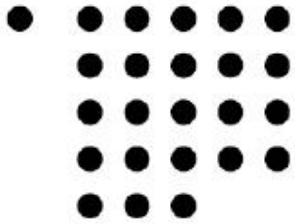


Fachhochschule Köln
University of Applied Sciences Cologne
Fakultät für Informations-, Medien-
und Elektrotechnik
Institut für Nachrichtentechnik
Labor für Informatik

Überarbeitung der Dokumentation der Bachelorarbeit

Thema: Entwicklung eines TeXmacs-to-XML-Parsers

Student: Sebastian Zittermann
Matrikelnummer: 11046747
Referent: Prof. Dr. phil. Gregor Büchel
Koreferent: Prof. Dr. rer. nat. Heiko Knopse

Inhaltsverzeichnis

1 Einführung

1.1 Beschreibung

Diese Arbeit beschäftigt sich mit der Realisierung eines Parsers, der mathematische Beweise aus „TeXmacs“ in eine zu erstellende XML-Struktur umwandelt. Die Struktur der XML-Datei wird durch eine ebenfalls zu entwickelnde DTD-Datei vorgegeben. TeXmacs ist ein WYSIWYG-Editor zum Beschreiben von mathematischen Texten. Die Arbeit findet im Rahmen der Forschungsgruppe Naproche (siehe Kapitel 2: Projektumfeld) statt.

Langfristig soll das Ergebnis der Arbeit in Naproche eingesetzt werden, um mathematische Beweise besser auf ihre logischen Aussagen zu überprüfen. Zu klären ist also, ob die logische Abfolge des erfassten TeXmacs-Beweises richtig in eine XML-Struktur umgesetzt wurde und alle dokumentierten Voraussetzungen und Beweisschritte im XML-Dokument syntaktisch richtig sind.

Die Anwender, die direkt in TeXmacs mit Beweisen arbeiten, sollen eine Rückmeldung erhalten. Bei einer fehlerhaften Rückmeldung soll auch die Stelle im Beweis gekennzeichnet werden, an der der Fehler aufgetreten ist. Dieser kann so schnellstmöglich korrigiert werden. Tritt keine Fehlermeldung auf, wird auch das als Information ausgegeben.

Diese Bachelorarbeit befasst sich mit der Realisierung des ersten Teils: Aus einer von TeXmacs erzeugten SCM-Datei soll mit einem Java-Programm die Struktur des mathematischen Textes ermittelt werden; es soll also eine Trennung von mathematischen Formelanteilen (Formeln, Gleichungen/Ungleichungen, mathematische Symbole, usw.) und beschreibenden Texten durchgeführt werden. Diese Struktur wird mit den Regeln einer DTD in ein XML-Format so umgewandelt, dass das resultierende XML-Dokument von den nachfolgenden Modulen des Naproche-Projekts bearbeitet werden kann.

1.2 Gliederung

Die Dokumentation der Arbeit ist in zwei inhaltliche Bereiche aufgeteilt. Zuerst werden die theoretischen Grundlagen und die gegebene Software mit ihren jeweiligen Dateiformaten beschrieben. Es folgt der Realisierungsteil, der die Konzepte der Entwicklung und deren Implementierung sowie einen Ausblick auf die künftige Entwicklung der Software enthält.

1.3 Anforderungen

Die Aufgabenstellung der Bachelorarbeit umfasst folgende Bereiche:

- Weiterentwicklung einer DTD zur Abbildung von mathematischen Beweisen
- Entwicklung einer Software zur Übertragung eines TeXmacs-Beweises in die durch die DTD vorgegebene XML-Struktur
- Erfassung von aussagekräftigen Beweisen zur Überprüfung der Leistungen der Software

Die Mindestanforderungen an die Software sind:

- Einlesen der TeXmacs-Daten aus den von TeXmacs-erzeugten SCM-Dateien
- Die TeXmacs-SCM-Klammerstruktur abbilden
- Bekannte Schlüsselwörter/Ebenen in der Struktur der SCM-Datei in ein möglichst MathML nahes XML-Format übertragen
- Übertragung des XML-nahen-Formates in die durch die DTD vorgegebene XML-Struktur
- Validierung des erstellten XML-Dokuments

2 Projektumfeld: Naproche

Naproche (**na**tural language **pro**of **che**cker) ist ein Projekt der Universitäten Bonn und Duisburg-Essen in Kooperation mit der Fachhochschule Köln (vgl. [l_Naproche]). Das Ziel des Projektes ist es, ein System zu entwickeln, das einen mathematischen Text auf Integrität und Korrektheit überprüft.

Üblicherweise sind solche mathematischen Texte in einer Mischung aus formalen Symbolen und umgangssprachlichen Ausdrücken geschrieben. Beispielsweise findet man in einigen Beweisen die kurze Darstellung „$\sqrt{2} \notin \mathbb{Q}$“, die andererseits auch umgangssprachlich in der Form „Wurzel 2 ist irrational“ erscheint.

Naproche analysiert diese Texte soweit, dass beide Ausdrucksformen erkannt und mit ihnen gleichwertig gearbeitet werden kann. Daher kann das Projekt als Hilfe sowohl für die Verfasser als auch die Leser von mathematischen Dokumenten eingesetzt werden. Ziel ist es, dass beide Gruppen jederzeit überprüfen können, ob ein aktuelles Dokument logische Fehler enthält oder ob innerhalb eines Beweises jede Schlussfolgerung - auch wenn sie in natürlicher Sprache erfolgt - richtig und damit der Beweis insgesamt korrekt ist.

Im Rahmen des Naproche-Projektes wurden schon einige Abschlussarbeiten geschrieben (unter anderem [Kolev] und [Fisseni]), bzw. werden parallel zu dieser verfasst; Herr Kühlwein arbeitet an der Überprüfung der Richtigkeit der einzelnen Beweisschritte.

Da es sich um ein vielschichtiges Projekt handelt, wurden die Zuständigkeiten aufgeteilt:
Der mathematische Aspekt wird hauptsächlich am mathematischen Institut der Universität Bonn erforscht. Der Schwerpunkt des linguistischen Teils wird an der Universität Duisburg-Essen bearbeitet. An der Fachhochschule Köln liegt der Hauptteil im informationstechnischen Bereich.

3 Theorie

In der Mathematik gibt es verschiedene Gebiete (unter anderem Lineare Algebra, Analysis und Aussagenlogik), in denen aus wenigen Axiomen viele Sätze abgeleitet werden. Informationen über die Inhalte stammen aus den Quellen [Schweizer], [Forster] und [Waerden]. Axiome sind elementare Bestandteile der Mathematik, die in der jeweiligen Struktur als allgemeingültig angenommen werden. Hieraus können neue Sätze und Erkenntnisse gewonnen werden. Diese müssen dann durch Beweise auf die Axiome oder schon bewiesene Sätze zurückgeführt werden.
Beweise sind im Allgemeinen wie folgt strukturiert: Man formuliert die konkreten Voraussetzungen eines Satzes, stellt die zu beweisende Behauptung auf und führt anschließend den Beweis durch.
Es gibt drei unterschiedliche vorherrschende Beweisverfahren einen mathematischen Beweis durchzuführen:

- Direkter Beweis
- Indirekter Beweis
- Vollständige Induktion

3.1 Direkter Beweis

In einem direkten Beweis geht man von einem bereits bekannten Satz oder den gegebenen Voraussetzungen aus. Diese werden mit Hilfe von Äquivalenz- bzw. Implikationsumformungen zu neuen Sachverhalten Schritt für Schritt verändert, bis sich schließlich die aufgestellte Behauptung ergibt. Bei einer Äquivalenzumformung ist der neue Sachverhalt gleichwertig zur Ursprungsaussage; diese Umformung kann also in beide Richtungen erfolgen. Eine Implikationsumformung stellt dagegen nur eine Wenn-Dann-Folgerung dar, die nur in eine Richtung gültig ist.

> Der direkte Beweis wird am Beispiel des Additionstheorems für sin(x) und cos(x) erläutert:

Satz: Additionstheoreme

Vor.: $x, y \epsilon \mathbb{R}$; komplexe e-Funktion

Beh.:

$$\cos(x+y) = \cos x \cos y - \sin x \sin y,$$
$$\sin(x+y) = \sin x \cos y + \cos x \sin y$$

Bew.: Aus der Funktionalgleichung der Exponentialfunktion

$$e^{j(x+y)} = e^{jx+jy} = e^{jx} e^{jy}$$

ergibt sich mit der Eulerschen Formel

$$\begin{aligned}\cos(x+y) + j\sin(x+y) &= (\cos x + j \sin x)(\cos y + j \sin y) \\ &= (\cos x \cos y - \sin x \sin y) + j(\sin x \cos y + \cos x \sin y)\end{aligned}$$

Vergleicht man Real- und Imaginärteil, erhält man die Behauptung.

q.e.d.

Die erste Angabe des Satzes ist der Titel. Es folgen die Voraussetzung und die Behauptung. Der Beweis beginnt mit einer bekannten Gleichung [hier: „Funktionalgleichung der Exponentialfunktion“]. Diese wird schrittweise [hier mit Hilfe der Eulerschen Formel] äquivalent umgeformt, so dass man eine Gleichungskette erhält. Hieraus kann man problemlos auf die Behauptung schliessen. Das „q.e.d.“(quod erat demonstrandum: was zu beweisen war) schließt den Beweis ab.

3.2 Indirekter Beweis

Bei einem indirekten Beweis wird die Behauptung in ihr kontradiktorisches Gegenteil umgekehrt. Diese Negation wird dann ähnlich wie beim direkten Beweis mit den nötigen Äquivalenz- bzw. Implikationsumformungen zu einem Widerspruch geführt.

Durch diesen Widerspruch der falschen Annahme ist bewiesen, dass die ursprüngliche Aussage richtig sein muss.
Ein klassisches Beispiel für den indirekten Beweis ist der Nachweis, dass $\sqrt{2}$ irrational ist:

Satz: Irrationalität von $\sqrt{2}$

Vor.: Die Beschreibung der Menge der rationalen Zahlen $\mathbb{Q}$ in folgender Form:

$$\mathbb{Q} = \left\{ \frac{p}{q} | p \epsilon \mathbb{Z} \wedge q \epsilon \mathbb{N} \wedge \mathrm{ggT}(p,q) = 1 \right\}$$

Beh.: $\sqrt{2} \not\epsilon \mathbb{Q}$, d.h. $\sqrt{2}$ ist irrational

Bew.: durch Widerspruch:

Annahme: Beh.: $\sqrt{2} \epsilon \mathbb{Q}$

Beweisschritte:

(1) Dann gibt es $p \epsilon \mathbb{Z}$ und $q \epsilon \mathbb{N}$, so dass $\sqrt{2} = \frac{p}{q}$, mit $\mathrm{ggT}(p,q) = 1$.

(2) Quadrieren ergibt $2 = \frac{p^2}{q^2}$.

(3) Daraus folgt $2q^2 = p^2 = pp$.

(4) Also ist 2 Teiler von p.

(5) Daher gilt $p = 2\alpha \,\mathrm{mit}\, \alpha \epsilon \mathbb{Z}$.

(6) Einsetzen in (3) ergibt $2qq = 4\alpha$.

(7) Damit gilt $qq = 2\alpha$.

(8) Also ist 2 Teiler von q.

(9) Damit ist $\mathrm{ggT}(p,q) = 2$.

(10) Das ist ein Widerspruch zu (1); also gilt $\sqrt{2} \not\epsilon \mathbb{Q}$.

q.e.d.

Wie schon beim direkten Beweis werden zuerst Satztitel, Voraussetzung und Behauptung genannt. Der erste Beweisschritt ist die Formulierung der Annahme, die sich aus der Negation der Behauptung ergibt [hier: „ $\sqrt{2}$ ist rational“]. Diese Negation wird im nächsten Schritt formalisiert. Weitere Umformungen führen schließlich zu einem Widerspruch zur Annahme. Abschließend erfolgt die Feststellung, dass aufgestellte Behauptung gültig ist.

3.3 Vollständige Induktion

Der direkte und indirekte Beweis sind Formen des deduktiven Beweisverfahrens, also eine strenge Abfolge von logischen und mathematischen Schlüssen. Das Verfahren einer (unvollständigen) Induktion wird in der Naturwissenschaft benutzt, um aus Einzeltatsachen wie Messungen allgemeingültige Erkenntnisse zu gewinnen. Diese sind jedoch mit einem Unsicherheitsfaktor belastet.
In der Mathematik garantiert das Prinzip der vollständigen Induktion die Gültigkeit einer Aussage $A(n)$ für alle natürlichen Zahlen $n \epsilon \mathbb{N}$.
Das Beweisverfahren der vollständigen Induktion erfolgt in zwei Teilen: der Induktionsverankerung und dem Induktionsschluss. In der Induktionsverankerung wird die Behauptung für eine natürliche Zahl n_0 nachgewiesen, damit gewährleistet ist, dass die Aussage für einen Startwert n_0 Gültigkeit besitzt. In dem Induktionsschluss wird bewiesen: wenn eine Aussage für eine natürliche Zahl n richtig ist, dann ist sie auch für ihren Nachfolger $n + 1$ gültig. Der Induktionsschluss hat oft die gleiche Struktur wie der direkte Beweis: Die Aussage für n wird zur Voraussetzung, die Aussage für $n + 1$ wird zur Behauptung, die im Induktionsbeweis direkt nachgewiesen wird.
Im Induktionsschluss können auch Beweise zu einem Widerspruch geführt werden; dann entspricht die Struktur der des indirekten Beweises.
Ein Beispiel für die vollständige Induktion ist die Summenformel:

> Satz: Summenformel für die ersten n natürlichen Zahlen
>
> Vor.: $n \epsilon \mathbb{N}$
>
> Beh.: Für n gilt: $1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$.
>
> Bew.: Wir setzen zur Abkürzung $S(n) = 1 + 2 + 3 + \ldots + n$ und zeigen die Gleichung
>
> $S(n) = \frac{n(n+1)}{2}$ durch vollständige Induktion.
>
> Induktionsverankerung: $n = 1$. Es ist $S(1) = 1$ und $\frac{1(1+1)}{2} = 1$ also gilt die Formel für $n = 1$.
>
> Induktionsschluss: $n \rightarrow n + 1$.

Induktionsvor.: Wir nehmen an, dass $S(n) = \frac{n(n+1)}{2}$ gilt.

Induktionsbeh.: Wir müssen zeigen, dass daraus die Formel $S(n+1) = \frac{(n+1)(n+2)}{2}$ folgt.

Induktionsbew.: Zunächst gilt: $S(n+1) = S(n) + (n+1)$,

wegen der Induktionsvoraussetzung folgt hieraus: $\frac{n(n+1)}{2} + n + 1 = \frac{(n+1)(n+2)}{2}$.

q.e.d.

Zuerst wird die Behauptung für jede natürliche Zahl n formuliert. Im Beweis wird die Gleichung dann vereinfacht und das Verfahren der vollständigen Induktion ausgewählt. Die Induktionsverankerung wird hier für $n = 1$ bewiesen. Für den Induktionsschluss $n \to n + 1$ werden anschließend Voraussetzung und Behauptung genannt. Der Beweis wird nun wie gewohnt in direkter Weise durchgeführt.

