Diplomarbeit

Umsetzung von LaTeX-Dateien in RTF-Format

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom–Ingenieurs
unter der Leitung von

Ao. Univ. Prof. Dipl.-Ing. Dr. Karl Riedling

Institut für Industrielle Elektronik und Materialwissenschaften der TU Wien
E366

Eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik

von

Spaller Gottfried
MNr.: 8732088
Zeitling 37
A–4320 Perg

Wien, im Mai 2002 .....

Kurzfassung

Das Ziel dieser Diplomarbeit war es, die Einbindung von LaTeX-Dokumenten in die MS-Word-Umgebung zu ermöglichen. Dies sollte durch die Konvertierung in das von Microsoft definierte Rich Text Format erfolgen, wobei das Ergebnis bearbeitbar bleiben muß.

Nach einer kurzen Einführung in LaTeX und RTF wird der Stand der Technik erfaßt. Der Suche nach allen verfügbaren Konvertern folgt die Festlegung der Bewertungskriterien. Anschließend werden die Konverter untersucht und bewertet. Aus dieser Untersuchung ist jener Konverter, der für diese Anwendung am besten geeignet ist, auszuwählen. Dieser Konverter ist durch Erweiterungen zu verbessern.

Die Verbesserungen orientieren sich an den vom Institut für Industrielle Elektronik und Materialwissenschaften der TU Wien zur Verfügung gestellten Dateien. Diese stellen einen Durchschnitt der zukünftige Aufgaben des Konverters dar.

Nach der Erklärung der Verbesserungen folgt die Anleitung zur Verwendung des Konverters. Als Anhang sind die Testdateien für die Bewertung angeführt.

Abstract

The intention of this thesis is the integration of LaTeX-documents in MS-Word. This should be done by conversions into Rich Text Format, defined by Microsoft whilst the result should remain alterable.

Following a short introduction in LaTeX and RTF the state of the art is described. After researching all available converters the evaluation criteria will be defined. Subsequently the converters are tested and evaluated. From this examination the converter which is most suitable for this application will be identified. This converter has to be improved by further developments.

These improvements follow datasets provided by the Institute of Industrial Electronics and Material Science at the TU Vienna. These datasets constitute an average of future tasks of the converter.

The next step after explaining the improvements is the development of a guideline for the application of the converter. Test data files used for the evaluation can be found in the appendix.

Danksagung

Bedanken möchte ich mich bei Herrn ao. Univ. Prof. Dipl.-Ing. Dr. Karl Riedling, der mir diese Diplomarbeit ermöglicht hat. Ihm habe ich die Bereitstellung der Dateien zu verdanken. Auch für die Anregungen und die Betreuung während der Diplomarbeit möchte ich mich herzlich bedanken.

Bedanken möchte ich mich auch bei meiner Familie. Meine Partnerin Angelika, die mich immer wieder ermutigt und unterstützt hat. Unsere Anna, die mich immer wieder aus den Tiefen der Diplomarbeit ins Leben zurückgebracht hat und meine Eltern, die die Rahmenbedingungen für das Studium geschaffen haben.

Perg, im Mai 2002 Spaller Gottfried

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung und Vorgangsweise

1.1 Aufgabenstellung

Es sollte ein Konverter gefunden und weiterentwickelt werden, der es ermöglicht, die unter LaTeX geschriebenen Dokumente unter MS-Word möglichst gut und ohne Aufwand weiterzubearbeiten.

Das größte Augenmerk lag auf die Konvertierung der Formeln. Idealerweise sollte ein LaTeX-Dokument nach der Bearbeitung einfach in ein Word-Dokument eingebunden werden können. Dieses betrifft zumindest den Text und erst in zweiter Linie die Formatierung.

Die Aufgabe einen Konverter zu entwickeln, der das Dokument so verarbeitet, daß kein „Nachbessern“ mehr notwendig ist, würde den Aufwand ins Unendliche treiben. Aus diesem Grund entstand auch die Einschränkung auf die möglichst gute Formelkonvertierung.

1.2 Vorgangsweise

1.2.1 Vorhandene Konverter suchen

Zu Beginn mußte eine Suche nach den bereits vorhandenen Konvertern gestartet werden. Dabei sollten möglichst alle existierenden Konverter in Betracht gezogen werden. Die Suche beschränkte sich hauptsächlich auf das Internet.

Ein Ankauf von Software wurde aus zwei Gründen nicht in Betracht gezogen. Erstens das Risiko, daß die Software nicht den Anforderungen entspricht. Zweitens die Wahr-

scheinlichkeit, daß der Quellcode nicht erhältlich ist und daher auch nicht ausgebaut werden kann.

1.2.2 Benchmark der vorhandenen Konverter

In diesem Schritt wurden die Konverter mit einigen konkreten Anwendungsbeispielen getestet und anschließend wurden die Ergebnisse bewertet.

Die Kriterien für die Bewertung mußten definiert werden. Es war nicht möglich, die einzelnen Ergebnisse direkt zu übernehmen. Die Kriterien mußten beschreibend gehalten werden, um auch ein vernünftiges Bild zu ermöglichen. Ein Schwerpunkt war, wie in der Aufgabenstellung schon festgehalten, das Konvertieren der Formeln.

Die Anwendungsbeispiele wurden vom Institut für Industrielle Elektronik und Materialwissenschaften zur Verfügung gestellt und sollten möglichst den späteren Anwendungen entsprechen, um den für diesen Zweck am besten geeigneten Konverter auswählen zu können.

1.2.3 Auswahl des zu verbessernden Konverters

Nach der Bewertung der Konverter sollte nun jener gefunden werden, der das größte Entwicklungspotential aufweist. Auf diesen Konverter sollte aufgebaut werden.

Die Auswahl erfolgt hauptsächlich nach dem Kriterium der Konvertierung der Formeln. Anschließend wurde noch versucht, die Ausbaumöglichkeit abzuschätzen.

1.2.4 Kriterien der Verbesserung

Anhand des ausgewählten Konverters sollten nun die zu verbessernden und auszubauenden Funktionen ausgesucht werden. Auch hier wurde wiederum der Schwerpunkt auf die Lesbarkeit und die Möglichkeit der Weiterverarbeitung gelegt.

Diese Funktionen sollten anschließend implementiert und wiederum mit den Anwendungsbeispielen getestet werden.

Kapitel 2

Was ist LaTeX

2.1 Die Entstehung von LaTeX

LaTeX entwickelte sich aus einem Textverarbeitungssystem, genannt TeX.

Dieses Textverarbeitungssystem wurde von Donald Knuth an der Stanford University entwickelt. Knuth schreibt im Vorwort zum TeX Buch [1]: „TeX [is] a new typesetting system intended for the creation of beautiful books - and especially for books that contain a lot of mathematics. By preparing a manuscript in TeX format, you will be telling a computer exactly how the manuscript is to the world's finest printers."(TeX [ist] ein neues System, mit dem Bücher - und vor allem Bücher, die eine Menge Formeln enthalten, so gesetzt werden können, daß sie schön aussehen. Bei Erstellung eines Manuskriptes im TeX-Format wird dem Computer genau mitgeteilt, wie die Seiten des Manuskriptes auszusehen haben, wobei die typographische Qualität der Seiten mit derjenigen der besten Setzer der Welt vergleichbar ist.)

In der Zwischenzeit haben sich viele, vor allem Wissenschafter, diese Sprache angeeignet, und benützen sie auch, um Artikel, Berichte oder Forschungsbeiträge in Buchform zu verfassen. TeX bietet viele Möglichkeiten das Format mit Hilfe einer Vielfalt von Befehlen selbst festzulegen. Vor allem für mathematische Formeln und für Ausdrucke in Buchqualität eignet sich TeX besonders. Außerdem gibt es TeX für viele verschiedene Rechnerplattformen. TeX verhält sich immer gleich, ganz egal welche Plattform man verwendet, oder ob man es am Drucker, Bildschirm oder Fotosatzgerät ausgibt.

Aus TeX entwickelte Leslie Lamport das System LaTeX. Dieses System verarbeitet die TeX-Basisbefehle, sodaß sich der Anwender hauptsächlich auf die Struktur und nicht mehr auf die Details der Formatierung konzentrieren muß. Er kann die Dokumente mit wenigen höheren Befehlen erstellen, und der Designer kann sich um das Layout kümmern.

2.2 Was ist LaTeX

LaTeX ist ein Textsatzsystem, bei dem, im Gegensatz zu anderen Textverarbeitungsprogrammen, den sogenannten WYSIWYG-Programmen (What You See Is What You Get), die Formatierung explizit im Quelltext angegeben wird. Bei den WYSIWYG-Programmen ist die Formatierung implizit festgelegt. Diese Festlegung ist meist binär und ist daher nicht lesbar.

Das LaTeX-Programm erzeugt aus der Quelltextdatei eine DVI-Datei (*Device Independent File*), die wiederum von weiteren Programmen für das jeweilige Ausgabegerät aufbereitet wird. Das Ausgabegerät kann eine professionelle Druckeranlage, der Bildschirm oder ein ganz billiger und einfacher Nadeldrucker sein.

Dieses Textsatzsystem wird auch als generisches Markup bezeichnet und ist von der Art her z. B. mit HTML zu vergleichen. LaTeX ist sehr gut geeignet um, wie oben bereits erwähnt, Berichte, Manuskripte und Bücher für den wissenschaftlichen Bereich zu erstellen.

2.3 Wie verwendet man LaTeX

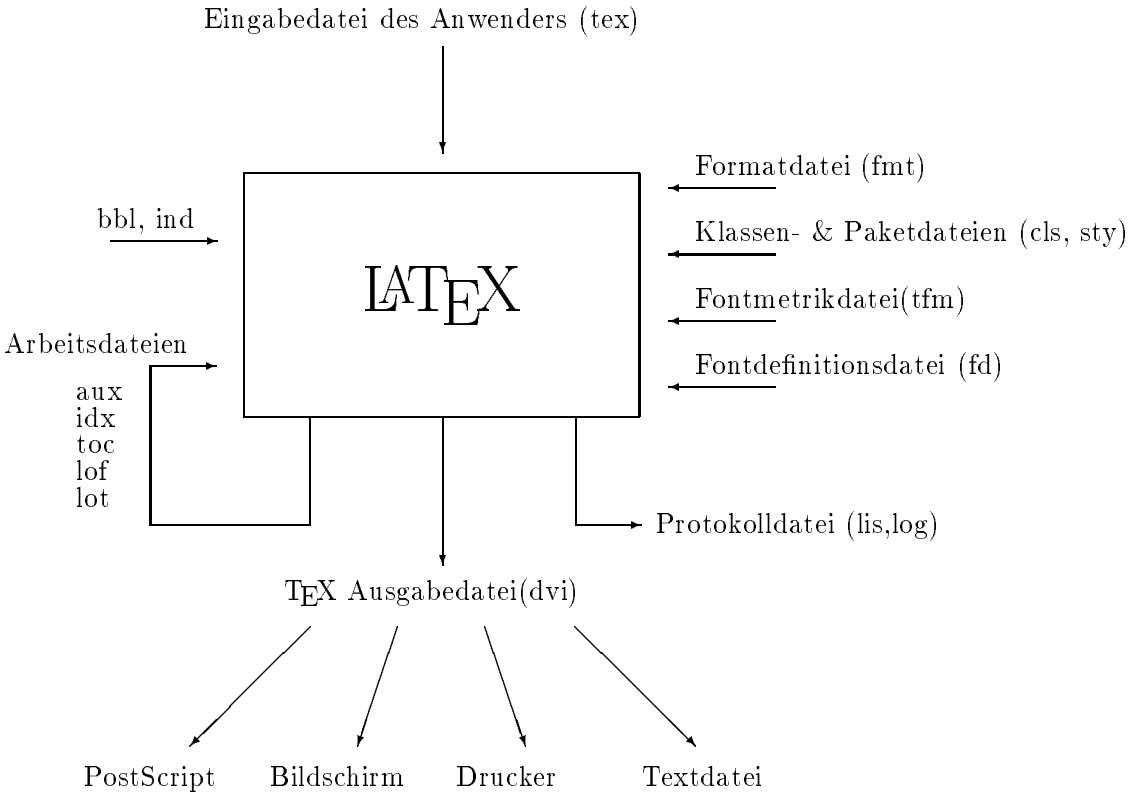
Mit LaTeX kann man mit Hilfe von einigen Zusatzprogrammen auch Indizes, Literaturverzeichnisse, Querverweise und Inhaltsverzeichnisse erzeugen sowie Graphiken einbinden lassen (siehe Abb. 2.1). Einen genaueren Überblick liefert der Artikel „The Components of TeX“ von Joachim Schrod [3]

Das Markanteste an LaTeX ist, daß alles in einem einfachen Editor geschrieben wird, und auf den ersten Blick für einen Ungeübten nicht lesbar erscheint. Alle Befehle und der gesamte Text werden mit der gleichen Schrift erstellt, und man kann eigentlich noch nicht sagen, wie das Dokument nach der Anwendung von LaTeX aussieht. Erst danach tritt das Ergebnis der Formatierung zu Tage.

In LaTeX gibt es Dokumentklassen, die ganz bestimmte Standardformatierungen besitzen. Desweiteren können noch verschiedene Zusatzpakete eingebunden werden (z. B. epic-Paket, german-Paket usw.). Die Klassen und Pakete können mit Optionen auf die Anwendung abgestimmt werden. Außerdem lassen sich die Klassen auch abändern und an spezielle Wünsche anpassen. Auch neue, eigene Klassen können erstellt werden.

Im darauffolgenden Definitionsteil können noch viele spezifische Anpassungen durchgeführt werden. Nach dem Definitionsteil beginnt das eigentliche Dokument. Hier wird der Text mit den notwendigen Formatierungsangaben eingegeben.

Nach der Eingabe des gesamten Dokuments und der weiteren evtl. benötigten Dateien (z. B. Referenzliste als eigene Datei) werden die notwendigen Zusatzprogramme und das LaTeX-Programm gestartet. Dadurch entsteht das formatierte Dokument als dvi-Datei.



Erläuterung	**Dateierweiterung**
LaTeX-Eingabedatei	.tex,.ltx
formatierte TeX-Ausgabedatei	.dvi
TeX-Protokolldatei	.log,.texlog,.lis,.list
METAFONT-Ausgabedatei	.mf
Fontdefinitionsdatei	.fd
Fontdatei	.pk
Fontmetrikdatei	.tfm
String Pool Datei	.pool,.poo,.pol
Formatdatei	.fmt
LaTeX Formatvorlagen- & Erweiterungsdatei	.clo,.cls,.dtx,.sty
LaTeX Hilfsdatei	.aux
Inhaltsverzeichnis	.toc
Abbildungsverzeichnisdatei	.lof
Tafelverzeichnisdatei	.lot
BibTeX-Dateien	.bbl,.bib,.blg,.bst
Index und *MakeIndex*-Dateien	.idx,.ilg,.ind,.ist

Abbildung 2.1: Die wichtigsten TeX- und LaTeX-Dateien [2]

2.4 Zukunft von LaTeX

Im Moment ist LaTeX2ϵ die aktuelle Version. Da jedoch LaTeX in immer neuen Anwendungsgebieten (Rechtstexte, Gedichte, Zeitschriften usw.) Verwendung findet, für die es von Anfang an eigentlich nicht gedacht war, wurden einige Mängel entdeckt.

Daraufhin wurde das Projekt LaTeX3 ins Leben gerufen. Der Sinn dieses Projekts ist es, LaTeX leistungsfähiger zu machen und die notwendigen Ergänzungen der Pakete durchzuführen. Das neue System soll eine vollständig integrierte Benutzeroberfläche für Layoutvorlagen enthalten und damit die Entwicklung und Pflege eigener Layoutvorlagen erleichtern.

Außerdem wurde eine Diskussionsrunde, NTS, für ein neues Schriftsatzsystem, „The New Typesetting System“, gegründet. Diese Runde soll jene Bereiche diskutieren und koordinieren, die am jetztigen TeX verändert und ergänzt werden müssen, um den neuen Ansprüchen verschiedenartiger Anwendungen gerecht zu werden, und sogenannte „masterpieces of the Publishing Art“[5] (Meisterwerke der Satzkunst) erstellen zu können.

2.5 Woher kann man LaTeX bekommen

LaTeX ist frei verfügbar und kann kostenlos aus dem Internet heruntergeladen werden. Es gibt für viele verschiedene Plattformen eine erhältliche Version. Hier einige Bezugsquellen:

http://www.ctan.org

http://www.dante.de

http://www.miktex.org

http://www.esm.psu.edu/mac-tex

Kapitel 3

Was ist RTF

3.1 Was ist RTF

„The Rich Text Format (RTF) Specification is a method of encoding formatted text and graphics for easy trasfer between applications. Currently, users depend on special translation software to move word-processing documents between different MS-DOS, Microsoft Windows, OS/2, Macintosh and Power Macintosh applications.“[7]

Die Spezifikation des Rich Text Format (RTF) ist eine Methode, formatierten Text und Graphiken für einen einfachen Transfer zwischen Applikationen zu kodieren. Nach dieser Kodierung kann der Text (inclusive den Graphiken) für verschiedene Ausgabegeräte, Systemumgebungen und Betriebssysteme verwendet werden. RTF verwendet ANSI- (American National Standards Institute), PC-8-, Macintosh- oder IBM PC-Zeichensätze zur Darstellung und Formatierung der Dokumente. Durch die Verwendung dieser standardisierten Zeichensätze ist der Austausch zwischen verschiedenen Plattformen möglich.

RTF wurde von der Firma Microsoft als allgemeines Format zum Austausch von Dokumenten definiert und findet aus diesem Grund breite Unterstützung in den Microsoft-Produkten. RTF ist das Produkt der Firma Microsoft und stellt bisher keinen weltweiten Standard dar, wodurch RTF großteils an die Software von Microsoft gebunden ist.

Im Gegensatz zu den meisten Produkten von Microsoft gibt es hier offengelegte Syntaxregeln. Daher ist der Quelltext auch nachvollziehbar, weshalb sich dieses Format für die Anbindung an die Microsoft-Produkte anbietet.

In Microsoft-Produkten gehört RTF zum Standard. Daher ist es für andere Anbieter von Textverarbeitungsprogrammen beinahe zwingend, zumindest einen RTF-Reader einzubauen, um den Datenaustausch mit anderen Anwendern garantieren zu können, da die Microsoft-Produkte einen großen Marktanteil besitzen.

3.2 Wie verwendet man RTF

Die Software, die einen formatierten Text in einen RTF-Code umwandelt, nennt man einen RTF-Writer. Der Writer trennt den Text von der Formatierung und schreibt den Text mit der zugehörigen Formatierung in ein RTF-Dokument.

Die Formatierung wird durch den RTF-Writer in sogenannten *Kontrollwörtern* und *Kontrollsymbolen* kodiert. Zusätzlich besteht noch die Möglichkeit, bestimmte Sequenzen durch Klammerung zu Gruppen zusammenzufassen.

Um aus dieser Kodierung wieder ein formatiertes Dokument zu erhalten, ist das Gegenstück des Writers, der sogenannte RTF-Reader, notwendig. Dieser Reader setzt die Formatierung für die jeweilige Applikation aus den *Kontrollwörtern* und *Kontrollsymbolen* wieder zusammen.

Normalerweise sind RTF-Writer und RTF-Reader in den Applikationen vorhanden. Für spezielle Anwendungen kann es aber durchaus möglich sein, angepaßte Writer und Reader zu schreiben.

3.3 Zukunft von RTF

Durch das World Wide Web werden immer mehr Dokumente bereits in HTML oder XML geschrieben. Daher stellt sich die berechtigte Frage, ob es nicht in Zukunft einfacher sein wird, die Dokumente über diese wirklichen Standards auszutauschen. Der große Vorteil ist die Standardisierung. Auch in diesen Formaten kann das Dokument mit genormten Zeichensätzen ausgetauscht werden. Darüber hinaus bleibt es wahrscheinlich keinem Hersteller von Textverarbeitungsprogrammen erspart, die Implemetierung für die Abspeicherung in HTML oder XML in seine Applikation aufzunehmen.

Die Entwicklung von RTF und die weitere Anwendung hängt sehr stark von der Firmenpolitik von Microsoft ab. Es kann daher keine Abschätzung geben, wie sich RTF in Zukunft entwickeln wird.

3.4 Woher kann man RTF bekommen

Einen einfachen RTF-Reader bietet Microsoft an. Dieser kann über die folgende Adresse gesucht (Suchbegriff: „RTF Reader“) und heruntergeladen werden:

http://www.microsoft.com/downloads/search.asp

Die Spezifikationen von Rich Text Format sind unter folgender Adresse zu finden (Aktuelle Version 1.6):

http:/msdn.microsoft.com/library/en-us/dnrtfspec/html/rtfspec_1.asp

Kapitel 4

Stand der Technik, ein Vergleich

4.1 Suchergebnisse

Die Suche im Internet ergab folgende Resultate:

4.1.1 Entwicklung

Die Suche nach Konvertern gestaltete sich durchaus schwierig. Im Internet existieren sehr viele verschiedene Links zu Konvertern. Nach langer Suche und Sortierung kam Folgendes zum Vorschein.

Es gibt zwei verschiedene Tendenzen. Erstens die frei verfügbaren Konverter. Zweitens die käuflich erwerbbaren Konverter.

Bei der ersten Kategorie gibt es zwei Startpunkte der Enwicklung. Den ersten Grundstein zur Entwicklung legte Julian Smart mit seinem Tex2RTF Konverter. Die Version 1.0 ist mit November 1993 datiert. Diese Entwicklunglinie wird von Julian Smart bis heute mit schwankender Intensität betreut.

Den zweiten Grundstein dieser Kategorie legte Ralf Schlatterbeck 1994 mit einer Gruppe von Studenten der Universität Wien. Er begann mit seiner Gruppe, einen Konverter mit der Bezeichnung ltx2rtf zu entwickeln. Der letzte Hinweis auf Ralf Schlatterbecks Arbeit ist die Version 1.3 vom Juli 1994. Im Jahre 1995 nahm eine weitere Gruppe von Studenten die Arbeit am Konverter auf und brachte die Verion 1.5 zum Entstehen.

Auf diesen Ansatz bauten zwei verschiedene Teams die weitere Entwicklung auf, wobei beide Gruppen nichts von der Entwicklung der anderen Gruppe wußten. 1997 begann Daniel Taupin diesen Konverter zu verbessern. Diese Entwicklung reicht bis zum heutigen Tag und ist unter der Bezeichnung ltx2rtf zu finden. Die zweite Weiterentwicklung wurde

von Georg Lehner 1998 in Angriff genommen. Im Jahre 2001 begann Scott Prahl den Konverter zu verbessern.

Die Zusammenstellung und die Entwicklung der Konverter basierend auf dem Grundstein von Ralf Schlatterbeck wird von Winfried Hennings durchgeführt. Hennings versucht, die Neuerungen gesammelt aufzulisten und auch die entsprechenden Links zu aktualisieren.

Die Kategorie der käuflich erwerbbaren Konverter beschränkt sich auf einen Anbieter. Die Firma K-Talk bietet das Programm TexPort an.

Neben diesen Konvertern gibt es noch das Programm TexAide, welches aber nur Formeln konvertiert, und aus diesem Grund nicht in die Betrachtung einbezogen wurde.

4.1.2 Bewertbare Programme

Es gibt latex2rt und tex2rtf für verschiedene Plattformen (Unix, Windows, MS-DOS OS2). Die Entscheidung ergab sich aus der höchsten verfügbaren Version unter den verschiedenen Plattformen.

TexPort für Windows
tex2rtf Version 1.62 für Windows
ltx2rtf Version 3.1 für Unix und MS-DOS
latex2rtf Version 1.9e für Unix und MS-DOS

4.2 Bewertungskriterien

Installation & Anwenderfreundlichkeit: Hier wurde die Einfachheit von Installation und Handhabung der Konverter getestet.

Funktionstüchtigkeit: Dieser Punkt ist das wichtigste Kriterium. Dieses ist sehr weit gestreut. Daher wurde hier in vier Unterpunkte gegliedert. Diese vier Punkte sind Text, Formatierung, Verweise und Formeln.

Text: Der Text sollte möglichst fehlerfrei und vollständig konvertiert werden.

Formatierung: Die Formatierung war von nachrangiger Bedeutung. Es ist jedoch für die Lesbarkeit von Texten wichtig, daß die Formatierung ebenfalls gut konvertiert wird.

