Recognition of mathematical texts

Bachelor thesis

Miroslav Hrdina

Brno, fall 2012
Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Miroslav Hrdina

Advisor:  doc. RNDr. Petr Sojka, Ph.D.
Acknowledgement

I would like to thank to my supervisor doc. RNDr. Petr Sojka, Ph.D for his time, suggestions and advices during my work on the thesis. My gratitude is also directed towards Mgr. Michal Růžička for his help and for answers to my questions. Finally I would like to thanks to my family for their support and nerves of steel.
Abstract

This work aims to resolve the OCR issue for EuDML project with the most important part being the recognition of mathematical content. Goal of this work was to create workflow for mathematical OCR, which transforms the mathematical publications to indexable recognized output. This was achieved by using the InftyReader OCR software. The chosen recognition workflow was tested and results were evaluated. During recognition numerous problems with the software and recognized data were encountered. Part of these problems were resolved, but some are still pending and will be resolved in the future.
Keywords

OCR, math recognition, InftyReader, Tralics, MathML, EuDML
Contents

1 Introduction ......................................................... 1

2 OCR Technologies and Formats ................................. 3
  2.1 OCR Process .................................................. 3
    2.1.1 Scanning a document .................................. 3
    2.1.2 Preprocessing the image .............................. 4
    2.1.3 Analyzing the layout of the page and segmentation 4
    2.1.4 Feature extraction .................................... 4
    2.1.5 Classification .......................................... 5
    2.1.6 Correction ............................................. 5
    2.1.7 Creation of output file in chosen file format ...... 5
  2.2 Mathematical OCR ........................................... 5
    2.2.1 Difference from text OCR ............................ 5
    2.2.2 OCR process on mathematical documents .......... 6
    2.2.3 Layout Analysis ....................................... 6
    2.2.4 Recognition ........................................... 7
    2.2.5 Documents Structure Analysis ....................... 8
  2.3 Software and formats ....................................... 8
    2.3.1 InftyReader ........................................... 8
    2.3.2 LaTeX .................................................. 9
    2.3.3 MathML ................................................ 9
    2.3.4 Tralics ................................................. 10

3 OCR workflow for EuDML ........................................... 11
  3.1 DML-CZ ....................................................... 11
  3.2 EuDML project ............................................... 11
    3.2.1 EuDML workflow .................................... 12
  3.3 Setup and context .......................................... 13
    3.3.1 Access ............................................... 13
    3.3.2 The Data ............................................. 13
    3.3.3 Using InftyReader ................................. 14
## 3.3.4 Infty.exe

3.3.5 Using Tralics

3.3.6 Possible workflows

### 3.4 Chosen method

3.4.1 Batch processing

3.4.2 InftyReader errors

3.4.3 Other issues

3.4.4 Post processing

3.4.5 Final decision

### 4 Testing the workflow

4.1 Evaluation methods

4.2 Error testing

4.3 OCR quality testing

4.4 Method testing evaluation

### 5 Conclusion and Future work

### A Batch scripts

### B Unix sed fixes
Chapter 1

Introduction

With the expansion of OCR technology many text documents were or are about to be digitalized. The projects to create digital libraries holding digitalized documents of cultural heritage started in many nations. The smaller digitalization projects on universities and libraries have progressed into cooperating with others to create a bigger common digital libraries which would provide better functionality to users. Even for the mathematical literature were created specialized digital libraries. For example the DML-CZ (see section 3.1) in Czech Republic or NUMDAM in France. To create common mathematical digital library in Europe, the project EuDML started (section 3.2). As a partner of this project DML-CZ has committed to cooperate in development.

The main purpose of digital library is to provide its users searching ability to find desired literature they are looking for. One of the goals of EuDML project is to allow their users to search math formulae. In order to achieve that goal, the output from OCR process needs to be indexable. This is not a problem in text documents but can be difficult to achieve with mathematics. If the math search has to be available, the OCR process needs to use special software which will recognize the math formula and its structure and then create output in format which can hold this information.

For this purposes was used software InftyReader (see sections 2.3.1 and 3.3.3). The work with this software and creating the workflow built on top of it is the main topic this thesis deals with. The goal of this work is to use the workflow to perform OCR on all DML-CZ

1. optical character recognition
data. Results of the OCR then will be included in the EuDML digital library. For the final output to be created, there are two possible methods that can be used. These methods need to be tested and the better one depending on the circumstances will be chosen.

The work was segmented into three chapters. The first one, OCR Technologies and Formats offers the information on OCR process, mainly the difference between the OCR of text and OCR of math. In addition in the chapter are mentioned the most important software and the file formats involved in the work. The second chapter is named OCR workflow for EuDML. Beginning of this chapter offers information about the EuDML project and its goals. This is followed by sections dealing with usage of the OCR software and choosing the method for OCR. The main issue of this chapter is creating the workflow and dealing with problems, which have occurred. Also section describing the recognizing data can be found in this chapter. The third chapter named Testing the workflow describes the testing of the chosen method and evaluation of the tests. The conclusion chapter summarizes the results of the work and deals with the possibility of improvement of the OCR in the future.
Chapter 2

OCR Technologies and Formats

The purpose of OCR remains the same, whether we are recognizing the newspapers, a book or a scientific document – to get the recognized document with the minimum differences from the original image. However, the chosen software and technology behind this software would not be the same, neither will be the output format.

2.1 OCR Process

The OCR process consists of several steps, which transform an image file to recognized text file. There were many OCR systems developed to perform OCR on documents, using several approaches and achieving various accuracy. Some can only recognize specific languages or types of documents. Basic steps that are used to OCR an ordinary text document are the following: scanning a document, preprocessing, analyzing the layout and segmentation, feature extraction, classification, correction, creation of the output.

2.1.1 Scanning a document

To perform OCR a file of non-recognized or partly recognized text is needed. This file is usually created via scanning a page from a book, magazine, newspaper etc. by scanner. Scanning resolution should be at least 300 DPI\footnote{dots per inch} in order to capture as much image information as possible. Good OCR system must understand a lot of image formats. Most common are BMP, JPEG, PNG, TIFF etc. PDF support is desired as well.

\footnote{dots per inch}
2. OCR Technologies and Formats

2.1.2 Preprocessing the image

OCR results are greatly influenced by quality of the source. It is important that before the character recognition, the image file is preprocessed. Preprocessing an image will remove image defects and achieve a better results in followed recognition. The typical image defect is noise. Noise can be removed or suppressed by using of correct noise reduction algorithm.[9] Loss of detail must be taken into account when choosing the algorithm. Another issue can be that text on page is not upright but skewed. This also needs to be corrected before recognition. OCR software might require binary image, therefore color or grayscale scanned images must be converted.