4 Werkzeuge / Hilfsmittel

In diesem Bereich werden die einzelnen Softwareprodukte, mit denen sich die Bachelorarbeit beschäftigt, kurz aufgeführt und näher beschrieben. Ein weiterer Unterpunkt erläutert die Datenformate, die rund um diese Software vorhanden sind bzw. genutzt werden.

4.1 Software

4.1.1 TeXmacs

Das Forschungsprojekt Naproche benutzt als mathematischen WYSIWYG-Editor TeXmacs. TeXmacs gehört zur TeX-Familie (vgl. [I_TeXmacs] und [I_LaTeX]). TeX selber bietet ein System, mit dem man verschiedene Textbausteine hintereinander setzen kann. Diese Textbausteine können auch aus Makros bestehen, die beispielsweise mathematische Symbole beinhalten. Bei der Erzeugung eines druckreifen Dokumentes einer TeX-Datei werden diese Maakros dann in Textsymbole im Fließtext umgewandelt.

TeXmacs ist speziell für wissenschaftliche und mathematische Dokumentationen entwickelt worden. Es können mathematische Formeln, Ausdrücke und Symbole in großer Auswahl in einen Fließtext leicht geschrieben werden. Außerdem kann das Programm Funktionen graphisch darstellen und leicht in einen Text einbinden.

Bestandteile des Programms sind unter anderem ein Text-Editor für mathematische Formeln und ein Werkzeug um Präsentationen zu erstellen. TeXmacs kann neben dem TeXmacs-Format insbesondere XML- und Scheme-Formate erzeugen, die im Unterpunkt „Datenformate“ näher beschrieben werden.
Diese erzeugbaren Formate sind alle so aufgebaut, dass sie leicht maschinell verarbeitet werden können. Sie bieten alle eine interne Datenstruktur, in der die Nutzdaten (z.b. stehen die Nutzdaten bei XML, sofern sie keine Attributdaten sind, zwischen einem öffnenden und schliessendem Tag) eindeutig erkennbar und damit schnell wei-

terverarbeitet werden können. Weiterhin gibt es die nützliche Möglichkeit, die Daten als PDF-Datei zu exportieren. PDF ist ein plattform übergreifendes Dateiformat zum Lesen von Dokumenten, auf das hier nicht weiter eingegangen wird.

4.1.2 Java

Java ist eine objektorientierte Programmiersprache. Sie wird von der Firma Sun Microsystems für alle Interessenten kostenlos zur Verfügung gestellt (vgl. [Java]).
Java bietet durch seine Plattformunabhängigkeit die Möglichkeit auf jedem Betriebssystem gleichbleibend zu funktionieren; dies stellt einen großen Vorteil in der Praxis dar. Die Voraussetzung für ein lauffähiges Java-Programm ist die Installation der Java Virtual Maschine. Sie liegt zwischen dem Betriebssystem und Java. Diese beinhaltet eine Laufzeit-Umgebung, auf der betriebssystemunabhängig die Java-Programme ausgeführt und kompiliert werden.

Das Kompilieren funktioniert nicht so wie in einigen anderen gängigen Sprachen wie zum Beispiel C++. Der Quellcode wird hier zuerst in einen maschinennahen „Java-Bytecode“ umgewandelt. Auf dem Computer des Anwenders wird dann dieser Bytecode in der Java Laufzeit-Umgebung ausgeführt. Die Startzeit eines Programms wird dadurch allerdings verzögert.

Sowohl für Einsteiger als auch für professionelle Entwickler ist Java gleichermassen interessant. Es ist z.B. problemlos möglich, einfache Browserapplets, aber auch komplexe Datenbankmodule zu programmieren.

Eine objektorientierte Programmierung wird durch Java ebenfalls unterstützt. Man erstellt gezielte Datengruppen und fasst sie in Objekten zu Klassen zusammen. Es existieren bereits viele vorgefertigte Klassen, die man benutzen und bei Bedarf anpassen kann (z.B. die Klassen des SAX-Parsers).

4.1.3 Parser

Das Wort „parsen“ bedeutet „zerlegen“ bzw. „analysieren“. Die Aufgabe eines Parsers in einem Computerprogramm ist es, ein Dokument oder eine Datei nach bestimmten Regeln aufzuteilen, zu analysieren und eine Aussage über die Wohlgeformtheit zu treffen (vgl. [l_selfHTML]). Die Gesamtheit der Regeln, die den Aufbau des zu parsenden Dokuments bestimmen, bildet eine Grammatik. Die Grammatik eines XML-Dokuments kann durch eine DTD gegeben werden (vgl. Kapitel 4.2.3). Ist das Dokument im Sinne dieser Grammatik korrekt, ist es valide.

In dieser Arbeit wird in Kombination mit einem SAX-Parser ein erstelltes XML-Dokument auf seine Wohlgeformtheit und in Kombination mit einer DTD auf seine Validität überprüft.
Der Unterschied zwischen der Wohlgeformtheit und der Validität eines Dokuments liegt in der Prüfungsart des Dokuments. Bei der Wohlgeformtheit wird die Struktur des Dokumentes überprüft (z.B. ob es zu jedem öffnenden Tag auch ein schliessendes Tag auf der richtigen Ebene gibt), bei der Validitätsprüfung wird die vorgegebene Grammatik in die Strukturüberprüfung mit einbezogen. Ein valides Dokument ist wohlgeformt; ein wohlgeformtes Dokument ist nicht zwingend valide.

4.2 Dateiformate

4.2.1 SCM

Die von TeXmacs erstellten Daten werden in SCM-Dateien exportiert und können daher mit der Programmiersprache Scheme (vgl. [l_scheme]) bearbeitet werden. Scheme ist eine nicht objektorientierte Programmiersprache, die volle Funktionalität bietet, aber nicht all zu sehr verbreitet ist. Sie ist wie eine programmierbare Programmiersprache aufgebaut. Es ist möglich, mit eigenen Makros diese Sprache um eigene Methoden zu erweitern.

Das Scheme aus TeXmacs liefert ein durch geschachtelte Klammern strukturiertes Format. Hier ein Beispiel:

(document (TeXmacs “1.0.6.5“) (style (tuple “generic“)) (body (document “Satz von Rolle“ “Bemerkung: Der Satz von Rolle besagt, dass zwischen zwei Nullstellen einer differenzierbaren Funktion eine Nullstelle der Ableitung liegt.“ (concat “Voraussetzung: Sei “ (math “a <less> b“) “ und “ (math “f : [a,b] <rightarrow> <bbb-R>“) “ eine stetige Funktion mit “ (math “f(a) = f(b)“) “. “) (concat “Die Funktion “ (math “f“) “ sei in “ (math “]a,b[“) “ differenzierbar. “) (concat “Beh.: Dann existiert ein “ (math “<xi> <in>]a,b[“) “ mit “ (math (concat “ f“ (rprime “’“) “(<xi>) = 0“)) “.“) (concat “Bew.: Falls “ (math “f“) “ konstant ist, ist der Satz trivial. Ist “ (math “f“) “ nicht konstant, so ergibt es ein “ (math (concat “x“ (rsub “0“) “ <in>]a,b[“)) “ mit “ (math (concat “f(x“ (rsub “0“) “) <gtr> f(a)“)) “ oder “ (math (concat “f(x“ (rsub “0“) “) <less> f(a)“)) “. Dann wird das absolute Maximum (bzw. Minimum) der Funktion “ (math “f: [a,b] <rightarrow> <bbb-R> “) “ in einem Punkt “ (math “<xi> <in>]a,b[“) “ angenommen. Nach dem Satz über lokale Maxima/Minima ist “ (math (concat “f“ (rprime “’“) “(<xi>) = 0“)) “. q.e.d.“))) (initial (collection (associate “language“ “german“))))

Scheme benutzt eine Präfixnotation: Hinter jedem öffnenden Klammersymbol der TeXmacsstruktur folgt eine Anweisung, die Aufschluss über den Inhalt der Klammer gibt. Der Inhalt kann je nach Anweisung wiederum aus weiteren Klammern mit beliebig vielen Unteranweisungen bestehen. Die im Editor zu erkennenden Text- und Symbolbausteine werden zwischen zwei Anführungszeichen geschrieben. Falls es sich um ein mathematisches Symbol handelt, wird dieses in spitzen Klammern gesetzt (z.B. <in> für das Symbol „ist Element von“).

4.2.2 CSV

CSV-Dateien sind Textdateien mit einem platzsparenden, klar strukturierten Aufbau. CSV steht für „Comma Seperated Values“, was soviel wie „von Komma getrennte Werte“ heißt. Die Dateneinträge sind durch ein Semikolon getrennt. In jeder neuen Zeile steht ein neuer Datensatz. Sie können Tabellen und Listen in unterschiedlicher

Länge enthalten. Für komplexe Strukturen sind XML-Dokumente besser geeignet.
Im Rahmen der Bachelorarbeit wurden diese Art von Dateien für verschiedene Bereiche eingesetzt: Es wird von der Software eine Datei „Satzliste.csv“ erstellt, in der alle vom Programm erfassten Beweise eine Satz-ID zugeordnet wird. Anhand eines Auszugs aus dieser Liste wird die CSV-Struktur leicht klar:

```
...
6;lokalesMaximum
7;zwischenwertsatz
8;Summenformel
9;Bernoulli
...
```

Die Zahl links neben dem Semikolon ist die vom Programm zugeordnete Satz-ID. Jeder ID wird ein Beweisname zugeordnet. Dieser Name steht rechts neben dem Semikolon.
Weiterhin tritt die CSV-Struktur auch in einer XML-Datei im „Statementtyp“ auf:

```
0;gl
1;ungl
0;gl,gl,ungl
0;gl,ungl
```

Dieser beinhaltet die Information, ob ein prädikatenlogischer Ausdruck vorkommt (1) oder nicht (0). Ein Gleichungstyp wird vom Programm ermittelt und als Tupelfolge durch das Semikolon getrennt ergänzt. Hier wird zwischen einer Gleichung und einer Ungleichung unterschieden. Es können auch mehrere Gleichungssymbole in einem Ausdruck vorkommen. Diese werden dann durch Kommata getrennt und hintereinander geschrieben und bilden so das Tupel, wie man an dem Beispiel erkennen kann.

4.2.3 DTD

Möchte man einen regelmäßigen Datenaustausch vornehmen (z.B. zwischen Lieferant und Kunde) muss beiden Seiten das Datenformat bekannt sein. Die DTD-Dateien haben die Aufgabe, Grammatiken für XML-Dateien festzulegen (vgl. [l_selfHTML] und [l_w3DTD]). Die Abkürzung DTD steht für „Dokument-Typ Definition“. Wenn also beide Seiten ihre XML-Dokumente anhand einer DTD aufbauen, können keine Komplikationen auftreten. Folgendes Beispiel verdeutlicht das:

```
<!ELEMENT Satz (SatzInformationen,Voraussetzungen*,Behauptung,Beweis)>
<!ATTLIST Satz
ID CDATA #REQUIRED
>
<!ELEMENT SatzInformationen (Satztitel,Mathemteilgebiet?,Quelle?,Bemerkung*)>
<!ELEMENT Satztitel (#PCDATA)>
<!ELEMENT Mathemteilgebiet (#PCDATA)>
<!ELEMENT Quelle (#PCDATA)>
<!ELEMENT Bemerkung (#PCDATA)>
```

Anhand dieses Auszugs einer im Rahmen der Bachelorarbeit erstellten DTD kann man die grundlegenden Eigenschaften einer DTD erkennen. Das Element „Satz“ besteht aus „Satzinformationen“, beliebig vielen Voraussetzungen (auch keine Voraussetzungen sind möglich), einer Behauptung und einem Beweis. Bis auf die Voraussetzung müssen die Einzelelemente genau einmal vorkommen. Es muss weiterhin eine ID zu jedem Satz mit übergeben werden. Der Datentyp der ID wird in der DTD als „CDATA“ näher spezifiziert; in diesem Fall wird die ID mit Zahlenwerten gefüllt.

Die „SatzInformationen“ bestehen aus einem Satztitel, optional einem mathematischen Teilgebiet, einer Quelle wie z.B. einem Autor und so vielen Bemerkungen wie erforderlich sind. Die vollständige DTD wird in dem Unterpunkt 5.3 näher und detaillierter erläutert.

4.2.4 XML

XML steht für EXtensible Markup Language; es ist ähnlich wie HTML eine Markup-Sprache (vgl. [I_selfHTML] und [I_w3XML]). Die Markup-Sprachen haben den Vorteil, dass sie sowohl für den Computer als auch für den Menschen leicht lesbar sind. EXtensible steht hierbei für die Erweiterbarkeit der Sprache. Während HTML bezüglich einer festen Menge von zulässigen Markups definiert ist, kann XML jederzeit leicht erweitert werden um beispielsweise neue mathematische Formeln aufzunehmen. XML enthält Markups (sogenannten „Tags“), wie man im folgenden Beispiel sehen kann:

<SATZ ID = “1“>
<Satz-Informationen>
<Titel>
<TEXT>Satz: Dreiecksungleichung für reelle Zahlen</TEXT>
</Titel>
</Satz-Informationen>
...
</SATZ>

Ein “Tag“ ist eine Kennung. Dabei gibt es öffnende und schliessende Tags. Zu jedem öffnenden Tag muss es auch einen zugehörigen geschlossenen in der selben Ebene geben. Im Beispiel des Auszuges muss der „Text“-tag beendet werden, bevor der „Titel“-Tag geschlossen wird, da er sich auf einer höheren Ebene befindet.
XML-Dokumente lassen sich mit Hilfe von Parsern auf ihre Wohlgeformtheit überprüfen. Diese Validierungsüberprüfung stellt fest, ob alle Richtlinien, die in der DTD getroffen wurden, auch wirklich eingehalten wurden.
Mit Hilfe von XML lassen sich verschiedene andere Sprachen definieren, sogenannte XML-Anwendungen. Beispiele hierfür sind Extensible HTML (XHTML), WML und MathML. Letztere wurde in dieser Arbeit angewandt, wie im folgenden Kapitel erläutert wird.

4.2.5 MathML

MathML ist ein XML-Standard, mit dem man mathematische Formeln und komplexe Ausdrücke darstellen kann (vgl. [I_MathML]). Sie besteht aus zwei Teilsprachen, Presentation MathML und Content MathML, die beliebig kombinierbar sind.

Presentation MathML ist hauptsächlich für die graphische Gestaltung von mathematischen Ausdrücken gedacht. Der konkrete Ausdruck im Präsentationsformat für die Gleichung $(a + b)^2$ sieht wie folgt aus:

```
<msup>
<mfenced>
<mrow>
<mi>a</mi>
<mo>+</mo>
<mi>b</mi>
</mrow>
</mfenced>
<mn>2</mn>
</msup>
```

Die einzelnen mathematischen Elemente werden in der höchsten Ebene einfach hintereinander geschrieben. Das Potenzieren findet in einer vorherigen Ebene statt.