Verweise: Mit diesem Punkt wurde die Fähigkeit getestet, mit dem Konverter Bezüge zu verarbeiten, inwieweit diese auch in der RTF-Datei vorhanden sind.

Formeln: Die Konvertierung der Formeln wurde in dieser Arbeit zum zentralen Thema. Die Formeln sollten möglichst richtig dargestellt werden, und auch im Formeleditor von MS-Word weiterbearbeitet werden können.

Stabilität: Unter diesen Begriff fällt die Absturzsicherheit des Konverters. Es geht darum, wie ein Konverter damit umgeht, wenn er Befehle nicht findet, etwas nicht konvertieren kann und vieles mehr. Ideal wäre es, wenn ein Konverter eine Warnung bzw. Meldung ausgibt, falls er etwas nicht konvertieren kann, aber trotzdem den Rest des Dokuments nach besten Möglichkeiten konvertiert.

aktuelle Entwicklung: Dies ist für die Erweiterungen und Verbesserungen von Bedeutung. Die aktuelle Qualität der Konverter hängt stark davon ab, ob an der Entwicklung gearbeitet wird, oder der Konverter im Stillstand verharrt.

Plattform: Die Plattform ist nicht von zentraler Bedeutung, sollte aber vollständigkeitshalber ebenfalls in die Bewertung einfließen, da es für die Anwender durchaus wichtig ist, auf welchen Plattformen die Konverter erhältlich sind. Hier kann es auch vorkommen, daß die Konverter für verschiedene Plattformen in verschiedenen Versionen erhältlich sind.

Gewichtung. Zur Bewertung der Ergebnisse wurde eine Gewichtung eingeführt. Diese Gewichtung legt den Schwerpunkt für die Bewertung fest. Es ist klar, daß eine andere Gewichtung auch das Ergebnis verändert. In dieser Arbeit war die Übersetzung und Weiterbearbeitung der Formeln das wichtigste Kriterium. Diesem Schwerpunkt wurde mit der gewählten Gewichtung Rechnung getragen.

4.3 Die Optionen der Konverter

4.3.1 TexPort

Der von der Firma K-Talk im Internet angebotene Konverter soll laut Beschreibung so gut wie alles können. Leider gibt es davon keine Testversion im Internet. Daher war es auch nicht möglich, diesen Konverter zu testen. Die Möglichkeit, eine kurze Testdatei einzusenden, die dann von dieser Firma konvertiert wird, wurde nicht in Anspruch ge-

nommen, da nicht feststellbar ist, ob das Ergebnis direkt aus dem Konverter kommt, oder ob es nachgebessert wurde.

Auch wenn dieser Konverter die besten Ergebnisse erreicht hätte, wäre eine Verbesserung wohl kaum möglich, da es umöglich ist, an den Quellcode zu kommen, und diesen zu erweitern bzw. zu ergänzen.

4.3.2 Tex2RTF

Dieser Konverter kann unter Windows oder im Textmodus betrieben werden. Der einfachere Fall ist der Betrieb unter Windows, der auch hier verwendet wurde, da sich auch im MS-DOS-Modus nicht mehr Einstellungen treffen lassen können. Um den Konverter zu verwenden, muß die Eingabedatei und die Ausgabedatei festgelegt werden. Es besteht vom Fenster aus die Möglichkeit, zwischen mehreren Ausgabeformaten zu wählen. Im Falle, daß ein selbstgeschriebenes Makro existiert, wird dieses beim Starten des Konverters aufgerufen. Auch Verwendung der Hilfe ist innerhalb des Fensters vorgesehen.

Das Ausgabeformat kann RTF, HTML, Winhelp RTF[1] oder wxWindows[2] sein.

Im Makro gibt es viele Optionen zum Einstellen. Diese sind viel zu zahlreich, um hier erwähnt zu werden. Sie beziehen sich auf

- allgemeine Optionen
- Darstellungsoptionen
- RTF und WinHelp Optionen
- HTML Optionen
- DDE Befehle

Neben diesen im Makro einstellbaren Optionen bietet das Makro auch die Möglichkeit, Befehle zu definieren, die der Konverter ansonsten nicht übersetzt. Dies können z. B. selbst erstellte Befehle sein. Dieses Makro wird jedesmal vor einer neuen Konvertierung aufgerufen.

Wenn das Makro richtig definiert ist, läßt sich dieser Konverter sehr einfach bedienen.

[1] Winhelp ist ein schneller RTF-Browser von Microsoft.

[2] wxWindows ist ein Werkzeug, um Anwendungen zu schreiben, die auf verschiedenen Plattformen anwendbar sind.

4.3.3 ltx2rtf

Mit ltx2rtf liegt ein Konverter vor, der sich unter MS-DOS und Unix betreiben läßt. Als Testplattform wurde MS-DOS verwendet. Der Syntax des Aufrufs ist

ltx2rtf [-V] [-m] [-M] [-D] [-o outputfile] [-C pagecode] [inputfile]

Die Optionen bedeuten:

-V gibt die Versionsnummer zurück

-m verarbeitet Formeln, die mit $$ beginnen und $$ enden, mit LaTeX und wandelt die Ergebnisse in Bilder (Bitmaps) um und fügt diese anstelle der konvertierten Formeln ein

-M gleich wie -m, nur daß auch die Formeln, die mit $ beginnen und $ enden, als Bilder eingefügt werden

-D debugging

-o legt den Namen der Ausgabedatei fest

-C es gibt zwei Optionen, 850 oder ISO, um 850-coded oder ISO-latin1-coded Schriftzeichen lesen zu können

Um die Formel als Bilder einfügen zu können, muß eine funktionierende Version von LaTeX auf der entsprechenden Plattform installiert sein. In der Beschreibung wird auf emtex[3] Bezug genommen. Der Formeleditor von MS-Word kann diese Bilder aber nicht bearbeiten.

Auch dieser Konverter ignoriert die benutzerdefinierten Makros. Es gibt aber für diesen Zweck in einem Unterverzeichnis ein spezielles, in Fortran77 geschriebenes Programm. Mit diesem Programm ist es möglich, die meisten benutzerdefinierten Befehle zu übersetzen bzw. zu extrahieren. Die Syntax für dieses Unterprogramm, das vor dem Aufruf von ltx2rtf gestartet wird, lautet:

exp-macr <source>.tex <translated-source>.anything

Um eine gute Konvertierung zu gewährleisten, muß man die Konfigurationsdateien noch anpassen.

Diese Dateien haben folgenden Zweck:

[3] ist eine TeX-Version für OS/2, PC-DOS und MS-DOS

direct.cfg Hier kann zu einem benutzerdefinierten LaTeX-Befehl seine RTF-Entsprechung eingegeben werden. ltx2rtf schreibt den Inhalt direkt in die RTF-Datei.

fonts.cfg Um die Schriftarten von LaTeX richtig zu konvertieren, weist man ihnen die entsprechenden RTF-Schriftarten zu.

ignore.cfg Benutzerdefinierte Befehle, die man hier einträgt, werden einfach ignoriert. Dies ist wichtig, damit der Konverter alle Befehle abarbeiten kann, und nicht ein unbekannter Befehl zum Absturz führt.

config.cfg Diese Datei muß adapiert werden, da der Konverter sonst nicht weiß, wo er nach den Programmen suchen soll, mit denen er aus den Formeln die Bilder erstellen kann.

Damit der Konverter diese Konfigurationsdateien auch findet, wird die Umgebungsvariable RTFPATH gesetzt. Diese Variable zeigt auf das Verzeichnis, in dem sich die .cfg Dateien befinden.

4.3.4 latex2rtf

latex2rtf kann unter MS-DOS und Unix installiert und verwendet werden. Aufgrund der Verwendung der anderen Konverter unter MS-DOS wurde auch dieses Programm unter MS-DOS getestet.

Schon bei der Syntax des Funktionsaufrufes ist die gemeinsame Wurzel mit ltx2rtf erkennbar. Die Syntax lautet

latex2rtf [-V] [-l] [-v #] [-o outputfile] [-a auxfile] [-b bblfile] [-i idiom] [inputfile]

Die Optionen bedeuten:

-V gibt die Versionsnummer zurück

-l ermöglicht die Verwendung von ISO 8859-1 (latein-1) Schriftzeichen

-v # debugging, hier wird der debugging level eingestellt, je höher, umso mehr Information erhält man. (Bereich: 0 bis 4)

-o hier kann der Name der Ergebnisdatei festgelegt werden

-a bietet die Möglichkeit, eine aux-Datei festzulegen. Diese aux-Datei wird bei der Verwendung von LaTeX erzeugt und dient dazu, die Referenzen unter LaTeX herstellen zu können.

-b die bbl-Datei, die hier einfügt, ist die Referenzdatenbank, die als tex-Datei erstellt wird, und aus der auch LaTeX seine benötigten Referenzen bezieht.

-i dieser Konverter besitzt eine Anzahl von language.cfg-Dateien. Diese enthalten die festgelegte Abschnittsnamen wie z. B. Literaturverzeichnis, Anhang und vieles mehr. Das idiom legt die Sprache der Bezeichnungen fest.

Dieser Konverter ignoriert benutzerdefinierte Befehle. Es gibt auch kein spezielles Programm, das die Einbindung ermöglichen würde. Für eine gute Konvertierung ist es auch hier notwendig, die Konfigurationsdateien gut anzupassen. Die Dateien sind, mit Ausnahme von config.cfg, die selben wie beim oben beschriebenen Konverter ltx2rtf. Auch die Verwendung ist gleich.

Der Unterschied ist, daß es anstatt der Datei config.cfg die Dateien <language>.cfg gibt. Es besteht die Möglichkeit, die vorhandenen Dateien zu korrigieren, oder neue, für den eigenen Gebrauch angepaßte Dateien zu schreiben.

Auch bei diesem Konverter muß der RTFPATH gesetzt werden, damit er die .cfg-Dateien findet.

4.4 Die zum Testen verwendeten Dateien

Zum Testen der Fähigkeiten der Konverter wurden einige Testdateien geschrieben und solche Dateien verwendet, die sich ausschließlich mit einem Gebiet beschäftigen. Diese Testdateien (siehe Anhang) sollten einen Überblick über die Verwendbarkeit der Konverter geben. Dabei wurde versucht, möglichst viele Gebiete abzudecken.

Außer diesen eigens dafür geschriebenen Testdateien wurden noch Dateien vom Institut für Industrielle Elektronik und Materialwissenschaften zur Verfügung gestellt. Diese Dateien waren sehr groß, und keinem dieser Konverter war es möglich, sie richtig zu verarbeiten. Eine Konvertierung konnte nur abschnittsweise durchgeführt werden.

Die Testdateien hatten den Vorteil, daß jede dieser Dateien auf ein Gebiet ausgerichtet war. Damit wurde versucht, die Stärken und Schwächen der einzelnen Konverter festzustellen.

Die Dateien sind den einzelnen Bewertungskriterien zugeordnet.

Text

char.tex Hier wird versucht, so viele Sonderzeichen wie möglich zu testen. Diese Sonderzeichen werden mit einem Befehl und dem zu verändernden bzw. zu ergänzenden Zeichen in LaTeX geschrieben. (siehe Anhang A.4 Seite 84)

fonts.tex Diese Datei testet zwei Schriftarten. Außerdem werden noch die möglichen Modifikationen einer Schrift getestet. (siehe Anhang A.7 Seite 86)

german.tex LaTeX kann das sogenannte german.sty-Paket verwenden. Diese Paket dient zur leichteren Verwendung der Umlaute und von ß. Diese Umlaute und das ß können auch ohne das Paket dargestellt werden, jedoch ist dann die Schreibweise in LaTeX wesentlich komplizierter, daher die Gegenüberstellung. (siehe Anhang A.10 Seite 87)

Formatierung

align.tex Diese Datei dient zum Testen der Ausrichtung des Textes. Normalerweise wird der Text bei LaTeX blockweise formatiert. Um von dieser Formatierung abzuweichen, gibt es geeignete Befehle, deren Konvertierung mit dieser Datei getestet werden. (siehe Anhang A.1 Seite 81)

article.tex Jedes Dokument in LaTeX gehört einer bestimmten Dokumentenklasse an. Jede Dokumentenklasse besitzt gewisse Formatierungen, die die Darstellung von Befehlen definieren (z. B. die Größe der Kapitelüberschriften usw). Zum Testen wurden zwei Dokumentenklassen herangezogen. Die erste dieser eben genannten Dokumentenklassen findet man in dieser Datei und die zweite wird in der Datei report.tex verwendet. (siehe Anhang A.3 Seite 83)

liste.tex Viele Arten von Listen stehen hier zum Testen bereit. Es werden einfache und nummerierte Aufzählungen verwendet, genauso wie Definitionslisten. Ebenso wird die Schachtelungstiefe berücksichtigt. (siehe Anhang A.13 Seite 90)

report.tex Im Anhang ist diese Datei nicht zu finden. Sie entspricht der später noch verwendeten Datei inhalt.tex, jedoch ohne Contents (Inhaltsverzeichnis). (siehe Anhang A.12 Seite 89)

tabular.tex Zum Erstellen von Tabellen gibt es in LaTeX eine eigene Umgebung. Mit dieser Datei wird diese Umgebung getestet. Es werden drei Möglichkeiten untersucht, und zwar als alleinstehende Tabelle, als Tabelle mitten im Text und als zentrierte Tabelle. (siehe Anhang A.15 Seite 94)

Verweise

cite.tex Um Referenzen zu testen, wird diese Datei verwendet. Dabei sind die Referenzen am Ende der Datei angefügt. Es wird auch eine undefinierte Referenz angegeben. Die zweite Variante, auch unter LaTeX verwendet, eine eigene Referenzdatei zu erstellen, steht den Konvertern im Test ebenfalls zur Verfügung. (siehe Anhang A.6 Seite 85)

footnote.tex Um die Fähigkeit der Konverter bezüglich Fußnoten zu testen, wurde diese Datei kreiert. (siehe Anhang A.8 Seite 86)

inhalt.tex Das Inhaltsverzeichnis zu erstellen, gehört unter LaTeX zu den Standardaufgaben. Die Datei report.tex stellt die Gliederung des Dokuments zur Verfügung, wobei aber der zum Erstellen des Inhaltsverzeichnisses notwendige LaTeX-Befehl integriert ist. (siehe Anhang A.12 Seite 89)

Formeln

array.tex Die Fähigkeit Matrizen darzustellen ist der Testansatz dieser Datei. Dabei wird auch die Ausrichtung der einzelnen Spalten verändert. (siehe Anhang A.2 Seite 82)

charmath.tex Hier wird versucht, soviele veränderbare Zeichen wie möglich in mathematischer Umgebung zu testen. (siehe Anhang A.5 Seite 84)

frac.tex Um die Konvertierung von einfachen Formeln zu untersuchen, werden in dieser Datei häufig gebrauchte LaTeX-Befehle verwendet. (siehe Anhang A.9 Seite 87)

hoch.tex Für speziellere Anwendungen in der Mathematik wird unter LaTeX das amsmath-.sty-Paket verwendet. Mit diesem Paket ist die Darstellung komplizierterer Formeln sehr gut möglich. (siehe Anhang A.11 Seite 88)

mathUmg.tex Mit dieser Datei werden die mathematischen Umgebungen auf ihre Konvertierbarkeit getestet. Es gibt verschiedene Möglichkeiten, diese Umgebungen aufzurufen. (siehe Anhang A.14 Seite 93)

zeichen.tex Hier werden sehr viele verschiedene Sonderzeichen im mathematischen Modus getestet. Unter anderem auch die Darstellung der griechischen Buchstaben und die Möglichkeiten von hoch- und tiefgestellten Zeichen. (siehe Anhang A.16 Seite 94)

Institutsdateien: Die Dateien, die vom Institut zur Verfügung gestellt wurden, mußten zuerst bearbeitet werden. Eine Bearbeitung war aus verschiedenen Gründen notwendig. Einige Dateien waren nicht vollständig. Es wurden Pakete verwendet, die nicht Standard sind. Es waren TEX-Befehle enthalten, die LATEX zwar akzeptiert, aber nicht mit LATEX konform sind, und daher auch nicht im Konverter implementiert sind. Es waren auch zusätzliche Zeichen enthalten, die LATEX zwar richtig verarbeitet bzw. unverarbeitet läßt, die aber bei der Verwendung von LATEX Fehlermeldungen bzw. Warnungen ausgeben.

Die Dateien waren teilweise so groß, daß sie zuerst zerlegt werden mußten, um sie anschließend, zumindest stückweise, konvertieren zu können. Die Konverter schaffen zwar auch große Dateien, aber aufgrund der im vorigen Absatz erwähnten Probleme stürzten die Konverter laufend ab. Dies war ein weiterer Grund, die Dateien zu zerteilen, denn damit konnten die Unkorrektheiten beseitigt werden.

Eigene Dokumente und Dokumentteile wurden geschrieben, um diese Dateien überhaupt testen zu können. Außerdem mußten die Definitionen in den Dokumenten zuerst expandiert werden, da keiner der Konverter die Definitionen bzw. Makros verarbeiten konnte.

Nach diesen Korrekturen war es teilweise möglich, die Dateien zu konvertieren.

Numerische Bewertung: Für jede Gruppe wurde eine Bewertungstabelle erstellt. In dieser Bewertungstabelle gibt es für jedes Ergebnis eine Note von eins bis fünf. Das Ergebnis für jede Variante ist die Summe der mit der Gewichtung multiplizierten Noten.

Die einzelnen Summen der Tabellen wurden in eine gewichtete Gesamttabelle eingebracht, um so die Ausrichtung der Bewertung auf den Schwerpunkt zu ermöglichen.

4.5 Bewertungsergebnisse

Die Bewertung erfolgte mit einer Gegenüberstellung der Ergebnisse der einzelnen Testdateien in den jeweiligen Gruppen.

4.5.1 Text

char.tex

Tex2RTF Von diesem Konverter werden nur die gängigeren Zeichen verarbeitet. Ein Teil der Zeichen wird nicht dargestellt. Der Rest wird als Buchstabenkombination

konvertiert. Diese Buchstabenkombination besteht aus dem Buchstaben, der in LaTeX als Befehl verwendet wird und dem Buchstaben, der modifiziert werden soll.

ltx2rtf Das Ergebnis ist eine RTF-Datei, die ebenfalls die Zeichen richtig darstellt, die vom obigen Konverter richtig dargestellt werden. Es werden mehr Zeichen dargestellt, dafür werden aber die Buchstaben, die nicht ausgelassen oder richtig konvertiert werden, nicht modifiziert dargestellt.

latex2rtf Dieser Konverter läßt keine Zeichen weg. Richtig modifiziert werden aber ebenfalls nur die zehn Zeichen, die auch die anderen Kandidaten richtig konvertieren konnten. Allen Zeichen, die nicht richtig konvertiert wurden, folgt nach einem Beistrich in den meisten Fällen die richtige Modifikation.

Von der Anzahl der richtig konvertieren Zeichen liefern alle das gleiche Ergebnis, aber latex2rtf läßt kein Zeichen aus. Daher ist bei der Nachbearbeitung der Konvertierung davon auszugehen, daß die nicht richtig verarbeiteten Zeichen erkennbar und korrigierbar sind. Die beiden anderen Kandidaten lassen Zeichen aus, die das Lesen einer verarbeiteten Datei sicherlich erschweren. Wenn latex2rtf noch lernt, die Modifikation nicht hinter, sondern über das Zeichen zu stellen, dann wäre die Ausbeute sehr groß.

fonts.tex

Tex2RTF Bei diesem Konverter sind ein paar Schriftformatierungsbefehle nicht implementiert und die Formatierung ist nur mittelmässig. Die Standardschrift ist serifenlos.

ltx2rtf Bis auf ein paar Kleinigkeiten werden die Schriften von ltx2rtf sehr gut konvertiert.

latex2rtf Auf den ersten Blick sind fast keine Unterschiede zu der mit LaTeX verarbeiteten Datei festzustellen.

Zwischen latex2rtf und ltx2rtf gibt es beinahe keine Unterschiede. Tex2RTF hat im Bereich der Schriftarten bzw. -formatierung noch ein wenig aufzuholen, ist aber trotz allem gut lesbar.

german.tex

Tex2RTF Mit den deutschen Sonderzeichen hat dieser Konverter große Probleme. Es war weder möglich, die Sonderzeichen mit der normalen Befehlssequenz darzustellen, noch ist es mit dem Befehl, der durch das german.sty-Paket implementiert ist,

gelungen. Die einzige Art, hier zum Erfolg zu kommen, ist die Darstellung mit der Befehlssequenz für Sonderzeichen.

ltx2rtf Die Darstellung mit der normalen in LaTeX implementierten Befehlssequenz ist gut gelungen. Die Befehle, die durch das german.sty-Paket zur Verfügung gestellt werden, werden nicht konvertiert. Es werden Anführungszeichen vor das Zeichen gestellt.

latex2rtf Dieser Konverter hat nur bei der Darstellung der Großbuchstaben, und auch hier nur bei der durch das german.sty-Paket bereitgestellten Befehlssequenz, Probleme.

Um Dateien in deutscher Sprache zu konvertieren, sind am besten latex2rtf und ltx2rtf zu verwenden. Bei ltx2rtf muß zwar mehr nachkorrigiert werden, aber dafür ist es eindeutig erkennbar, wo ein Umlaut oder ein ß stehen soll. latex2rtf konvertiert mehr Zeichen, dafür reduziert es beim german.sty-Paket die großgeschriebenen Umlaute auf Vokale.

Tabelle 4.1: Bewertungsergebnis der Gruppe Text

Ergebnis Gruppe Text							
Datei		Konverter					
		Tex2RTF		ltx2rtf		latex2rtf	
Name	Gew.	Note	gew.	Note	gew.	Note	gew.
char	0,4	4	1,6	4	1,6	3	1,2
fonts	0,4	3	1,2	2	0,8	1	0,4
german	0,2	5	1	2	0,4	2	0,4
Gesamtnote Formeln			3,8		2,8		2

4.5.2 Formatierung

align.tex

Die Ergebnisse unterscheiden sich nur darin, daß Tex2RTF den Text nicht blockweise ausrichtet. Ansonsten liefern alle drei Konverter dieselben Resultate. Die Befehle, mit denen LaTeX eine Umgebung startet sind implementiert. Leider fehlen die einfachen Befehle für zentrieren, rechts- und linksbündig.

article.tex

Tex2RTF Die höchste Gliederungsebene, die hier Verwendung findet, ist die Unterteilung des Artikels in Teile. Die Formatierung der Teile wird teilweise unterdrückt. Der Titel des Teils ist mit normaler Schrift dargestellt. Anschließend beginnt jedoch eine neue Seite. Die restliche Struktur ist in der gleichen Größe formatiert. Die Nummerierung der Abschnitte funktioniert gut, bis auf die Paragraphen. Diese werden als Überschriften der dritten Hierarchieebene konvertiert, und nicht wie in LaTeX auf einer tieferen Ebene verwendet.

ltx2rtf In diesem Fall ist die Gliederung in Teile mit der dritten Hierarchieebene gleichgestellt. Ansonsten wird die Struktur sehr gut untergliedert. Die Paragraphen werden nicht nummeriert.

latex2rtf Dies ist der einzige Konverter, der die Gliederung in Teile richtig konvertiert, abgesehen davon, daß vor und nach dieser Gliederungsebene eine neue Seite begonnen wird. Die Strukturen Paragraph und Unterparagraph werden nicht in der richtigen Hierarchieebene verwendet. Paragraph wird auf der gleichen Ebene wie der Unterabschnitt verwendet und der Unterparagraph wird als Überschrift der dritten Ebene gehandhabt.