2.1.3 Analyzing the layout of the page and segmentation

In this step the page is analyzed in order to separate areas of text from other content of page (pictures, charts, tables, mathematical formulae etc.). Then text parts of the page are segmented. By applying good segmentation techniques, performance of OCR can be increased, see [6]. Segmentation subdivides text into lines then into words and finally into single characters. This might be problematic, when the quality of source file is not very good. For example adjacent characters are not divided by white area but touching each other or one character is not intact but broken down into several pieces.

2.1.4 Feature extraction

Before classification can begin, many OCR programs perform what is known as feature extraction. Feature extraction tries to find and measure certain features in characters. This helps to reduce the data used in classification. There are number methods to provide feature extraction [1] e.g. template matching, zoning, geometric moments, zernike moments, fourier descriptors and more[12]. Some of them could be used on grayscale images, others might require binary images.
2. OCR TECHNOLOGIES AND FORMATS

2.1.5 Classification

In this step each character is classified. OCR software are using several algorithmic approaches. In its most basic form, characters on the image are simply compared to a library of characters. More complicated algorithms use extracted features to determine a best fit. Examples of used approaches: matrix matching, fuzzy logic, structural analysis, neural networks. Depending on OCR technique every character is converted into appropriate character code.

2.1.6 Correction

In the recognition process, situations might occur, where OCR is not sure which character code to produce. In such situations the built-in dictionary can help, which will resolve the correct character. Some OCR software like Finereader have correction utility, which allows software user to choose proper word from a list or type in the word if not in the list.

2.1.7 Creation of output file in chosen file format

Final step in the process is choosing the output format. Chosen format depends on the purpose of OCR and on the content of the document itself. If the document needs to be editable we can use formats like plain text, Rich Text Format, Microsoft Word Format, Open Document, TeX etc. When the purpose of document is to be shared on Internet better choice is PDF format. Advantage of this format is the possibility to have more layers. First can store the original file, under which in second layer will be the recognized text.

2.2 Mathematical OCR

2.2.1 Difference from text OCR

While OCR of text is generally resolved issue, situation with mathematical OCR is different. Basic problem with mathematical OCR is the structure of mathematical formulae, see. Text document written in English have basically simple structure – lines. Every char-
acter in line is on the same vertical level. This makes segmenting into words and characters simple. On the other hand mathematical formulae does not have such clear structure. If we try to divide the formula to lines, we realize it is not always possible to achieve that single character is located in single line. Even if we successfully divide the formula to horizontal lines, result of the recognition will not be reflecting the true meaning of mathematical formula structure but only its visual appearance\cite{4, 5}. Keeping the formulae structure similar to its pattern is important for searching to be viable.

With searching also the choice of output format is associated. Text formats are obviously not very suitable for mathematical documents. Good choice would be XML format using MathML markup or TeX format. These MathML or TeX documents can be then used in PDF as a one layer under original image. More information about MathML and TeX is in section 2.3 Software and formats.

Another difference is the existence of mathematical symbols which are not always the same width and height\cite{4, 5}. As most OCR software is using information about height and width of character, this might cause incorrect recognition.

2.2.2 OCR process on mathematical documents

The OCR process will be described as done by InftyReader. As input, image file or PDF is taken. In order to recognize PDF, it is automatically converted to TIF files. The process flow consists of three main phases: layout analysis, recognition, document structure analysis. After all phases are finished, recognized mathematical document is created and saved in suitable math-description format.

2.2.3 Layout Analysis

At the beginning of the first phase source image is preprocessed (binarization, noise removal). Then all connected components on the image are extracted. After that image is separated into areas of text, tables and figures\cite{11, 4}. Text area in this stage includes math expression and is being segmented into lines. In case text does not in-
2. OCR Technologies and Formats

Include mathematical expression, lines are extracted by searching of periodic local minimum on the horizontal projection histogram of the image. In mathematical documents, lines are built by concatenation of connecting components in a certain neighborhood.

2.2.4 Recognition

This phase consists of 3 steps. First is the initial character recognition and math-text separation, second is the automatic correction of recognition results using clustering and third is the structure analysis of mathematical expressions.

Initial character recognition and math-text separation: Both character recognition and math-text separation are performed simultaneously and cooperatively. Recognition is done by using two complementary recognition engines. One, commercial engine, is used on ordinary texts and other, original, for mathematical formulae. Commercial engine achieves good recognition on ordinary text files, but fails on mathematical expressions. Sometimes it might happen that commercial OCR engine will misrecognize mathematical expression as ordinary word. That is why, after initial recognition, parts separated as ordinary text are verified and mistakes in separation are selected as mathematical expressions. Then original recognition engine is used on mathematical expressions. Recognition results are candidates from which the most probable will be chosen by the structure analysis.

Automatic correction of recognition results using clustering: Clustering is used in order to lower the misrecognized characters either in ordinary texts or in mathematical expressions. With this method, characters misrecognized because of slight differences in shape, will be correctly recognized. The characters are divided into clusters according to their shapes. Every character in a cluster will be recognized as the same character. However this method can cause new misrecognitions if clustering is not "mild".
2. OCR Technologies and Formats

Structure analysis of mathematical expressions: This step has three roles [11]. First is to create tree representation of mathematical expressions, which is needed for converting the mathematical document into math-description formats (XML, LaTeX...). Second is to fix character recognition on mathematical expressions. Third is to detect ordinary texts misrecognized as mathematical expressions. The structure analysis is done by search of minimum-cost spanning tree\(^4\) on the weighted digraph. Each character candidate selected by original recognition engine represents a node in the digraph. By finding the minimum-cost spanning tree in the graph, first two roles are fulfilled. If tree is not found, then the third role is fulfilled as well, because in this case, the mathematical expression is in fact ordinary text.

2.2.5 Documents Structure Analysis

Document Structure Analysis is the final phase in recognition process. Purpose of this phase is to maintain original document structure in recognized documents [10]. It is done by detection of titles, authors, sections itemization, references, theorems, lemmas etc.

2.3 Software and formats

In this section the most important software and file formats used in this work are described.

2.3.1 InftyReader

InftyReader is OCR software used to recognize scientific documents including mathematical formulae and it is probably the only one known commercial OCR software with this capabilities. It was developed by R&D organization called InftyProject consisting of researchers from the different universities and research institutes. InftyReader was chosen as main software used for OCR in this work. InftyReader comes in three versions. First version for normal license, second for 1-year license and third is EuDML version.

