping, but not touching lines can severely impact its performance. Also, the limits on large equations are sometimes erroneously treated as separate lines. Therefore, a major improvement would be to use a bottom up approach for line finding, employing image blurring, or to take into account more of the information available including the semantics of the page.

In general, the current strict order of first identifying lines and then linearising these separately is not ideal as it precludes some of the advantages of our grammar for linearising multi-line mathematical expressions. As demonstrated in [2], our grammar is capable of recognising and marking up certain alignment points when parsing multiline mathematical expressions. However, when parsing each line separately, these alignments can not be detected. This effect can be seen in our example, where the Multi Line Math expression is not correctly aligned.

Other obvious alignment problems can be observed in the section heading in the example in Fig. 2, which, while being correctly recognised as a Flushleft Line, is nevertheless too spread out. This is a consequence of the current mechanism for detection of white space, which divides it into only five classes, which are computed relative to the average distance between characters in

expressions. While these spacing classes correspond to the spacing design in LaTeX [6], we do not use any absolute thresholds but only relative values. This makes us independent from the actual tool that has produced the PDF as well as from any specialised fonts used by the authors. Obviously, the spacing information can therefore only be seen as a heuristic guide and is not necessarily adhered to by all documents that we consider.

While this system to classify internal space in formulae is deliberately coarse grained in order to aid the assignment of semantic information to components of single mathematical formulae (see [4] for details), in the setting of text lines it has the drawback that all white spaces above the relative threshold are not further distinguished and consequently replaced with the same explicit large space. A more sophisticated treatment of white space, possibly with the explicit representation of distances, might ameliorate this problem.

This could then also lead to a more effective approach to horizontally separate text areas in single lines or in vertically separate paragraphs. For example, the very first line of Fig. 2 is classified exclusively as a Spanning Line. However, it would be desirable to be able to split it at the position of the largest white space in order to enable better recognition of the single components, i.e., running header and page number. This would then also allow assigning certain semantic properties to areas, such as title, section headings and page numbers, as the Infty system does, and which would lead to a more human-like LaTeX translation and therefore to a better modelling of vertical space.

Finally, our current algorithm for deciding whether a line is primarily mathematics containing some embedded text or primarily text including some inline mathematics is rather ad hoc. A more sophisticated mechanism to distinguish mathematics from text and, in particular, display mathematics from inline expressions, such as the techniques proposed in [5], will be explored in the future.

5 Conclusion

In this paper we have presented significant improvements over previous iterations of our software. By automating the location of mathematical formulae, we have removed the most costly component, in terms of operator time, from the system; that of manual clipping. We have also shown how the system can be extended to not only deal with formula recognition, but also full layout analysis. However, this is at an early stage of development and we envisage significant improvements in the future, some of which we have discussed in the previous section. For a full comparison to the Infty system [9], we refer the reader to [3]. These results demonstrate the effectiveness of our approach.

Acknowledgements. This work is supported by the EuDML project, which is in turn partly financed by the European Union through its Competitiveness and Innovation Programme (Information and Communications Technologies Policy Support Programme, “Open access to scientific information”, Grant Agreement No. 250503).

References

1. Anderson, R.H.: Syntax-Directed Recognition of Hand-Printed Two-dimensional Mathematics. Ph.D. thesis, Harvard University, Cambridge, MA (1968).
2. Baker, J., Sexton, A., Sorge, V.: A linear grammar approach to mathematical formula recognition from PDF. In: Proceedings of Intelligent Computer Mathematics (2009).
3. Baker, J., Sexton, A.P., Sorge, V., Suzuki, M.: Comparing approaches to mathematical document analysis. In: 11th International Conference on Document Analysis and Recognition (to appear) (2011).
4. Baker, J., Sexton, A., Sorge, V.: Faithful mathematical formula recognition from PDF documents. In: 9th IAPR International Workshop on Document Analysis Systems, Extended Abstracts. pp. 485–492. ACM Press, Boston, USA (2010).
5. Garain, U.: Identification of mathematical expressions in document images. In: Document Analysis and Recognition, International Conference on. pp. 1340–1344. IEEE Computer Society, Los Alamitos, CA, USA (2009).
6. Mittelbach, F., Goossens, M.: The LaTeX Companion. Pearson Education, 2e edn. (2005), TeX spacing table, page 525.
7. Sternberg, S.: Theory of functions of a real variable (2005), http://www.math.harvard.edu/~shlomo/docs/Real_Variables.pdf
8. Suzuki, M., Uchida, S., Nomura, A.: A ground-truthed mathematical character and symbol image database. In: Proc. of ICDAR. pp. 675–679. IEEE Computer Society (2005).
9. Suzuki, M.: Infty (2011), http://www.inftyproject.org