Es ist keiner der Konverter fehlerfrei. Abhängig davon, worauf man Wert legt, kann man zwischen der korrekten Untergliederung in Teile oder der richtigen Verwendung der Paragraphen bzw. Unterparagraphen wählen.

liste.tex

Tex2RTF Der große Nachteil, den dieser Konverter besitzt, ist, daß das Ende einer Aufzählungsumgebung nicht richtig erkannt wird. Daraus folgend schreibt der Konverter den darauffolgenden Text unter dem letzten Punkt der Liste. Die Aufzählungen sind ansonsten gut verarbeitet. Solange im Quelltext labels gegeben sind, wenn sie auch leer sind, funktioniert der Konverter gut. Sind keine labels gegeben, wird das Listenelement nicht als solches erkannt und in den normalen Text übernommen. Die verschachtelten Aufzählungen funktionieren gut, aber die Variation der Kennzeichnung der Punkte der Liste ist auf Punkte oder arabische Ziffern beschränkt.

ltx2rtf Die Konvertierung mit ltx2rtf funktioniert gut. Die Listen sind gut lesbar. Nur die Variation der Kennzeichnung der Aufzählungspunkte läßt noch sichtbare Verbesserungen zu.

latex2rtf Mit latex2rtf lassen sich die Kennzeichnungen variieren. Leider ist bei diesem Konverter die Verwendung von labels nicht korrekt implementiert. Die Listen mit labels werden in Aufzählungen konvertiert, und die labels stehen in eckigen Klammern am Beginn des Textes.

Alle drei Konverter haben ihre Eigenheiten. Wenn Tex2RTF das Ende der Liste noch richtig verarbeitet, sind alle Ergebnisse durchaus gut lesbar und herzeigbar. Ohne Nachkorrektur bekommt man bei keinem der Konverter das völlig richtige Resultat.

report.tex

Hier bietet sich bei allen Kandidaten ein ähnliches Bild wie bei der Testdatei article.tex. Die Unterschiede zur obigen Beschreibung in article.tex sind äußerst minimal. Alle drei Konverter gehen bei der Nummerierung der Kapitel nicht auf die Gliederung in Teile ein. Erwähnenswert ist, daß bei latex2rtf die Unterdrückung der Nummerierung der Gliederung nicht funktioniert.

tabular.tex

Tex2RTF Das Ergebnis war ein dreiseitiges Dokument. Nach einer Bearbeitung des Ergebnisses kamen dann doch noch drei Tabellen zum Vorschein, die aber am Bildschirm zuerst nicht leicht erkennbar waren. Der Text, der zwischen den Tabellen steht, ist hier nicht mehr sichtbar. Alle drei Tabellen haben die gleiche Position. Sie sind linksbündig im Dokument angeordnet. Auch die Einträge in die Tabelle sind alle linksbündig angeordnet.

ltx2rtf Dieser Konverter produziert drei gut lesbare Tabellen, die in MS-Word gut weiter formatiert werden können. Ein kleiner Schönheitsfehler ist die Tabelle mitten im Text, denn diese wird nicht im Text, sondern nach dem Text positioniert. Die Ausrichtung der Tabellen entspricht ansonsten den Angaben der Quelldatei. Auch die Ausrichung der Tabelleneinträge ist korrekt.

latex2rtf Die Tabellen, die latex2rtf ausgibt, sind alle zentriert. Sie entsprechen den Tabellen, wie sie auch unter MS-Word bearbeitet werden können. Die Ausrichtung der Tabelleneinträge entspricht der in der Quelldatei vorgegebenen Ausrichtung.

Die Resultate von latex2rtf und ltx2rtf sind beinahe identisch, außer daß erstgenannter alle Tabellen zentriert. Tex2RTF produziert ein Ergebnis, das der Anwender zuerst formatieren muß, um brauchbar zu sein.

Tabelle 4.2: Bewertungsergebnis der Gruppe Formatierung

Ergebnis Gruppe Formatierung							
Datei		Konverter					
		Tex2RTF		ltx2rtf		latex2rtf	
Name	Gew.	Note	gew.	Note	gew.	Note	gew.
align	0,2	3	0,6	3	0,6	3	0,6
article	0,2	2	0,4	2	0,4	2	0,4
liste	0,2	3	0,6	2	0,4	3	0,6
report	0,2	3	0,6	3	0,6	3	0,6
tabular	0,2	5	1	2	0,4	2	0,4
Gesamtnote Formeln			3,2		2,4		2,6

4.5.3 Verweise

cite.tex

Referenzen können direkt am Ende in der Quelldatei stehen oder in eine eigene Datei geschrieben werden. Dies erfordert unter LaTeX die Verwendung von BibTeX. Dieses Programm bearbeitet die Referenzdatei derart, daß LaTeX die angegebenen Referenzen auch richtig verwenden kann. Beide Arten wurden zum Test verwendet.

Tex2RTF Beide Arten der Referenzen funktionieren gut. Der Schwachpunkt, der sich hier einschleicht, ist, daß sich die Ergebnisdatei im ersten Fall über drei Seiten erstreckt, wobei die erste Seite der Text mit den Referenzmarken ist, die zweite Seite nur aus der Überschrift „1. References“ besteht und die dritte Seite die Referenzen wiedergibt. Die Zuordnung ist sehr gut, nur eine kurze Nachbearbeitung der Formatierung bleibt nicht erspart. Die zweite Variante ist ähnlich der ersten, nur daß die Referenzen bereits auf der zweiten Seite, unter der oben angeführten Überschrift stehen.

ltx2rtf Mit diesem Konverter war es nicht möglich, irgend ein Ergebnis zu erzielen. Es wurden weder die Referenzmarken, noch die Referenzen verarbeitet.

latex2rtf In diesem Fall ist das beste Ergebnis erzielt worden. Die Referenzen und die Referenzmarken stimmten mit dem Original überein.

Um Referenzen konvertieren zu können, darf man nicht ltx2rtf verwenden. Gute Ergebnisse, die nach der Formatierung auch sehr brauchbar sind, liefert Tex2RTF. Für die Konvertierung mit dem geringsten Aufwand muß man latex2rtf verwenden.

footnote.tex

Hier weist nur latex2rtf Mängel auf. Die beiden anderen Konverter leisten gute Arbeit. latex2rtf verarbeitet zwar die Fußnoten , aber es ist die Zuordnung schwierig, da bei der Auflistung der Fußnoten der erste Buchstabe des Fußnotentextes hochgestellt wird, und nicht wie gewünscht, die im Text angeführten Nummern. Aus unerklärlichen Gründen wird im Text jedesmal vor der Fußnote der Buchstabe l hochgestellt.

inhalt.tex

Die Inhaltsverzeichnisse werden bei allen drei Konvertern mit MS-Word erstellt. Die Übersetzung der Strukturierung ist bereits unter report.tex beschrieben. Einziger Unterschied ist, daß latex2rtf die Gliederung in Teile zwar richtig konvertiert, aber MS-Word diese Gliederung nicht ins Inhaltsverzeichnis einfließen läßt. So entsteht ein Inhaltsverzeichnis, das den beiden anderen Inhaltverzeichnissen sehr ähnlich ist.

Tabelle 4.3: Bewertungsergebnis der Gruppe Verweis

Ergebnis Gruppe Verweis							
Datei		Konverter					
		Tex2RTF		ltx2rtf		latex2rtf	
Name	Gew.	Note	gew.	Note	gew.	Note	gew.
cite	0,4	1	0,4	5	2	1	0,4
footnote	0,3	1	0,3	1	0,3	3	0,9
inhalt	0,3	2	0,6	2	0,6	2	0,6
Gesamtnote Formeln			1,3		2,9		1,9

4.5.4 Formeln

Der Vergleich im mathematischen Umfeld ist der für diese Arbeit wichtigste. Es wird mit dem Erkennen der mathematischen Umgebungen begonnen. Danach wird die Darstellbarkeit modifizierter Zeichen und von Sonderzeichen untersucht. Dies wird gefolgt von

einer Matrizendarstellung. Mit den letzten beiden Dateien wird versucht, kompliziertere Formeln zu konvertieren.

Ausgegangen wird davon, daß die Formeln mit dem Formeleditor von MS-Word nachbearbeitet werden können. Eine Darstellung der Formeln als Bild wird nicht in Betracht gezogen. Dies trifft vor allem ltx2rtf, da dieser Konverter die Möglichkeit bietet, Formeln als Bild wiederzugeben.

mathUmg.tex

Tex2RTF Bei diesem Konverter sind nur vier der neun mathematischen Umgebungen implementiert. Bei den fehlenden Implementierungen werden die LaTeX-Befehle als Text in das Ergebnis geschrieben. Es fehlen dabei aber auch einige Steuerzeichen, wie der Backslash und Klammern, sodaß nur ein geübter LaTeX-Anwender sofort erkennen kann, daß hier eine mathematische Umgebung gestartet werden sollte. Die Möglichkeit zum Durchnummerieren der Formeln funktioniert bei Tex2RTF nicht.

ltx2rtf Hier werden bereits sieben von neun mathematischen Umgebungen verwendet. Es fehlt aber noch die Implementierung von Formelfeldern. Von den erkannten Umgebungen wird der Großteil richtig konvertiert. Die Nummerierung der Formeln ist leider noch nicht realisiert.

latex2rtf Alle neun Umgebungen werden erkannt und auch richtig angewendet. Auch die Nummerierung der Formeln funktioniert.

Aus diesem Ergebnis folgt, daß bei der weiteren Untersuchung der mathematischen Fähigkeiten der Konverter auf die Implementierung der Umgebungen Rücksicht genommen werden muß. Es wird versucht, möglichst nur solche Umgebungen zu verwenden, die auch von allen drei Konvertern verstanden werden, da sonst die Ergebnisse bereits indirekt beeinflußt werden.

charmath.tex

Tex2RTF Hier wird gänzlich auf eine Konvertierung verzichtet. Das Ergebnis besteht aus den Buchstaben des LaTeX-Befehls und aus dem zu modifizierendem Zeichen.

ltx2rtf Dieser Konverter kann sechs der insgesamt vierzig Testobjekte richtig verarbeiten. Der Großteil wird unverändert wiedergegeben. Einige der zu modifizierenden Zeichen werden überhaupt weggelassen.

Tabelle 4.4: Beispiele zur Datei mathUmg.tex

Befehl	LaTeX	Tex2RTF	ltx2rtf	latex2rtf
$ $	$l = x - 45$	l=x-45	$l = x - 45$	$l = x - 45$
\begin{math} \end{math}	$a = x + y$	beginmath a=x+y endmath	$a = x + y$	$a = x + y$
\begin {eqnarray*} \end {eqnarray*}	$l^2 = r + t * 6$	begineqnarray* l^2=r+t*t endeqnarray*	begin of env: EQNARRAY* l^2=r+t+6 end of env: EQNARRAY*	$l^2 = r + t * 6$

latex2rtf Es werden keine Zeichen ausgelassen. Richtig verarbeitet werden zehn Zeichen. Die restlichen Zeichen werden durch das unveränderte Zeichen, gefolgt von einem Beistrich und der Modifikation dargestellt. Die dem Beistrich folgenden Modifikationen sind bis auf ein Zeichen korrekt. Mit diesem Wissen können diese modifizierten Zeichen in MS-Word leicht richtiggestellt werden.

Die Bewertung ist schwierig durchzuführen. Die Frage ist, ist es besser, die Modifikationen oder die Zeichen auszulassen, oder durch den LaTeX-Befehl darzustellen. latex2rtf läßt nichts aus, aber auch hier muß man wissen, wie die Zeichen vom Konverter dargestellt werden.

zeichen.tex

Diese Datei könnte man leicht in drei Gruppen unterteilen. Die Gruppen wären das griechische Alphabet, die mathematischen Sonderzeichen und die Hoch- und Tiefstellungsmöglichkeiten.

Das griechische Alphabet bietet keine großen Unterschiede. Alle drei Konverter können den Großteil richtig darstellen. Aus diesem Grund wird nicht weiter darauf eingegangen.

Tex2RTF Von allen Sonderzeichen wurden sechzig richtig verarbeitet, zwei falsch. Die restlichen Zeichen wurden durch die Buchstaben des LaTeX-Befehls dargestellt. Auf Hoch- und Tiefstellungen wurde keine Rücksicht genommen. Es wurden die Zeichen genauso wiedergegeben, wie sie im Quelltext stehen.

ltx2rtf Dieser Konverter erreichte mit nur siebenundvierzig richtigen Darstellungen den niedrigsten Wert. Der Rest der Zeichen wurde im Ergebnis einfach ausgelassen. Der

Tabelle 4.5: Beispiele zur Datei charmath.tex

Zeichen		LaTeX		Tex2RTF		ltx2rtf		latex2rtf	
\hat{o}	\hat{x}	$\hat{o}$	$\hat{x}$	hato	hatx	ó		$\hat{o}$	$x,\hat{}$
\check{o}	\check{x}	$\check{o}$	$\check{x}$	checko	checkx	o	x	$o,\check{}$	$x,\check{}$
\acute{o}	\acute{x}	$\acute{o}$	$\acute{x}$	acuteo	acutex	ó		$\acute{o}$	$x,\acute{}$
\grave{o}	\grave{x}	$\grave{o}$	$\grave{x}$	graveo	gravex	ò		$\grave{o}$	$x,\grave{}$
\tilde{o}	\tilde{x}	$\tilde{o}$	$\tilde{x}$	tildeo	tildex	õ		$\tilde{o}$	$x,\tilde{}$
\bar{o}	\bar{x}	$\bar{o}$	$\bar{x}$	baro	barx	o	x	$o,\bar{}$	$x,\bar{}$
\vec{o}	\vec{x}	$\vec{o}$	$\vec{x}$	veco	vecx	o	x	$o,\vec{}$	$x,\vec{}$
\dot{o}	\dot{x}	$\dot{o}$	$\dot{x}$	doto	dotx	o	x	$o,\dot{}$	$x,\dot{}$
\ddot{o}	\ddot{x}	$\ddot{o}$	$\ddot{x}$	ddoto	ddotx	o	x	$\ddot{o}$	$x,\ddot{}$
\breve{o}	\breve{x}	$\breve{o}$	$\breve{x}$	breveo	brevex	o	x	$o,\breve{u}$	$x,\breve{u}$
\hat{O}	\hat{X}	$\hat{O}$	$\hat{X}$	hatO	hatX	Ó		$\hat{O}$	$X,\hat{}$
\check{O}	\check{X}	$\check{O}$	$\check{X}$	checkO	checkX	O	X	$O,\check{}$	$X,\check{}$
\acute{O}	\acute{X}	$\acute{O}$	$\acute{X}$	acuteO	acuteX	Ó		$\acute{O}$	$X,\acute{}$
\grave{O}	\grave{X}	$\grave{O}$	$\grave{X}$	graveO	graveX	Ò		$\grave{O}$	$X,\grave{}$
\tilde{O}	\tilde{X}	$\tilde{O}$	$\tilde{X}$	tildeO	tildeX	Õ		$\tilde{O}$	$X,\tilde{}$
\bar{O}	\bar{X}	$\bar{O}$	$\bar{X}$	barO	barX	O	X	$O,\bar{}$	$X,\bar{}$
\vec{O}	\vec{X}	$\vec{O}$	$\vec{X}$	vecO	vecX	O	X	$O,\vec{}$	$X,\vec{}$
\dot{O}	\dot{X}	$\dot{O}$	$\dot{X}$	dotO	dotX	O	X	$O,\dot{}$	$X,\dot{}$
\ddot{O}	\ddot{X}	$\ddot{O}$	$\ddot{X}$	ddotO	ddotX	O	X	$\ddot{O}$	$X,\ddot{}$
\breve{O}	\breve{X}	$\breve{O}$	$\breve{X}$	breveO	breveX	O	X	$O,\breve{u}$	$X,\breve{u}$

Vorteil liegt in der Darstellung der Hoch- und Tiefstellungen, die gut funktioniert. Hier werden die hoch- bzw. tiefgestellten Zeichen auch in einer kleineren Schriftgröße dargestellt. Es gibt die Einschränkung, daß innerhalb einer hoch- bzw. tiefgestellten Gruppe keine weitere Hoch- bzw. Tiefstellung mehr möglich ist.

latex2rtf Hier wurde die größte Anzahl der richtig dargestellten Sonderzeichen mit dreiundsiebzig festgestellt. Die Hoch- und Tiefstellungen bringen daselbe Ergebnis wie ltx2rtf, bis auf die Einschränkung, daß die hoch- bzw. tiefgestellten Zeichen die gleiche Schriftgröße wie das Zeichen auf der Grundlinie aufweisen.

Der große Unterschied in dieser Kategorie ist die Darstellbarkeit der Hoch- und Tiefstellungen. Tex2RTF kann hier nicht mit den anderen Konvertern mithalten. latex2rtf ist in Bezug auf die Anzahl der dargestellten Sonderzeichen führend.

Tabelle 4.6: Beispiele zur Datei zeichen.tex

Befehl		LaTeX		Tex2RTF		ltx2rtf		latex2rtf	
\approx	\aleph	$\approx$	$\aleph$	$\approx$	$\aleph$	$\approx$		$\approx$	$\aleph$
\asymp	\angle	$\asymp$	$\angle$	ASYMP	$\angle$				$\angle$
\bowtie	\bot	$\bowtie$	$\bot$	bowtie	$\bot$				
\cong	\exists	$\cong$	$\exists$	$\cong$	$\exists$		$\exists$	$\cong$	$\exists$
\dashv	\flat	$\dashv$	$\flat$	DASHV	flat				
\doteq	\forall	$\doteq$	$\forall$	DOTEQ	$\forall$		$\forall$		$\forall$
\equiv	x^5	$\equiv$	x^5	$\equiv$	x^5	$\equiv$	x^5	$\equiv$	x^5
\frown	x_3^7	$\frown$	x_3^7	„Smilie“	x_3^7		${x_3}^7$		${x_3}^7$
\ge	x_{i_5}	$\geq$	x_{i_5}	$\geq$	x_i_5		x_{i5}	$\geq$	x_{i5}
\gg	x_{i^6}	$\gg$	x_{i^6}	$\gg$	x_i^6		${x_i}^6$	>>	${x_i}^6$
\in	x^{i^7}	$\in$	x^{i^7}	$\in$	x^i^7	$\in$	x^{i7}	$\in$	x^{i7}
\le	x_i{}^b	$\leq$	$x_i{}^b$	$\leq$	x_i^b		${x_i}^b$	$\leq$	${x_i}^b$
\ll	x^{i_8}	$\ll$	x^{i_8}	$\ll$	x^i_8	$\ll$	${x^i}_8$	<<	${x^i}_8$
\mid	{}_2x_i	$\mid$	${}_2x_i$	$\mid$	_2x_i		${}_2x_i$		${}_2x_i$

Die Hoch- und Tiefstellungen bei latex2rtf sind in normaler Schriftgröße

array.tex

Tex2RTF Matrizen werden mit diesem Konverter nicht dargestellt. Der einzige Unterschied zum Quelltext ist die Erkennung und die Verarbeitung der Zeilenschaltung. Ansonsten wird der Text innerhalb der Matrizenumgebung unverändert wiedergegeben.

ltx2rtf Die Umgebung eines Arrays wird von ltx2rtf nicht verarbeitet. Es wird zwar der Beginn einer neuen Umgebung erkannt, daher wird die Matrize auch vom Text abgesetzt. Umgesetzt werden aber nur die Trennzeichen zwischen den einzelnen Einträgen und die Zeilenschaltungen.

latex2rtf Der Ansatz, den latex2rtf beim Konvertieren macht ist gut. Die Matrize wird bereits als solche erkennbar verarbeitet. Lediglich die letzte Zeile wird nicht richtig formatiert, aber das kommt daher, daß das Zeilenabschlußzeichen fehlt. Fügt man diese Zeichen im Quelltext ein, wird auch die letzte Zeile richtig konvertiert werden. Eine Spaltenausrichtung fließt ebenfalls in die Matrize ein. Hierbei wird die linke Ausrichtung zu einem zentrierenden Tabulatur und die beiden anderen Ausrichtungen werden zu linksbündigen Tabulatoren.

Außer latex2rtf ist kein Konverter in der Lage, Matrizen brauchbar zu verarbeiten.

Tabelle 4.7: Beispiele zur Datei array.tex

Konverter	Ergebnis
LaTeX	$w = \begin{array}{lll} x+y & y+2c & z*x^2 \\ x & y & z \\ x & y & z \\ x & y & z \end{array}$
Tex2RTF	w = lcrx+y&y+2c&z*x^2 x&y&z x&y&z x&y&z
ltx2rtf	w = ***begin of environment: ARRAY *** lcr x+y y+2c z*x^2 x y z x y z x y z ***end of environment: ARRAY ***
latex2rtf	w = x+y y+2c z*x^2 x y z x y z x y z

frac.tex

Tex2RTF Dieser Konverter beendet seine Arbeit nach der ersten Formel, und selbst diese ist nicht lesbar konvertiert. Das Ergebnis besteht aus einer Mischung aus verkürzten LaTeX-Befehlen und den einzelnen Zeichen der Formel. Es war auch mit verschiedenen Variationen nicht möglich, Tex2RTF zur weiteren Konvertierung zu bewegen.

ltx2rtf Die Ergebnisse dieses Kandidaten waren schon etwas besser. Der Konverter verarbeitete das gesamte Dokument. Das Resultat war aber nicht gut genug, um die Formeln auch richtig lesen zu können. So wurden Brüche komplett unterdrückt, das Wurzelzeichen nicht entsprechend verlängert und das Summen- und Produktzeichen einfach weggelassen.

latex2rtf Hier wurden keine Zeichen weggelassen. Alles wurde konvertiert. Aber auch hier gibt es noch Mängel. So wird ein Bruch nicht als solcher geschrieben, sondern

lediglich als Schrägstrich angedeutet. Das Wurzelzeichen wird ebenfalls zu kurz dargestellt. Mit einiger Übung ist es möglich, diese Formeln nach der Konvertierung zu lesen.

Brauchbare Ergebnisse liefern nur latex2rtf und teilweise auch ltx2rtf. Hier erhält man zumindest ein Resultat, daß man anschließend verbessern kann, und man ist nicht gezwungen, die gesamten Formeln neu einzugeben.

Tabelle 4.8: Beispiele zu den Dateien frac.tex und hoch.tex

LaTeX	Tex2RTF	ltx2rtf	latex2rtf
$\frac{1}{2} = \frac{\alpha+\beta}{\gamma+\delta}$	frac12 = frac $\alpha+\beta\gamma+\delta$	$12 = \alpha+\beta\gamma+\delta$	$1 / 2 = \alpha+\beta / \gamma+\delta$
$z = \sqrt{x+y}$	z = sqrtx+y	$z = \sqrt{x+y}$	$z = \sqrt{x+y}$
$\int_0^\infty \frac{a}{\sqrt{x+c}}\,dx$	int_0^∞ fracasqrtx+c	$\int_0^\infty a\sqrt{x+c}dx$	$\int_0^\infty a / \sqrt{x+c}dx$
$\prod_0^\infty \frac{a}{\sqrt{x+c}}\,dx$	prod_0^∞ fracasqrtx+c dx	${}_0^\infty a\sqrt{x+c}dx$	$\prod_0^\infty a / \sqrt{x+c}dx$
$\sum_{0\le i\le m} E_i$	sum_0≤ i≤ m E_i	${}_{0im}E_i$	$\sum_{0\le i\le m} E_i$
$lim_{y\to 1} f(y)$	lim _yto 1f(y)	$lim_{y1} f(y)$	$lim_{y1} f(y)$

hoch.tex

Auch bei dieser Datei kam es erwartungsgemäß zum gleichen Ergebnis wie bei der soeben oben beschriebenen Datei frac.tex. Der einzige Unterschied ist, daß die Befehle, die vom amsmath.sty-Paket zur Verfügung gestellt werden, von keinem einzigen der drei Kandidaten konvertiert werden konnten.

Tabelle 4.9: Bewertungsergebnis der Gruppe Formeln

Ergebnis Gruppe Formeln							
Datei		Konverter					
		Tex2RTF		ltx2rtf		latex2rtf	
Name	Gew.	Note	gew.	Note	gew.	Note	gew.
mathUmg	0,3	4	1,2	3	0,9	1	0,3
charmath	0,15	4	0,6	4	0,6	3	0,45
zeichen	0,15	4	0,6	3	0,45	3	0,45
array	0,15	5	0,75	4	0,6	3	0,45
frac	0,15	5	0,75	4	0,6	3	0,45
hoch	0,1	5	0,5	4	0,4	4	0,4
Gesamtnote Formeln			4,4		3,55		2,5

4.5.5 Institutsdateien

Auf Verwendung der Institutsdateien zur Bewertung der Konverter wurde verzichtet. Aus den bereits am Anfang erwähnten Schwierigkeiten war es nur möglich, die Dateien teilweise als Tests laufen zu lassen. Zur sinnvollen Nutzung dieser Dateien mußten die Definitionen zuerst entfernt und die entsprechenden LaTeX-Befehle eingesetzt werden. Auch die Suche nach Sequenzen, die, aus welchen Gründen auch immer, nicht konvertiert werden konnten, gestaltete sich als sehr schwierig.