Content MathML beschreibt ausschließlich die strukturelle Funktion einer Formel. Die Formel $(a + b)^2$ lässt sich in diesem Format so darstellen:

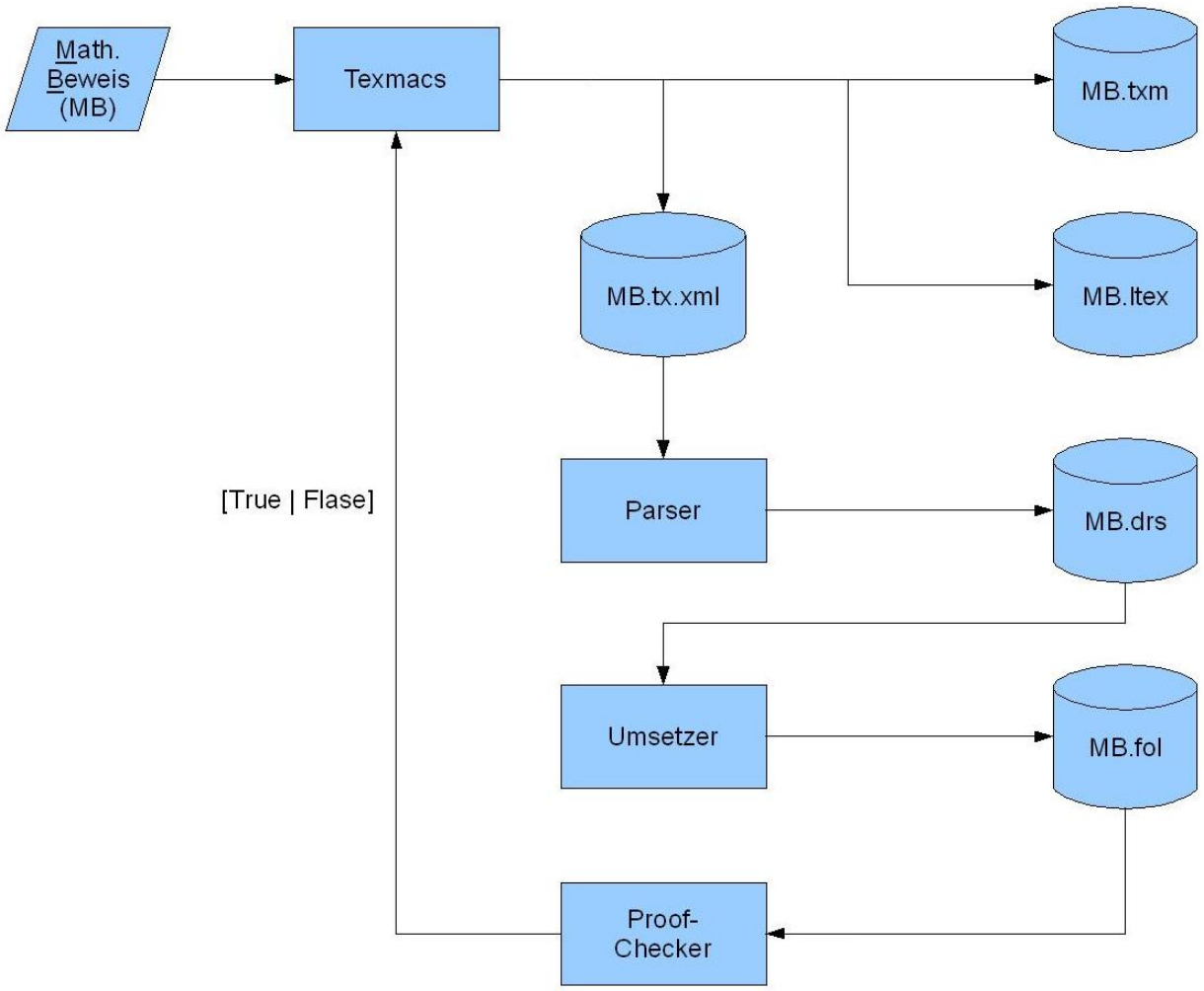
```
<apply>
<power/>
<apply>
<plus/>
<ci>a</ci>
<ci>b</ci>
</apply>
<cn>2</cn>
</apply>
```

Hier werden die Elemente funktional zusammengefasst. Die Variablen a und b sind Elemente der Addition (<plus/>), während <power/> das Quadrat dieser Addition wiedergibt.
In der Bachelorarbeit wurden einige in Content MathML definierte Ausdrücke genutzt, um einzelne Texmacs-Elemente XML-fähig darzustellen.

5 Planung / Entwurf

5.1 Strukturdarstellung

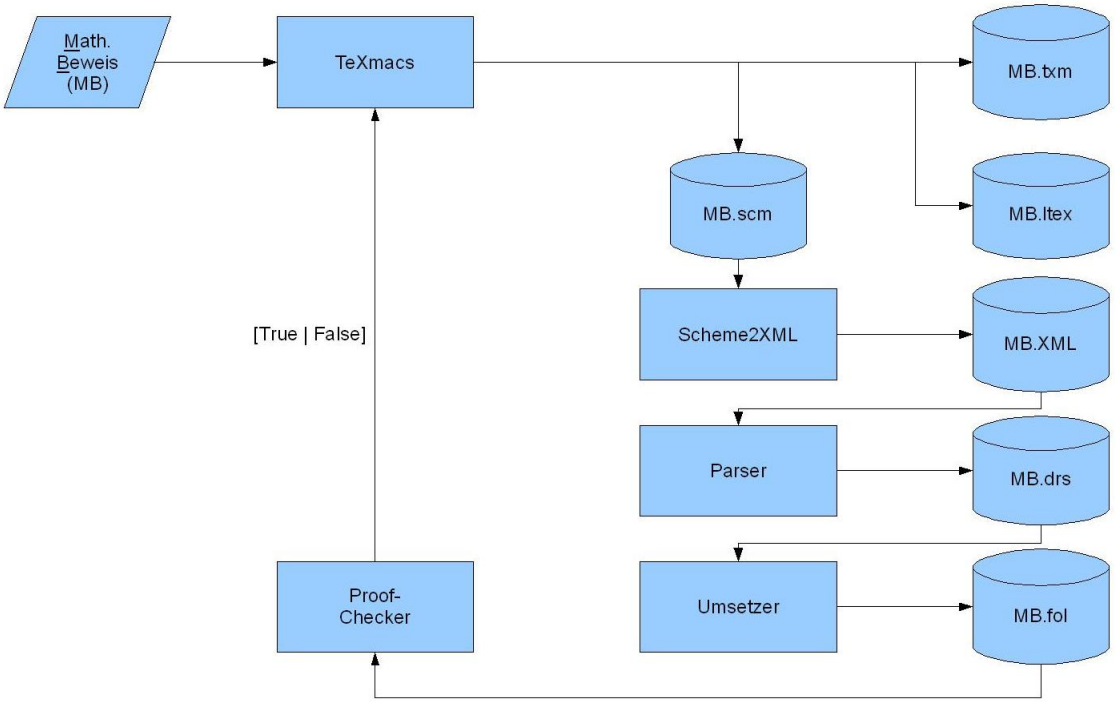
Im Projekt NAPROCHE wird zurzeit (Stand 2008, siehe [Material]) nach folgendem Datenflussplan gearbeitet:



Ein Beweis wird von einem Autor mit dem Programm TeXmacs editiert. TeXmacs kann jetzt diesen Beweis in verschiedene Formate exportieren.
TXM-Dateien sind ähnlich aufgebaut wie LaTex-Dokumente, aber nicht kompatibel mit LaTex. Sie stellen den Beweis in einer Baumstruktur dar. LTEX-Dateien entsprechen dem LaTex-Format. Diese Dateien können problemlos mit LaTex geöffnet und bearbeitet werden. TeXmacs kann ein „spezielles“ XML-Format erzeugen.

Diese XML-Dateien werden mit Hilfe eines Parsers, der von Herrn Kolev im Rahmen seiner Magisterarbeit entwickelt wurde, in das DRS-Format umgewandelt.
DRS steht für Diskurs Repräsentation Struktur und ist eine klar definierte Struktur, die mit dem Umsetzer in das FOL-Format (First Order Logic; Prädikatenlogik 1. Stufe) übertragen wird.
Die FOL-Datei wird über den Proof-Checker analysiert. Ist der Beweis in sich schlüssig, wird der Wert „true“ an TeXmacs geschickt. Ist das nicht der Fall wird „false“ gesendet. Die Module Parser, Umsetzer und Proof-Checker sind in Prolog programmiert.

Der Ansatz der Bachelorarbeit ist es, einen alternativen Weg vorzubereiten:



Anstelle des TeXmacs-XMLs soll ein XML anhand einer gegebenen DTD erzeugt werden. Dieses XML soll als Grundlage das ebenfalls von TeXmacs exportierbare SCM-Format benutzen und es nach den Gegebenheiten der DTD entsprechend umzuwandeln. Langfristig muss auch der an das Projekt angrenzende Parser von Herrn Kolev mit der Struktur der DTD synchronisiert werden, damit dieser mit dem erzeugten XML-Format umgehen kann.

5.2 Analyse der SCM-Daten

Schreibt man einen Beweis in TeXmacs, besteht die Möglichkeit, diesen in ein SCM-Format zu exportieren. Diese Arbeit setzt auf dem SCM-Format auf. Dieses Format wird als Quelle für die Umwandlung in das Ziel-XML-Format genutzt. Hierzu muss man die Daten in der SCM-Datei auf ihre Struktur hin analysieren. Der Aufbau einer SCM-Datei wird anhand eines Beispiels näher erläutert:

> (document (TeXmacs “1.0.6.5“) (style (tuple “generic“)) (body (document “Satz von Rolle“ “Bemerkung: Der Satz von Rolle besagt, dass zwischen zwei Nullstellen einer differenzierbaren Funktion eine Nullstelle der Ableitung liegt.“ (concat “Voraussetzung: Sei “ (math “a <less> b“) “ und “ (math “f : [a,b] <rightarrow> <bbb-R>“) “ eine stetige Funktion mit “ (math “f(a) = f(b)“) “. “) (concat “Die Funktion “ (math “f“) “ sei in “ (math “]a,b[“) “ differenzierbar. “) (concat “Beh.: Dann existiert ein “ (math “<xi> <in>]a,b[“) “ mit “ (math (concat “ f“ (rprime “’“) “(<xi>) = 0“)) “.“) (concat “Bew.: Falls “ (math “f“) “ konstant ist, ist der Satz trivial. Ist “ (math “f“) “ nicht konstant, so ergibt es ein “ (math (concat “x“ (rsub “0“) “ <in>]a,b[“)) “ mit “ (math (concat “f(x“ (rsub “0“) “) <gtr> f(a)“)) “ oder “ (math (concat “f(x“ (rsub “0“) “) <less> f(a)“)) “. Dann wird das absolute Maximum (bzw. Minimum) der Funktion “ (math “f: [a,b] <rightarrow> <bbb-R> “) “ in einem Punkt “ (math “<xi> <in>]a,b[“) “ angenommen. Nach dem Satz über lokale Maxima/Minima ist “ (math (concat “f“ (rprime “’“) “(<xi>) = 0“)) “. q.e.d.“))) (initial (collection (associate “language“ “german“))))

Folgende Eigenschaften lassen sich hieraus erkennen:

- Es sind sehr viele Anführungszeichen vorhanden.
- Es gibt viele runde Klammerelemente, die auch in einander verschachtelt sein können.
- Die wesentlichen Elemente des Beweises (z.B. Titel, Behauptung, Beweis) lassen sich wieder finden.

- einzelne Beweisschritte werden durch mathematische Formeln und Symbole wiedergegeben, man findet auch natürlichsprachliche Sequenzen.
- Einige mathematische Symbole werden XML-ähnlich in Tags dargestellt(z.B. <in>).

Diese Eigenschaften müssen jetzt im Einzelnen analysiert werden:

- Da für Textbereiche immer ein Anführungszeichen-Paar nötig ist, weisst eine ungerade Anzahl der Anführungszeichen also auf ein nicht geschlossenes Paar und damit auf einen Fehler hin.
- Die für den Beweis relevanten Daten stehen immer in einer „(body ...)“-Struktur.
- Der Body-Bereich enthält eine verschachtelte Klammerstruktur, die ausschließlich aus Klammerpaaren besteht. Dabei ist zu beachten, das auch runde Klammern vorkommen, die als Elemente des Beweises vorhanden sind, z.B. bei „f(a)“.
- Es muss unterschieden werden, ob ein Klammerpaar in der SCM-Struktur vom Benutzer eingegeben oder von TeXmacs automatisch erzeugt wurde; z.B. (rsub...) von TeXmacs und f(a) vom Benutzer.
- Der Beweistext der Datei wird nach Schlüsselwörtern, z.B. „Behauptung“, durchsucht, um zu erkennen, ob die Datei die wesentlichen Beweismerkmale enthält.
- Die Struktur muss in mathematische Bereiche und Textelemente aufgeteilt werden.
- Die XML-ähnlichen Tags müssen in ein MathML-Tag umgewandelt werden.

5.3 DTD

Im Rahmen der Bachelorarbeit wird die DTD-Datei „Satz.dtd“ entwickelt. Als Anregung für den Entwurf diente einerseits eine Vorlage von Prof. Dr. phil. Gregor Büchel

(vgl. [Material]), zwei Magisterarbeiten aus der Forschungsgruppe Naproche ([Fisseni] und [Kolev]) und andererseits eigene Überlegungen anhand unterschiedlicher Mathematik-Fachbücher.

Aus der Arbeit von Herrn Kolev wurde eine BNF übernommen. BNF steht für Backus-Naur-Form und ist eine Metasprache, die benutzt wird, um kontextfreie Grammatiken darzustellen. Unter anderem lassen sich die gängigen Programmiersprachen alle in dieser Syntax beschreiben. Als Beispiel für den Aufbau wird hier die BNF zur „Expected Proof Structure“ (erwartete Beweisstruktur) aufgeführt:

```
<text>                  ::= [<open-assumption>]*,
                            [<definition>]*,
                            [<proof>]*;
<proof>                 ::= “theorem.“,
                            [<free-assumption>]*,
                            [<definition>]*,
                            [statement]+,
                            “proof.“,
                            [<close-assumption> | statement | <lemma>]+,
                            “q.e.d.“;
<lemma>                 ::=“lemma.“,
                           [<assumption>]*,
                           [<definition>]*,
                           [statement]+,
                            “proof.“,
                           [<close-assumption> | statement]+,
                            “q.e.d.“;
<open-assumption>       ::=<assumption>;
<closed-assumption>     ::=<assumption>,
                           [statement]*,
                           <assumption-close>;
<defintion>             ::=“define“,
                           statement,
                           [“iff“ | “if and only if“],
                           statement;
<assumption>            ::=[ “assume that“ | “assume for a contradiction that“ |
                             “let“ | “consider“],
                           statement;
<assumption-close>      ::=“thus“,
                           statement;
```

Auffallend bei der BNF sind die vielen Oder-Verbindungen. Es werden auch recht viele konstante sprachliche Ausdrücke festgelegt, wie z.B. „iff“. Sollten diese ergänzt

oder geändert werden, müsste man die komplette BNF anpassen.
Eine recht komplexe und ausführliche DTD wurde in der Magisterarbeit von Herrn Fisseni dem Standard OMDoc folgend entwickelt. Diese enthält noch wesentlich mehr Elemente (über 30), die meistens durch umfangreiche Oder-Verknüpfungen verbunden werden. Festgelegte sprachliche Ausdrücke kommen, genau wie in dem Beispiel der BNF von Herrn Kolev, recht häufig vor. Zudem treten auch rekursive Aufrufe auf. Diese DTD finden ist im Anhang 11.2.3 aufgeführt.