\(^4\) a graph with the minimal sum of the cost of all edges of a tree
InftyReader require binary images, grayscale can cause lower quality of recognition. Images should have at least 400 DPI. InftyReader can erase small noise, but is recommended to denoise images before processing. InftyReader works only on Windows 7, Vista or XP and has both UI version and command-line version. Supported input formats are TIF (TIFF), BMP, GIF, PNG, PDF. Possible output formats are IML, LaTeX, HR-TeX, XHTML (MathML), Microsoft Word 2007 (XML).

2.3.2 LaTeX

LaTeX is document preparation system and document markup language used to create high-quality typeset documents. It is mainly used to publish scientific documents, because LaTeX automatizes most aspects of typesetting and desktop publishing.

LaTeX contains features for: typesetting journal articles, technical reports, books etc., control over large documents containing sectioning, cross-references, tables, figures, typesetting of mathematical formulae, automatic generation of bibliographies, multi-lingual typesetting and more...

In this work LaTeX format is used as one of the output formats for recognized pages by InftyReader. Mainly because of mathematics representation and possibility to easily convert LaTeX documents into PDF. PDF can be then compared with source documents and recognition accuracy can be evaluated. More information about LaTeX can be found at: [http://www.latex-project.org/](http://www.latex-project.org/)

2.3.3 MathML

MathML is an XML-based encoding for describing mathematical expressions developed by W3C. It was created to provide better way of describing mathematical equations in Web pages and became standard format for math in XML.

Two separate vocabularies are used to describe mathematical equa-
2. OCR Technologies and Formats

MathML is used in many areas such as science, technical or medical publishing, e-learning, or accessibility. But the reason for choosing it as the output format for the whole OCR process in this work is searching, especially because MathML has the possibility of mathematical keyword searching. More information at: [http://www.w3.org/Math/](http://www.w3.org/Math/)

2.3.4 Tralics

Tralics is software we are using as a connection between LaTeX and MathML. The purpose of this software is to convert LaTeX documents into XML documents. Conversion is based on three objectives, which were behind the creation of this software: easily analyzable result, preserved structure, and univocal translation.

Tralics does not have GUI version, it can only be executed in command line. If the source LaTeX document has mistakes in syntax in it, errors might occur during conversion. Error log files are created, but despite the errors, XML file is always produced. Additional information about Tralics can be found at: [http://www-sop.inria.fr/marelle/tralics/](http://www-sop.inria.fr/marelle/tralics/)

5. having only one meaning
Chapter 3

OCR workflow for EuDML

3.1 DML-CZ

DML-CZ is finished project to create Czech digital mathematic library. Data collected in the library represents around 300000 pages of most important scientific and professional works published within Czech Republic. These works were scanned and digitalized and now are presented in digital library available at [dml.cz](http://dml.cz). Digitalization process involved image preprocessing (thresholding, noise removal, deskewing) and text recognition using FineReader and InftyReader software. The double layer PDF of articles were created containing both image layer and recognized text layer. Also system for newly published works was created, allowing to automatically add them to DML-CZ.

3.2 EuDML project

EuDML (The European digital mathematics library) is a project, partially funded by EC, which aims to design and build a digital library service. Project started on 1. February 2010 and will end in 31. January 2013. The digital library will contain large quantity of mathematical literature from partner countries of this project. Today is EuDML formed by 15 partners from 9 European countries. Since the Czech Republic is one of the partner countries, their DML-CZ will be incorporated in EuDML.

The digital library goals are to preserve the whole digital mathemat-
3. OCR workflow for EuDML

The EuDML partners developed a workflow, which uses a toolset for image and text processing and metadata enhancements. This toolset consists of five subsystems, which are made from smaller software called eutools. The five subsystems are OCR, extraction, conversion, analysis and refinement. The OCR subsystem is made of eutools which handles the character recognition. The extraction subsystem take care of PDF documents. It either extracts the MathML from PDF if possible or indicates that PDF needs to be recognized. The conversion subsystem uses eutools to conversion between LaTeX and MathML.

1. data about data
during math search. The analysis subsystem are used to extract bibliographic references from the text and then parse them. The last subsystem refinement handles finding and presenting the metadata for publications, other than that refinement subsystem uses eutools to optimize the PDF documents. [8]

3.3 Setup and context

The project’s most important part in the terms of this work is the math searching. To provide this feature, the digitalization of the content in this digital library will use such a procedure, where both text and math parts of literature will be recognized and indexed.

Searchable text could be achieved by various OCR software but searchable math, as previously mentioned, is not trivial and needs special software. Ideally one that can recognize both text and math at the same time. The situation in math recognition area does not really give a chance to select a software, as the only available OCR software which can handle OCR of math is InftyReader. For this reason, InftyReader was tested in trial version and later 1 year license was bought and is now used to perform OCR task.

3.3.1 Access

For purposes of OCR was created a virtual machine using Windows 7 environment, which is necessary to use InftyReader. The machine was able to be accessed via Remote Desktop at infty.ics.muni.cz. InftyReader was then installed as well as Tralics software. Other important software installed was TeX distribution TeXWorks which was used for checking LaTeX files.

3.3.2 The Data

Although the DML-CZ data are already recognized, for EuDML purposes they need to be recognized again, for the math search feature to be possible. Data includes 13 journals, 6 proceedings and 4 monographs in multiple languages (Czech, Slovak, English, German, Russian, French and more). Total amount of data to be recognized is
26213 articles. These articles contain almost 250000 pages. The pages arrive both in PDF and TIF formats. Most of TIFs are 600 DPI or above with 1-bit depth, but large quantity of them are grayscale. Since binary images are recommended to use with InftyReader, images in grayscale might create worse quality output than binary.

The DML-CZ data were copied on the machine from DSpace repository where they are stored. At first only small amount of data for recognition testing were accessed, later whole journals were copied to be batch-processed. The data are stored in directory structure, which holds information about the collection, volume and article. Example: AplMat\01-1956-5\tiff\article-3

### 3.3.3 Using InftyReader

First InftyReader tests were made on PDF files, which were later replaced by TIFs, because they had higher quality and gave better OCR results. The GUI version was used at first, on the figure 3.2 is main window of the software. The OCR is simply done by selecting either one file or file folder, choosing input file format and output file format. The GUI allows to select only one output format. Other than that there is image quality selection (400,600), language selection (English, Japanese), math level selection and newline code selection (either at the end of each line or paragraph). If LaTeX output is selected upon clicking Start OCR another windows is shown, where LaTeX output can be modified. On this window LaTeX preamble can be adjusted, which will be put on the beginning of the LaTeX output document. Default LaTeX package settings in preamble should cover all output InftyReader can produce.
3. OCR WORKFLOW FOR EU DML

Although, there are only English and Japanese in the language selection, OCR results on other languages are good as well. There is an option to use FineReader plugin, which can help with text recognition and supports more languages. But first tests showed that PDFs with Czech content are recognized reasonably well enough, so it was not necessary to buy FineReader plugin. The only tested language with not readable OCR results is Russian, due to Cyrillic characters.