Dies war noch alles zu meistern, aber die Bewertbarkeit der Ergebnisse brachte neue Schwierigkeiten. Es hätten die Dateien so weit zerlegt werden müssen, um auf Kriterien zu kommen, die auch vernünftig und anwendbar sind. Die meisten dieser Dateien überstreichen das gesamte Gebiet. Es ist nicht möglich, zu komplexe Ergebnisse miteinander zu vergleichen, ohne daß es zu stark subjektiver Bewertung kommt.

Die Hauptaufgabe der Institutsdateien war Schluß endlich, die wichtigsten Kriterien herauszufinden. Es war Aufgabe dieser Diplomarbeit den besten Konverter für diese Aufgabe zu suchen, und diesen zu erweitern. Daher kam den Institutsdateien auch die Aufgabe zu, die Verbesserungen in die Richtung zu lenken, daß diese Dateien am Ende auch soweit wie möglich konvertiert werden können.

4.5.6 Gesamtergebnis

Die nun folgende Tabelle ist die gesammelte Berechnung der Noten der einzelnen Konverter. Auch die Gewichtung zeigt wiederum die Ausrichtung auf den Schwerpunkt bezüglich der Formeln.

Tabelle 4.10: Gesamtbewertung

Gesamtergebnis							
Gruppen		Konverter					
		Tex2RTF		ltx2rtf		latex2rtf	
Kategorie	Gew.	Note	gew.	Note	gew.	Note	gew.
Text	0,2	3,8	0,76	2,8	0,56	2	0,4
Formatierung	0,2	3,2	0,64	2,4	0,48	2,6	0,52
Verweis	0,2	1,3	0,26	2,9	0,58	1,9	0,38
Formeln	0,4	4,4	1,76	3,55	1,42	2,5	1
Gesamtnote			3,42		3,04		2,3

4.6 Fazit

Es stellte sich heraus, daß nur fehlerfreie Quelldateien vernünftig konvertiert werden können. Jeder Fehler, den LaTeX bemängelt, führt unwillkürlich zu großen Problemen der Konverter. Es gibt aber auch Sequenzen, die von LaTeX nicht bemängelt werden, die aber eine Mischform von LaTeX und TeX darstellen. Auch solche Sequenzen erschweren das Arbeiten mit den Konvertern.

Aus diesen Gründen stellt sich die erste Vorbedingung für die Verwendung von Konvertern. Diese gilt für alle Kandidaten gleichermaßen. Es ist sogut wie unmöglich eine Quelldatei zu konvertieren, die Fehler aufweist oder aus Mischformen besteht.

Um den Konvertern möglichst die gleichen Chancen zu geben, wurden alle Quelldateien möglichst so geschrieben, daß die bereits einmal in einem Test aufgetretenen Schwächen nicht das Ergebnis eines anderen Test beeinflussen. Das beste Beispiel hierfür ist die Verwendung der mathematischen Umgebungen. Es wurden hier alle Test so umgeschrieben, daß die bereits festgestellte fehlende Implementierung einer Umgebung keine Rückwirkung auf die anderen Tests hatte.

Um aus einem Konvertierer auch das Beste herauszuholen, muß man sich zuerst einmal mit all seinen Eigenheiten auseinandersetzen. Diese Beschäftigung kann das Ergebnis völlig verändern. Wenn nicht auf die Eigenheiten der Konverter Rücksicht genommen wird, verschlechtert sich das Ergebnis rapid.

4.6.1 Tex2RTF

Installation & Anwenderfreundlichkeit: Unter Windows ist das Installieren und das Handhaben sehr einfach. Mit einem einfachen Setup läßt sich dieser Konverter sehr einfach unter Windows als Programm installieren und braucht keinen MS-DOS-Modus. Daher ist auch die Anwenderfreundlichkeit sehr groß.

Funktionstüchtigkeit: Dieser Konverter verarbeitet ein Dokument sehr gut, solange seltene Schriftzeichen nicht verwendet werden. In diese Kategorie kann man auch die in der deutschen Sprache verwendeten Umlaute zählen. Mit diesen Zeichen hat er große Schwierigkeiten. Auch in der Testgruppe Formatierung zeichnen sich Probleme ab. Die Tabellen waren auf den ersten Blick nicht erkennbar und erst nach einer Nachbearbeitung einigermaßen brauchbar. Mit den Listen hat dieser Kandidat ebenfalls zu kämpfen. Die Aufzählung ist zwar durchnummeriert und auch einigermaßen zufriedenstellend eingerückt. Leider erkennt er das Ende der Aufzählung bzw. Liste nicht, und hängt so den folgenden Text unmittelbar an den letzten Punkt an.

Der größte Nachteil von Tex2RTF ist die Konvertierung von Formeln. Hier kann man eigentlich nur sagen, daß dieser Teil nicht unterstützt wird. Auf diesem Gebiet braucht dieser Konverter sicher Erweiterungen, um mit den anderen Kandidaten mithalten zu können.

Dafür hat dieser Konverter seine große Stärke bei den Verweisen. In dieser Kategorie ist er nicht zu schlagen. Es werden alle getesteten Arten von Verweisen gut verarbeitet. Die noch notwendigen Nachbearbeitungen sind sofort erkennbar und auch nicht aufwendig.

Sollte ein Dokument vorliegen, das keine Formeln beinhaltet, und in dem nicht zu viele Sonderzeichen vorkommen, so kann dieser Konverter empfohlen werden. Für eine deutschsprachige Konvertierung ist dieser Konverter nur sehr eingeschränkt brauchbar.

Die Fehlerdiagnose, die angezeigt wird, ist nicht sehr hilfreich. Die Meldungen sind nur für geübte Anwender brauchbar. Es können die Fehlermeldungen als log-Datei abgespeichert und durchgearbeitet werden. Es ist jedoch sinnvoller, bereits die Quelldatei z. B. unter LaTeX auf Fehlerfreiheit zu untersuchen.

Stabilität: Die Wahrscheinlichkeit, eine konvertierte rtf-Datei mit MS-Word öffnen zu können, ist sehr hoch. Der Nachteil, den dieser Konverter besitzt, ist, daß man das Dokument zwar öffnen kann, aber dafür wird die Übersetzung oft mitten im Dokument abgebrochen und nicht mehr weitergeführt.

aktuelle Entwicklung: Es fand sich ein Hinweis, daß es auch für dieses Programm Verbesserungen und Weiterentwicklungen gibt. Leider war es nicht möglich, eine verbesserte Version im Internet zu finden.

Plattform: Dieser Konverter ist für Windows und Unix-Systeme verfügbar.

4.6.2 ltx2rtf

Installation & Anwenderfreundlichkeit: Die Installation erfolgte durch das Entpacken einer zip-Datei. Anschließend mußte man zumindest die config.cfg-Datei anpassen. Der MS-DOS-Modus verlangte eine Definition von RTFPATH. Das Starten des Konverters erfolgte unter MS-DOS im Zeilenmodus.

Um den vollen Umfang von ltx2rtf nutzen zu können, muß man eine lauffähige LaTeX-Version installiert haben. Die Installation dieses Programms hat sich unter MS-DOS als umständlich erwiesen.

Nach der gelungenen Installation von LaTeX müssen noch die Konfigurationsdateien und die Batchdateien angepaßt werden, damit der Konverter auch in der Lage ist, LaTeX zu benutzen. Wenn die entsprechenden Optionen beim Start des Konverters angegeben sind, braucht er LaTeX um die Formeln im Quelltext zu formatieren und anschließend in ein Bild umzuwandeln. Dieses Bild wird anstelle der konvertierten Formel eingesetzt. Dieses Bild kann natürlich nicht mit dem Formeleditor bearbeitet werden. Daher wurde auch diese Verwendung in der Bewertung nicht berücksichtigt.

Funktionstüchtigkeit: Auch dieser Konverter hat seine Probleme mit den Sonderzeichen. Ohne diese ist das Ergebnis durchaus gut. Die Definitionen des german.sty-Pakets für die deutschen Umlaute werden nicht richtig verwendet, aber es wird ein Anführungszeichen vor den betroffenen Buchstaben gesetzt, und er ist daher als solcher erkennbar.

In der Gruppe Formatierung schneidet der Konverter gut ab. Bei der Verarbeitung von Überschriften gibt es nur das Problem, daß die Untergliederung in Teile nicht richtig konvertiert wird. Das Ergebnis der verarbeiteten Listen ist sehr gut, nur die Variation der Labels läßt Wünsche offen. Auch mit Tabellen kann dieser Konverter ganz gut umgehen.

Kein Resultat zeigt der Konverter bei der Verarbeitung von Referenzen. Dafür ist das Ergebnis bei Fußnoten perfekt. Das Inhaltsverzeichnis wird mit MS-Word erstellt, bietet daher auch das gleiche Bild, wie die Konvertierung der Überschriften.

Probleme hat dieser Konverter aber bei der Verarbeitung von Formeln. Es sind fast alle mathematischen Umgebungen implementiert. Es fehlt nur die Realisierung der Umgebung Equationarray und darüberhinaus auch die Verarbeitung der Umgebung Array, die zur Darstellung von Matrizen benötigt wird. Bei den mathematischen und bei den modifizierten Schriftzeichen hat dieser Konverter einen Nachholbedarf. Auch etwas anspruchsvollere Formeln werden nur teilweise, unter Auslassen der nicht konvertierbaren Teile, verarbeitet.

Dieser Konverter ist für Dokumente zu empfehlen, die mit ganz einfachen Formeln auskommen, und die kaum Literaturverweise beinhalten. Diese Dokumente werden gut verarbeitet, auch wenn sie in deutscher Sprache abgefaßt sind.

Die Fehlerdiagnose dieses Programms sollte noch verbessert werden. Sie ist wie bei allen anderen Kandidaten ebenfalls nicht einfach zu lesen und kann bei diesem Konverter nicht in einer Datei abgespeichert werden, ohne die Standardausgabe zu verändern.

Stabilität: In diesem Punkt weist dieser Konverter Schwächen auf. Wenn in der tex-Datei Befehle oder andere Sequenzen vorkommen, die dieses Programm nicht übersetzen bzw. ignorieren kann, bricht es die Konvertierung ab, und die rtf-Datei kann nicht mit MS-Word geöffnet werden, da die richtigen Abschlußsequenzen einer rtf-Datei fehlen. Diese Abschlußsequenzen können auch nicht einfach vom Benutzer angefügt werden, da der Konverter nicht bei jeder Datei in der gleichen Verschachtelungsebene und in der gleichen Umgebung abbricht. Der Aufwand, nach der fehlenden Sequenz zu suchen, ist um vieles höher, als die Quelldatei von Fehlern zu säubern und eventuelle Abbruchstellen kurzerhand aus der Datei zu streichen.

aktuelle Entwicklung: Dieser Konverter befindet sich in der Weiterentwicklung. Daniel Taupin erweitert und verbessert den Konverter ständig.

Plattform: Dieses Programm ist für MS-DOS und für Unix-Systeme im Internet verfügbar.

4.6.3 latex2rtf

Installation & Anwenderfreundlichkeit: Dieser Konverter kann als zip-Datei aus dem Internet bezogen werden. Durch das Entpacken der zip-Dati erfolgt die Installation der Dateien in den richtigen Verzeichnissen. Zum Starten dieses Programms kann man entweder die mitgelieferte Batchdatei modifizieren und mit dieser arbeiten, oder man muß den RTFPATH setzen und das Programm direkt ohne Hilfe der Batchdatei starten. Ob mit oder ohne Batchdatei erfolgt die Benützung des Programms im MS-DOS Textmodus. Die Version, die von Windows unterstützt wird, ist älter, und wurde daher auch nicht zur Bewertung herangezogen.

Funktionstüchtigkeit: Hier gilt für modifizierte Schriftzeichen in der Textumgebung, als auch in der mathematischen Umgebung, daß auch nicht mehr korrekt verarbeitet werden, als bei den anderen Kandidaten. Sollte aber ein Zeichen nicht richtig modifiziert dargestellt werden können, so folgt die Modifikation dem Zeichen, durch einen Beistrich getrennt, nach. So sind diese Zeichen bei der Nachbearbeitung erkennbar. Auch die Befehle des german.sty-Paketes werden für kleine Buchstaben richtig verwendet. Die Formatierung der Schriftzeichen kommt hier dem idealen Ergebnis schon sehr nahe.

Dieser Konverter ist der einzige, der auch die Gliederung des Dokuments in Teile erkennt. Leider gibt es bei der Erstellung des Inhaltsverzeichnisse unter MS-Word keinen entsprechenden Gliederungsbefehl, der angewendet wird, und daher fehlt diese Unterteilung auch im Inhaltsverzeichnis. Der Nachteil dieses Konverters bei der Formatierung ist, daß die Paragraphen und Unterparagraphen in eine falsche Hierarchieebene versetzt werden.

Gut umgehen kann dieser Konverter ebenfalls mit Literaturverweisen. Andererseits hat er Probleme mit den Fußnoten. Diese werden zwar im Text richtig verwendet, aber der Auflistung am unteren Ende der Seite fehlt die richtige Zuweisungsnummer.

Listen und Aufzählungen werden sehr gut verarbeitet, nur die Beschreibungslabels sind noch nachzubearbeiten. Tabellen werden gut, aber zentriert, im Dokument wiedergegeben.

Dieser Konverter ist der einzige, bei dem alle mathematischen Umgebungen implementiert sind. Er ist der einzige Konverter, der einigermaßen mit Matrizen umgehen und auch die meisten mathematischen Sonderzeichen von allen Konvertern darstellen kann. Bei anspruchsvolleren Formeln werden alle Zeichen und Operationen, wenn nicht richtig konvertiert, zumindest richtig angedeutet.

Mit diesen Vorgaben empfiehlt sich dieser Konverter für die Verwendung bei Dokumenten mit mathematischen Formeln. Diese Formeln müssen zwar nachbearbeitet werden, aber sie werden meist, zumindest lesbar, dargestellt.

Der Grad der Fehlerdiagnose läßt sich hier einstellen. Die Fehlermeldungen lassen sich in eine Datei ausgeben. Auch hier braucht man einige Übung, um mit den Fehlermeldungen etwas anfangen zu können.

Stabilität: Solange die Quelldatei fehlerfrei ist und möglichst wenig Benutzerdefinitionen vorkommen, läuft der Konverter gut. Es können die meisten Ergebnisdateien sofort im MS-Word weiterbearbeitet werden. Nur wenn das Programm abstürzt und die Schlußsequenz der rtf-Datei nicht mehr in die Datei schreiben kann, ist es nicht möglich, direkt mit MS-Word weiterzuarbeiten. Auch hier gilt das, was bereits für ltx2rtf gesagt wurde, daß ein korrektes Dokument allem andern vorzuziehen ist.

aktuelle Entwicklung: Dieses Programm wird phasenweise ausgebaut und erweitert. Es gab in der letzten Zeit einige neue Versionen.

Plattform: latex2rtf gibt es als Download für MS-DOS, Windows und Unix-Systeme.

Kapitel 5

latex2rtf

5.1 Wieso latex2rtf

Die Entscheidung, auf den Konverter latex2rtf1.9e aufzubauen, fiel aufgrund der eindeutig besten Konvertierung der Formeln. Mit der Aufgabenstellung, daß die Formeln im Formeleditor von MS-Word weiterzubearbeiten sein sollten, gab es eigentlich keine Alternative.

Das Programm ist auf vielen Plattformen anwendbar. Es ist in C geschrieben und sehr modular aufgebaut. Auch das war ein wichtiger Grund, da man hier Funktionen leicht einfügen oder ersetzen kann, ohne im Programm gravierende Änderungen vornehmen zu müssen, die vielleicht ungeahnte Auswirkungen auf die Stabilität oder die sonstigen Funktionen des Konverters haben.

5.2 Die Struktur/Funktionsweise des Konverters

Der Konverter besteht aus einer großen Anzahl von C-Dateien und den dazugehörigen Headerdateien. Die Funktion dieser Module ist hier kurz angeführt. Die restlichen C-Dateien enthalten Initialisierungsroutinen, Hilfsfunktionen und die Implementierungen der zu konvertierenden Befehle und werden hier nicht extra angeführt.

Neben diesen Programmteilen gibt es noch einige .cfg-Dateien. Der Großteil dieser Dateien sind die sogenannten *language*.cfg-Dateien. Die drei restlichen verbleibenden Dateien sind direct.cfg, ignore.cfg und fonts.cfg. Dieses .cfg-Dateien können sehr leicht verändert und vom jeweiligen Benutzer angepasst bzw. erweitert werden.

main.c: Dieser Programmteil verarbeitet die Argumente, mit denen der Konverter gestartet wird. Main öffnet und schließt die RTF- und die erste TEX-Quelldatei, liest die `.cfg`-Dateien und die eingestellte Sprache ein, initialisiert den Stack, die Schriftarten und die LATEX-Parameter, konvertiert den Definitionsteil des Dokuments und schreibt den notwendigen RTF-Header. Es wird hier auch die eigentliche Konvertierung des Dokuments gestartet. Am Ende schließt dieser Programmteil die RTF- und die TEX-Datei.

convert.c: Dies ist das Herzstück des Konverters und wird von main.c aufgerufen, um den Textteil des Dokuments zu konvertieren. Dieser Programmteil lies die Befehle, Parameter und Zeichen mit Hilfe von parser.c ein und verarbeitet sie. Dieser Teil entscheidet, worum es sich handelt und wie damit weiter umgegangen werden soll. Wenn es sich um einen Befehl handelt, sucht convert.c diese Befehle der Reihe nach in commands.c, in direkt.cfg und in ignore.cfg gesucht. Falls der Befehl dort gefunden wird, erfolgt die Bearbeitung entsprechend, ansonsten gibt das Programm eine Warnung aus und ignoriert den Befehl.

commands.c: In diesem Teil stehen die implementierten Befehle aufgelistet. Ihnen zugeordnet ist ein Funktionsaufruf mit eventuell dazugehörigen Parameter. Falls ein Befehl hier gefunden wird, erfolgt der dazugehörige Funktionsaufruf und das Programm kann den Befehl konvertieren.

stack.c: Dieses Modul steuert den Rekursionslevel und den Bracelevel. Der Rekursionslevel stellt die Anzahl der rekursiven Aufrufe der Funktion Convert dar. Der Bracelevel ist die Anzahl der geschwungenen Klammern, wobei die öffnenden inkrementieren und die schließenden dekrementieren. Hier wird die Verschachtelung („nest“-Level) der Konvertierung gesteuert. Die richtige Steuerung von Rekursionslevel und Bracelevel sind wichtig, da sonst der Konverter seine Arbeit vorzeitig abbricht, oder er aber am Ende des Dokuments anlangt, das er aber nicht richtig erkennt, weil der Barcelevel noch auf einem höheren Wert steht.

parser.c: Das programm liest alle Zeichen der Quelldatei(en) über Funktionen dieses Teils ein. Bereits mit und in diesen Funktionen wird eine gewisse Aufbereitung der Zeichen vorgenommen. Auch die Handhabung von verschieden Quelldatei(en) erfolgt hier. Hier ist die entsprechenden Aufbereitung der Daten von großer Bedeutung. Nur gut vorbereitete Daten können auch gut verarbeitet werden.

***language*.cfg:** Diese Dateien enthalten die Übersetzung der von LaTeX feststehenden Abschnittsbezeichnungen in der entsprechenden Sprache. Dies bedeutet, daß z. B. „CONTENTS“ in der deutschen Sprache als „Inhaltsverzeichnis“ bezeichnet wird.

direct.cfg: Hier stehen Befehle, die bei der Konvertierung durch eine einfachen Text ersetzt werden. Diese Datei kann vom Anwender, falls dieser die RTF-Entsprechung für einen Befehl weiß, erweitert werden.

ignore.cfg: Alle Befehle, LaTeX-Variable und benutzerdefinierte Variable, die in dieser Datei stehen, werden auf eine definierte Weise ignoriert. Dies ist wichtig, da damit viele Fehlerquellen eliminiert werden können. Es müssen bei einem zu ignorierenden Befehl auch der bzw. die dazugehörigen Parameter ignoriert werden. Auch hier kann der Anwender Befehle hinzufügen, die der Konverter ignorieren soll. Dies ist in manchen Fällen für einen fehlerfreien Lauf des Konverters sinnvoll.

fonts.cfg: In dieser Datei wird festgelegt, wie die Schriftarten konvertiert werden. Hier ist jeder LaTeX-Schriftart eine RTF-Schriftart zugordnet.

5.3 Verbesserungen

Es folgen nun die Beschreibungen der Verbesserungen. Diese wurden hauptsächlich im Bereich der Vorverarbeitung des Quelltextes erstellt. Es handelt sich dabei großteils um neu erstellte Funktionen. Sollte dies nicht der Fall sein, so weist die Beschreibung darauf hin, daß auf eine bestehende Funktion aufgebaut wurde.

5.3.1 „invalid pointer“

Beim Konvertieren einer LaTeX-Datei gab es oft eine „invalid pointer“-Fehlermeldung. Dieser Fehler trat immer beim Aufruf der Funktion free(langfn) in der Funktion ReadLanguage() im Programmodul cfg.c auf. Dieser Fehler führte des öfteren zum Absturz des Konverters.

Der Verursacher dieser Fehlfunktion war eine unrichtige Speicherplatzreservierung. Dies trat als Folge der Nichtberücksichtigung des Null-Bytes auf. Durch die Erhöhung des zu reservierenden Speicherplatzes um eins konnte der Fehler behoben werden.

5.3.2 Der include-Befehl mit Fehlern

Mehrere LaTeX-Quelldateien lassen sich bequem durch die Verwendung des `\include`-Befehls zu einem Dokument zusammenfassen. (Es ist jedoch zu beachten, daß sich solche `\include`-Befehle nicht verschachteln lassen. Für solche Fälle gibt es den Befehl `\input`, der eine ähnliche Wirkung besitzt.) Dadurch können Dokumente teilweise neu formatiert werden, indem als Argument für den den Befehl `\includeonly` nur die Name der Dateien eingesetzt werden, die LaTeX neu formatieren soll. Bei allen anderen mit `\include`-Befehlen geladenen Dateien liest LaTeX nur die aktuellen Werte der Zähler (page, chapter, table, figure, equation ...) aus den entsprechenden `.aux`-Dateien ein, sofern diese bereits in einem früheren Lauf erzeugt wurden.

Wenn die Informationen in den `.aux`-Dateien aktuell sind, ist es somit möglich, nur einen Teil eines Dokuments zu formatieren und trotzdem im neu formatierten Teil die korrekten Zähler , Querverweise und Seitennummern zu erhalten. Wenn sich jedoch ein Zählerregister oder die Position eines `\label`-Befehls im neu formatierten Teil ändert, kann es sein, daß das gesamte Dokument erneut formatiert werden muß, um fortlaufende und korrekte Indexregister, Verzeichnisse und Literaturverweise zu erhalten. [2]

Die Implementierung des `\include`-Befehls funktioniert dadurch, daß in der Funktion CmdInclude der Zeiger auf die momentane Quelltextdatei gesichert wird. Weiters speichert die Funktion noch die Zeilennummer und die Bezeichnung der aufrufenden Datei. Anschließend wird der Dateizeiger, den das Programm für das Lesen des Quelltextes standardmäßig verwendet, auf die mit `\include` eingebundene Datei abgeändert. Nun wird die Funktion Convert aufgerufen, die mit der Konvertierung beginnt. Der Konverter liest weiterhin über seine Standardzeiger ein, der aber jetzt auf die eingebundene Datei zeigt. Ist diese Datei zu Ende, wird auch die Funktion Convert in CmdInclude beendet. Danach setzt das Programm den Dateizeiger wieder auf die vorherige Datei zurück. Ebenso wird mit den gesicherten Werten der Zeilennummer und der Bezeichnung verfahren. Damit kann der Konverter seine Arbeit in dieser Datei wieder fortsetzen.

Im Konverter war der Befehl `\include` zwar implementiert, aber es lag ein Problem vor. Jedesmal wenn in einer Datei, die mit `\include` eingebunden wurde, eine Formel zu konvertieren war, beendete der Konverter seine Arbeit am Ende der Formel.