Die OMDoc-DTD wurde nach dem Bottom-Up-Verfahren entwickelt. Im Bottom-Up-Verfahren werden zunächst einzelne Programmelemente (z.B. Klassen und Methoden) definiert. Auf dieser Basis werden immer größere Elemente aus den schon vorhandenen Elementen zusammengesetzt. Das System ist dann komplett, wenn alle vorkommenden Daten verarbeitet werden können. Bei der Implementierung ist darauf zu achten, dass die einzelnen Softwarekomponenten so programmiert sind, dass diese auch für andere Projekte wiederverwendet werden können. Die Realisierung kann theoretisch auch schon während der Entwicklungsphase beginnen.

Im Gegensatz zu dem Bottom-Up-Verfahren gibt es noch das Top-Down-Verfahren. Hier beginnt man mit der übergeordneten Systemstruktur, um sich zuerst einmal einen Überblick über das gesamte System zu verschaffen. Die einzelnen Teilbereiche werden zunächst grob oder sogar umgangssprachlich formuliert. Sie werden im Laufe des Entwicklungsprozesses immer genauer, bis sie letztendlich die kompletten Spezifikationen beinhalten. Der Schwerpunkt bei diesem Designverfahren liegt also auf dem Verständnis des Aufbaus der Software. Mit der Realisierung kann erst recht spät begonnen werden.

In der Praxis werden die beiden Ansätze häufig kombiniert. Das Verständnis des Gesamtsystems ist notwendig um die einzelnen Unterelementene richtig zuordnen zu können. Diese sollen so programmiert werden, dass sie jederzeit wiederverwendet werden können.

Ziel der DTD-Datei dieser Bachelorarbeit ist es, mit möglichst wenigen Oder- Verknüpfungen auszukommen. Rekursive Aufrufe sollen minimal gehalten werden. Die

festgelegten Textelemente sollen mit Hilfe einer leicht pflegbaren XML-Datei von der Software eingelesen und überprüft werden. All diese Vorgaben sollen dazu führen, dass die DTD leicht zu lesen und zu verstehen ist. Die DTD soll nach dem Top-Down-Designverfahren entwickelt werden.

5.4 Richtlinien

Die folgenden Richtlinien für die Dokumentation mathematischer Beweise in TeXmacs sollen vom Anwender eingehalten werden, um den Beweis in ein gültiges XML-Format übertragen zu können:

- Dem Beweis / Satz muss ein Name gegeben werden.
- Das Dokument ist in einzelne Bereiche aufzuteilen:
  - Satzinformationen
  - Voraussetzung(en)
  - Behauptung (Die explizite Kennzeichnung einer Behauptung ist notwendig.)
  - Beweisanfang
  - Beweisende (q.e.d.)
- Man soll versuchen, möglichst wenig verschiedene mathematische Bereiche in TeXmacs zu öffnen. Ein Beispiel: Die Gleichung „x = y“ soll erfasst werden. Theoretisch könnte man für jedes Symbol einen eigenen Formelbereich öffnen (hier beispielhaft durch das Zeichen „$“ dargestellt): „$x$ $=$ y“. Die Formel kann auch in einem einzigen Formelbereich geschrieben werden, was auch eindeutig zu bevorzugen ist: „$x = y$“
- Bei der Dokumentation des Beweises sollte man darauf achten, „sprachliche“ Kommentare nicht in die Bereiche der Formel zu schreiben: „$x \leq y$ wobei x gerade ist“ sollte zu „$\$x \leq y\$$ wobei x gerade ist“ werden.

6 Realisierung

6.1 Umsetzung in Java

Das Programm „Scheme2XML“ wurde mit der Programmiersprache Java implementiert. Wie schon erwähnt besteht das eigentliche Quelldatenformat SCM aus einer Baumstruktur. Die bei der Implementierung erstellten Klassen sind so konzipiert worden, dass auf eine interne Darstellung der Baumstruktur verzichtet wurde. Alternativ wurden folgende grundlegende Variablen benutzt: Die SCM-Datei wird während der Verarbeitung in einer Stringvariable „zeile“ zwischengespeichert. Eng mit dieser Zeile ist das mehrdimensionale Integer-Array „klammern[][]“ verbunden. Die Klammerstruktur wird in diesem Array gespeichert.
Ein Beispiel hierfür ist der Auszug aus der SCM-Datei „SatzvonRolle.scm“:

```
46          0                          2       1                        42     41
(document (TeXmacs "1.0.6.5") (style (tuple "generic")) (body (document "Satz

von Rolle" "Bemerkung: Der Satz von Rolle besagt, dass zwischen zwei

Nullstellen einer differenzierbaren Funktion eine Nullstelle der Ableitung
         8                                  3                                4
liegt." (concat "Voraussetzung: Sei " (math "a <less> b") " und " (math "f :
                                                                 7        5
[a,b] <rightarrow> <bbb-R>") " eine stetige Funktion mit " (math "f(a) =
  6
f(b)") ". ") (concat "Die Funktion " (math "f") " sei in " (math "]a,b[") "

differenzierbar. ") (concat "Beh.: Dann existiert ein " (math "<xi>  <in>

]a,b[") " mit " (math (concat " f" (rprime "'") "(<xi>) = 0")) ".") (concat

"Bew.: Falls " (math "f") " konstant ist, ist der Satz trivial. Ist " (math "f")

"nicht konstant, so ergibt es ein " (math (concat "x" (rsub "0") " <in>

]a,b[")) " mit " (math (concat "f(x" (rsub "0") ") <gtr> f(a)")) " oder "

(math (concat "f(x" (rsub "0") ") <less> f(a)")) ". Dann wird das absolute

Maximum (bzw. Minimum) der Funktion " (math "f: [a,b] <rightarrow> <bbb-R> ")

" in einem Punkt " (math "<xi> <in> ]a,b[ ") " angenommen. Nach dem Satz über

lokale Maxima/Minima ist " (math (concat "f" (rprime "'") "(<xi>) = 0")) ".
          45       44          43
q.e.d.")))  (initial (collection (associate "language" "german"))))
```

Die Zahlen über den öffnenden Klammersymbolen sind zur Veranschaulichung der

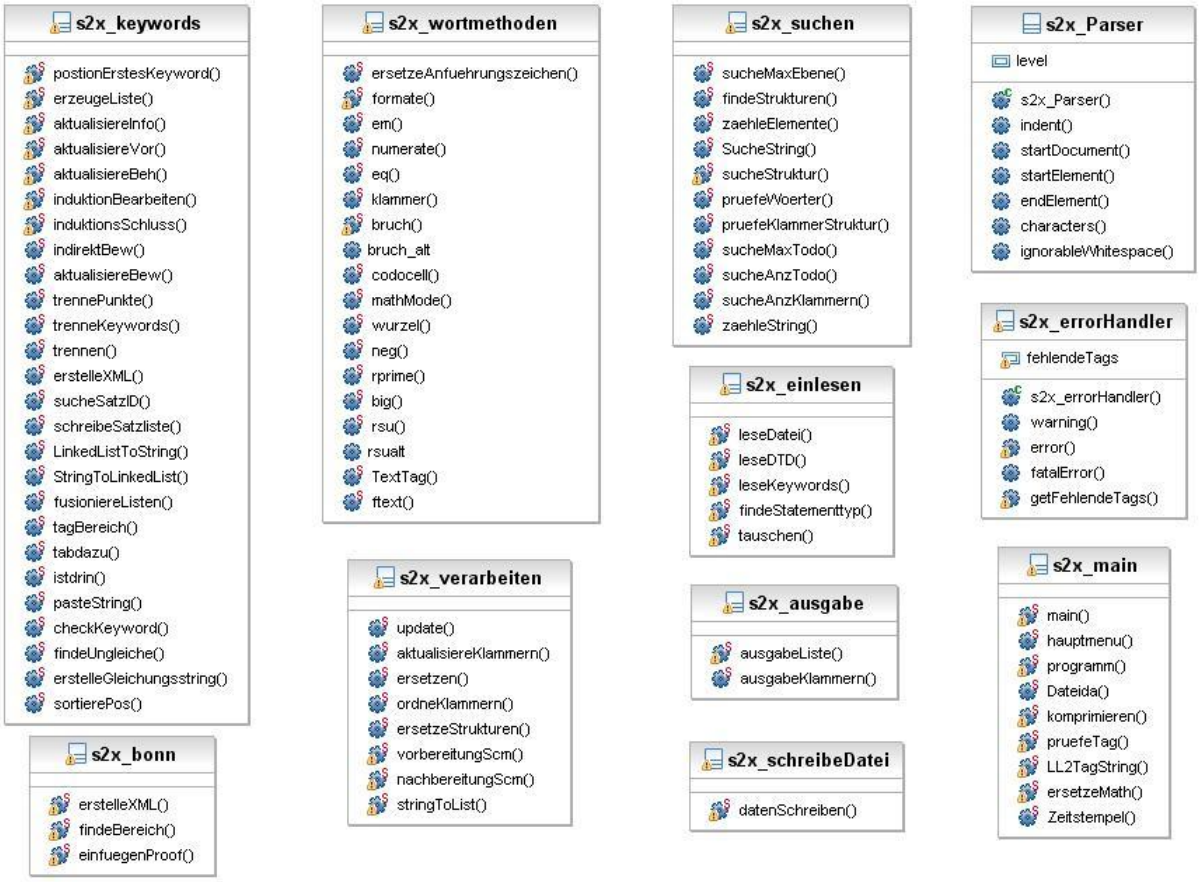
folgenden Erklärung zusätzlich eingefügt worden. Sie entsprechen den „Nummern“ in der unten stehenden Liste, die die Klammerstruktur dieser SCM-Datei ausschnittsweise darstellt:

```
K: 0  Anfang: 11    Ende: 30    Ebene: 1
K: 1  Anfang: 39    Ende: 55    Ebene: 2
K: 2  Anfang: 32    Ende: 56    Ebene: 1
K: 3  Anfang: 273   Ende: 291   Ebene: 4
K: 4  Anfang: 301   Ende: 339   Ebene: 4
K: 5  Anfang: 379   Ende: 381   Ebene: -1
K: 6  Anfang: 386   Ende: 388   Ebene: -1
K: 7  Anfang: 371   Ende: 390   Ebene: 4
K: 8  Anfang: 243   Ende: 396   Ebene: 3
[...]
K: 41 Anfang: 64    Ende: 1156  Ebene: 2
K: 42 Anfang: 58    Ende: 1157  Ebene: 1
K: 43 Anfang: 1180  Ende: 1210  Ebene: 3
K: 44 Anfang: 1168  Ende: 1211  Ebene: 2
K: 45 Anfang: 1159  Ende: 1212  Ebene: 1
K: 46 Anfang: 1     Ende: 1213  Ebene: 0
```

In der Klammerstruktur „klammern[][]“ wird in dem ersten des mehrdimensionalen Arrays die laufende Nummer hochgezählt. Der zweite Klammer kann drei verschiedene Elemente beinhalten:
An erster Stelle wird die Position der aufgehenden Klammer in der „zeile“ gespeichert. Die Position der schließenden Klammer steht an zweiter Stelle. Die dritte Stelle enthält die Ebene der Klammer. Eine „0“ an der Stelle bedeutet, dass sich das Klammerpaar auf der untersten Ebene befindet. Sollte als Inhalt eine „-1“ hinterlegt sein, handelt es sich um eine „inhaltliche“ Klammer aus dem Beweistext, also eine Klammer, die nichts mit der TeXmacs-Struktur zu tun hat. Dies trifft beispielsweise auf die Klammerpaare 5 und 6 zu: „f(a)“ und „f(b)“.

6.2 Klassendiagramm

Zur Übersicht über alle implementierten Klassen und Methoden wird im Folgenden das Klassendiagramm abgebildet:

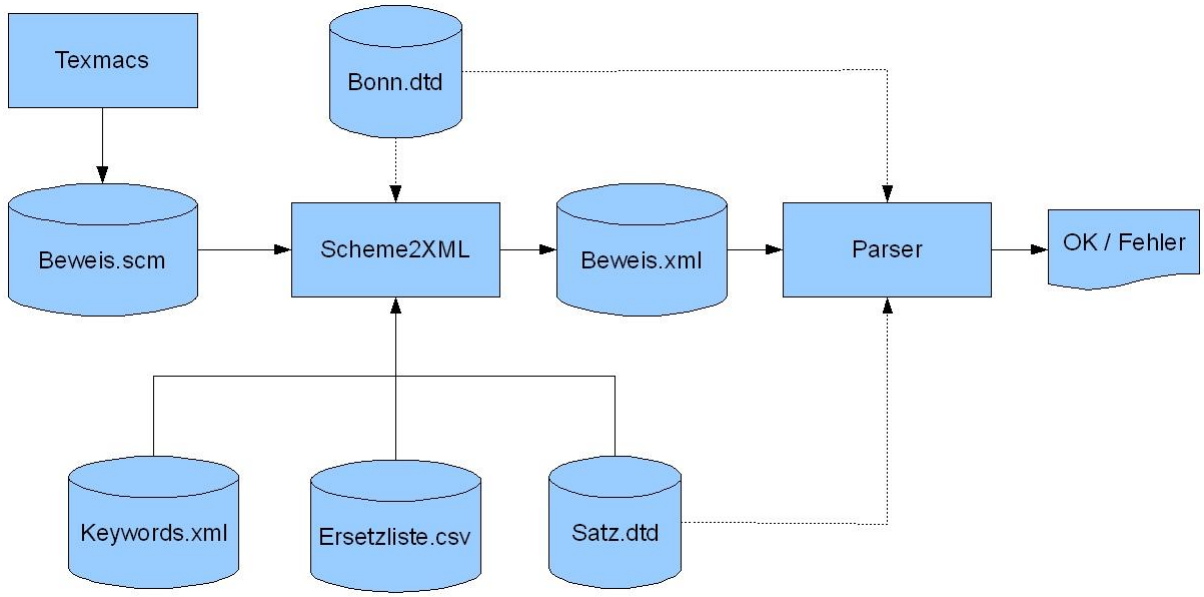


Das Programm startet mit der Klasse „s2x_main“. Im Main stehen weiterhin eine Menu-Methode, die die Eingaben des Anwenders steuert, eine Methode, die überprüft ob eine Datei vorhanden ist und die Methode, die mit den übergebenen Werten das eigentliche Programm startet. Die externen Datenquellen werden in den Methoden der Klasse „s2x_einlesen“ eingelesen. Die Klasse „s2x_verarbeiten“ enthält die Methoden, die zur Umwandlung der Scheme-Struktur in das XML- ähnliche Format benötigt werden. Die Analyse der Klammerstruktur erfolgt über die Klasse „s2x_suchen“. Zum Austausch der TeXmacs- Strukturschlüsselwörter benutzt das Programm die Klasse „s2x_wortmethoden“. Die Umwandlung der Daten in die von einer DTD vorgegebenen XML-Struktur findet entweder über die Methoden der Klasse „s2x_keywords“ oder die der Klasse „s2x_bonn“ statt. Die Bildschirmausgabemethoden befinden sich in der Klasse „s2x_ausgabe“. Eine fertige XML-Struktur wird mit Elementen der Klasse „s2x_schreibeDatei“ geschrieben und kann über die Klas-

se „s2x_parser“ validiert werden.