During the process of recognition (described in section 2.2.4), the progress.dat file is created in the folder where recognized files are located. This file stores data collected from recognized files. Moreover, log.txt is created, containing information about the processed file and recognition phases. At the end of the recognition process the KML

3. special XML format produced by InftyReader, which stores detailed information about the recognition

---

Figure 3.2: InftyReader - Main Window
file is created from progress.dat. This file is then converted into IML file. Which is then used to create the output on the basis of selection. Regardless of chosen output format, the IML file will remain in the folder. In addition, the folder named * _images might be created. This folder will contain .eps and .png image files. They are created, if the recognized document contains images, graphs, tables or any other parts of image which are considered by InftyReader to be neither text or math formula. These images containing non-recognizable parts of document are then used in output. The output file names are similar to image files if single file is processed. If PDF is chosen as input format, before the recognition every page is converted to image file. Images extracted from PDF are then used for recognition.

Recognition of folder with multiple images will create one common output for whole folder named after the folder name. There is an exception, when processing multiple PDF files in one folder, then each PDF has its own output named after PDF.

3.3.4 Infty.exe

Another way to run InftyReader is using the command line version Infty.exe. Running Infty example.tif in Windows command prompt will process the file example.tif and create output named example.iml (iml is default output format). If Infty.exe will be run on folder, it will process all image files (PDF can not be used as input with Infty.exe) in folder and create output file named after the folder.

To change the behavior of OCR, Infty.exe uses couple parameters:
-f which will specify output formats (iml, kml, tex, html, mathml, hrtex)
-o to specify output folder name
-n to specify output file name
-r (400, 600 - specify image quality)
-preamble/-nopreamble (specify latex preamble file)
-l (language)
3. OCR workflow for EuDML

-s (high speed mode, without commercial OCR software).

3.3.5 Using Tralics

To convert LaTeX output into XML with MathML the Tralics converter was used. Tralics has to be run from command line, where many options to adjust converting are offered. To successfully convert a .tex file created via Infty into .xhtml the Tralics needs to be run with correct configuration. The default configuration that was used in the beginning did not cover all the macros generated by Infty. This led to error messages during conversion. The EuDML uses their own configuration, but even with this configuration, were found correct LaTeX codes, which could not be recognized. The tralics with EuDML configuration was running using:

```
tralics.exe -verbose -noentnames -utf8 -oe8
-type=eudml -leftquote=2018 -rightquote=2019
-usequotes -notrivialmath -confdir="C:/tralics/
eudml-v1.0"
```

3.3.6 Possible workflows

From the beginning the preferred way to create indexable form of output was using InftyReader’s LaTeX output, then using program Tralics to convert LaTeX document to XML+MathML. Reason behind this, is that every MathML output in EuDML was created via Tralics and this output was then tested for indexing math formulae. Later InftyReader’s XHTML+MathML output was tested and became viable option. Since both LaTeX and XHTML outputs are simply created as conversion from one common IML file, there should be no differences in recognition results. The difference is in the production of correct output. Both IML to LaTeX and IML to XHTML conversion can and sometimes will create incorrect output. These incorrect output files then cause errors when being converted or validated.
3.4 Chosen method

After initial OCR test and checking, it was found that both outputs contain errors (see 3.4.2). To decide which method to choose, it was necessary to find out what kind of errors InftyReader can produce and how can they be fixed. In order to find these errors, a larger amount of images had to be processed and error-checked. For this purposes was designed a method for batch processing images.

3.4.1 Batch processing

The GUI version of IntyReader provides some degree of processing of larger amounts of data, if the folder is used as output, but it lacks the possibility to create output in multiple formats. Given that the GUI version was abandoned and rather command line version Infty.exe were used. The Infty.exe was run with these parameters:

`Infty.exe <folder> -r 600 -f iml -f kml -f tex -f mathml -f hrtex -preamble "C:\Program Files(x86)\sAccessNet\InftyReader\bin\preamble.tex"`

With these settings Infty will perform OCR on given folder and create output in 5 formats. To not have to run Infty on every folder manually, 2 cmd scripts were created (see Appendix A). First script infty-all.cmd uses for loop, which will recursively look for subfolders with name using prefix "clanek-". When matching folder is found second cmd script infty-one-folder.cmd is called with parameter storing folder path.

Second script will check if folder path received from first script contains any TIFs. If TIFs are found, then script checks if folder does not already have output LaTeX file. In this case recognition will not be run and script will finish. Eventually if both conditions are unmet, Infty starts and process TIFs in the folder. Script also creates log file to track what is being recognized.

Because of the amount of data needed to be processed the speed of OCR started to be issue, as the single image takes almost 9 seconds
3. OCR workflow for EuDML

to process. To perform OCR faster, the virtual machine was adjusted so it allowed to use 4 threads, thus giving opportunity to run 4 Infty processes at the same time. However the speed of single Infty process became a bit slower.

To batch process tralics on LaTeX output, another simple batch script were created (see Appendix A). Script uses for loop to process every LaTeX file located inside of the given directory tree. The tralics command is followed by general options, which are used with EuDML tralics configuration.

3.4.2 InftyReader errors

To validate LaTeX output was used program pdfLatex (part of TeX-Works TeX distribution), which was used to review the OCR results. pdfLatex will take a LaTeX document and convert it to PDF document. During conversion errors are signaled and written in log file created by pdfLaTeX. Another program used to find errors in LaTeX output was Tralics, which would also signal errors and create log file. However errors signaled by Tralics were not always caused by InftyReader, but rather by wrong configuration of Tralics. To validate XHTML was used online validator at: http://validator.w3.org/. Below are the errors, divided into three categories.

**The InftyReader errors signaled by pdfLatex are:**

! Text line contains an invalid character.
– in some cases Infty produces an invalid character. It was found that these character are control characters.

! Double superscript or ! Double subscript.
– this error is related to not correct notation of superscripts and subscripts in math formulae.

! Misplaced alignment tab character &.
– ampersand character has special meaning in LaTeX, error suggests that & is incorrectly used.

! Undefined control sequence (\Cyrillic_be related)
– LaTeX does not recognize this macro. This error is mostly followed by another errors.
3. OCR workflow for EuDML

! Missing $ inserted
– or similar error message indicating that problem is with LaTeX math mode. Error in math mode often causes follow-up errors.