Das Problem bestand darin, daß latex2rtf den „nest“-Level, siehe Abschnitt 5.2, nach der Verarbeitung einer Formel um eine geschlossene geschwungene Klammer zuviel dekementierte. Dies bedeutet für den Konverter, daß er einen Funktionsaufruf von convert schließt. Daher war auch der Befehl `\include` bereits für den Konverter erledigt, und er sprang wieder auf die Datei zurück, in der die Einbindung der Datei aufgerufen worden war.

Um den Fehler des „nest“-Levels zu beheben, wurde eine zusätzliche Erhöhung und eine Erniedrigung in der Implementierung des Befehls `\include` eingeführt.

Zu diesem Zweck wurde im Modul funct1.c in der Funktion CmdInclude vor der Anweisung Convert() ein PushBrace() und nach der Anweisung Convert() ein PopBrace() eingefügt. Ein PushBrace() erhöht den „nest“-Level und ein PopBrace() erniedrigt, vereinfacht gesagt, den „nest“-Level.

Der zweite Fehler, der in dieser Funktion auftrat war ein „invalid pointer“. Dieser stammte von der Anweisung free(fname). Da fname kein Pointer ist, für den ein Speicherplatz allokiert wurde, trat beim Aufruf ein Fehler auf.

Nach der Beseitigung diese Aufrufs lief der Befehl `\include` einwandfrei.

5.3.3 Definitionen

Der Befehl `\def` stammt noch aus TEX. Er wird aber auch von LATEX unterstützt und dient dazu, die Arbeit zu erleichtern. So kann mit diesem Befehl eine Folge von Zeichen, die im Text immer wieder benötigt wird, als Kurzbefehl abgespeichert werden. Diese Art der Arbeitserleichterung bietet nebenbei auch eine gewisse Fehlersicherheit an, da einmal definierte Zeichenfolgen immer richtig eingesetzt werden.

LATEX expandiert die neu geschaffenen Befehle, bis nichts mehr zu expandieren ist, d. h. nur mehr Primitive übrigbleiben. Neben dem Begriff „Definition“ hat sich auch der Begriff „Makro“ eingebürgert, da quasi eine mikroskopisch kleine Anweisung einen makroskopischen Effekt haben kann.

Eine Definitition hat immer den folgenden Aufbau:

```
\def\<name>{Ersatzname}
```

Sie wird immer mit der Kontrollsequenz `\def` eingeleitet, der dann der Name der neu zu definierenden Kontrollsequenz, inklusive Backslash, folgt. Zum Schluß kommt der Text, der aus der Kontrollsequenz expandiert wird, in geschwungenen Klammern. Die geschwungenen Klammern haben hier nicht die Wirkung einer Gruppierungsklammer.[6]

Bis zu diesem Zeitpunkt ignorierte das Programm den Befehl `\def`. Um dies zu ändern, wurde zwei neue Funktion in das Programm eingefügt. Die erste Funktion CmdDef liest die Definitionen ein und speichert diese. Definitionen haben für den Konverter das Aussehen eines Befehls. Daher wurde die zweite Funktion so programmiert, daß der Konverter, falls er einen Befehl nicht findet, in den Definitionen nachsieht, ob dieser Befehl einer Definition entspricht und wenn dies der Fall ist, daß er den Ersatztext der Definition konvertiert.

CmdDef: Diese Funktion wurde, um nur den notwendigen Speicherplatz zu belegen, mit dynamischen Zeigerarrays programmiert. Dafür definiert die Funktion die Struktur deftag, die aus zwei Zeiger auf char besteht. Der erste Zeiger *defshort zeigt auf den Speicherplatz, der die für die Kurzform der Definition zur Verfügung steht. *deflong zeigt auf den Speicherplatz, der den Ersatztext aufnimmt. Dazu wurde der Zeiger *Definition geschaffen, der auf die Struktur zeigt. So ist es möglich, erst wenn eine Definition auftritt, den dafür notwendigen Speicherplatz zu allokieren.

```
typedef struct deftag {
    char *defshort;
    char *deflong;
    } deftag;
static deftag *Definition=NULL;
```

Die Definition diese Struktur steht am Begin des Programms funct1.c. Ebenso wird die Variable DefNumber an dieser Stelle im Programm definiert und mit null initialisiert. Die Variable DefNumber enthält immer die Anzahl der bereits abgespeicherten Definitionen.

Nachdem die Funktion CmdDef gestartet ist, wird bis zum ersten Zeichen, das kein Leerzeichen ist, eingelesen. Grundsätzlich sollten keine Leerzeichen an dieser Stelle stehen, aber um auch solche Fehler meistern zu können, führt das Programm dies durch. Da eine Definition mit einem Backslash beginnt, steht am Anfang eine Abfrage, die feststellt, ob diese Bedingung auch erfüllt ist. Falls dies nicht der Fall ist, endet die Funktion, und eine Fehlermeldung wird ausgegeben.

Im Anschluß daran nimmt die Variable buffer die Kurzform der Definition auf. Die Kurzform endet mit einem Leerzeichen oder mit einer öffnenden geschwungenen Klammer. Daraufhin wird ein Null-Byte buffer angehängt. Die Funktion ignoriert alle Leerzeichen, die bis zur öffnenden geschwungenen Klammer auftreten. Diese Klammer muß wieder zurückgeschrieben werden, um die bereits vorhandene Funktion getParam zum Einlesen des Ersatztextes verwenden zu können. Der Rückgabewert, der von der Funktion getParam geliefert wird, ist der gesamte Ausdruck zwischen den geschwungenen Klammern, und die Variable temp nimmt diesen Wert auf.

Anschließend versucht die Funktion, den benötigten Speicherplatz zu allokieren, und bei Erfolg speichert sie die Kurzform und die Langform der Definition ab und erhöht die Variable DefNumber, die die Anzahl der bereits gespeicherten Definitionen enthält.

Diese Funktion berücksichtigt nicht, daß Definitionen mit einem neuen Befehl `\def` und dem gleichen Kurztext überschrieben werden können. LaTeX nimmt in so einem Fall immer die letzte Definition. Dieses Problem löst die Funktion TryDefConvert, idem sie die

neuen Definitionen ebenfalls abspeichert, und anschließend von unten zu suchen beginnt, und und somit die letzte Definition verwendet.

Um diese Funktion auch verwenden zu können, mußte die Zuweisung des `\def`-Befehls geändert werden. Im Programm commands.c bekommen die Befehle die dazugehörigen Funktionen zugewiesen. Die Syntax lautet

```
{"Befehl", Funktion, Parameteranzahl}.
```

Hier mußte dem Befehl `\def` die Funktion CmdDef zugeordnet werden. Die Parameteranzahl ist null, da beim Funktionsaufruf keine Parameter übergeben werden.

TryDefConvert: Diese Funktion wurde so programmiert, daß sie gleich den anderen Aufrufen zum Suchen von Befehlen in convert.c verwendet werden kann. Der Aufruf erfolgt mit der Definition als Parameter. Die Funktion definiert die Variable i und weist dieser den Wert DefNumber zu.

Nun suchte eine Schleife nach der Definition. Diese Schleife endet, wenn entweder die Definition gefunden wird, oder die Variable i den Wert null erreicht. Innerhalb dieser Schleife vergleicht die Funktion die gesuchte Definition mit der unter dem Index i abgespeicherten Definition. Stimmen diese beiden Definition überein wird die Langform der Definition mit dem Befehl ConvertString aufgerufen und konvertiert. Nach dieser Konvertierung endet die Funktion mit dem Rückgabewert TRUE. Stimmen die beiden Definitionen beim Vergleich nicht überein, wird die Variable i um eins dekrementiert. Erreicht i nun den Wert null, so endet die Schleife. Die Funktion schließt mit dem Rückgabewert FALSE.

Mit diesem Rückgabewert erkennt der Konverter, ob der von ihm, an die Funktion TryDefConvert zur Verarbeitung übergebene LaTeX-Befehl eine Definition ist, die diese Funktion auch konvertiert hat.

Der Funktionsaufruf von TryDefConvert steht im Programm convert.c. Zuerst wird der gesuchte Befehl mit den Befehlen verglichen, die in den Konfigurationsdateien direkt.cfg und ignore.cfg gespeichert sind. Anschließend wird der Befehl durch den Aufruf der Funktion TryDefConvert mit den Definitionen verglichen. Sollte auch diese Funktion kein Ergebnis bringen, kann dieser Befehl nicht konvertiert werden.

5.3.4 getTexUntil

Die in parser.c definitierte Funktion getTexUntil liest den Quelltext bis zum Beginn von „Ziel“ ein. Diese Funktion benötigt der Konverter, um logisch zusammenhängende

Programmsequenzen vor der eigentlichen Konvertierung einlesen zu können. Dies ist notwendig, um die Textteile in der richtigen Umgebung zu konvertieren.

Die Suche mit getTexUntil nach dem „Ziel“ war bisher nur dann erfolgreich, wenn der gesuchten Ausdruck nicht in Form einer Definition vorlag. Gab es eine Definition, suchte die Funktion darüber hinweg bis zur ersten Verwendung des Ausdrucks und nicht der Definition. Dies führte meist dazu, daß das „Ziel“ nicht gefunden wurde, und die Suche am Ende der Quelldatei endete und der Konverter somit Probleme hatte, seine Arbeit richtig auszuführen.

getDefinition: Als erstes mußte eine Funktion geschrieben werden, die die Definition für das „Ziel“ sucht und an die Funktion getTexUntil übergibt. Dieser Funktion erhält als Parameter den zu suchende Ausdruck. Es werden zwei Variable definiert. Erstens die Variable help, die die Langform der Definition aufnehmen kann, und zweitens eine Zählervariable i. Dieser Variablen i wird der Wert der globalen Variablen iDefinitionCount, die die Anzahl der Definitionen enthält, zugewiesen.

Nun startet eine Schleife, die solange läuft, bis die Variable i den Wert null erreicht hat oder der gesuchte Ausdruck mit einer Definition übereinstimmt. Am Ende jedes Schleifendurchlaufs erniedrigt sich der Wert der Variable i um eins.

Nach Beendigung der Schleife untersucht die Funktion, ob der Wert von i kleiner null ist, denn in diesem Fall wurde keine Definition gefunden und die Funktion gibt den Wert NULL zurück. Ist der Wert von i nicht kleiner null, so werden ein Backslash und ein Null-Byte am Anfang in die Variable help eingefügt. Diese zwei Zeichen sind deshalb notwendig, weil die Langformen der Definitionen ohne den normalerweise am Anfang stehenden Backslash abgespeichert sind und die Verwendung des C-Befehls strcat das Null-Byte benötigt. Dieser Befehl hängt nun die Langform der Definition an den Backslash an. Der Rückgabewert der Funktion ist ein Zeiger auf Variable help, die die Definition samt Backslash enthält.

getTexUntil: Diese Funktion bekommt beim Aufruf zwei Parameter übergeben, und zwar als ersten den zu suchenden Ausdruck (geht als Variable target in die Funktion ein) und als zweiten einen Parameter, der steuert, welche Funktion, getTexChar oder getRawTexChar, die Zeichen einliest. Von der einlesenden Funktion hängt die Manipulation der Zeichen vor der Verwendung in getTexUntil ab.

Die Funktion beginnt mit der Abfrage, ob der gesuchte Ausdruck ein Befehl ist. Wenn dies der Fall ist, so folgt der Aufruf von getDefinition, um zu prüfen, ob eine Definition für diesen Befehl existiert. Trifft dies zu, so gibt getDefinition diesen Wert zurück und die

Variable target2 bekommt ihn zugewiesen. Existiert keine Definition, so hat die Variable target2 den Wert Null. Mit der gleichen Abfrage wird auch der Wert der Variablen g_parser_search_command definiert. Diese Variable benötigt die Funktion getTexChar in parser.c, um die Kommentare, die unter LaTeX mit einem Prozentzeichen beginnen, richtig zu verarbeiten. Der anschließende Aufruf der Funktion SetScanAhead(TRUE) verhindert, daß sich beim Suchen des Ausdrucks die Zeilennummer erhöht. Diese Erhöhung der Zeilennummer darf nur bei der Konvertierung und nicht beim vorherigen Einlesen einer Quelltextsequenz erfolgen.

Nun startet eine Schleife. Der Durchlauf dieser Schleife endet, wenn der letzte Index die Größe der Variablen buffer übersteigt, oder der Ausdruck bzw. seine Definition gefunden ist. Die erste Anweisung innerhalb der Schleife liest ein Zeichen aus der Quelltextdatei ein und weist dieses Zeichen den Speicherplatz mit dem Index i in der Variablen buffer zu. Der zweite Parameter, den die Funktion getTexUntil erhält, definiert den Grad der Vorverarbeitung der Rohdaten. Beim Erreichen des Endes der Quelltextdatei ist der Wert von buffer[i] null.

Der nächste Schritt ist die Abfrage, ob buffer[i] den Wert null hat. Ist dies der Fall, so wird die Variable end_of_file_reached gleich TRUE gesetzt und break bricht die Schleife ab.

Anschließend folgt der Vergleich der eingelesenen Zeichen mit dem gesuchten Ausdruck. Dieser Vergleich ist für die Variable target als Ablaufdiagramm (siehe Abb. 5.3) und als Programmsequenz (siehe Abb. 5.1) dargestellt. Beide Darstellungen beginnen bei der gleichen Anweisung. Die Variablen j, l, Diffl und Diffj bekommen am Beginn der Funktion den Wert null zugewiesen. Die erste Entscheidung besteht darin, ob das aktuelle Zeichen der Variablen buffer mit dem aktuellen Zeichen von target übereinstimmt. Ist dies der Fall, so muß nun unterschieden werden, ob es das erste Zeichen (j==0) ist, das übereinstimmt, und ob bei der Variablen target2 noch keine Übereinstimmung, die vor diesem Zeichen begonnen hat, (l==0) vorliegt. Trifft dieser Fall zu, so wird j um eins inkrementiert und mit der Funktion SaveFilePosition , siehe Abbildung 5.2, die Position des Zeigers, der gerade auf das nächste noch nicht eingelesene Zeichen der Quelltextdatei zeigt, gespeichert.

Sollte der Fall eintreten, daß bereits vor diesem Zeichen eine zusammenhängende Übereinstimmung mit target2 begonnen hat ($l \neq 0$), so wird die Varialbe j um eins inkrementiert. Die Anzahl der Zeichen, um die die Übereinstimmung mit target2 früher begonnen hat, wird in der Variablen Diffj , um den Wert 1 korrigiert, gespeichert.

```
if (buffer[i]==target[j]) {
    if (l==0 && j==0) {
        SaveFilePosition();
        j++;
    } else if (j==0) {
        j++;
        Diffj=l-1;
    } else {
        if (j<(strlen(target)-1)) j++;
        else {
            target_found=TRUE;
            target_true=TRUE;
        }
    }
} else {
    if (j>0) {
        if (l==0) {
            RestoreFilePosition(0);
            i=i-j-Diffl;
            j=0;
            Diffj=0;
        } else {
            j=0;
            Diffj=0;
        }
    }
}
```

Abbildung 5.1: Programmsequenz des Vergleichs

Hat die Übereinstimmung mit target bereits vor dem aktuellen Zeichen begonnen ($j \neq 0$), so wird verglichen, ob die Anzahl, der in Folge bereits gleichen Zeichen kleiner ist als die Länge von target. Ist die Bedingung erfüllt, so wird j inkrementiert, anderenfalls ist der gesuchte Ausdruck target gefunden, und die beiden boolschen Variablen target_found und target_true werden TRUE gesetzt.

Wenn das eingelesene Zeichen nicht mit dem durch den Index j angezeigten Wert von target übereinstimmt, muß untersucht werden, ob der Index größer null ist. Ist diese Bedingung nicht erfüllt, wird die Fallunterscheidung für target somit beendet. Anderen-

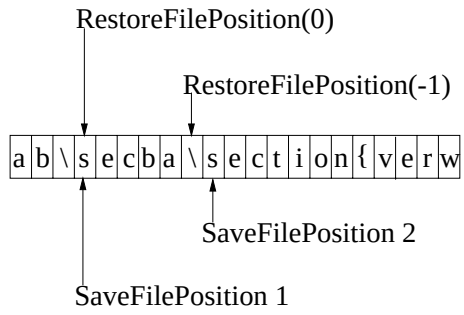


Abbildung 5.2: SaveFilePosition und RestoreFilePosition

falls gibt es eine eine weitere Fallunterscheidung. Diese bezieht sich auf den Index l von target2. Ist diese Variable l null, so wird der mit SaveFilePosition gespeicherte Wert wieder zum aktuellen Zeiger auf das nächste zu lesende Zeichen im Quelltext. Der Index i der Variablen buffer muß um den Wert (j+Diffl) zurückgesetzt werden, um den auch der Zeiger auf die Zeichen im Quelltext rückgesetzt wurde. Anschließend bekommen die Variablen j und Diffl den Wert null zugewiesen. Wenn die Variable l nicht gleich null ist, so folgt nur die Nullsetzung der Variablen j und Diffl.

Nach der Fallunterscheidung für target folgt die Abfrage, ob target2 den Wert NULL besitzt. Dies ist der Fall, wenn keine Definition für target existiert. Ist target2 NULL, so wird die nächste Programmsequenz, die im nächsten Absatz kurz angesprochen wird, übersprungen.

Die Variable target2 ist ungleich NULL. Nun sind die gleichen Fallunterscheidungen wie für target zu treffen, nur daß die Variable j mit l, Diffl mit Diffj und target mit target2 die Plätze tauschen. Die Zuweisung der boolschen Variable target_true=TRUE wurde in diesen Programmblock entfernt. Ansonsten hat diese Programmsequenz die gleichen Bedingungen wie der in Abb. 5.3 und Abb. 5.1.

Die letzte Anweisung, die innerhalb der Schleife steht, ist die Inkrementierung der Variablen i um eins.

Nach dem Schleifenende wird abgefragt, ob die Schleife endete, weil der gesamte Speicherplatz der Variablen buffer aufgebraucht ist. Trifft dies zu, so folgt die Ausgabe eines Fehlers und der Abbruch des Konverters.

Die nächste Entscheidung im Programm ist die Abfrage, ob das Ende der Schleifen nicht durch das Ende der Quelldatei eintrat. Ist diese Bedingung wahr, so muß die Schleife durch das Auffinden von target oder target2 beendet worden sein. Nun wird die Länge des in Frage kommenden Ausdrucks berechnet. In die Variable buffer wird beim Index i minus die eben berechnete Länge ein Null-Byte eingefügt, und der Zeiger im Quelltext wird auf das erste Zeichen des gefundenen Ausdrucks rückgespeichert, denn dort muß

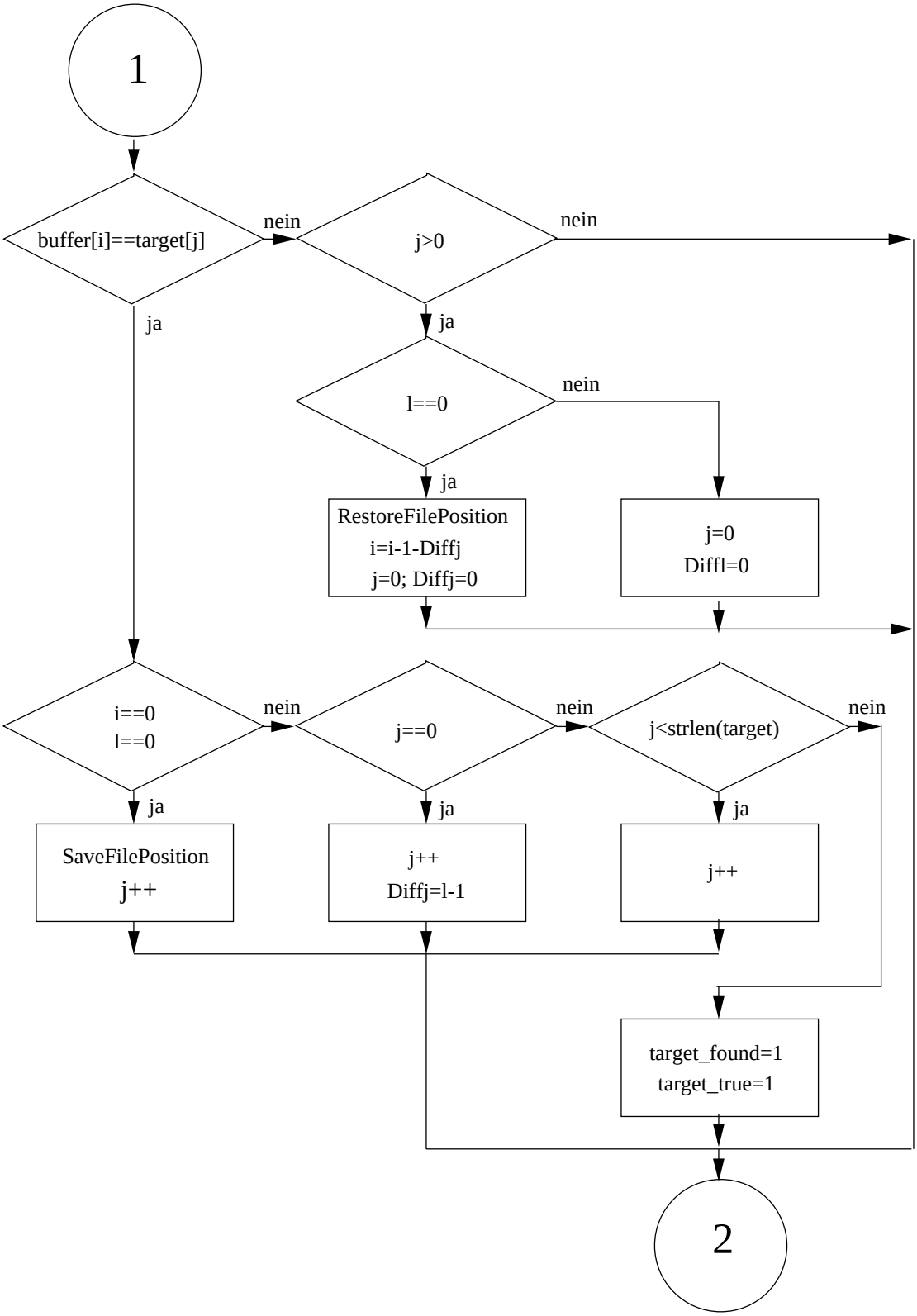


Abbildung 5.3: Entscheidungsablauf des Vergleichs

der Konverter seine Arbeit nach der Abarbeitung der eben eingelesen Quelltextsequenz wieder fortsetzten.

Der folgende Aufruf der Funktion SetScanAhead(FALSE) startet die Zeilenzählung wieder, die am Anfang der Funktion ausgeschaltet wurde. Auch bekommt die Variable g_parser_search_command wieder den Wert FALSE. Die Funktion strdup weist der Variablen buffer einen Speicherplatz zu und gibt einen Zeiger darauf zurück. Dieser Zeiger ist der Rückgabewert.

5.3.5 getSection

Ein Dokument ist in verschiedene Abschnitte gegliedert. Die Abschnitte gehören logisch zusammen. Diese Gliederung ist das Grundgerüst des Dokuments und ist für die richtige Strukturierung verantwortlich. Von dieser Struktur hängt auch die Nummerierung ab. Diese wiederum ist für die Erstellung von Überschriften, Inhaltsverzeichnissen, Referenzen, Abbildungsverzeichnissen usw. wichtig.

LaTeX stellt mehrere Befehle zur Gliederung des Dokuments zur Verfügung. Aufgabe der Funktion getSection ist es, diese Gliederungsbefehle zu finden, und den zusammengehörenden Text zwischen diesen Gliederungsbefehlen als Block einzulesen, und diesen der Konvertierung zu übergeben.

Diese Funktion existierte in einer Form, die diese Gliederungsbefehle in der Normalform fand. Sehr viele Anwender schreiben aber für oft verwendete Befehle eine Definition, mit der sie die Befehle durch eine kurze Buchstabensequenz ersetzten können. getSection verarbeitete diese Definitionen nicht. Daraus folgt eine Konvertierung mit Fehlern.