6.3 Modulübersicht

Eine Modulübersicht der Software „Scheme2XML“ wird mit folgendem Datenflussplan gegeben:



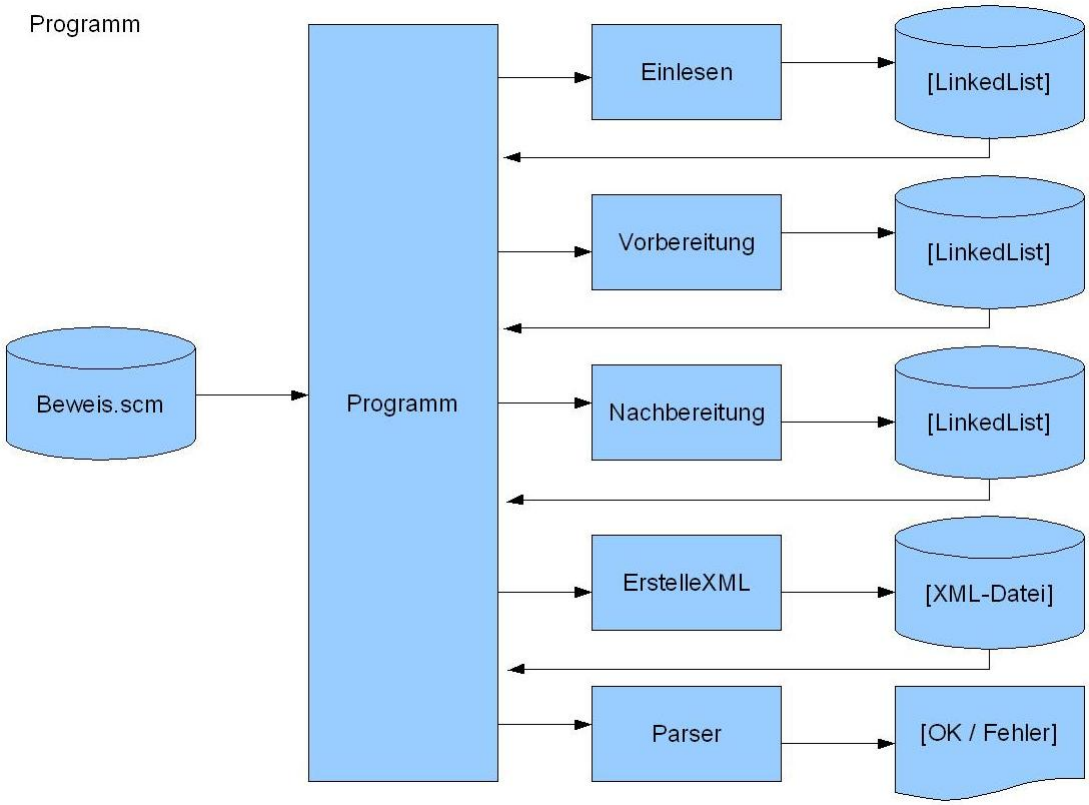
Der in TeXmacs erfasste Beweis wird als scm-Datei exportiert. Das Java-Programm „Scheme2XML“ ließt diese Datei ein. Neben der Datei gibt aus außerdem noch weitere Datenquellen, auf die im Laufe des Programms zurückgegriffen werden kann: Keywords.xml, Ersatzliste.csv, Satz.dtd und Bonn.dtd. „Scheme2XML“ erstellt daraus eine XML-Datei, die anschliessend mit einem Parser validiert wird. Die Module sind folgendermaßen implementiert:

- Modul Parser: durch die Klassen s2x_Parser und s2x_errorHandler
- Modul Scheme2XML: durch alle übrigen Klassen des Klassendiagramms (Kap. 6.2)

Die Dateien Bonn.dtd, Satz.dtd, Keywords.xml und Ersatzliste.csv befinden sich im Anhang.

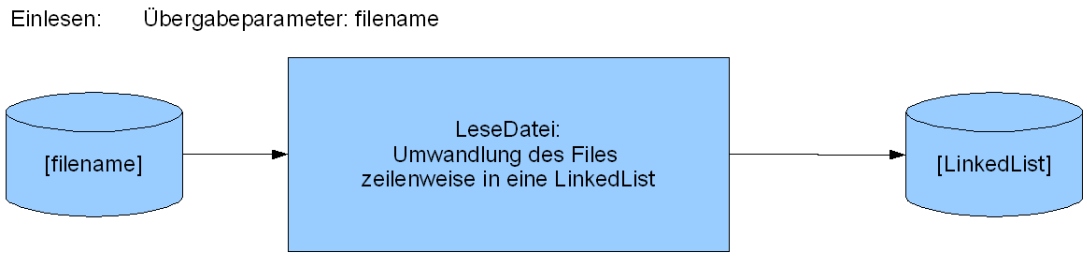
6.4 Hauptmodule

Der Programmplan sieht wie folgt aus, wobei das Modul Programm durch die Methode „programm()“ der Klasse s2x_main implementiert wurde:



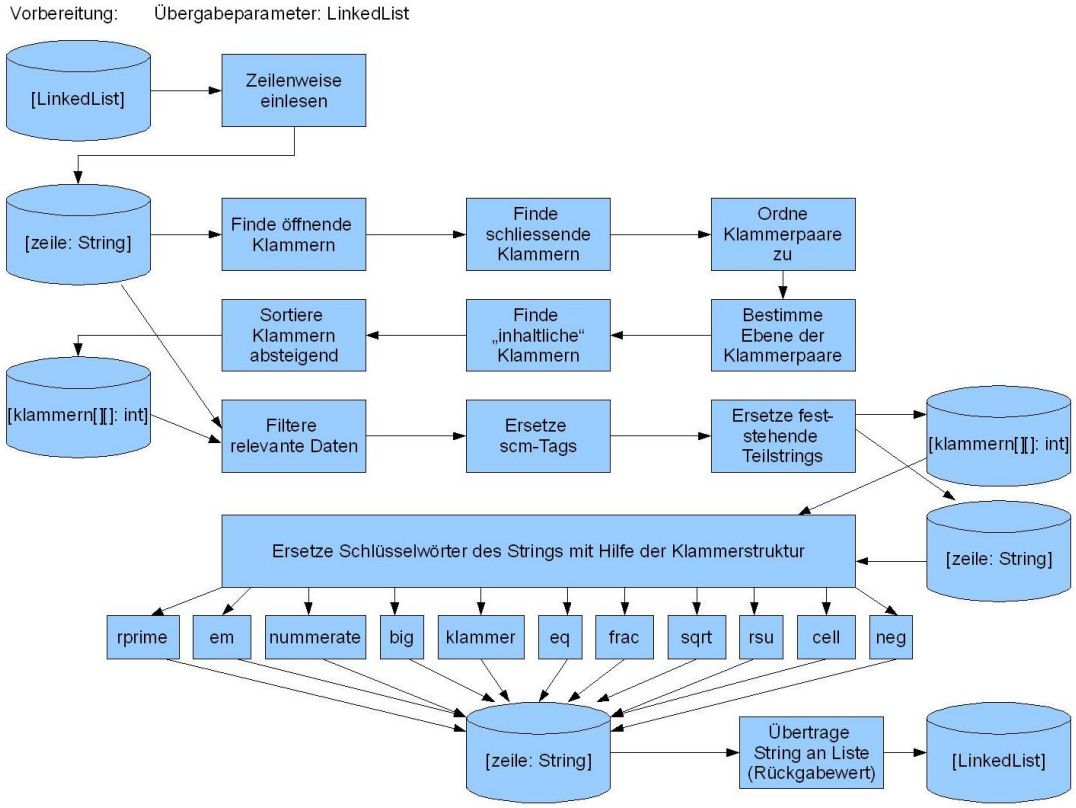
Die Hauptmodule dieser Software sind die grundlegenden Methoden, die in der Main- Funktion aufgerufen werden und ihrerseits Nebenmodule aufrufen:

- Einlesen [Methode „leseDatei()“ der Klasse „s2x_einlesen“]:



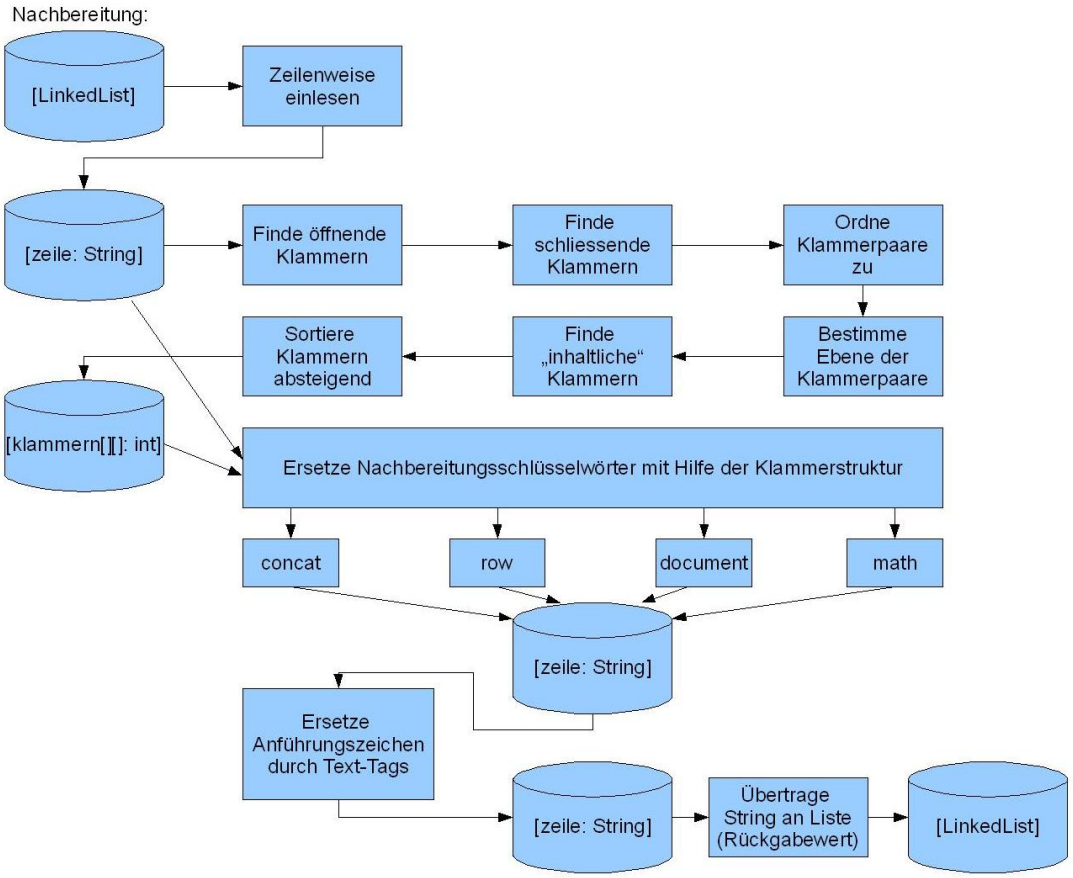
  Der Übergabeparameter ist ein String mit dem Pfand und dem Namen der Datei. Die Datei wird zeilenweise in eine LinkedList eingelesen und zurückgegeben.

- Vorbereitung der SCM [Methode „vorbereitungScm()“ der Klasse „s2x_verarbeiten“]:



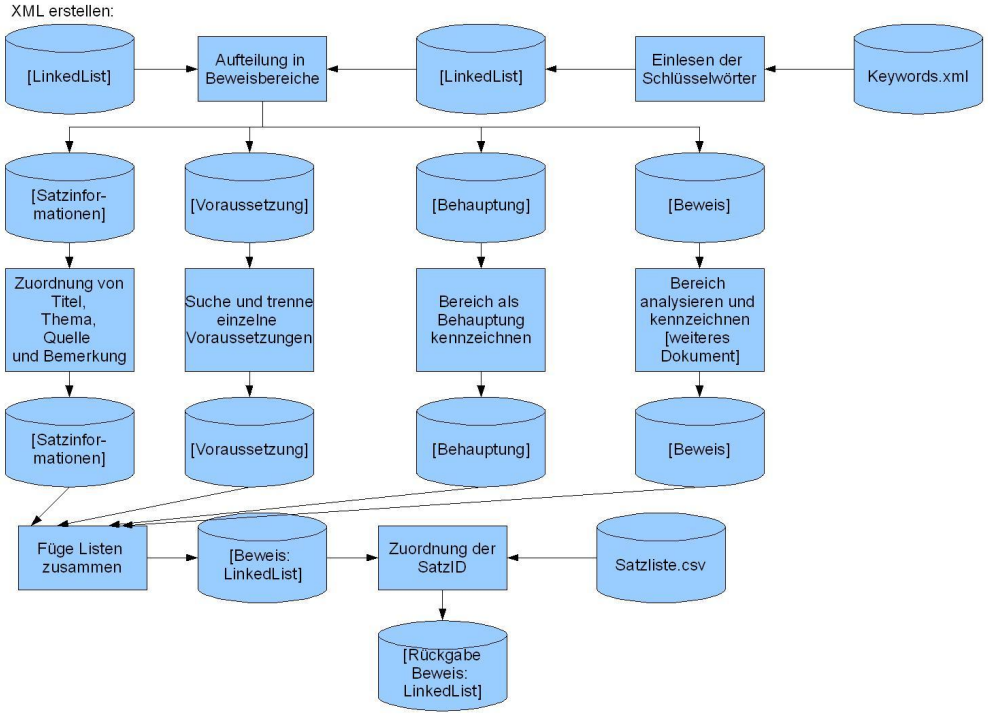
Der Übergabeparameter ist eine LinkedList. Aus dieser Liste wird zunächst die Klammerstruktur über die Methode „aktualisiereKlammern()“ der Klasse „s2x_verarbeiten“ ermittelt. In dieser Methode werden die Hilfsmethoden „findeStrukturen()“, „pruefeKlammerStruktur()“ der Klasse „s2x_suchen“ und „ordneKlammern()“ der Klasse „s2x_verarbeiten“ verwendet. Die Daten und die Klammerstruktur werden auf die relevanten Einträge im scm-body reduziert. Anschliessend werden durch den Aufruf diverser „Wort“- Methoden (siehe Nebenmodule) die Elemente, die bei der Vorbereitung nicht direkt erkannt und bearbeitet werden konnten, durch XML-Tags ersetzt. Die so entstandende Liste wird zurückgegeben.