THE INFTYREADER ERRORS SIGNALLED BY TRALICS:

Undefined command \urcorner and other similar errors.
– special LaTeX characters \urcorner, \lrcorner, \ulcorner, \llcorner from amsmath package are correct, but Tralics will not recognize them, this is due to wrong configuration files.
Double superscript
– see pdfLaTeX errors.
Missing dollar inserted
– see pdfLaTeX errors.
Undefined command \Cyrillic.
– see pdfLaTeX errors.
Error in accent, command = \"
– issue are characters with umlaut accent. For example to produce character ü, InftyReader used code "\"{u} while to correct notation should be \"{u}

THE INFTYREADER ERRORS SIGNALLED BY XHTML ONLINE VALIDATOR ARE:

Non SGML character number X error (where X is some number)
– this is XHTML equivalent of invalid character error from LaTeX.
&Prim; and &tprim; entities in XHTML
– non-defined entities &Prim; and &tprim; can be found in XHTML, correct entities should be &Prime; and &tprime;.
&rcheck; &obar; and similar entities error
– characters with caron and macron have incorrect XHTML codes, for example incorrect forms &rcheck; &obar; where correct forms should be &rcaron; and &omacron;.
element "u" undefined
– related to <u \> XHTML tag usage.
Required attribute "src" not specified and required attribute "alt" not specified errors
– related to images produced by InftyReader during recognition, 2
errors are given for every image created.

Figure 3.3: ConvertTable.tbl

<table>
<thead>
<tr>
<th>Soubor</th>
<th>Úpravy</th>
<th>Format</th>
<th>Zobrazení</th>
<th>Nápověda</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01CC</td>
<td>{1}</td>
<td>&lt;mi&gt;ì&lt;/mi&gt;</td>
<td>Igrave</td>
<td>0x00CC</td>
</tr>
<tr>
<td>0x01CD</td>
<td>{2}</td>
<td>&lt;mi&gt;´&lt;/mi&gt;</td>
<td>Iacute</td>
<td>0x00CD</td>
</tr>
<tr>
<td>0x01CE</td>
<td>{3}</td>
<td>&lt;mi&gt;ĉ&lt;/mi&gt;</td>
<td>Inhat</td>
<td>0x00CE</td>
</tr>
<tr>
<td>0x01CF</td>
<td>{4}</td>
<td>&lt;mi&gt;¨&lt;/mi&gt;</td>
<td>Unlaut</td>
<td>0x00CF</td>
</tr>
<tr>
<td>0x01D1</td>
<td>{5}</td>
<td>&lt;mi&gt;ñ&lt;/mi&gt;</td>
<td>Ntildes</td>
<td>0x00D1</td>
</tr>
<tr>
<td>0x01D2</td>
<td>{6}</td>
<td>&lt;mi&gt;ò&lt;/mi&gt;</td>
<td>Ograve</td>
<td>0x00D2</td>
</tr>
<tr>
<td>0x01D3</td>
<td>{7}</td>
<td>&lt;mi&gt;´&lt;/mi&gt;</td>
<td>Acute</td>
<td>0x00D3</td>
</tr>
<tr>
<td>0x01D4</td>
<td>{8}</td>
<td>&lt;mi&gt;ĉ&lt;/mi&gt;</td>
<td>Ocirc</td>
<td>0x00D4</td>
</tr>
<tr>
<td>0x01D5</td>
<td>{9}</td>
<td>&lt;mi&gt;ñ&lt;/mi&gt;</td>
<td>Ontilde</td>
<td>0x00D5</td>
</tr>
<tr>
<td>0x01D6</td>
<td>{10}</td>
<td>&lt;mi&gt;¨&lt;/mi&gt;</td>
<td>Umlaut</td>
<td>0x00D6</td>
</tr>
<tr>
<td>0x01D9</td>
<td>{11}</td>
<td>&lt;mi&gt;ø&lt;/mi&gt;</td>
<td>Oslash</td>
<td>0x00D8</td>
</tr>
<tr>
<td>0x01DA</td>
<td>{12}</td>
<td>&lt;mi&gt;û&lt;/mi&gt;</td>
<td>Ucirc</td>
<td>0x00DA</td>
</tr>
<tr>
<td>0x01DB</td>
<td>{13}</td>
<td>&lt;mi&gt;ñ&lt;/mi&gt;</td>
<td>Untilde</td>
<td>0x00DB</td>
</tr>
<tr>
<td>0x01DC</td>
<td>{14}</td>
<td>&lt;mi&gt;¨&lt;/mi&gt;</td>
<td>Umlaut</td>
<td>0x00DC</td>
</tr>
<tr>
<td>0x01DD</td>
<td>{15}</td>
<td>&lt;mi&gt;ˆ&lt;/mi&gt;</td>
<td>Yacute</td>
<td>0x00DD</td>
</tr>
</tbody>
</table>

Some of these errors were managed to be fixed by changing values in ConvertTable.tbl, which is one of the configuration files InftyReader uses. This table contains columns (see figure 3.3) of values. The first column is OCR, which is character coding InftyReader use during OCR. The important ones are second and third columns, because they represent LaTeX and MathML entities. Values in those columns are used in conversion to corresponding format.

On the table 3.4 are LaTeX and XHTML errors which were able to be fixed by changing the wrong values in ConvertTable.tbl. Incorrect column contains original LaTeX and XHTML values whereas column labeled Correct contains the corrected values, which are now being used. Although it seems that \texttt{&Prim;} and \texttt{&tprim;} errors could be fixed this way as well, ConvertTable.tbl already contains correct values \texttt{&Prime;} and \texttt{&tprime;}.
Solution to fix \texttt{Cyrillic\_be} error was also found, it involved changing the preamble of the \LaTeX{} output and then making changes in ConvertTable.tbl. The new syntax for \texttt{Cyrillic\_be} would look like \texttt{\textbackslash cyrbe}. However this notation would cause difficulties with the subsequent processing.

For output files containing errors which could not be fixed, reports were written and together with source documents which they were created from, were then sent via e-mail to prof. Masakazu Suzuki, who is one of the InftyReader creators. Later when an updated version of InftyReader was released, the files giving error output were recognized again to see if error was fixed.

### 3.4.3 Other issues

The errors in output files were not the only issue, when processing files with InftyReader. In some cases in the OCR process, part of the text, the math formulae or even the whole page is incorrectly evaluated as image. This happens mainly on the TIFs, that are not binary
and this issue became much less occurring after 2.9.4 version of InftyReader. Binarization was tried to reduce the occurrence of this issue. (see section 4.2 Error testing)

Next issue is that Infty.exe is missing the newline code selection and inserts newline after every sentence, this might affect the further processing of output.