Um diese Konvertierungsfehler zu beseitigen, wurde die Funktion getSection erweitert, sodaß sie auch die Definitionen handhaben kann. Um die Normalform von der Definition zu erhalten, mußte die Funktion expandmacro in abgeänderter Form unter dem neuen Namen getSectionexpandDef abgespeichert werden.

getSectionexpandDef: Die Funktion expandmacro dient zum Verarbeiten von Definitionen. Die Definition wird expandmacro übergeben, und von dieser in der Liste der gespeicherten Definitionen gesucht und konvertiert. Für die Verwendung in der Funktion getSection, die die Texte voreinliest, ist expandmacro nicht brauchbar, da es die Definitionen sofort konvertiert.

getSectionexpandDef ist auf expandmacro aufgebaut. Die Befehlssquenzen, die die Konvertierung bewirken wurden gestrichen.

Das Einlesen der Definitionen erfolgt auf gleiche Weise. Anstatt die Entsprechung der Definition zu konvertieren, wird sie auf einem Speicherplatz abgelegt. Die Funktion getSectionexpandDef hat den Zeiger auf diesen Speicherplatz als Rückgabewert.

Diese Funktion kann nun von getSection aufgerufen werden, und liefert somit die Langform der Definition zurück.

getSection: Ausgehend von der bestehenden Funktion getSection, erfolgt die Erweiterung zur Verwendung mit Definitionen.

Die Funktion besitzt drei Adreßparameter, über die die Rückgabe des eingelesenen Textteils, die aktuelle Abschnittsüberschrift und einen eventuellen gefundenen Label des Abschnitts erfolgt.

Die wichtigsten Definitionen die am Beginn der Funktion stehen, sind die Definitionen von zwei Felder. Im ersten Feld, mit den Namen command, steht eine Anzahl von Befehlen, die den Anfang von Abschnitten und Umgebungen darstellen und aus wenigen Befehlen, die einer eigenen Behandlung bedürfen. Es ist so aufgebaut, daß die Befehle, die eine eigene Umgebung starten, am Anfang der Liste stehen. Die korrespondierenden Befehle, die die Umgebungen wieder schließen, stehen im zweiten Feld (ecommand). Die Anzahl der Elemente sind in den Variablen ncommands und ecommands gespeichert.

Die erste Anweisung nach den Definitionen ist die Reservierung eines Speicherplatzes für die Varialble section_buffer, die den Text aufnehmen soll. Nach dieser Reservierung setzt die Funktion die Zeiger der Parameter auf NULL. Es wird SetScanAhead(TRUE) aufgerufen, um die Zeilenzählung auszuschalten, da es sich um ein Einlesen vor der Konvertierung handelt.

Die Schleife, die jetzt gestartet wird, erstreckt sich beinahe über den gesamten Rest der Funktion und hat keine Abbruchbedingung vorgegeben. Nach jedem Schleifendurchlauf wird die Index delta, der auf den aktuellen Speicherplatz in der Variablen section_buffer zeigt, um eins inkrementiert.

Die erste Anweisung, die in der Schleife steht prüft, ob der Index delta um mindestens zwei kleiner ist als der Wert, der die Größe des reservierten Speicherplatzes enthält. Ist dies nicht der Fall, so wird die Funktion increase_buffer_size aufgerufen, die den reservierten Speicherplatz, und die Variable die die Größe des Speicherplatzes enthält, verdoppelt. Diese Anweisung steht jedesmal, bevor die Funktion ein Zeichen in die Variable section_buffer schreibt, und findet aus diesem Grund in der restlichen Beschreibung keine Erwähnung mehr.

Nun wird ein Zeichen aus der Quelltextdatei eingelesen und in die Variable section_buffer mit dem Index delta geschrieben. Ist dieses Zeichen ein Null-Byte, so bricht die

Schleife ab. Sollte das Zeichen ein Prozentzeichen sein, das nicht hinter einem Backslash steht, und die Darstellung des Prozentzeichens angibt, bedeutet dies, daß der Rest der Zeile im Quelltext ein Kommentar ist. Diesen Rest liest die Funktion Zeichen für Zeichen ungeprüft bis zum Ende der Zeile ein und schreibt sie, unter Verwendung von Index delta, in die Variable section_buffer. Das letzte Zeichen, das in section_buffer geschrieben wird ist die Zeilenschaltung.

Ist die nächste Abfrage, daß das letzte Zeichen in section_buffer ein Backslash ist, erfüllt, so wird eine Variable bs_count gesetzt. Diese Variable wird benützt, um festzustellen, ob zwei Backslashes hintereinanderstehen, und um in der oben erwähnten Abfrage zwischen der Darstellung eines Prozentzeichens und einem Kommentar zu unterscheiden. Diese nächste Abfrage unterscheidet, ob es sich um eine ungerade Anzahl von unmittelbar aufeinanderfolgende Backslashes handelt. Trifft dies zu, so starten zwei Initialisierungsschleifen. Die erste Schleife setzt die Werte eines Feldes von Integer (match) auf TRUE. Dieses Feld hat ncommands Elemente. Jedes Element gehört zu einem Befehl aus dem Feld command, welcher den gleichen Index besitzt. Die zweite macht das selbe für ein Feld von Integer (match_def), das genau so viele Elemente hat, als Definitionen existieren (iDefinitionCount). Der Index index wird, für den gefundenen Backslash, der das erste gesuchte Zeichen ist, auf eins gesetzt. Mit dem Befehl continue wird der nächste Schleifendurchgang gestartet. Sollte die Bedingung einer ungeraden Anzahl von Backslashes nicht erfüllt sein, so wird bs_count null gesetzt, und zur nächsten Anweisung weitergegangen.

Falls index gleich null sein sollte, wird der nächste Schleifendurchgang gestartet. Die Variable index gibt die Zahl an, bis zum wievielten Zeichen mindestens ein Befehl oder eine Definition mit der letzten Befehlssequenz, beginnend mit einem Backslash, übereinstimmt.

Die Variablen possible_match, possible_match_def, found und found_def werden FALSE gesetzt. Es beginnt eine Schleife (siehe Abb. 5.4 und 5.5) zu laufen, die genau einmal für jeden Befehl im Feld command durchlaufen wird. Die Schleife wird durch den Index i gesteuert, der auch die einzelnen Elemente im Feld referenziert. Die Variable match[i] ist solange wahr, solange jedes Zeichen seit dem letzten Backslash mit dem Befehl command[i] übereinstimmt. Ist match[i] FALSE, so wird die Schleife mit dem nächsten Index i gestartet. Anderenfalls wird das aktuelle Zeichen mit der aktuellen Stelle im Befehl command[i] verglichen. Sind diese beiden verschieden, so wird match[i] FALSE gesetzt. D. h. dieser Befehl weicht von der Befehlssequenz im Quelltext ab. Mit continue wird der nächste Schleifendurchlauf gestartet. Beim ersten Befehl, der auch bei diesem Zeichen noch mit der Sequenz aus dem Quelltext übereinstimmt, wird possible_match auf TRUE gesetzt. Dies bedeutet, daß bis zum Index index zumindest bei einem Befehl noch alle Zeichen übereinstimmen.

```
for(i=0; i<ncommands; i++) {
    if (!match[i]) continue;
    if (*(section_buffer+delta)!=command[i][index]) {
        match[i] = FALSE;
        continue;
    }
    possible_match=TRUE;
}
```

Abbildung 5.4: Schleife für den Vergleich der einzelnen Zeichen auf der jeweiligen Position

Die gleiche Schleife wird auch für die Definitionen durchlaufen, nur werden hier die für die Definitionen korrespondierenden Variablen verwendet (iDefinitionCount, match_def, possible_match_def und statt command Definitions.name) Auch die Position, die die Variablen index referenziert, muß um eins reduziert werden, da die Definitionen ohne Backslash gespeichert werden.

Vor den folgenden beiden Abfragen wird das nächste Zeichen eingelesen, in der Variablen cNext gespeichert und wieder in die Quelltextdatei zurückgeschrieben. Dieses Zeichen wird für die nächsten Entscheidungen benötigt.

Die beiden anschließenden Bedingungen sind einander wieder ähnlich. Abbildung 5.6 zeigt die Bedingung für die Definitionen. Es wird wieder die Schleife für alle Definitionen durchlaufen. Ist match_def FALSE, so startet ein neuer Schleifendurchlauf mit dem nächsten Index j. Ist index gleich der Länge der Definition, so wird die Variable cNext getestet, ob sie alphanumerisch ist. Wenn dies nicht erfüllt ist, so ist die eingelesene und verglichene Befehlssequenz mit dem Beginn eines anderen Befehls ident, und es entspricht nicht der Definition, daher startet ein neuer Schleifendurchlauf. Ist cNext nicht alphanumerisch, so wurde eine Definition gefunden. found_def erhält den Wert TRUE, und die Schleife bricht ab.

Für die Befehle im Feld command ist der Ablauf, unter Verwendung der korrespondierenden Variablen fast gleich. Der Unterschied ist, daß sich hier die Frage, ob cNext alphanumerisch ist, nicht stellt. Daher wird cNext nicht benötigt und die Schleife verkürzt. Beim Längenvergleich muß mit index + 1 gearbeitet werden, da bei den Befehlen der Backslash mitgespeichert ist. Die Laufvariable ist jetzt i, weil i und j den gefundenen Befehl bzw. die gefundene Definition referenzieren.

Die nächste Anweisung ist die Abfrage, ob ein Befehl gefunden wurde. In diesem Fall ist found auf TRUE gesetzt. Ist found TRUE, so wird noch die Korrektheit für jene Befehlen verifiziert, die nach der gesuchten Befehlssequenz ein Leerzeichen oder eine

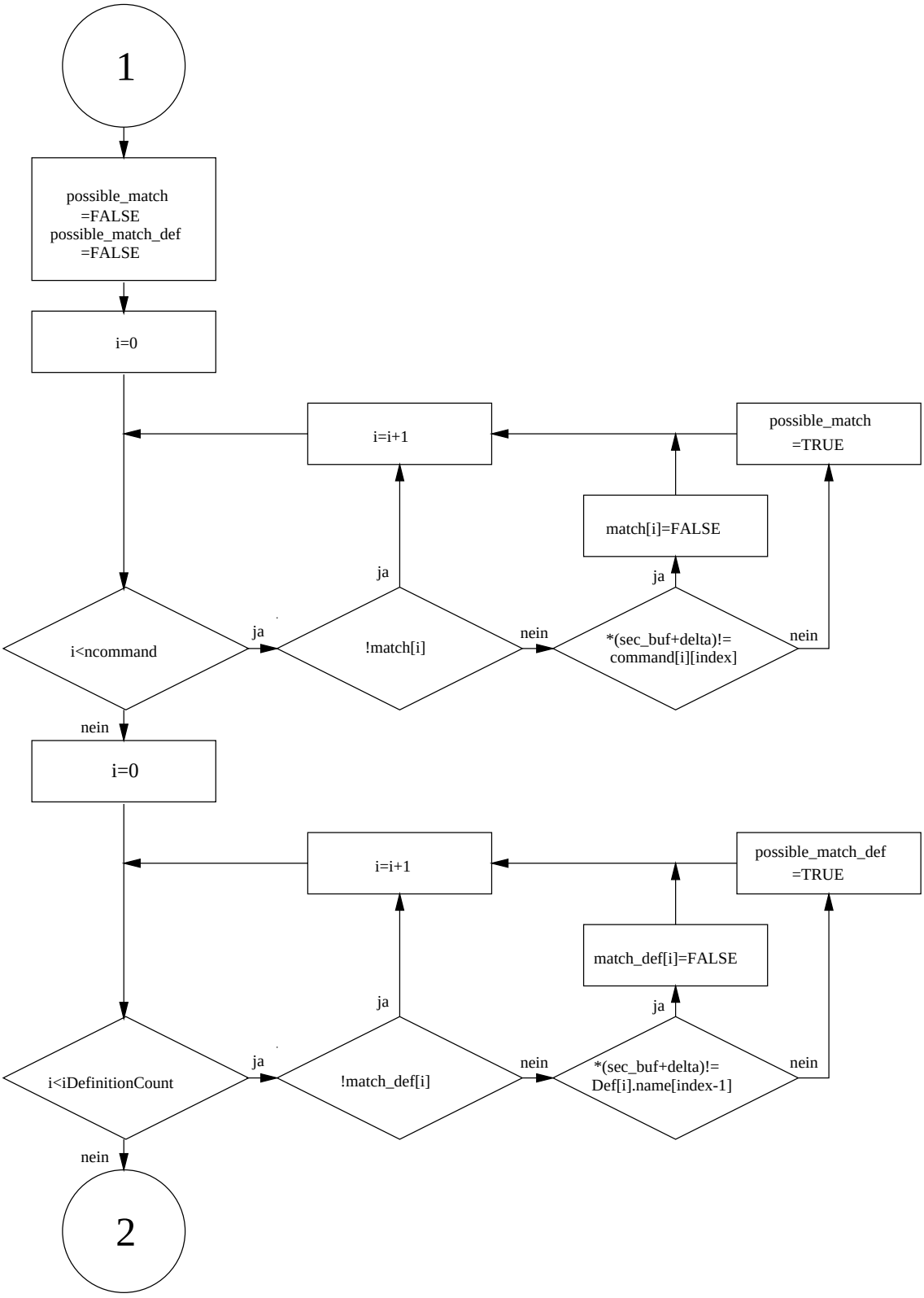


Abbildung 5.5: Vergleich letzten Zeichens in section_buffer mit den aktuellen Zeichen jedes Befehls und jeder Definition

```
for(j=0; j<iDefinitionCount; j++) {
    if (!match_def[j]) continue;
    if (index == strlen(Definitions[j].name)) {
        if (isalnum(cNext)) continue;
        found_def = TRUE;
        break;
    }
}
```

Abbildung 5.6: Schleife ob bereits der gesamte Befehl gleich ist

öffnende geschwungen Klammer haben müssen. Diese Befehle sind im Feld command hintereinander angeordnet, sodaß die, zu durchsuchenden Element mit einem beginnenden und einem beendenden Index begrenzt werden können. Ist die Bedingung des Leerzeichens oder der geschwungenen Klammer nicht erfüllt, so wird match[i] (für den jeweiligen Befehl) und found FALSE gesetzt.

Nun werden die beiden Variablen possible_match und possible_match_def abgefragt. Sind beide FALSE, so stimmt kein Befehl bzw. keine Definition in allen Zeichen mit der, seit dem letzen Backslash gefundenen, Zeichenfolge überein. D. h. die Befehlsequenz entspricht keinem der gesuchten Befehle und keiner der vorhandenen Definitionen. Daher wird der Index index auf 0 gesetzt, damit die neuerliche Suche beim nächstem Backslash wieder starten kann. Danach beginnt ein neuerlicher Schleifendurchgang. Ist eine der beiden abgefragten Variablen aber TRUE, so wird index um eins erhöht, um das nächste Zeichen für die bis jetzt richtige Zeichenfolge vergleichen zu können.

Die nächste Zeile fragt ab, ob noch kein Befehl oder keine Definition korrekt gefunden wurde, d. h. wenn found und found_def FALSE sind. In diesem Fall wird ein neuer Schleifendurchlauf begonnen anderenfalls geht es im Programm weiter, und es folgt die Behandlung des gefundenen Ausdrucks.

Der nächste Block beschäftigt sich mit der Abarbeitung einer gefundenen Definition. Die Eingangsbedingung ist, daß found_def TRUE ist. Am Beginn steht der Aufruf der Funktion getSectionexpandDef. Die Variable defbuffer speichert den Rückgabewert. Anschließend erfolgt die Korrektur von delta um die Länge der Kurzform der gefundenen Definition und der Test, ob die Variable section_buffer groß genug ist, um die extrahierte Form der Definition aufnehmen zu können, ansonsten wird die Funktion increase_buffer_size aufgerufen, um den notwendigen Speicherplatz zur Verfügung zu stellen.

Jetzt beginnt die Suche, ob ein Befehl aus dem Feld command der extrahierten Definition entspricht. Alle Elemente des Feldes match werden TRUE gesetzt. Die Stringlänge

von defbuffer begrenzt die Anzahl der Durchläufe, der nun startenden Schleife. In dieser Schleife steht der Ablauf aus Abbildung 5.4, nur daß nicht mit *(section_buffer+delta) verglichen wird, sondern mit dem gerade indizierten Zeichen von defbuffer. Die Verwendung von possible_match ist nicht mehr notwendig.

Was darauf folgt, entspricht der Abbildung 5.6 für das Feld command, wie oben bereits beschrieben, und daß anstatt index die um eins erhöhte Laufvariable k verwendet wird.

Ist found TRUE, dann entspricht die Definition einem Befehl. Sollte sich dieser innerhalb der bereits oben beschriebenen Sequenz befinden, so starten zwei Abfragen nach dem nächsten Zeichen. Sollte die Definition gleich lang als der gefundene Befehl sein, so wird wie oben das nächste Zeichen eingelesen, zurückgeschrieben und verglichen. Die zweiten Abfrage, die eintritt, wenn der gefundene Befehl kürzer ist als die Definition, untersucht das nächsten Zeichen der Definition. Falls dieses Zeichen nicht ein Leerzeichen oder eine öffnende geschwungene Klammer ist, so setzt die Funktion die Variablen found und match[i] auf FALSE.

Bleibt die Variable found TRUE, so vergleicht die Funktion, ob die Definition länger ist als der gesuchte Befehl. Ist dies der Fall, so schreibt getSection die verbleibenden Zeichen in die Quelldatei zurück. Die Indexvariable delta wird um die Länge von der Definitionskurzform plus eins, für den Backslash, vermindert, und der gefundene Befehl an section_buffer angehängt, wonach delta wieder das letzte Zeichen von section_buffer indiziert. Die letzte Schleife endet mit einem break.

Ist in der Definition kein Befehl enthalten, d. h. fount hat den Wert FALSE, so wird delta um die um eins erhöhte Länge der Definitionskurzform vermindert. Die entsprechende Definition wird an section_buffer angehängt, der Index index auf null gesetzt, und mit einem neuen Schleifendurchlauf das nächste Zeichen zum Vergleich eingelesen.

Startet hier kein neuer Schleifendurchgang, d. h. found ist TRUE, so folgen einige Unterscheidungen, die gefundene Befehle betreffen, die aber keinen neuen Abschnitt beginnen. Diese Befehle müssen hier verarbeitet werden, damit getSection den nächsten Abschnittsbeginn auch richtig finden kann. Nach jeder dieser Abarbeitungen, wird der Index index wieder auf null gesetzt, und ein neuer Schleifendurchlauf gestartet.

Ist der gefundene Befehl aber ein Abschnittsbeginn, so wird der dazugehörige Parameter eingelesen, und der Befehl mit dem Parameter im Speicher abgelegt. Die Variable next_header speichert die Anfangsadresse. Die Übergabevariable header zeigt auf next_header. Die Indexvariable delta wird auf das Zeichen vor die gefundenen Befehlssequenz zurückgesetzt und mit einen Null-Byte abgeschlossen. Daraufhin endet die Schleife, die Zeichen für Zeichen einliest und vergleicht. Es wird section_buffer ein Speicherplatz

zugewiesen und die Variable text zeigt auf den Beginn dieses Speicherplatzes. Die Übergabevariable body zeigt auf text. SetScanAhead(FALSE) schaltet die Zeilenzählung ein und die Funktion getSection endet.

5.3.6 PrepareEQ

Der Konverter arbeitet sequentiell. Dies ist für die Verarbeitung der meisten LaTeX-Befehle ausreichend, da die Parameter nach diesen Befehlen stehen. So hat der Konverter die Möglichkeit, den Befehl einzulesen, die richtige Funktion dazu aufzurufen, und die entsprechenden Parameter aus der Quelltextdatei zu bekommen. Die Syntax der meisten LaTeX-Befehle lautet

\Befehl[Optionen]{Parameter1}...{Parametern}.

Von Befehl zu Befehl ist abhängig, ob Optionen verwenden werden können, und wieviele Paramter der Befehl benötigt. Diese Syntax ermöglicht es, zuerst den Befehl, und erst anschließend die Parameter, bzw. Optionen einzulesen.

LaTeX ist aus TeX entstanden, und daher werden noch einige Befehle von TeX verwendet. Zu dieser Gruppe gehört auch der Befehl `\over`, der nur in einer mathematischen Umgebung verwendet werden kann. Verwendet wird dieser Befehl folgendermaßen:

{Parameter1 \over Parameter2}

Wie bereits aus der Syntax erkennbar ist, wird hier zuerst der Parameter1 eingegeben, und erst anschließend der Befehl. Dieser Befehl bewirkt unter LaTeX verwendet folgendes:

$$\frac{Parameter1}{Parameter2}.$$

Die Entsprechung unter LaTeX ist der Befehl `\frac`. Dieser Befehl hat folgende Syntax und Wirkung:

\frac{Parameter1}{Parameter2}

$$\frac{Parameter1}{Parameter2}$$

Hier ist zu sehen, daß dieser Befehl der Verarbeitung durch den Konverter entgegenkommt, weil zuerst der Befehl geschrieben steht, und erst dann die Parameter im Quelltext vorkommen.

Daher ist es sinnvoll, im Fall des Auftretens einer mathematischen Umgebung, diese zuerst einzulesen, und vorzuverarbeiten. Erst danach erfolgt die Konvertierung.

Naheliegend ist, den Befehl `\over` in den Befehl `\frac` umzuwandeln, da dieser der sequentiellen Konvertierung entgegenkommt.

Um dies zu realisieren, wurden zwei Funktionen, ConvertEQ und PrepareEQString, geschrieben, die den Quelltext innerhalb der mathematischen Umgebung zuerst einlesen und anschließend die Befehle `\over` in die LaTeX-Ensprechung `\frac` umwandeln. Der Aufruf erfolgt in den Funktionen CmdMath und CmdDisplayMath, die den Beginn einer mathematischen Umgebung behandeln.

CmdMath und CmdDisplayMath: Diese beiden bereits bestehenden Funktionen schalten den Konverter in einen mathematischen Modus um. Es gibt zwei verschiedene Modi. Der erste ist für Formeln im Text und der zweite startet die abgesetzten Formeln. In diese Funktionen wird nach dem Umschalten des Modus die Funktion ConvertEQ(Parameter) aufgerufen. Der Parameter hängt von der Zeichenfolge ab, mit der die Umgebung gestartet wird. Dies ist wichtig, um das Ende der Formel im Quelltext auch richtig zu erkennen.

Die einzigen Änderungen in diesen Funktionen sind die Einfügungen, die ConvertEQ aufrufen. Ansonsten wurde in diesen Funktionen nichts hinzugefügt.

ConvertEQ: Als Parameter wird dieser Funktion, wie bereits oben erwähnt, der Befehl, der den mathematischen Modus startet, in Form einer Integerzahl mitgeteilt. Mit dieser Integerzahl, deren Wert durch Konstanten definiert ist, wird nun über eine switch-Anweisung (Abb. 5.7) entschieden, wie die Funktion getTexUntil aufgerufen wird. Die Funktion getTexUntil braucht als Parameter die Befehlsequenz, nach der sie suchen, und bis zu der sie den Quelltext einlesen und zurückgeben soll. Eben diese Befehlssequenz ist durch den Parameter feststellbar. Je nach Parameter wird die Funktion aufgerufen, und liefert den Zeiger auf den Speicherplatz zurück, in dem der voreingelesene Text abgespeichert ist. Zur näheren Beschreibung von getTexUntil siehe Abschnitt 5.3.4 auf Seite 52.

Die Variable s bekommt diesen Zeiger zugewiesen. Nach der switch-Anweistung erfolgt der Aufruf der Funktion PrepareEQString(s). Als Parameter wird der Funktion der Zeiger auf den voreingelesenen Text übergeben. Die Zeigervariable convert erhält den Rückgabewert dieser Funktion. Dieser Zeiger convert wird anschließend dazu verwendet, um der Funktion ConvertString, deren Aufruf nun folgt, den zu konvertierenden Textblock zu übergeben. Nach der Konvertierung mit convert schließt die Funktion ConvertEQ.