- Nachbereitung der SCM [Methode „nachbereitungScm()“ der Klasse „s2x_verarbeiten“]:



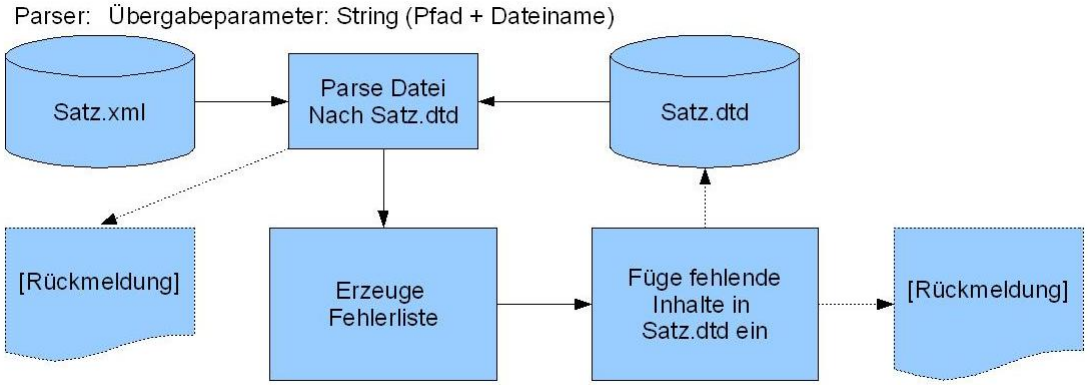
Wie auch bei der Vorbereitung sind die Übergabeparameter eine LinkedList und ein String. Hier werden ebenfalls zunächst die Klammerstruktur über die Methode „aktualisiereKlammern()“ der Klasse „s2x_verarbeiten“ ermittelt. Anschliessend werden durch den Aufruf der Wort- Methoden (siehe Nebenmodule) die Elemente, die bei der Vorbereitung nicht direkt erkannt und bearbeitet werden konnten, durch XML-Tags ersetzt. Bevor die so entstandende Liste zurückgegeben wird, werden alle noch vorhandenen Anführungszeichen durch eine weitere Wort- Methode ersetzt.

- XML erstellen [Methode „erstelleXML()“ der Klasse „s2x_keywords“]:



Die Übergabeparameter sind eine LinkedList und ein String. Es werden zuerst die Schlüsselwörter, die zum Trennen der vier Beweisbereiche (Satzinformationen, Voraussetzung, Behauptung, Beweis) benötigt werden, über die Methode „leseKeywords()“ der Klasse „s2x_einlesen“ eingelesen und ihre Positionen in der Liste mit der Methode „positionErstesKeyword()“ der Klasse „s2x_keywords“ bestimmt. Mit ihnen werden die verschiedenen Beweisbereichslisten aus der XML-Liste (Methode „erzeugeListe()“ der Klasse „s2x_keywords“) erstellt. Diese werden entsprechend der vier Bereiche eines Beweises durch Methoden (siehe Nebenmodule) bearbeitet und letztendlich mit der Methode „fusioniereListen()“ der Klasse „s2x_keywords“ zusammengefasst und aus Gründen der Übersichtlichkeit mit der Methode „tabdazu()“ aus der Klasse „s2x_keywords“ eingerückt. Der übergebene String enthält den Namen der Beweisdatei. Er wird benötigt, um die ID des Satzes mit der Methode „sucheSatzID()“ der Klasse „s2x_keywords“ zu bestimmen. Der Rückgabewert ist die zusammengefasste Liste der XML-Elemente mit Nutzdaten des Satzes und seines Beweises.

- Parsen [Klassen „s2x_Parser“ und „s2x_errorHandler“]:



  Zum Parsen wird ein Standard- SAX- Parser verwendet, dessen entsprechende Methoden des Default- und Error- Handler als Callback- Funktion aufgerufen werden. Der Parser wird in der Methode „programm()“ der Klasse „s2x_main“ aufgerufen. Sollten beim Parsen fehlende Tags gefunden werden, werden diese in einer Liste zwischengespeichert, die anschliessend mit der Methode „komprimieren()“ der Klasse „s2x_main“ so zusammengefasst wird, das jeder fehlende Tag genau einmal enthalten ist. Die Liste wird anschliessend zur Zeichenkette umgewandelt (Methode „LL2TagString()“ der Klasse „s2x_main“) und mit der Methode „ersetzeMath()“ der Klasse „s2x_main“ die entsprechenden Stellen der Datei „Satz.dtd“ aktualisiert (Methode „datenSchreiben()“ der Klasse „s2x_schreibeDaten“). Das Programm ist in diesem Punkt selbstlernend. Wird der „fehlerhafte“ Beweis ein zweites Mal geparst, können die Tags nun zugeordnet werden. Wurde Beispielsweise der Tag „<in>“ im SCM-Format gefunden, wird er nach Programmdurchlauf in die DTD übernommen.

Die Arbeitsweise der Hauptmodule wird im Unterpunkt „Programmablauf - Beispiel“ anhand des Beispiels „Dreiecksungleichung für reelle Zahlen“ näher erläutert.

6.5 Nebenmodule

In dieser Bachelorarbeit bezieht sich der Großteil der Nebenmodule auf die Umwandlung von SCM-Strukturen in XML-Formate.

Diese Nebenmodule werden bei der Vor- und der Nachbereitung der SCM-Datei und der Umwandlung nach den Regeln der Satz.dtd in das endgültige XML-Format aufgerufen. Aus zeitlichen Gründen lag der Schwerpunkt während der Realisierung der entsprechenden Methoden auf ihrer Funktionalität. Eine Konsequenz daraus ist, dass sich die Übergabewerte und Aufrufe der Methoden nur in wenigen Fällen unterscheiden. Die Methoden „bruch()“ und „wurzel()“ bekommen wie die meisten Methoden den Zeilenstring, das Klammerarray und die Position des Klammerpaares übergeben.

Die Nebenmodule der Hauptmodule „vorbereitungSCM“ und „nachbereitungSCM“ haben in der Regel als Übergabeparameter einen String, in dem die zu verarbeitende Zeile steht. Außerdem werden ihnen die komplette Klammerstruktur und die Position des Klammerpaares, das auf den TeXmacs-Befehl hin überprüft werden soll, übergeben. Der Bereich zwischen diesem Klammerpaar wird temporär zwischengespeichert und analysiert. Anschliessend wird über die Methode „ersetzen()“ der Klasse „s2x_verarbeiten“ das SCM-Format durch die entsprechenden Tags ersetzt. Diese Analyse ist abhängig von der jeweiligen Methode; folgende Methoden der Klasse „s2x_wortmethoden“ existieren:

Methodenname	**TeXmacs-Befehl(e)**	**Aufgerufen bei**
rprime	rprime	Ableitungen
em	em	Kursivschrift
neg	neg	Negierung eines Bereichs
numerate	enumerate-numeric	nummerierten Aufzählungen
big	big	„großen“ Operatoren (z.B: $\sum, \int$)
formate	hide-preamble	versteckten Einstellungen
klammer	left, right	linke bzw. rechte TeXmacs-Klammer
eq	equation eqnarray	mehrere mathematische Reihen

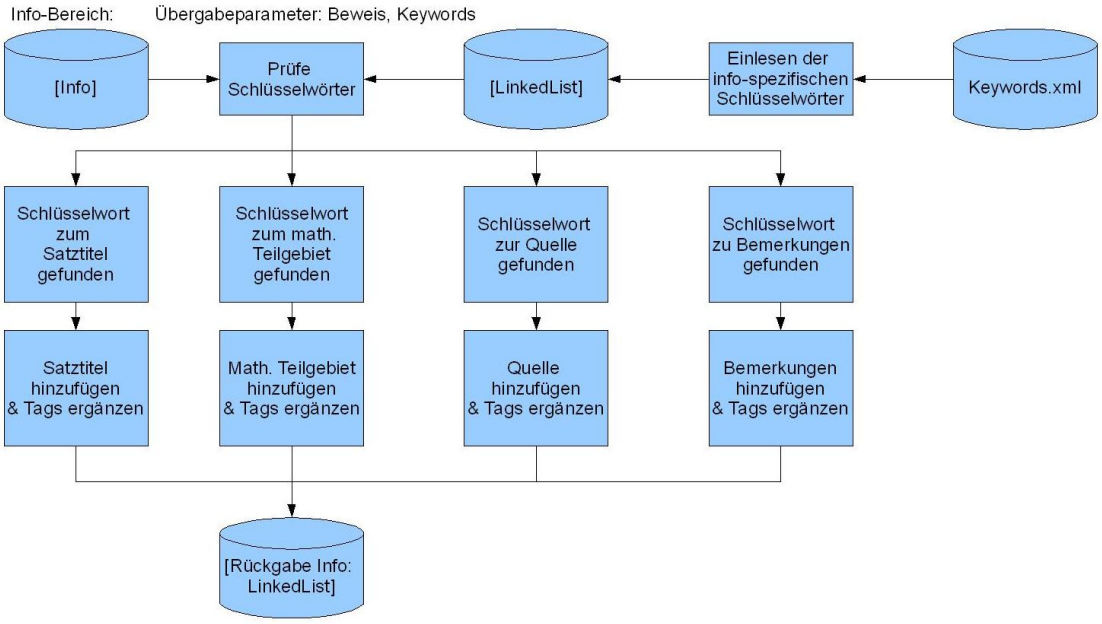
mathMode	with „mode“ „math“ math	mathematischer Zeile Formel (in der selben Zeile)
bruch	frac	Brüchen
wurzel	sqrt	Wurzel-Symbolen
rsu	rsup rsub	hochgestellten bzw. untergestellten Inhalten
codocell	concat, document, cell, row	verbindende Strukturen untergeordnete Dokumentbereiche Zellen und Reihen
TextTag	“ “	Textbereichen
ersetzeAnfuer- ungszeichen	-	TextTag
ftext	-	Text in Mathebereichen

Zur Veranschaulichung betrachtet man den TeXmacs-Befehl „frac“. Beispielsweise wird der Bruch $\frac{x}{y}$ im zwischengespeicherten Inhalt nach einem Zähler und einem Nenner durchsucht und zugeordnet. Diese Bereiche werden in Tags geschrieben, so dass sich für den Bruch folgender Rückgabewert ergibt:
<BRUCH><Zaehler>x</Zaehler><Nenner>y</Nenner></BRUCH>.
Dieser String wird dann von der Methode „vorbereitungSCM“ weiterverarbeitet.

Ein weiteres Beispiel ist der TeXmacs-Befehl für „rsup“. Auch hier wird wie im Beispiel des Bruchs der zwischengespeicherte Inhalt nach gewissen Kriterien untersucht und neu zugeordnet. So sieht ein mögliches Ergebnis für den Befehl „rsup“ wie folgt aus:
<MSUP><mi> Basis </mi><mi> Exponent </mi></MSUP>
Eine Ausnahme in der Nachbereitung ist die Methode „TextTags“. Mit ihrer Hilfe werden alle noch vorhandenen Anführungszeichen in dem Zwischenergebnis durch Text-Tags ersetzt.
Die Methoden zu den Befehlen der SCM-Datei-Verarbeitung bieten also je nach TeXmacs-Befehl unterschiedliche Tag-Strukturen an.

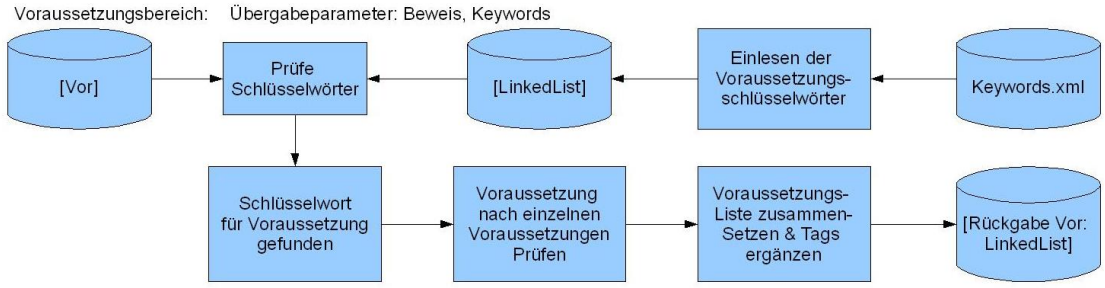
Ähnlich wie die Anpassung der Strukturen der TeXmacs-Befehle findet auch die Umwandlung der in der „Satz.dtd“ stehenden Strukturen statt. Die Methoden wurden in der Klasse „s2x_keywords“ implementiert. Die Übergabeparameter für jede Funktion sind zwei Listen: die eine beinhaltet die Bereichsinhalte, aus denen die Strukturen entstehen sollen; die andere die Schlüsselwörter, die für die Erstellung dieser Strukturen benötigt werden:

- aktualisiereInfo:



In dieser Methode werden die verschiedenen Satzinformationen zugeordnet. An Hand der Schlüsselwörter für diesen Informationsbereich (Klasse „s2x_einlesen“ Methode „leseKeywords()“) wird der Inhalt der übergebenen Liste daraufhin analysiert, wie der Satztitel lautet und der entsprechende Bereich zugeordnet (Methode „tagBereich()“). Die optionalen Bereiche „mathematisches Teilgebiet“, „Quelle“ und „Bemerkungen“ werden nur dann zugeordnet, wenn entsprechende Schlüsselwörter im Text vorkommen.

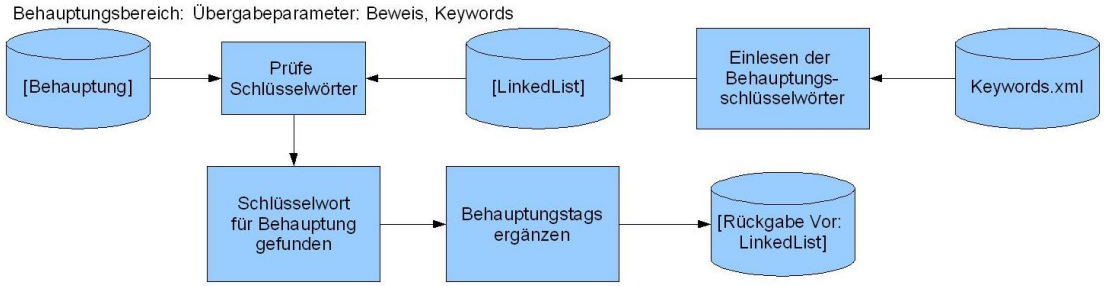
- aktualisiereVor:



Die übergebene Liste enthält in diesem Falle den Voraussetzungsbereich. Da eine Voraussetzung aus mehreren Einzelvoraussetzungen zusammengesetzt wird, erfolgt ihre Aufteilung durch den Aufruf der Methode „trennen()“, die anhand von Satzzeichen oder Schlüsselwörtern die Liste neu zusammenstellt.