One of the other problems found is Infty not creating output files and ending with error code. This is caused by certain images. If folder is being processed and Infty encounters this type of image, this results in no output creation as well. To get the error codes, the batch script was adjusted. Searching in 22211 articles (213940 TIFs) turned out that 76 articles were without output. Approximately in 0,34% (or in 1 of 292) articles Infty failed to create output. Log files of this articles revealed that this error is not bound to any specific recognition phase. From 76 articles, 25 stopped in preprocessing phase, 12 in recognition 2 phase, 12 in verifying, 6 in structure analysis, 2 in recognition 3 phase, 1 at saving. In all these cases Infty ended with error code 95. Then there were 14 articles, which logs contain Infty OCR Dictionary Initialize Error! and error code 1. Last 4 articles do not even include log files with Infty ending with error code 0 on them. The images causing this problem, were withdrawn from the articles and articles were then recognized again.

Last problem is related to batch processing. When data is being processed, after some random time the batch tends to stop processing. During the stop, batch window will accept input. Pressing random key will make the batch to continue. This turned out to be a problem in later phase of the work when the large amounts of data were recognized and Infty was running in 4 parallel threads to speed up the recognition. This error leads to other changes in the batch script (see Appendix A).

Idea was to create script that will automatically send key presses to batch window, which will restore the recognition. To create such a script, a freeware program named AutoIt was used. This program uses scripting language that is designed to automatize Windows GUI.
3. OCR workflow for EuDML

It can simulate key strokes, mouse movement, window manipulation etc. (More information about this software at: [http://www.autoitscript.com/site/autoit/](http://www.autoitscript.com/site/autoit/))

The script has five lines (see Appendix B). First will take parameter (in this case the batch window title), which it is started with and save it into variable sTitle. The WinWait function will wait until batch window exist and returns the handle of the window. The handle is saved in the hHandle variable. The third line uses while loop, which uses function WinWaitClose. This function waits until the batch window with given handle does not exist, then returns 1. If six hundred seconds pass and batch window did not end, WinWaitClose returns 0. Zero returned means, that while condition is true and ControlSend function is called. ControlSend will send enter key stroke, to the batch window. After that the while loop starts again.

AutoIt program allows to transform the script into executable program. This option was used and run.exe program was created. This program is now started in the infty-all.cmd script before the second infty-one-folder.cmd is called.

All issues mentioned in this section were reported to the creators of InftyReader and possibly will be fixed in the future versions.

3.4.4 Post processing

Some errors from section 3.4.2 were not yet resolved although being reported. To achieve error free output, some fixes were made. These fixes will resolve some errors in LaTeX output and all errors encountered with XHTML output. Fixes use two modified unix sed scripts made by Thierry Bouche, who is member of French EuDML partner team. One script corrects the invalid character errors in both XHTML and LaTeX output, the second fixes other XHTML errors. The sed fixes (see Appendix B) are started with xhtml_repair.sh script, that provide looping through all XHTML documents. On the XHTML output are then used kill-bad-utf.sed and sanitize-HTML.sed scripts, which basically modify the output files by either deleting the causing string or replacing with correct one.
3. OCR WORKFLOW FOR EuDML

3.4.5 Final decision

Since the OCR results are same for both LaTeX+Tralics and XHTML methods, the decision did not depend on OCR quality. The main reasons for choosing the method were errors related to the method. The LaTeX errors were harder or not at all solvable. The dollar sign math mode related LaTeX problem were the one of the reasons, the difficulties with Tralics configuration was the other. On the other hand, the post processing made possible to produce error free XHTML output. Because of this facts the XHTML method was chosen for now. It is possible that this will be reconsidered in the future, if the LaTeX output will be fixed.

Thus, the workflow used for OCR consist of using the updated Infty-all and Infty-one-folder script with the run.exe. On the MathML output are then ran Unix sed fixes using the XHTML_repair.sh script.
Chapter 4

Testing the workflow

4.1 Evaluation methods

To evaluate the chosen workflow, 2 tests were used. First, which will test the number of errors in XHTML output in the sample of 50 articles and second to test the OCR quality in XHTML output by counting the incorrectly recognized characters.

To perform the tests, the set of 50 articles were chosen from the DML-CZ data and OCR were performed on them. The articles were chosen in order to cover multiple languages and math levels as well as different image quality. For the mistakes in OCR it was not possible to count every error on that many pages. Therefore 10 pages were chosen, again containing text in multiple languages, multiple levels of math and image quality.

On the table 4.1 is the distribution of TIFs used in evaluation according to the color depth and the DPI. Most of the images have 1 bit color depth. Totally, there are 32 articles, which contain binary TIFs. The remaining 18 articles are grayscale with 8-bit color depth. TIFs also have various DPI, but mostly over 600.
4. TESTING THE WORKFLOW

Figure 4.1: Articles.tbl

<table>
<thead>
<tr>
<th>Count</th>
<th>Depth</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPI</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>399 PixelsPerInch</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>402 PixelsPerInch</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>424 PixelsPerInch</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>600 PixelsPerInch</td>
<td>187</td>
<td>63</td>
</tr>
<tr>
<td>644 PixelsPerInch</td>
<td>84</td>
<td>65</td>
</tr>
<tr>
<td>Total</td>
<td>271</td>
<td>181</td>
</tr>
</tbody>
</table>

4.2 Error testing

The errors in XHTML were tested on the full 50 articles sample. The articles were batch processed using latest batch script. During the recognition no problems occurred and output was created for every article. XHTML output was then validated by W3C online validator.

XHTML output of all tested files contained 306 errors in total, giving average of 6.12 errors per articles. The articles containing non-binary images (3.33 errors per article) or lower DPI images (5 errors per article) did not produce more errors then the binary ones. Approximately half of the errors (153) were caused by &Prime; code instead of ′. Second big group were errors caused by images created during recognition (118). Next error was &tprim; code instead of &tprime; (26). Then the element "u" undefined error (6) and at Non SGML character errors (3). This means at least one error, from every error type listed in the error section (3.4.2), which were not fixed in ConvertTable.tbl was produced. Seven articles contained more than ten errors, in five the reason was caused mostly by the image folder created during recognition. Remaining two articles were from one journal and had 63 and 72 errors. Both of the article had 600 DPI TIFs with 1 bit color depth and the higher number of errors was caused by &Prim; error.