PrepareEQString: Diese Funktion ist der Kernteil der Vorverarbeitung des Textblocks. Es wird der übergebene Teil durchsucht, um beim Auftreten eines `\over`-Befehls diesen in den äquivalent `\frac`-Befehl umzuformen, und den umgeformten Textblock der Konvertierung zur Verfügung zu stellen.

```
switch(code){
    case EQN_DOLLAR_DOLLAR:
        diagnostics(5,"ConvertEQ switch(code) code=$$");
        s=(char *)getTexMathUntil("$$",0);
        break;
    case EQN_BRACKET_OPEN:
        diagnostics(5,"ConvertEQ switch(code) code=\\[");
        s=(char *)getTexMathUntil("\\]",0);
        break;
    case EQN_MATH:
        diagnostics(5,"ConvertEQ switch(code) code=\\begin{math}");
        s=(char *)getTexMathUntil("\\end{math}",0);
        break;
    .
    .
    .
}
```

Abbildung 5.7: Ein Teil der switch-Anweisung

Innerhalb der Funktion werden zwei Konstanten und zwei Felder, left und right, deren Elementanzahl durch die zwei Konstanten bestimmt wird, definiert. Die Elemente des ersten Feldes enthalten mögliche Modifikationen der öffnenden geschwungenen Klammer. Das zweite Feld enthält in seinen Elementen die korrespondierenden Teile für die schließende geschwungene Klammer. Diese Modifikationen bestehen aus der Größenanpassung der Klammern. Es handelt sich dabei auch um eine explizite Klammerung für die Darstellung, und nicht um die Klammerung, die für den Befehl `\over` verwendet wird. Diese Definitionen sind notwendig, um, wie später beschrieben, diese Klammern von den gesuchten unterscheiden zu können. Es ist jederzeit möglich diese Felder anzupassen, da bis jetzt nur die verwendet werden, die auch vom Konverter verarbeitet werden.

Als erstes werden für die Zeigervariablen converts, buffer und temp die notwendigen Speicherplätze reserviert. Diese werden auf die doppelte Größe des zu bearbeitenden Blocks ausgelegt.

Nachdem die Speicherplätze reserviert sind, folgen drei Abfragen. Die erste testet die Länge des Strings, ob dieser zu kurz ist, um einen `\over`-Befehl zu enthalten. Die zweite untersucht den Textblock, ob kein `\over` zu finden ist und die dritte Abfrage untersucht den gesamten Block, ob die Anzahl der öffnenden und schließenden geschwungenen Klammer ungleich groß ist. Jede der drei Abfragen beendet, bei ihrem Eintreten, die Funktion

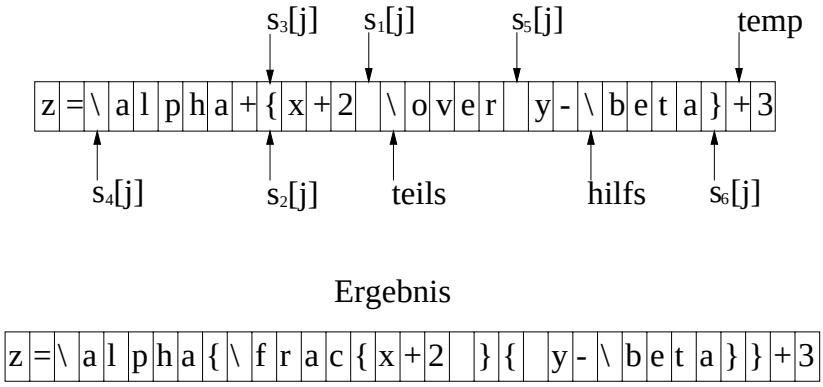


Abbildung 5.8: PrepareEQ, aktuelle Positionen und Ergebnis

und gibt den Zeiger auf den Textblock s als Rückgabewert zurück. Die zweite Abfrage liefert einen Zeiger auf den, mit dem `\over`-Befehl beginnenden, restlichen Textblock zurück. Dieser wird für die spätere Verwendung in der Variablen teils (siehe Abb. 5.8) gespeichert.

Nun wird die Indexvariable j berechnet. Diese Berechnung erfolgt durch die Differenzbildung der Länge des Textblocks mit der Länge des nach dem Auftreten des `\over`-Befehls verbleibenden Strings. Dieser wird um eins vermindert, um das Zeichen vor dem Befehl zu indizieren (s_1[j] in Abb. 5.8). Es beginnt die Suche nach der öffnenden geschwungenen Klammer. Dies wird durch eine Schleife, siehe Abb. 5.9, realisiert, die mit dem Index j beginnt, mit jedem Schleifendurchlauf j dekrementiert, und höchstens bis j gleich null laufen kann. Zeichen für Zeichen arbeitet sich die Schleife, beginnend mit dem Zeichen vor dem `\over`, rückwärts durch den Textblock. Jedes dieser Zeichen wird untersucht, ob es eine öffnende oder schließende geschwungene Klammer ist. Für eine schließende Klammer wird die Variable brace, die mit null initialisiert ist, um eins inkrementiert, und für eine öffnende Klammer um eins dekrementiert. Die Schleife endet, wenn brace einen negativen Wert annimmt (s_2[j] in Abb. 5.8), oder der Index j bei null angekommen ist.

Endet Suche nach den Klammern, so muß der Wert von brace kleiner null sein. Sollte dies nicht der Fall sein, so ist die für den `\over`-Befehl notwendige öffnende geschwungene Klammer nicht vorhanden, und es wird die Funktion mit dem Rückgabewert s beendet.

Ist eine öffnende Klammer gefunden, so fehlt noch eine weitere Untersuchung, die testet, ob es sich um eine modifizierte Klammer handelt, die nicht für diesen Befehl zu verwenden ist. Zur Durchführung dieser Untersuchung startet eine Schleife mit dem Index l (s_3[j] in Abb. 5.8), der auf die notwendige öffnende Klammer zeigt, und maximal solange läuft, bis l den Wert null hat. In dieser Schleife wird rückwärts nach dem ersten Auftreten eines Backslashes gesucht, der das Startzeichen eines Befehls ist. Wird ein

```
for (j;j>=0;j--) {
    switch (s[j]){
        case '}': brace++;
            break;
        case '{': brace--;
            break;
        default:
            break;
    }
    if (brace<0) break;
}
```

Abbildung 5.9: Die Suche nach der öffnenden Klammer

Befehl gefunden (s_4[j] in Abb. 5.8), so schreibt die Funktion die Zeichen aufwärts, mit dem Backslash beginnend, in den String leftstring. Dieses Schreiben endet etweder nach acht Zeichen, oder wenn der Index vor der öffnenden Klammer erreicht wird. In einer Schleife, die höchstens einmal für jedes Element des Feldes left durchlaufen wird, vergleicht die Funktion den gefundenen Befehl, der in leftstring steht, mit den Elementen von left. Sollte einer dieser Vergleiche erfüllt sein, so bricht die Funktion ab und gibt den Zeiger s zurück.

Nach diesem Test werden die Zeichen zwischen der öffnenden geschwungenen Klammer und dem Befehl `\over` in die Variable buffer geschrieben. Als Abschluß bekommt buffer noch eine schließende geschwungene Klammer und ein Null-Byte angefügt.

Der vor der gesuchten Klammer stehende Teil des Textblocks, inklusive der Klammer, wird zeichenweise in die Variable converts geschrieben. Nun erfolgt das Anhängen der Sequenz „ \\frac{“ an converts. Dies ist der Befehl, in den `\over` umgewandelt werden soll. Daraufhin hängt die Funktion die Variable buffer an converts an, womit der erste Parameter für den Befehl `\frac` bereits vorverarbeitet ist.

Der nächste Schritt ist die Suche nach dem zweiten Parameter und somit nach der schließenden geschwungenen Klammer. Als ersten Schritt schreibt die Funktion eine öffnende geschwungene Klammer als erstes Zeichen in die Variable buffer, die den zweiten Parameter aufnehmen soll. Der Index j wird durch Differenzbildung zwischen der Länge des gesamten Textblocks und der Länge des Textblocks ab dem Befehl `\over`. Dieser Index wird um fünf erhöht, um auf dem ersten Zeichen nach dem Befehl `\over` zu stehen (s_5[j] in Abb. 5.8).

Die Schleife, die nun beginnt, und die nach jedem Schleifendurchlauf j um eins erhöht, läuft, solange j kleiner ist als die gesamte Länge des Textblocks, und sie nicht vorher abbricht. Innerhalb dieser Schleife werden die öffnenden und schließenden geschwungenen Klammern gezählt. Eine öffnende dekrementiert und eine schließende Klammer inkrementiert den Wert der Variablen brace. Diese Variable besitzt vor dieser Schleife noch den Wert aus der vorherigen Suche, und ist minus eins. Erreicht brace den Wert null, so ist die gesuchte Klammer gefunden (s_6[j] in Abb. 5.8), und die Schleife bricht ab. Während der Suche wird jedes Zeichen in die Variable buffer geschrieben. Nach Abbruch der Schleife steht in buffer der zweite Parameter ohne der abschließenden Klammer. Zur Verwendbarkeit als String, erhält buffer noch ein Null-Byte angeschlossen. Erreicht brace bis zum letzten Zeichen nicht den Wert null, so wird die Vorverarbeitung abgebrochen, und der Rückgabewert ist der Zeiger s.

Um aus buffer einen vollständigen String zu machen, wird ein Null-Byte an buffer angehängt. Es folgt ein Durchsuchen von buffer in Rückwärtsrichtung, ob ein Backslash vorhanden ist. Der Ergebniswert dieses Tests ist der Zeiger hilfs (siehe Abb. 5.8), der auf den Teil von buffer zeigt, wo der letzte Backslash auftritt. Wenn dieser Zeiger ungleich null ist, so beginnt erneut eine Schleife zu laufen. Diese Schleife läuft für jedes Element des Feldes right höchstens einmal. Innerhalb dieser vergleicht die Funktion den String, auf welchen hilfs zeigt mit den einzelenen Elementen von right. Dies ist die Abfrage, ob es sich um eine modifizierte Klammer handelt, die auch von LaTeX nicht als Ende des `\over`-Befehls erkannt wird. Sollte eine Übereinstimmung vorhanden sein, so bricht die Funktion ab und der Rückgabewert ist der Zeiger s.

Die Funktion hängt an buffer eine schließende geschwungene Klammer an. Damit steht in buffer nun der in geschwungene Klammern gefaßte zweite Parameter. Ein rekursiver Aufruf von PrepareEQString untersucht buffer, ob ein `\over`-Befehl enthalten ist. Der Rückgabewert dieses rekursiven Aufrufs wird an converts angehängt.

Die Klammer, die sich vor dem ersten Parameter des `\over`-Befehls befindet, steht jetzt vor dem Befehl `\frac`. Um die Klammerung abzuschließen, fügt die Funktion eine schließende geschwungene Klammer nach dem zweiten Parameter ein, indem sie diese an converts anhängt.

Der Rest des Textblocks, der nach der schließenden geschwungenen Klammer noch übrig ist, wird in die Variable temp geschrieben. Ist die Länge von temp (siehe Abb. 5.8) gößer als sechs, so kann es möglich sein, daß ein weiterer `\over`-Befehl enthalten ist. Daher ruft sich die Funktion mit temp als Parameter rekursiv auf. In diesem Fall wird der Rückgabewert dieses Aufrufs an converts angefügt. Ansonsten hängt die Funktion den Rest des Texblocks, der kleiner gleich sechs Zeichen ist, an converts an.

Zum Abschließen des Strings converts muß noch ein Null-Byte angehängt werden. Anschließend wird der Speicherplatz von temp und buffer freigegeben. Die Funktion endet hier, und der Rückgabewert ist der Zeiger auf converts.

5.3.7 Falsche Zeilennummern

In der Ausgabe der Fehler, Warnungen usw. wird jeweils die Zeilennummer und der aktuelle Dateiname ausgegeben. Dies soll helfen, Fehler rückverfolgen zu können und die Fehlerquelle im Quelltext bzw. im Konvertierungsprogramm zu beheben. Als schwierig erwies sich, daß die angegebenen Zeilen nicht mit denen übereinstimmen, in denen die Fehler aufgetreten sind. Daher ist der Sinn der Zeilennummern nicht erfüllt.

Die Fehler lagen hauptsächlich daran, daß beim Voreinlesen und Vorverarbeiten zwar die Zeilenzählung ausgeschalten wurde, aber durch die Aufrufe von Funktionen, wie z. B. getTexUntil, siehe Abschnitt 5.3.4 auf Seite 52, innerhalb eines solchen Voreinlesens die Zeilenzählung ungewollt wieder eingeschaltet wurde. Ein weiterer Grund für die Abweichungen der Zeilennummern war die Verwendung der Befehle `\include` und `\input`. Diese Befehle werden in der Funktion getSection, siehe Abschnitt 5.3.5 auf Seite 58 behandelt. Diese beiden Befehle verwendet LaTeX um andere Quelldateien einbinden zu können. Beim Voreinlesen wird von Abschnitt zu Abschnitt eingelesen. Dabei kann es vorkommen, daß der Abschnitt in der ersten Datei beginnt, aber der nächste Abschnitt sich in der neuen Datei befindet. Beim Verarbeiten dieser beiden gerade genannten Befehle wird die aktuelle Datei, aus der eingelesen wird, richtigerweise getauscht. Das Weiterzählen der Zeilennummern bleibt ebenso ausgeschaltet, aber als aktuelle Zeilennummer wird die Nummer eins verwendet. Nach dem Voreinlesen beginnt der Konverter mit der Konvertierung des voreingelesenen, dateiübergreifenden Abschnitts nach der Abschnittsüberschrift mit der Zeilennummer eins, welche in den meisten Fällen nicht die erste Zeile der neuen Datei ist. Aus diesem Grund ist der Wert der Zeilennummer bereits erhöht, wenn der Konverter zur Zeile eins in der neuen Datei gelangt. Auch der Dateiname der Quelldatei hat bereits den neuen Dateinamen gespeichert und gibt diesen bei Meldungen aus, die die vorherige Datei betreffen. Dies erschwert die Fehlersuche.

Um diesem Problem zu begegnen, wurde die Funktion getBraceParam und getSection modifiziert.

getBraceParam: Hier wird g_parser_scan_ahead_switch als neue boolsche Variable eingeführt, und mit FALSE initialisiert. Es gibt zwei neue Abfrage. Die erste Abfrage befindet sich am Anfang der Funktion. Diese unterscheidet, ob g_parser_scan_ahead gesetzt

ist. Trifft dies nicht zu, so verwendet getBraceParam die Funktion SetScanAhead(TRUE), um die Zeilenzählung zu vermeiden, und setzt die neu eingeführte Variable auf TRUE.

Mit der zweiten Abfrage, am Ende vor dem return eingefügt ist, wird untersucht, ob g_parser_scan_ahead_switch gesetzt ist. Wenn dies der Fall ist, so schaltet SetScanAhead die Zeilenzählung wieder ein, und die Variable g_parser_scan_ahead_switch wird auf FALSE gesetzt.

Diese Umschaltung ist notwendig, um zwischen den verschiedenen Fällen der Anwendung von getBraceParam unterscheiden zu können, und die Zeilenzählung, wenn sie vermieden werden soll, auszuschalten.

getSection: Als zweiter Lösungsansatz ist hier die Anwendung der Funktion SetScanAhead zu betrachten. Diese Funktion schaltet die Zeilenzählung ein und aus. Leider gibt es die Konstellation, daß die Zählung ausgeschaltet ist, aber eine Funktion aufgerufen wird, in der SetScanAhead am Anfang gesetzt und am Ende rückgesetzt wird. Dies betrifft die Funktion getTexUntil (Abschnitt 5.3.4). Diese Funktion wird sowohl während der Konvertierung zum Voreinlesen von Textteilen verwendet, als auch durch die Funktion getSection (Abschnitt 5.3.5). Während die, in dieser Funktion am Ende stehende Rücksetzung von SetScanAhead für das Voreinlesen beim Konvertieren gewünscht ist, setzt sie andererseits, bei der Verwendung durch getSection, SetScanAhead ungewollt zurück. Durch Einfügen der Funktion SetScanAhead nach dem Aufruf von getTexUntil in getSection, wird die Zeilenzählung wieder ausgeschaltet.

Um die Zeilendifferenz bei dateiübergreifenden Abschnitten zu verhindern, wurde in die voreinlesenden Funktionen getSection der Zähler count eingebaut, der am Beginn der Funktion den Wert null erhält, und der jede Zeilenschaltung mitzählt. Dies ist in verschiedenen Fällen möglich. Eine Erhöhung von count tritt nach einer Zeilenschaltung, die in der normalen Umgebung verwendet wird, ein. Weiters inkrementiert sich count am Ende einer Kommentarzeile. Das Auftreten jeder Zeilenschaltungen innerhalb der Parameter der Befehle `\verb` und `\url` erhöht count und der mit getTexUntil eingelesen Text muß auf Zeilenschaltungen untersucht werden. Dies wird mit einer Schleife (siehe Abb. 5.9) realisiert, die solange läuft, solange eine Zeilenschaltung im eingelesenen Text zu finden ist. In jedem Schleifendurchlauf erhöht sich count.

Die Anwendung von count erfolgt in dem Abschnitt von getSection, der die Befehle `\include` und `\input` behandelt, denn nur hier kann es zu einem Dateiwechsel kommen. Die Zeilennummer, die nach dem Dateiwechsel verwendet wird, ist in der Variablen g_parser_line gespeichert. Diese Variable vermindert sich um den Wert von count. Aus diesem Grund beginnt die Konvertierung des dateiübergreifenden Abschnitts mit negativen Zeilennummern. Als Dateiname gibt der Konverter den Namen der neuen Datei aus.

```
hilfs1=s;
while((hilfs2=strchr(hilfs1,'\n'))!=NULL) {
    hilfs2++;
    hilfs1=hilfs2;
    count++;
}
```

Abbildung 5.10: Zählen der Zeilenschaltungen

Die Zeilennummern sind negativ, solange der Quelltext noch aus der vorherigen Datei stammt.

Mit dem Wissen, wie mit negativen Zeilennummern umzugehen ist, kann die richtige Zeile leicht gefunden werden.

5.3.8 Ausgabedatei des eingelesenen Quelltextes

Es gibt eine neue zusätzliche Startoption

-s: Ausgabedatei.

Die Verwendung dieser Option ermöglicht die Angabe eines Dateinamens, in die der Konverter in jeder Zeile zuerst den aktuellen Dateinamen, die Zeilennummer und anschließend jedes Zeichen, das eingelesen wurde und der Konvertierung zur Verfügung steht, schreibt. Dabei sind die Definitionen bereits expandiert. D. h. in dieser Datei stehen die Zeichen und Befehle so, wie sie vom Konverter verarbeitet werden. Die einzelnen Quelltextdateien werden dabei in die festgelegte Datei geschrieben. Sollte der Konverter seine Arbeit abbrechen, enden auch die Einträge in diese Datei.

Diese Option trägt zur Vereinfachung der Fehlersuche bei. So stehen nur jene Zeichen bzw. Zeilen in der Datei, die auch konvertiert wurden. Damit lassen sich die Abbruchstellen auffinden, und auch die richtige Verwendung von Definitionen kontrollieren.

Um dies zu realisieren, wurden in der Programmdatei main.c vier neue Funktionen definiert.

OpenSourceFile: Als Parameter besitzt diese Funktion einen Zeiger auf den Namen der zu öffnenden Datei und einen Parameter, mit dem die Adresse des Dateizeigers übergeben werden kann. Ist der Dateiname nicht NULL, so öffnet oder erstellt diese Funktion die Datei mit dem angegebenen Namen und weist ihr einen Dateizeiger zu.

CloseSourceFile: Diese Funktion hat als einzigen Parameter die Adresse des Dateizeigers. Sie ist das Gegenstück zur Funktion im vorigen Absatz, und schließt die Datei, wenn der Dateiname nicht NULL ist.

OpenSourceFile wird von main am Anfang der Konvertierung unmittelbar nach dem Öffnen der RTF-Datei aufgerufen. Die Verwendung von CloseSourceFile erfolgt dementsprechen nach dem Schließen der RTF-Datei am Ende von main.

putSourceChar: Der Parameter dieser Funktion ist das Zeichen, das in die neue Datei geschrieben werden soll. Sie ist durch eine Abfrage realisiert, die falls der Dateiname ungleich NULL ist, das Zeichen in die Datei schreibt. Ist der Dateiname NULL, so bleibt der Funktionsaufruf ergebnislos.

putSourceString: Dies ist die gleiche Funktion wie putSourceChar, nur daß hier anstatt eines Zeichens ein Zeiger auf ein String übergeben wird, und dieser, unter der Bedingung, daß der Dateiname ungleich NULL ist, in die neue Datei geschrieben wird. Mit Hilfe dieser Funktion kann das Programm den Dateinamen und die Zeilennummer in die Datei schreiben.

Um putSourceChar und putSourceString nur dann zu verwenden, wenn konvertiert wird, und nicht bereits beim Voreinlesen, gibt es die boolsche Variable g_processing_parameter. Diese Variable wird je nach momentaner Tätigkeit des Konverters gesetzt oder rückgesetzt. Sie bestimmt, daß beim Voreinlesen der Parameter das Programm nichts in die Datei schreibt.

Die Variable bekommt an vier verschiedenen Stellen im Konverter einen Wert zugewiesen. Zu Beginn weist ihr main den Wert FALSE zu. Weiters wird sie am Ende von ConvertString und nach der Konvertierung eines Befehls auf FALSE gesetzt. Den Wert TRUE erhält sie nur am Anfang der Funktion getBraceParam, die die Parameter vor der Konvertierung einliest, damit nicht eine zweifache Ausgabe der Parameter erfolgt.

Die Auflistung der Aufrufe von putSourceChar und putSourceString bringt keine Verbesserung des Verständnisses, und würde nur aus einer Beschreibung der Zeilen bestehen, wo die Aufrufe stehen.

5.3.9 openBrace

LaTeX aktzeptiert manche Fehler, und stellt das Dokument trotzdem richtig dar. Es werden Warnungen und Meldungen ausgegeben, die diese Fehler andeuten, aber die

meisten Anwender verzichten darauf, diese Fehler zu suchen, da LaTeX das gewünschte Ergebnis liefert.

Einer dieser Fehler ist, daß eine öffnende geschwungene Klammer zuviel im Text steht. Ausgenommen sind spezielle Umgebungen, in diesen akzeptiert auch LaTeX dieses zusätzliche Zeichen nicht. Dem Zeichen fehlt die korrespondierende schließende Klammer. LaTeX gibt eine Meldung (\end occured inside a group at level 1) aus, die jedoch nicht den Status einer Fehlermeldung besitzt. Ist das verfaßte Dokument sehr lang, so ist es mühsam, den Fehler zu suchen.

Der Konverter akzeptiert diesen Fehler nur teilweise. Es wird die Konvertierung durchgeführt, aber es ist nicht möglich, die RTF-Datei mit MS-Word zu öffnen, da zwar das Dateiende erreicht ist, aber der Konverter sich noch in einem höheren „Bracelevel“ befindet, und diesen nicht mehr schließt. Es ist auch nicht möglich, eine einfache RTF-Sequenz anzuschließen, um dieses Problem zu umgehen. Es gibt keine Fehlermeldung und auch keinen Hinweis darauf, weshalb sich die Datei nicht weiterverarbeiten läßt.

Aus diesem Grund wurde eine Abfrage in den Programmteil main.c eingebaut. Diese Abfrage testet, ob vor dem Schließen der RTF-Datei der „Bracelevel“ größer als eins ist. Der „Bracelevel“ wird mit einer öffnenden geschwungenen Klammer inkrementiert, und mit einer schließenden geschwungenen dekrementiert. Sollte die Abfrage zutreffen, so gibt es eine dieser öffnenden geschwungenen Klammern, der das korrespondierende Zeichen fehlt. In diesem Fall wird vom Konverter eine Warnung ausgebenen, daß mindestens eine Klammer zuviel vorhanden ist.

Den umgekehrten Fall, einer schließenden geschwungenen Klammer zuviel, akzeptiert auch LaTeX nicht und zeigt die Zeile mit dem Fehler an. Der Konverter akzepiert diese Klammer zwar, aber nur solange, bis er mit dem „Bracelevel“ auf eins kommt. Zu diesem Zeitpunkt ist für den Konverter der Zustand erreicht, in dem er die Arbeit beendet.

Kapitel 6

Zusammenfassung

Die Durchführung der Diplomarbeit gliederte sich in drei große Aufgabenbereiche. Der erste Aufgabenbereich stand im Zeichen der Suche nach vorhandenen Konvertern. Diese Suche gestaltete sich umfangreicher als angenommen. Es gibt im Internet jede Menge Links zu Konvertern. Diese Links mußten untersucht werden. Durch unterschiedliche Bezeichnungen und die verschiedenen vorhandenen Versionen gestaltete sich dieses Unterfangen als langwierig. Nach dem Sortieren und Auflösen der Links kam zum Vorschein, daß es sich dabei um eine kommerzielle und drei freie Entwicklungen von Konvertern handelt.