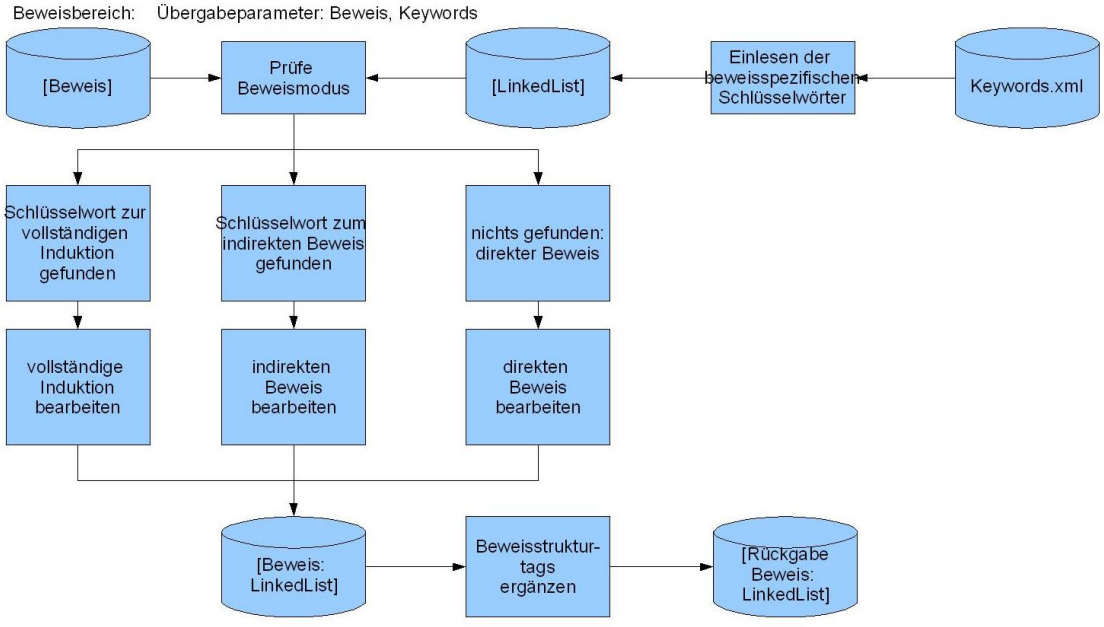
Jeder Einzelvoraussetzung wird eine ID zugewiesen.

- aktualisiereBeh:



Zu jedem Beweis gibt es genau eine Behauptung, die von der Methode entsprechend mit XML-Tags bearbeitet und zurückgegeben wird.

- aktualisiereBew:

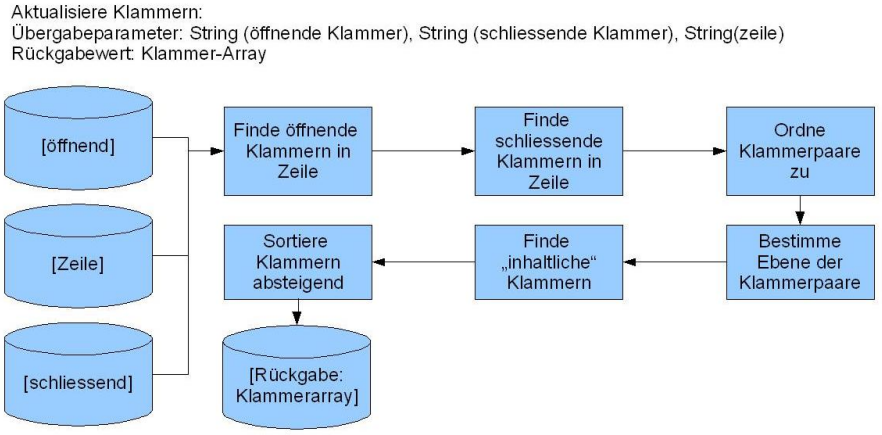


Der Beweis ist beinhaltet die komplexeste Struktur des Satzes. Zuerst muss der Beweismodus ermittelt werden, um den Beweis korrekt umwandeln zu können. Das wird über die Methode „istdrin()“ geregelt, die überprüft, ob es sich um einen Beweis mit vollständiger Induktion oder einen indirekten Beweis handelt. Ist beides nicht der Fall, so wird der Modus auf direkter Beweis gestellt. Die Beweisargumentationsfolge wird dann je nach vorhandenem Modus anders aufgebaut: Bei einem direkten Beweis werden die einzelnen Beweisschritte

getrennt (Methode „trennen()“) und über Argumente („Arg“) nummeriert. Die Trennkriterien für die Liste sind neben den Keywords die Satzzeichen „.“ und „:“. Zu jedem indirekten Beweis (Methode „indirektBew()“) wird zunächst das kontradiktorische Gegenteil mit Hilfe der Methoden „postionErstesKeyword()“, „erzeugeListe()“ und „LinkedListToString()“ formuliert. Die eigentliche Beweisführung entspricht der des direkten Beweises (Methode „trennen()“), die dann in dem Widerspruch zur Behauptung endet. Eine vollständige Induktion (Methode „induktionBearbeiten()“) teilt den Beweis zunächst in die Bereiche Induktionsverankerung (selbe Methoden wie beim indirekten Beweis) und Induktionsschluss (Methode „induktionsSchluss()“) auf. Der Induktionsschluss wird mit den Methoden „postionErstesKeyword()“, „erzeugeListe()“ und „LinkedListToString()“ wiederum in die Induktionsannahme (entspricht der Voraussetzung), Induktionsbehauptung und dem Induktionsbeweis gegliedert. Die Annahme wird nach den Kriterien der Voraussetzung bearbeitet, die Behauptung nach der Behauptung und der Beweis dem direkten Beweis entsprechend.

Weitere allgemeine Nebenmodule sind:

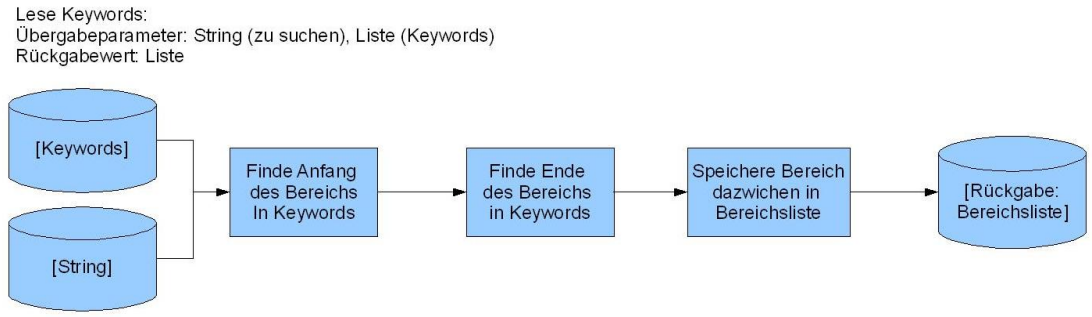
- aktualisiereKlammern (in der Klasse s2x_verarbeiten):



Die Übergabeparameter sind drei Strings. Einer beinhaltet eine Zeichenkette für das öffnende und ein anderer für das schließende Symbol. In diesem Falle sind es die runden Klammern (). Der dritte String ist die Zeile, in der nach den Symbolen gesucht werden soll. Zurückgegeben wird ein mehrdimensionales

Integer-Feld, welches die Positionen sortiert nach ihren Ebenen zurückgibt, wobei die öffnenden und schließenden Klammern absteigend sortiert sind.

- leseKeywords:



Der Übergabeparameter sind ein String und eine Liste. Die Liste wurde aus der Datei „Keywords.xml“ erzeugt. Der String beinhaltet den Bereich, aus dem die Schlüsselwörter in der XML-Datei stammen sollen. Zurückgegeben wird eine LinkedList mit allen zum Bereich zugehörigen Schlüsselwörtern.

Diese Nebenmodule werden ebenfalls im Unterpunkt „Programmablauf - Beispiel“ anhand des Beispiels „Dreiecksungleichung für reelle Zahlen“ näher erläutert. Die Hilfsmethoden, die weder bei den Haupt- noch bei den Nebenmodulen zu finden sind, werden in der folgenden Tabelle mit ihren Klassen aufgelistet, aber ihre Wirkungsweisen werden nur kurz beschreiben bzw. in dem Beispiel (6.6) gegebenenfalls aufgeführt: Es folgt eine Liste aller Methoden mit ihren zugehörigen Klassen, die Einteilung in Haupt-, Neben- und Untermodule sowie einer Kurzbeschreibung der Funktion:

Klasse	Methode	Beschreibung
s2x_ausgabe	ausgabeKlammern()	Gibt Klammerpaare aus
s2x_ausgabe	ausgabeListe()	Gibt LinkedList aus
s2x_einlesen	findeStatementtyp()	Statementtyp bestimmen
s2x_einlesen	tauschen()	Ersetzt konstante SCM-Tags
s2x_einlesen	leseDTD()	Funktion speziell zum DTD-einlesen
s2x_keywords	checkKeyword()	Einzelelement oder mit Inhalt?

s2x_keywords	erstelleGleichungsstring()	Hinterer Statementtyp-Bereich
s2x_keywords	findeUngleiche()	Vergleicht zwei Arrays
s2x_keywords	pasteString()	String einfügen
s2x_keywords	schreibeSatzliste()	Schreibe die Datei „Satzliste.csv“
s2x_keywords	sortierePos()	Bubblesort des arrays
s2x_keywords	StringToLinkedList()	String in LinkedList
s2x_keywords	sucheSatzID()	Bestimmt mögliche SatzID
s2x_keywords	trenneKeywords()	Trennt nach Schlüsselwörtern
s2x_keywords	trennePunkte()	Trennt nach Satzzeichen
s2x_main	Dateida()	Existiert File?
s2x_main	hauptmenu()	Menu
s2x_main	pruefeTag()	Tag schon vorhanden?
s2x_main	Zeitstempel()	Zeitstempel in DTD schreiben
s2x_suchen	pruefeWoerter()	Kommen Schlüsselwörter vor?
s2x_suchen	sucheAnzKlammern()	Anzahl der relevanten Klammern
s2x_suchen	sucheAnzTodo()	Anzahl der Einträge > 0 des übergebenen Feldes
s2x_suchen	sucheMaxEbene()	Suche die tiefste Ebene in klammern[][]
s2x_suchen	sucheMaxTodo()	Gibt höchsten Wert des übergebenen Feldes
s2x_suchen	SucheString()	Suche Start- und End- Inhalte in einem String
s2x_suchen	sucheStruktur()	Suche Start- und End- Inhalte einer Liste
s2x_suchen	zaehleElemente()	Bestimmt alle Positionen in einem String
s2x_suchen	zaehleString()	Wie oft ein String in einem anderen vorkommt
s2x_verarbeiten	stringToList()	Erstellt aus einem String eine LinkedList
s2x_verarbeiten	update()	Aktualisierung des Klammerarrays
s2x_bonn	erstelleXML()	Erstellt XML nach Bonn.dtd
s2x_bonn	findeBereich()	Finden und zuordnen der Beweisbereiche
s2x_bonn	einfuegenProof()	Zusammenstellen des Proof-Bereichs

6.6 Programmablauf - Beispiel

Zu Beginn des Programms wird die Main- Funktion aufgerufen. Die Beschreibung der kompletten Software ist im Anhang hinterlegt. Hier wird zuerst das Menu aufgebaut; es wartet auf die Eingabe des Anwenders:

```
-------------------------------------------------
[1] Umwandeln einer bereits erfassten SCM-Datei
[2] Umwandeln einer noch nicht erfassten SCM-Datei
-------------------------------------------------
[9] Einstellungen des Programms
-------------------------------------------------
[0] Programm beenden
-------------------------------------------------
1
---------------------------------------
Folgende Dateien wurden erfasst:
[1] potenzen
[2] irrationaleZahlen
[3] SatzVonRolle
[4] Dreiecksungleichung_reell
[5] SumNullfolge
[6] lokalesMaximum
[7] zwischenwertsatz
[8] Summenformel
[9] Bernoulli
[10] SummegeraderZahlen
[11] hauptideal
[12] additionstheoreme
[13] produktregel
[14] Stammfunktion
Welcher Eintrag soll umgewandelt werden?
[0] um zurück zu springen
---------------------------------------
```

Der Anwender wählt in diesem Beispiel die Datei „Dreiecksungleichung_reell“ aus, indem er die Zahl „4“ eingibt. Die Hauptmethode „programm()“ wird gestartet: Die gewählte SCM-Datei wird dann mit Hilfe der Methode „leseDatei()“ eingelesen:

```
(document (TeXmacs "1.0.6.5") (style (tuple "generic")) (body (document "Satz:
Dreiecksungleichung für reelle Zahlen" (concat "Vor.: " (with "mode" "math" "a,b <epsilon> <bbb-
R>")) "Beh: " (eqnarray* (document (tformat (table (row (cell (concat (mid "|") "| a| - |b| |<leq> | a +
b | <leq> | a| + | b |")) (cell "") (cell ""))))))) "Bew: Weil alle Seiten nicht negativ sind, ist Quadieren
eine Äquivalenzumformung:" (eqnarray* (document (tformat (table (row (cell (concat "a" (rsup "2")
" - 2 |a b|+ b" (rsup "2") "  <leq>  a" (rsup "2") " + 2 a b + b" (rsup "2"))) (cell (concat "<leq>  a"
(rsup "2"))) (cell (concat "+ 2 |a b | + b" (rsup "2")))))))) (concat "Das ist äquivalent zur
Ungleichung " (with "mode" "math" " - 2 | a b | <leq> 2 a b <leq> 2 |a b |")) (eqnarray* (document
(tformat (table (row (cell "- 2 | a b | <leq> 2 a b <leq> 2 |a b|") (cell "") (cell "")))))) (concat "welche
gilt, weil " (with "mode" "math" "- | x | <leq> x <leq> | x|") " für alle reellen " (with "mode" "math"
"x") ".") "q.e.d.")) (references (collection (associate "auto-1" (tuple "1" "?")) (associate "auto-3"
(tuple "3" "?")))))
```