To fix all errors found in the output, the Unix sed fixes were used. The output XHTML files were then inspected again to find if every error was fixed. No errors were found.
In addition the image folder creation was observed. The image folder were created in 21 articles and in 12 of them image contained images of unrecognized text or math formulae. Seven of these articles had 8-bit color depth and five were binary. Some of the problems in binary ones are almost certainly caused by the font used on the source images. The eight grayscale articles were further processed to find out if binarizing the images will affect creating of the unrecognized images. For the binarizing was used the software ImageMagick. Two binarizing algorithms were tested. One using `convert.exe -color space gray +dither -colors 2 -normalize` and second using `convert.exe -monochrome`. Both will create the output TIF with 1-bit color depth. The recognition of the binarized document with both algorithms showed improvement, as 5 from 7 articles no longer contained images of unrecognized text or math formulae in the image folder.

### 4.3 OCR quality testing

From the articles used for previous testing 10 pages were chosen. The output xhtml was opened in the Firefox and visually compared with the original. Number of recognition errors was manually counted. The errors found were divided into 4 categories: errors in text, errors in math, errors in accents or diacritic and errors in layout or formatting of text. Errors in text category contains all misrecognized characters in text parts of the images. If the general character is correct but has wrong accent, this is counted in the accent category. Errors in math contains errors in math formulae, including the characters misrecognition in the math formulae. Others less important errors like layout issues, spacing errors, wrong usage of math mode in text or similar problems, which are not related to character classification are included in the layout errors. Those errors was not taken into account upon creating the statistics of successful recognition.
List of compared files:

   Image quality: 600 DPI, 1-bit depth, very good preprocessing, with no artifacts in background and only few spaces in letters or connected letters.
   Content: English text without math symbols.

2. CasPestMat/101-1976-4/clanek-1/0331.tif
   Image quality: 600 DPI, 1-bit depth, very good preprocessing, with few artifacts in background and almost none connected or split letters, using an older almost bold looking font.
   Content: English paper, lot of math formulae with few lines of text.

   Image quality: 424 DPI, 8-bit depth, good preprocessing, contains some split letters
   Content: English text, with some math formulae.

4. AplMat/35-1990-6/clanek-6/0060.tif
   Image quality: 600 DPI, 8-bit depth, very good preprocessing, with few artifacts in background and almost none connected or split letters.
   Content: English paper, lot of math formulae with few lines of text.

5. CasPestMat/078-1953-1/clanek-12/0048.tif
   Image quality: 600 DPI, 1-bit depth, good preprocessing, few artifacts in background, some connected letters, using an older almost bold looking font.
   Content: Czech text without math symbols.

   Image quality: 600 DPI, 1-bit depth, very good preprocessing, with no artifacts in background and only few spaces in letters or connected letters.
   Content: Czech paper, half text, half math.

4. Testing the Workflow

Image quality: 600 DPI, 1-bit depth, very good preprocessing, almost none split or connected letters.
Content: German text, with some math formulae.

8. CasPestMatFys/065-1936-4/clanek-1/0378.tif
Image quality: 600 DPI, 1-bit depth, good preprocessing, with some noise in the bottom of the page, some connected letters, using an older almost bold looking font.
Content: French text, with some math formulae.

Image quality: 600 DPI, 8-bit depth, very good preprocessing, with no artifacts or issues with letters
Content: Slovak paper, lot of math formulae and few lines of text.

Image quality: 402 DPI, 8-bit depth, very good preprocessing, with no artifacts or issues with letters
Content: English paper, lot of math formulae and few lines of text.

The table 4.2 contains image ID, DPI, depth, number of errors in all for 4 categories and total number of errors. The second table 4.3 contains total number of characters on the page, characters in text, characters in math and then percentage of correct recognition in text, math and total. The percentage of success was affected by several factors. Mainly the DPI and/or color depth, the language, the preprocessing, the used font and others. The tables shows that four out of ten tested pages scored over 99%. Two pages out of ten scored under 95%. The sixth image has 42 errors in math, but 37 of them caused by one uncommon repeating character (≠). Without these errors the image has score 97,2%. The tenth image turns to be the worst of all scoring only 89,36%. This is due to combination of low DPI and 8-bit depth.
In order to get better results on the image ten, the binarizing was tested on the image. The first algorithm lowered the number of errors to these values: T:20, M:3, A:0, L:2, giving the 97.22% success rate. The second one lowered the values as well but add more errors in math: T:31, M:22, A:0, L:3 with the success rate of 93.59%. The algorithms were also tested on the image number six and this time the text errors got lowered to 16 but part of one math formula did not appear in the XHTML output. The image file of this formula was not created either. This example of issue will be reported to the developers to get fixed.
Because the Infty also offers possibility to select image quality as option, the 400 DPI setting was tested on the lower DPI images to try if it affects the recognition. However, the results were exactly the same.

The binarization was further tested on 5 additional images from the set of 50. The results are in the table 4.4. Three images had less than 10 errors and scored over 99%. Image 12 had 37 errors, with 97.69% success rate. Binarization rose the rate to 98.36% with both algorithms. The image 15, containing Czech text, was unreadable and it was not possible to count errors. The binarization did not help at first, because the OCR classified the whole page as image. This happened in both algorithms. Only after using the 400 DPI option on Infty managed to solve this issue and recognized output was created. First algorithm scored 95.59% success, second 95.86%. Overall the binarization helped to get better results of OCR on these five pages.

![Testing table 3.tbl](image)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>399</td>
<td>8</td>
<td>9</td>
<td>99.32</td>
<td>6</td>
<td>95.58</td>
</tr>
<tr>
<td>12</td>
<td>600</td>
<td>8</td>
<td>37</td>
<td>97.69</td>
<td>22</td>
<td>98.36</td>
</tr>
<tr>
<td>13</td>
<td>600</td>
<td>8</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>600</td>
<td>8</td>
<td>8</td>
<td>99.43</td>
<td>6</td>
<td>95.63</td>
</tr>
<tr>
<td>15</td>
<td>402</td>
<td>8</td>
<td>n/a</td>
<td>n/a</td>
<td>148</td>
<td>95.66</td>
</tr>
</tbody>
</table>

### 4.4 Method testing evaluation

The error testing results show that the XHTML method might be viable to use in EuDML workflow, because errors in this output format given by InftyReader could be bypassed using the sed fixes. Additionally, testing shows that the lower DPI or higher color depth had no impact on error production. The issue with the creation of images of unrecognized text or math formulae still persist. Binarization did help to remove some cases of this issue on grayscale images. However, it is not just a problem with the images not being binary, because even the good quality input can suffer from this problem.
The quality of output in terms of misrecognition heavily depends on the source it is created from. Factors like the lower DPI in combination with grayscale can cause number of errors in output, but again the binarization helps to reduce the number of errors. The articles using nonstandard font can also cause lot of misrecognitions even on 600 DPI binary files, this could only be improved by enhancing the software. The overall recognition accuracy after binarizing the problematic pages was 98.05%. The testing showed that OCR using InftyReader achieve very good results on both text and math sections, but still offers room for improvement.
Chapter 5

Conclusion and Future work

My task in the work was to get familiar with the OCR of mathematical texts and with the existing solution from master’s thesis of R. Panak and T. Mudrak. I have studied the topic and the important matter in the terms of the work is summarized in the chapter (OCR Technologies and formats).