Im zweiten Schritt folgte der Vergleich der gefundenen Konverter. Zu diesem Zweck wurden spezielle, auf gewisse Befehlsgruppen ausgerichtete Testdateien geschrieben. Mit diesen Testdateien fand eine Untersuchung und Bewertung der freien Konverter statt. Um zu einer Bewertung zu gelangen, erfolgte eine Benotung der Konvertierungsergebnisse. Die Benotung unterlag weiters einer Gewichtung, die zur Festlegung des Schwerpunkts diente. Diese Schwerpunktsfestlegung war von entscheidender Bedeutung, da sich die Reihung durch die Verschiebung des Schwerpunkts verändern kann.

Nach der Festlegung auf den Konverter, der dem Schwerpunkt am besten entsprach, begann der dritte Teil der Aufgabe. Es folgte die Verbesserung des Konverters. Diese orientierten sich an den vom Institut zur Verfügung gestellten Dateien, die einen Querschnitt über die zukünftige Verwendung darstellen. Als Endergebnis sollte es möglich sein, diese Dateien so zu konvertieren, daß sie als RTF-Dateien unter MS-Word weiterverarbeitet werden können.

Zu Beginn der Verbesserungen mußte zuerst die Struktur des Konverters erkannt werden. Dies stellte keine leichte Aufgabe dar, da sich dieser aus über vierzig Dateien zusammensetzt, und mehrere Dateien an der Steuerung des Ablaufs beteiligt sind. Durch die Kenntnis der Struktur stellte sich heraus, daß die wichtigsten Verbesserungen im Bereich

der Vorverarbeitung des Quelltextes notwendig waren. Darauf gingen die Erweiterungen des Konverters ein.

Während der Durchführung der Diplomarbeit wurde auch von Anderen der Konverter verbessert. Diese Verbesserungen waren unter der Internetadresse

http://Sourceforge.net/projects/latex2rtf/

zu finden. Diese Adresse stellt den Kontakt zwischen den einzelnen Entwicklern des Konverters dar. Unter dieser Adresse wurden von Zeit zu Zeit neue Versionen zur Verfügung gestellt. Die durchgeführten Verbesserungen fanden immer auf Basis der neuesten Version statt. Die letzte Version, unter der Verbesserungen stattfanden, hatte die Nummer 1.19.12. Alle, im Rahmen der Diplomarbeit erstellten Erweiterungen, wurden diesen Entwicklern zur Verfügung gestellt. Diese Erweiterungen fanden in durch die Entwickler abgeänderter Form Eingang in den Konverter.

Nach den Verbesserungen war es möglich, den Großteil der Dateien zu konvertieren. Das Ergebnis waren Dateien, die den mit LaTeX erzeugten Dokumenten schon sehr nahe kamen, und die unter MS-Word weiterverarbeitet werden können. Probleme bleiben weiterhin bei Quelldateien bestehen, für die auch LaTeX Zusatzpaket benötigt, um eine Verarbeitung zu ermöglichen. Dies ist vor allem bei Formeln, die selbst LaTeX nur mit Mathematik-Paketen verarbeitet, der Fall.

Die Konvertierung hat sich mit jeder Erweiterung verbessert. Es scheint jedoch eine endlose Beschäftigung zu sein, denn sobald eine Verbesserung funktionierte, taten sich wieder neue Betätigungsfelder auf, die vor der Erweiterung als nicht wichtig, oder nicht existent erschienen. So setzt sich die Arbeit an dem Konverter ins Unendliche fort. Es bleibt die Frage bestehen, ob es in absehbarer Zeit überhaupt möglich ist, einen Konverter zu erstellen, der die gleichen Fähigkeiten besitzt, die auch LaTeX eigen sind.

Kapitel 7

Dokumentation & Anleitung

7.1 Vorwort

Die letzte Version von latex2rtf, auf die Verbesserungen aufbauen, ist 1.19.12. Diese Version ist frei verfügbar und kann aus dem Internet bezogen werden. In der Zwischenzeit ist die Version 1.19.13 in Vorbereitung.

In dem Paket von latex2rtf ist auch eine Dokumentation über die Installation, die Anwendung usw. enthalten. Darin sind noch detailliertere Informationen enthalten, als hier aufgeführt sind.

Diese Dokumentation unterscheidet sich nur durch die neu eingeführte Option -s, die in Version 1.19.12 nicht enthalten ist. Diese Option ist im Abschnitt 5.3.8 auf Seite 73 beschrieben.

7.2 Installation

latex2rtf kann unter Unix, Windows, MacOS und MS-DOS installiert und verwendet werden. Die entsprechenden Installationschritte sind in der oben genannten Dokumentation enthalten, und werden daher nicht explizit angeführt.

7.3 Hinweise

Vor der Verwendung von latex2rtf gibt es noch einige Hinweise zu beachten.

Am allerwichtigsten ist es, darauf hinzuweisen, daß der Konverter nur Quelldateien konvertieren kann, die als korrekte LaTeX-Dokumente verfaßt wurden. Kann LaTeX das

Dokument nicht richtig verarbeiten, funktioniert auch die Konvertierung nicht. Es ist ratsam, die Fehler zuerst mit Hilfe von LaTeX zu suchen, und zu beheben, und erst anschließend die Datei zu konvertieren.

Um bessere Ergebnisse zu erhalten, sind noch Anpassungen notwendig. Diese Anpassungen betreffen

RTFPATH: ist diese Umgebungsvariable nicht gesetzt, so sucht der Konverter die cfg-Dateien in seinem Standardpfad. Durch Setzen des RTFPATH kann ein anderer Suchpfad eingegeben werden, der als erster durchsucht wird. Falls latex2rtf die benötigten Datein nicht findet, so sucht er diese in seinem Standardverzeichnis. Ist auch dort nichts zu finden, bricht er ab.

fonts.cfg die Anpassung dieser Datei ermöglicht die richtige Zuordnung der verwendeten Schriftarten. Alle Schriftarten, die LaTeX verwendet, und die keine Zuordnung in dieser Datei haben, werden mit der Standardschrift dargestellt. Sind einer LaTeX-Schriftart mehrere Schriften zugeordnet, so wird nur die erste Zuordnung verwendet. Alle hier verwendeten Schriftarten finden sich im RTF-Header wieder.

ignore.cfg in dieser Datei steht, wie die speziellen Befehle zu ignorieren sind. LaTeX-Variable, benutzerdefinierte Variable und einige Befehle werden ignoriert. Dem Konverter wird in dieser Datei mitgeteilt, wie er diese zu ignorieren hat.

direkt.cft ist die RTF-Entsprechung für einen Befehl, der nicht konvertiert werden kann, bekannt, so ist dies hier einzutragen. latex2rtf sieht in dieser Datei nach, und schreibt die hier eingetragene RTF-Entsprechung in die Ergebnisdatei.

<language>.cfg eine Anpassung dieser Dateien ist nur dann erforderlich, wenn die Bezeichnungen nicht mit den gewünschten Ergebnissen übereinstimmen.

7.4 Anwendung

Die Befehlssyntax lautet

latex2rtf [options] inputfile.[tex]

Die Optionen bedeuten:

-a bietet die Möglichkeit, eine aux-Datei festzulegen, wenn der Name dieser Datei nicht mit dem Name der Quelltextdatei übereinstimmt. Ist diese Option nicht angegeben, so versucht der Konverter die Datei inputfile.aux zu verwenden um die Querverweise

verarbeiten zu können. Sollte dies nicht möglich sein, so können die Querverweise nicht erstellt werden.

-b hat die bbl-Datei einen anderen Namen als die Quelltextdatei, so ist diese Option zu verwenden. Diese Datei wird durch BIBTEX erzeugt, und enthält die Referenzen.

-d# debugging, hier wird der debugging level eingestellt, je höher, umso mehr Information erhält man. Der Bereich geht von 0 bis 7 und beginnt damit, daß nur Fehlermeldungen erscheinen, bis hin zur höchsten Stufe, in der beinahe jede Tätigkeit des Konverters ausgegeben wird.

-i dieser Konverter besitzt eine Anzahl von language.cfg-Dateien. Diese enthalten die festgelegten Abschnittsnamen wie z. B. Literaturverzeichnis, Anhang und vieles mehr. Die entsprechenden Ausdrücke werden normalerweise durch den Befehl `\usepackage[language]{babel}` festgelegt. Ist dies nicht der Fall, so bietet diese Option die Möglichkeit, die Sprache der Abschnittsnamen festzulegen.

-l ermöglicht die Verwendung von verschiedenen Schriftarten beim Einlesen. Dies ist für die Abwärtskompatibilität inkludiert.

-o hier kann der Name der Ergebnisdatei festgelegt werden, wenn der Name anders lauten soll als die Standardausgabedatei inputfile.rtf

-s diese Option ist neu, und ermöglicht es, den gelesenen Quelltext in eine Datei mit frei wählbarem Namen zu schreiben, wobei vor jeder Zeile der Dateiname und die Zeilennummer erscheinen

-C wird für die Angabe von Schriftsätzen verwendet, die im LATEX Dokument Verwendung finden. Diese Schriftsätze werden im Normalfall bereits durch den Befehl `\usepackage[codepage]{inputenc}` angegeben, und auch von Konverter verwendet.

-V gibt die Versionsnummer auf der Standardausgabe zurück und beendet den Konverter.

-W schreibt die Warnungen direkt in die RTF-Datei.

-Z# nach dem Ende der Quelldatei werden die angegebene Anzahl von schließenden geschwungenen Klammern konvertiert. Diese Option kann von Nutzen sein, wenn die RTF-Datei nicht korrekt geschlossen wurde (siehe Abschnitt openBrace).

Anhang A

Testdateien

A.1 align.tex

Dies ist zentriert mit begin{center} und end{center}. Dies ist zentriert mit begin{center} und end{center}. Dies ist zentriert mit begin{center} und end{center}. Dies ist zentriert mit begin{center} und end{center}. Dies ist zentriert mit begin{center} und end{center}.

zentriert mit centering

Dies ist ein einfacher Text, der links ausgerichtet ist mit begin{flushleft} und end{flushleft}. Dies ist ein einfacher Text, der links ausgerichtet ist mit begin{flushleft} und end{flushleft}. Dies ist ein einfacher Text, der links ausgerichtet ist mit begin{flushleft} und end{flushleft}. Dies ist ein einfacher Text, der links ausgerichtet ist mit begin{flushleft} und end{flushleft}. Dies ist ein einfacher Text, der links ausgerichtet ist mit begin{flushleft} und end{flushleft}.

linksbuendig mit raggedright

Dies ist ein einfacher Text, der rechts ausgerichtet ist mit begin{flushright} und end{flushright}. Dies ist ein einfacher Text, der rechts ausgerichtet ist mit begin{flushright} und end{flushright}. Dies ist ein einfacher Text, der rechts ausgerichtet ist mit begin{flushright} und end{flushright}. Dies ist ein einfacher Text, der rechts ausgerichtet ist mit begin{flushright} und end{flushright}. Dies ist ein einfacher Text, der rechts ausgerichtet ist mit begin{flushright} und end{flushright}.

rechtsbuendig mit raggedleft

A.2 array.tex

Alle drei Arten von Ausrichtung werden getestet,

$$w = \begin{array}{ccc} x+y & y+2c & z*x^2 \\ x & y & z \\ x & y & z \\ x & y & z \end{array} \tag{A.1}$$

$$w = \left(\begin{array}{ccc} x+y & y+2c & z*x^2 \\ x & y & z \\ x & y & z \\ x & y & z \end{array} \right) \tag{A.2}$$

A.3 article.tex

Part I

das ist der erste teil

1 erster Abschnitt

1.1 erster Unterabschnitt

1.1.1 erster Unterunterabschnitt

erster Paragraph

erster Unterparagraph

1

A.4 char.tex

$\mathring{o}$	$\mathring{x}$	$\mathring{X}$	$\mathring{O}$
$\grave{o}$	$\grave{x}$	$\grave{X}$	$\grave{O}$
$\acute{o}$	$\acute{x}$	$\acute{X}$	$\acute{O}$
$\hat{o}$	$\hat{x}$	$\hat{X}$	$\hat{O}$
$\ddot{o}$	$\ddot{x}$	$\ddot{X}$	$\ddot{O}$
$\tilde{o}$	$\tilde{x}$	$\tilde{X}$	$\tilde{O}$
$\bar{o}$	$\bar{x}$	$\bar{X}$	$\bar{O}$
$\dot{o}$	$\dot{x}$	$\dot{X}$	$\dot{O}$
$\check{o}$	$\check{x}$	$\check{X}$	$\check{O}$
$\c{o}$	$\c{x}$	$\c{X}$	$\c{O}$
$\d{o}$	$\d{x}$	$\d{X}$	$\d{O}$
$\b{o}$	$\b{x}$	$\b{X}$	$\b{O}$
$\u{o}$	$\u{x}$	$\u{X}$	$\u{O}$

A.5 charmath.tex

$\hat{o}$	$\hat{x}$	$\hat{X}$	$\hat{O}$
$\check{o}$	$\check{x}$	$\check{X}$	$\check{O}$
$\acute{o}$	$\acute{x}$	$\acute{X}$	$\acute{O}$
$\grave{o}$	$\grave{x}$	$\grave{X}$	$\grave{O}$
$\tilde{o}$	$\tilde{x}$	$\tilde{X}$	$\tilde{O}$
$\bar{o}$	$\bar{x}$	$\bar{X}$	$\bar{O}$
$\vec{o}$	$\vec{x}$	$\vec{X}$	$\vec{O}$
$\dot{o}$	$\dot{x}$	$\dot{X}$	$\dot{O}$
$\ddot{o}$	$\ddot{x}$	$\ddot{X}$	$\ddot{O}$
$\breve{o}$	$\breve{x}$	$\breve{X}$	$\breve{O}$

A.6 cite.tex

This is a citation for which the citation should be known [1]. We follow this with an unknown citation [?].

To test various combinations we need to cite more than one author. To this end, I cite two known authors [1, 2]. This might include one known and one unknown [?, 1]. Another bizarre case might be one known, one unknown and another known [1, ?, 2].

References

[1] A. Known and B. Betterknown. Fast reactions in latex conversion. *J Nonreproducible Results*, 24:319–34, 1978.

[2] A. Known and B. Betterknown. Fast reactions in latex conversion, a reprise. *J Nonreproducible Results*, 34:119–34, 1979.

A.7 fonts.tex

sans serif

roman

Serifenschrift

Serifenlose Schrift

`Maschinenschrift`

Medium

Halbfett

Aufrecht

Kursiv

Geneigt

KAPITAELCHEN

Hervorgehoben

Defaultschrift des Dokuments

A.8 footnote.tex

Dies ist nur ein kurzer[1] Test der Fussnotenverarbeitung.[2] Es ist auch moeglich, daß dies nicht so ist, aber dann kann entweder der Konverter[3] die Fussnoten nicht richtig verarbeiten.

[1] weil der Text sehr kurz ist

[2] sollte am Ende der Seite aufscheinen

[3] latex2rtf, Tex2rtf oder ltx2rtf

A.9 frac.tex

\frac

$$\frac{1}{2} = \frac{\alpha + \beta}{\gamma + \delta}$$

\over

$$\frac{1}{2} = \frac{\alpha + \beta}{\gamma + \delta}$$

\sqrt

$$z = \sqrt{x + y}$$

$$z = \sqrt{x^2 + y^2}$$

\frac und \sqrt

$$z = \frac{1}{\sqrt{x + y}}$$

\frac und \int

$$\int_0^\infty \frac{a}{\sqrt{x^2 + c}}\, dx$$

\frac und \sum

$$\sum_0^\infty \frac{a}{\sqrt{x^2 + c}}\, dx$$

\frac und \prod

$$\prod_0^\infty \frac{a}{\sqrt{x^2 + c}}\, dx$$

A.10 german.tex

ohne german.sty	mit german.sty
Ä	Ä
Ö	Ö
Ü	Ü
ä	ä
ö	ö
ü	ü
ß	ß

A.11 hoch.tex

$$\int_0^\infty \frac{a}{\sqrt{x^2+c}}\,dx$$

$$\iint_0^\infty x * y\,dx\,dy$$

$$\iiint_0^\infty x^4 + y + u\,dx\,dy\,du$$

$$\sum_{j<P} \prod_{\lambda} \lambda R(r-i)$$

$$\sum_0^\infty \frac{a}{\sqrt{x^2+c}}\,dx$$

$$a \sum_{0 \le i \le m} b\,E_i$$

$$\sum_{\substack{a \\ 0 \le i \le m}}^{b} \sum_c^d E_i$$

$$\prod_0^\infty \frac{a}{\sqrt{x^2+c}}\,dx$$

$$\prod_{*}^{*} \prod_{k}^{*}$$

$$\cfrac{1}{\sqrt{2}+\cfrac{1}{\sqrt{3}+\cfrac{1}{\sqrt{4}}}}$$

$$\frac{1}{\sqrt{2}+\frac{1}{\sqrt{3}+\frac{1}{\sqrt{4}}}}$$

$$lim_{y \to 1} f(y)$$

A.12 inhalt.tex + report.tex

Contents

Part I

das ist der erste teil

Chapter 1

das ist das erste Kapitel

Part II

das ist der zweite teil

Chapter 2

das ist das erste Kapitel zweiter Teil

Chapter 3

das ist das zweite Kapitel

ohne Numerierung

Kapitel ohne Nummerierung

A.13 liste.tex

The enumerate environment with labels

frog this is a frog

spider this is an arachnid

1. this has no label

   this has an empty label

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

The enumerate environment without labels

1. this is a frog
2. this is an arachnid
3. this is a bone

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

The itemize environment with labels

frog this is a frog

spider this is an arachnid

- this has no label

  this has an empty label

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

The itemize environment without labels

- this is a frog

- this is an arachnid
- this is a bone

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

The description environment with labels

frog this is a frog

spider this is an arachnid

this has no label

this has an empty label

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

The description environment without labels

this is a frog

this is an arachnid

this is a bone

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

Now to test nested environments. First the enumerate environment without labels

1. this is the first sublist
   (a) this 1a a frog
   (b) this 1b an arachnid
   (c) this 1c a bone
2. this is the second sublist
   - this a sub sub list

(a) this is 2 (a) frog

(b) this is 2 (b) an arachnid

(c) this is 2 (c) bone

- this is an arachnid
- this another sub sub list

(a) this is 2 (a) frog

(b) this is 2 (b) an arachnid

(c) this is 2 (c) bone

3. this is 3 an arachnid
4. this is 4 the third sublist

(a) this (a) is sub sub list

i. i. this is a frog

A. this A. is a frog

B. this B. is an arachnid

C. this C. is a bone

ii. this is ii. an arachnid

iii. this is iii. a bone

(b) this is an arachnid Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

(c) this another sub sub list Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

i. this i. is a frog

ii. this ii. is an arachnid

iii. this iii. is a bone

iv. this iv. is a frog

v. this v. is an arachnid

vi. this vi. is a frog

vii. this vii. is an arachnid

viii. this viii. is a bone

ix. this ix. is a frog

x. this x. is an arachnid

Here is some text so that the paragraph break after this is more clear than it is without this sentence which of course must be very long so that it will reach the end of the line and I see what is happening very clearly.

A.14 mathUmg.tex

mathematische Umgebungen

Formeln im fortlaufenden Text

gestartet mit \begin{math} und beendet mit \end{math}

$a = x + y$

gestartet mit \(und beendet mit \)

$t = t - 10 * \tau$

gestartet mit $ und beendet mit $

$p = r^2$

Abgesetzte Formeln

gestartet mit \[und mit \] beendet

$$z = a + b$$

gestartet mit \begin{displaymath} und mit \end{displaymath} beendet

$$r = 4 - t * z$$

gestartet mit \begin{eqnarray*} und mit \end{eqnarray*} beendet

$$l^2 = r + t * 6$$

gestartet mit $$ und mit $$ beendet

$$l = x - 45$$

abgesetzte Formeln mit Numerierung

gestartet mit \begin{equation} und mit \end{equation} beendet

$$c = c_0 + 34 * u \tag{A.3}$$

gestartet mit \begin{eqnarray} und mit \end{eqnarray} beendet

$$t = z * t^3 \tag{A.4}$$

A.15 tabular.tex

\begin{tabular} und \end{tabular}

left	center	right
left	center	right
left	center	right
left	center	right

Mitten im Text	left	center	right
	left	center	right
	left	center	right
	left	center	right

Zentriert

left	center	right
left	center	right
left	center	right
left	center	right

A.16 zeichen.tex

$$\amalg * \bigcirc \bigtriangledown \bigtriangleup \bullet \cap \cdot \circ \cup \dagger \ddagger \diamond \div \mp \odot \ominus \oplus \oslash \otimes \pm \setminus \sqcap \sqcup \star \times \triangleleft \triangleright \uplus \vee \wedge \wr$$

$$\approx \asymp \bowtie \cong \dashv \doteq \equiv \frown \geq \gg \in \leq \ll \mid \models \ni \parallel \perp \prec \preceq \propto \sim \simeq \smile \sqsubseteq \sqsupseteq \subset \subseteq \succ \succeq \supset \supseteq \vdash$$

$$\not< \not> \not= \not\approx \not\asymp \not\cong \not\equiv \not\geq \not\in \not\leq \not\ll \not\parallel \not\prec \not\preceq \not\sim \not\simeq \not\sqsubseteq \not\sqsupseteq \not\subset \not\subseteq \not\succ \not\succeq \not\supset \not\supseteq$$

$$\downarrow\Downarrow\hookleftarrow\hookrightarrow\leftarrow\Leftarrow\nleftarrow\not\leftarrow\leftrightarrow\Leftrightarrow\longleftarrow\Longleftarrow\longleftrightarrow\Longleftrightarrow$$

$$\mapsto\longrightarrow\Longrightarrow\mapsto\nearrow\nwarrow\to\Rightarrow\not\to\not\rightarrow\rightleftharpoons\searrow\swarrow\uparrow\Uparrow\updownarrow\Updownarrow$$

$$\bigcap\bigcup\bigodot\bigoplus\bigotimes\bigsqcup\biguplus\bigvee\bigwedge\coprod\int\oint\prod\sum$$

$$()[]\{\}\lfloor\rfloor\lceil\rceil\langle\rangle/\backslash|\|$$

$$\alpha\beta\gamma\Gamma\delta\Delta\epsilon\varepsilon\zeta\eta\theta\Theta\vartheta\iota\kappa\lambda\Lambda\mu\nu\xi\Xi\pi\Pi\varpi\rho\varrho\sigma\Sigma\varsigma\tau\upsilon\Upsilon\phi\Phi\varphi\chi\psi\Psi\omega\Omega$$

$$\aleph\angle\bot\clubsuit\diamondsuit\ell\emptyset\exists\flat\forall\hbar\heartsuit\Im\imath\infty\jmath\nabla\natural\neg\partial\wp\Re\sharp\spadesuit\surd\top\triangle\wp\|$$

x_1

x^5

x_3^7

x_{i_5}

x_{i^6}

x^{i^7}

$x_i{}^b$

x^{i_8}

${}_2x_i$

Literaturverzeichnis

[1] D. E. Knuth, *The TEXbook, Band A von Computers and Typesetting* (Addison-Wesley, Reading, 1986) ISBN 0-2011-3447-0.

[2] M. Goossens, F. Mittelbach, A. Samarin, *Der LATEX-Begleiter* (Addison-Wesley, München, 2000) ISBN 3-8273-1689-8.

[3] J. Schrod, *The Components of TEX* (March 1991)
http://www.ntg.nl/doc/schrod/etexkomp.pdf
http://www.uni-koeln.de/ftp/tex/info/components-of-TeX/etexkomp.tex.

[4] M. K. Dalheimer, *LATEX kurz & gut* (O'Reilly, Köln, 1998)
ISBN 3-89721-204-8.

[5] P. Taylor, The future of TEX. In *Proceedings of the 7th European TEX Conference, Prague* September 1992

[6] F. Cremer, *Das kleine TEXBuch* (5. 7. 1993)
http://www.ntg.nl/doc/cremer/texbuch.pdf
http://www.fmi.uni-passau.de/archive/tex/.

[7] rtf online references, (Version 1.6)
http://msdn.microsoft.com/library/en-us/dnrtfspec/html/rtfspec_1.asp

Tabellenverzeichnis

Abbildungsverzeichnis