The core task of the work was to design a workflow for mathematical OCR using the InftyReader and then add another step to create indexable output through LaTeX. Since the creation of recognized files was accompanied with difficulties with errors in the output documents, the most important part of the work became to be choosing between the LaTeX Infty output followed by Tralics conversion to MathML or directly creating MathML output with Infty. As the error-free output was crucial in further processing of the output in the EUDML workflow, from these two possible ways were chosen the direct creation of MathML. Errors occurred using this method managed to be fixed, while the LaTeX output suffered from errors, which were not fixed despite reporting them to the developers.

Last task was to test and evaluate the quality of OCR using the sample set and then perform OCR at the whole DML-CZ data set. The testing and evaluation described in the last chapter as well as recognition of the whole DML-CZ showed, that at the moment the chosen workflow will create output with no errors. However, the additional issues were identified during the work, such as non creation of output, wrongly recognizing text or math formulae as images on page and misrecognition in the output. The first one was reported and could not be avoided as it is internal problem of InftyReader.
5. Conclusion and Future work

The second and third issues offers some degree of solution using the binarization of the input data and as well reporting to creators of InftyReader.

Since the new journals that will be added to the collections in the future, would come already in digital form, this workflow will be used mainly for enhancing the current recognized output. The enhancements may include the binarization, if in the future the decision will be made to binarize the whole DML-CZ collection. The algorithms mentioned in the work are not final, they were only used for the testing purposes and further research in this direction will be needed. Another possibility of improving the output could bring the FineReader plugin for InftyReader, although the early tests on the beginning of the work did not indicate much of a improvement. What could definitely improve the recognition results is further development of newer versions of InftyReader.
Bibliography


[8] Petr Sojka, Krzysztof Wojciechowski, Nicolas Houillon, Michal Růžička, and Radim Hatlapatka. Toolset for Image and Text
5. Conclusion and Future Work


Appendix A

Batch scripts

infty-all.cmd

ECHO OFF
SET PATH=%PATH%;C:\Program Files (x86)\sAccessNet\InftyReader\bin
FOR /R /D %%G in (clanek-*) DO infty-one-folder.cmd %%G

infty-one-folder.cmd

@ECHO OFF
IF NOT EXIST "%1\*.tif" GOTO end
IF EXIST "%1\*.tex" GOTO done
ECHO %1
ECHO %1 » Infty.log
Infty.exe %1 -r 600 -f iml -f kml -f tex -f mathml -f hrtex -preamble "C:\Program Files (x86)\sAccessNet\InftyReader\bin\preamble.tex"
:done
ECHO %1 Already done
ECHO %1 Already done » Infty.log
:end

tralics-all.cmd

@ECHO OFF
set PATH=
FOR /R %G in (*.tex) DO tralics.exe -verbose
        -noentnames -utf8 -oe8-type=eudml -leftquote=2018
        -rightquote=2019 -usequotes -notrivialmath -confdir
        ="C:/tralics/eudml-v1.0" %G

**infty-all.cmd updated:**

@ECHO OFF
SET PATH=%PATH%;C:\Program Files (x86)\sAccessNet\InftyReader\bin
start run.exe %1
FOR /R /D %%G in (clanek-*) DO test_infty-one-folder.cmd %%G %1

**infty-one-folder.cmd updated:**

@ECHO OFF
title %2
IF NOT EXIST "%1\*.tif" GOTO end
IF EXIST "%1\*.tex" GOTO done
ECHO %1
ECHO %1 » Infty.log
Infty.exe %1 -r 600 -f iml -f kml -f tex -f mathml
    -f hrtex -preamble "C:\Program Files (x86)\sAccessNet\InftyReader\bin\preamble.tex"
ECHO ErrorLevel = %ERRORLEVEL%
ECHO ErrorLevel = %ERRORLEVEL% » Infty.log
:end
ECHO %1 Already done
ECHO %1 Already done » Infty.log
$sTitle = $CmdLine[1]
$hHandle = WinWait($sTitle)
While WinWaitClose($hHandle, "", 600) == 0
ControlSend($hHandle, "", "", "ENTER")
&WEnd
Appendix B

Unix sed fixes

XHTML_repair.sh

find . -name "*.xhtml" -exec sed -i -f kill-bad-utf.sed -f sanitize-HTML.sed \;

kill-bad-utf.sed

s,\x0,g
s,\x1,g
s,\x2,g
s,\x3,g
s,\x4,g
s,\x5,g
s,\x6,g
s,\x7,g
s,\x8,g
s,\x9,g
s,\xa,g
s,\xb,g
s,\xc,g
s,\xd,g
s,\xe,g
s,\xf,g
s,\x10,g
s,\x11,g
s,\x12,g
s,\x13,g
s,\x14,g
B. **Unix sed fixes**

```
s, \x15ng
s, \x16ng
s, \x17ng
s, \x18ng
s, \x19ng
s, \x1ang
s, \x1bng
s, \x1cng
s, \x1dng
s, \x1eng
s, \x1fg
```

```
s, \xc2\x80ng
s, \xc2\x81ng
s, \xc2\x82ng
s, \xc2\x83ng
s, \xc2\x84ng
s, \xc2\x85ng
s, \xc2\x86ng
s, \xc2\x87ng
s, \xc2\x88ng
s, \xc2\x89ng
s, \xc2\xa0ng
s, \xc2\x8ang
s, \xc2\x8bng
s, \xc2\x8cng
s, \xc2\x8dng
s, \xc2\x8eng
s, \xc2\x8fng
s, \xc2\x90ng
s, \xc2\x91ng
s, \xc2\x92ng
s, \xc2\x93ng
s, \xc2\x94ng
s, \xc2\x95ng
s, \xc2\x96ng
s, \xc2\x97ng
s, \xc2\x98ng
s, \xc2\x99ng
```
s,\xc2\xa"g
s,\xc2\xb"g
s,\xc2\xc"g
s,\xc2\xd"g
s,\xc2\xe"g
s,\xc2\xef"g

**sanitize-HTML.sed**

s,<u/>\"g
s,<img/>\"g
s,<math xmlns="http://www.w3.org/1998/Math/MathML">
<mi mathvariant="normal">Reject</mi></math>\"g
s,<mi mathvariant="normal">Reject</mi>\"g
s,&Prim;,,\&Prime;,\"g
s,&tprim;,,\&prime;,\"g