Die SCM-Datei wird nun mit der Methode „vorbereitungScm()“ weiter verarbeitet: Zuerst wird die Klammerstruktur zu der SCM-Datei, die in einen String umgewandelt

wurde, ermittelt. Diese ordnet jedem offenen Klammersymbol ein geschlossenes zu. Es folgt die Bestimmung der Ebene, auf der sich das Klammerpaar befindet. Anschließend wird die Klammerstruktur mit einem Bubble-Sort absteigend sortiert.
Dann werden aus diesem String die für den Beweis irrelevanten Daten entfernt. Da TeXmacs in seiner XML- ähnlichen Struktur keine schließenden Tags zu konstanten Ausdrücken bereit stellt, werden diese auch gleich durch abgeschlossene Tags ersetzt (z.B. aus <epsilon> wird <epsilon/>). Als nächstes werden die konstanten Ausdrücke in der SCM- Datei, die auf diese Weise einfach ersetzt werden können, ersetzt. Die Liste, in der diese Ausdrücke stehen, wird zu diesem Zweck aus einer CSV-Datei eingelesen. Sie kann nach Belieben jederzeit bearbeitet werden.
Nun folgt der Ersatz der TeXmacs- Befehle. Die Inhalte der Klammer werden zwischengespeichert und es wird nach TeXmacs- Befehlen gesucht. Ist ein solcher Befehl gefunden, wird eine entsprechende Methode aufgerufen. In dieser Methode wird dann die TeXmacs-Klammer durch XML-Tags ersetzt. Wird ein gültiger Wert übergeben, wird die Variable mit dem kompletten Inhalt ebenfalls angepasst und die Klammerstruktur erneut analysiert.
Ein Sonderfall bei der Übertragung ins XML-Format stellen die Klammern da. Wenn runde Klammern als öffnende Klammersymbole in TeXmacs vorhanden sind, sollen sie als Klammertyp erkennbar sein. Deswegen werden sie kurzfristig in einen String umgewandelt, der am Ende der Vorbereitung die Klammern wieder ersetzt. Die Zeile wird abschließend als Liste zurückgegeben, wo sie dann von der Nachbereitung verarbeitet wird. Vor der Nachbereitung sieht die „zeile“ so aus:

```
 (document "Satz: Dreiecksungleichung für
reelle Zahlen" (concat "Vor.: " (with "mode"
"math" "a,b <epsilon/> <bbb-R/>")) "Beh: "
<MATHreihen>
 (row
<CELL>
 (concat <OR/> "| a| - |b| |<leq/> | a + b | <leq/> |
a| + | b |")
</CELL>
  )
</MATHreihen>
  "Bew: Weil alle Seiten nicht negativ sind, ist
Quadieren eine Äquivalenzumformung:"
<MATHreihen>
 (row
<CELL>
 (concat
<MSUP>
<mi>
 a
</mi>
<mi> 2
</mi>
</MSUP>
<MSUP>
<mi>
 - 2 |a b|+ b
</mi>
<mi> 2
</mi>
</MSUP>
<MSUP>
<mi>
   <leq/>  a
</mi>
<mi> 2
</mi>
</MSUP>
<MSUP>
<mi>
  + 2 a b + b
</mi>
<mi> 2
</mi>
</MSUP> )
</CELL>
<CELL>
 (concat
<MSUP>
<mi>
  <leq/>  a
</mi>
<mi> 2
</mi>
</MSUP> )
</CELL>
<CELL>
 (concat
<MSUP>
<mi>
 + 2 |a b | + b
</mi>
<mi> 2
</mi>
</MSUP> )
</CELL>
 )
</MATHreihen>
  (concat "Das ist äquivalent zur Ungleichung "
(with "mode" "math" " - 2 | a b | <leq/> 2 a b
<leq/> 2 |a b |"))
<MATHreihen>
 (row
<CELL>
 "- 2 | a b | <leq/> 2 a b <leq/> 2 |a b|"
</CELL>
  )
</MATHreihen>
  (concat "welche gilt, weil " (with "mode" "math"
"- | x | <leq/> x <leq/> | x|") " für alle reellen "
(with "mode" "math" "x") ".") "q.e.d.")
```

Als erstes wird in der Nachbereitung ebenfalls die Struktur der Klammern ermittelt. Die TeXmacs- Befehle, die jetzt umgewandelt werden, sind teilweise in der Vorbereitung in Kombination mit anderen Befehlen bereits verarbeitet worden. Die noch übrig gebliebenen werden nun mit Hilfe der entsprechenden Methoden ersetzt und die Klammerstruktur angepasst. Danach werden alle noch vorhandenen Anführungszeichen durch Text-Tags ersetzt.

Der Klammertyp der öffnenden Klammersymbole muss wieder gesondert behandelt werden. Dadurch dass der öffnende Tag den Klammertyp in Anführungszeichen übergeben bekommt, dürfen diese nicht als Text-Elemente erkannt werden. Um

das zu verhindern wird wieder ein temporärer String am Schluss der Nachbereitung durch die Anführungszeichen ersetzt. Die endgültige Zeile wird mit einer speziellen Methode in eine vollwertige Liste umgewandelt. Hier folgt das Zwischenergebnis nach der Nachbereitung:

```
<TEXT>Satz: Dreiecksungleichung für reelle
Zahlen</TEXT>
<TEXT>Vor.:</TEXT>
<MATHmodus>
<TEXT>a,b <epsilon/> <bbb-R/></TEXT>
</MATHmodus>
<TEXT>Beh:</TEXT>
<MATHreihen>
<REIHE>
<CELL>
<OR/> <TEXT>| a| - |b| |<leq/> | a + b | <leq/> |
a| + | b |</TEXT>
</CELL>
</REIHE>
</MATHreihen>
<TEXT>Bew: Weil alle Seiten nicht negativ sind,
ist Quadieren eine
Äquivalenzumformung:</TEXT>
<MATHreihen>
<REIHE>
<CELL>
<MSUP>
<mi>
a
</mi>
<mi>2
</mi>
</MSUP>
<MSUP>
<mi>
- 2 |a b|+ b
</mi>
<mi>2
</mi>
</MSUP>
<MSUP>
<mi>
<leq/> a
</mi>
<mi>2
</mi>
</MSUP>
<MSUP>
<mi>
+2 a b + b
</mi>
<mi>2
</mi>
</MSUP>
</CELL>
<CELL>
<MSUP>
<mi>
<leq/> a
</mi>
<mi>2
</mi>
</MSUP>
</CELL>
<CELL>
<MSUP>
<mi>
+2 |a b | + b
</mi>
<mi>2
</mi>
</MSUP>
</CELL>
</REIHE>
</MATHreihen>
<TEXT>Das ist äquivalent zur
Ungleichung</TEXT>
<MATHmodus>
<TEXT>- 2 | a b | <leq/> 2 a b <leq/> 2 |a b |
</TEXT>
</MATHmodus>
<Statementtyp>0;ungl,ungl</Statementtyp>
<MATHreihen>
<REIHE>
<CELL>
<TEXT>- 2 | a b | <leq/> 2 a b <leq/> 2 |a b|
</TEXT>
</CELL>
</REIHE>
</MATHreihen>
<TEXT>welche gilt, weil</TEXT>
<MATHmodus>
<TEXT>- | x | <leq/> x <leq/> | x|</TEXT>
</MATHmodus>
<Statementtyp>0;ungl,ungl</Statementtyp>
<TEXT>für alle reellen</TEXT>
<MATHmodus>
<TEXT>x</TEXT>
</MATHmodus>
<TEXT>.</TEXT>
<TEXT>q.e.d.</TEXT>
```

Die Liste mit dem erzeugten TeXmacs-XML wird nun mit einer weiteren Hauptmethode in die durch die in der Datei „Satz.dtd“ gegebenen Struktur umgewandelt: In der Methode „erstelleXML()“ werden zunächst aus der Datei Keywords.XML die Schlüs-

selwörter für die Beweisbereiche Satzinformationen, Voraussetzungen, Behauptung und Beweis eingelesen. Mit den Wörtern wird die Liste in kleinere Listen aufgeteilt; für jeden dieser Bereiche entsteht eine eigene Liste. Voraussetzungen sind in der Unterteilung nicht zwingend erforderlich, so dass auch ein Beweis ohne Voraussetzung umgewandelt werden kann. Die Listen werden mit den jeweiligen Methoden analysiert und mit Hilfe von ergänzenden Tags umstrukturiert.

Sind alle Teilbereiche ordnungsgemäß bearbeitet, werden die Listen zu einem strukturierten Gesamtbeweis vereint. Die nächste Aufgabe ist es zu überprüfen, ob der Beweis bereits in der Liste von bereits erfassten Beweisen vorhanden ist, was hier der Fall ist. Dazu wird in der Datei „Satzliste.csv“ nachgesehen, ob der Name vorhanden ist, und wenn ja, welche ID im zugeordnet wurde (hier „4“). Die Gesamtbeweisliste wird als Ergebnis zurückgegeben (Abbildung siehe folgende Seite):

```
<SATZ ID = "4">
	<Satz-Informationen>
		<Titel>
			<TEXT>Satz: Dreiecksungleichung für reelle Zahlen</TEXT>
		</Titel>
	</Satz-Informationen>
	<Voraussetzungen>
		<Vor ID= "0">
			<Statementfolge>
				<TEXT>Vor.:</TEXT>
			</Statementfolge>
		</Vor>
		<Vor ID= "1">
			<Statementfolge>
				<MATHmodus><TEXT>a ,b <epsilon/> <bbb-R/></TEXT></MATHmodus>
			</Statementfolge>
		</Vor>
	</Voraussetzungen>
	<Behauptung>
		<Statementfolge>
			<TEXT>Beh:</TEXT><MATHreihen><REIHE><CELL><OR/> <TEXT>| a| - |b| |<leq/> | a +
b | <leq/> | a| + | b |</TEXT></CELL></REIHE></MATHreihen>
		</Statementfolge>
	</Behauptung>
	<Beweis>
		<BewModus>
			DirBew
		</BewModus>
		<BewArgFolge>
			<DirBew>
				<Arg ID= "0">
					<Statementfolge>
						<TEXT>Bew: Weil alle Seiten nicht negativ sind, ist
Quadieren eine Äquivalenzumformung:</TEXT><MATHreihen><REIHE><CELL><MSUP><mi>a</mi><mi>
2</mi></MSUP><MSUP><mi>- 2 |a b|+ b</mi><mi> 2</mi></MSUP><MSUP><mi><leq/>  a</mi><mi>
2</mi></MSUP><MSUP><mi>+ 2 a b + b</mi><mi> 2</mi></MSUP></CELL><CELL><MSUP><mi><leq/>  a</mi><mi>
2</mi></MSUP></CELL><CELL><MSUP><mi>+ 2 |a b | + b</mi><mi>
2</mi></MSUP></CELL></REIHE></MATHreihen>
					</Statementfolge>
				</Arg>
				<Arg ID= "1">
					<Statementfolge>
						<TEXT>Das ist äquivalent zur
Ungleichung</TEXT><MATHmodus><TEXT>- 2 | a b | <leq/> 2 a b <leq/> 2 |a b |
</TEXT></MATHmodus><Statementtyp>0;ungl,ungl</Statementtyp><MATHreihen><REIHE><CELL><TEXT>- 2 | a b |
<leq/> 2 a b <leq/> 2 |a b|</TEXT></CELL></REIHE></MATHreihen>
					</Statementfolge>
				</Arg>
				<Arg ID= "2">
					<Statementfolge>
						<TEXT>welche gilt, weil</TEXT><MATHmodus><TEXT>- |
x| <leq/> x <leq/> | x|</TEXT></MATHmodus><Statementtyp>0;ungl,ungl</Statementtyp> <TEXT>für alle
reellen</TEXT><MATHmodus><TEXT>x</TEXT></MATHmodus><TEXT>.</TEXT>
					</Statementfolge>
				</Arg>
				<Arg ID= "3">
					<Statementfolge>
						<TEXT>q.e.d.</TEXT>
					</Statementfolge>
				</Arg>
			</DirBew>
		</BewArgFolge>
	</Beweis>
</SATZ>
```

In der Methode „programm()“ werden jetzt noch die XML-Einstellungen, die für das XML-Dokument gelten sollen, an erster Stelle der Liste hinzugefügt. Anschliessend wird die Datei unter ihrem SCM-Namen in dem Unterordner XML als XML-Datei gespeichert: „XML/Dreiecksungleichung_reell.XML“. Diese Datei wird nun mit Hilfe eines üblichen SAX-Parsers auf ihre Wohlgeformtheit und Validität überprüft. Das Ergebnis wird dem Anwender in der Konsole angezeigt.

Eine vollständige Bedienungsanleitung ist im Anhang hinterlegt.

6.7 Einzubindende Dateien

6.7.1 Satz - DTD

Im Entwurf der Datei Satz.dtd geht es darum, die Gesetzmäßigkeiten von mathematischen Beweisen möglichst allgemeingültig zu formalisieren und zu dokumentieren. Mit Hilfe von einer Vorlage, die von Prof. Dr. phil. Gregor Büchel erstellt wurde, und einiger mathematischer Bücher (unter anderem „Analysis 1“ von Otto Forster) wurden verschiedene Beweise auf ihre Strukturen analysiert. Diese Ergebnisse stehen in der DTD-Datei, die im Folgenden näher erläutert wird:

Das grundlegende Element eines Beweises ist der Satz. Der Satz kann aus bis zu vier verschiedenen Bereichen bestehen: Informationen und Hinweise zum jeweiligen Satz, alle Voraussetzungen sowie die Behauptung und der eigentliche Beweis werden hier getrennt. Um eine Eindeutigkeit der Sätze gewährleisten zu können, wird jedem Satz eine eindeutige ID zugeordnet.

```
<!ELEMENT Satz (SatzInformationen,Voraussetzungen*,Behauptung,Beweis)>
<!ATTLIST Satz
ID CDATA #REQUIRED
>
```

Die „SatzInformationen“ setzen sich aus dem Titel, dem mathematischen Teilgebiet, der Quelle und beliebig vielen Bemerkungen zum Satz zusammen.
Der „Satztitel“ ist der Titel, unter dem der Satz in der Fachliteratur üblicherweise zu finden ist.
Optional kann das mathematische Teilgebiet (z.B. Geometrie, Analysis, Algebra) angegeben werden, um den Satz näher zu klassifizieren.

In der ebenfalls optinalen Quelle wird die Herkunft des Beweises hinterlegt. Die Herkunft kann beispielsweise der Fundort im Internet, der Autor oder auch ein Fachbuch sein.
Zu den Bemerkungen können historische Hintergründe oder auch Einordnungen des Satzes in mathematischen Kontexten gehören. Als historisches Beispiel könnte für den Satz „Summenformel“ einen Hinweis auf die Entdeckung durch C.F. Gauß dienen. Das Element „TEXT“ und die Entität „Mathe“ werden im Verlauf des Textes näher spezifiziert.

```
<!ELEMENT SatzInformationen (Satztitel,Mathemteilgebiet?,Quelle?,Bemerkung*)>
<!ELEMENT Satztitel (TEXT|%Mathe;)>
<!ELEMENT Mathemteilgebiet (TEXT|%Mathe;)>
<!ELEMENT Quelle (TEXT|%Mathe;)>
<!ELEMENT Bemerkung (TEXT|%Mathe;)>
```

Jeder Satz kann auf beliebig vielen Voraussetzungen basieren; diese werden mit Hilfe einer jeweils zugeordneten ID satzspezifisch durchnummeriert. Jede Voraussetzung besteht aus einer Statementfolge (siehe unten).

```
<!ELEMENT Voraussetzungen (Vor+)>
<!ELEMENT Vor (Statementfolge)>
<!ATTLIST Vor
ID CDATA #REQUIRED
>
```

Eine Behauptung besteht aus einer Statementfolge, die weiter unten näher erläutert wird.

```
<!ELEMENT Behauptung (Statementfolge)>
```

Das Element „Beweis“ enthält den Beweismodus und die Beweisargumentfolge. Im Modus wird hinterlegt, ob es sich um eine vollständige Induktion, einen indirekten Beweis oder einen direkten Beweis handelt. Entsprechend der Auswahl wird das Element der Beweisargumentfolge zugeordnet.

```
<!ELEMENT Beweis (BewModus,BewArgFolge)>
<!ELEMENT BewModus (#PCDATA)>
<!ELEMENT BewArgFolge (VollstIndukt|IndBew|DirBew)>
```

Der direkte Beweis beinhaltet mindestens ein Argument, welches aus einer Statementfolge besteht. Durch die ID eines Argumentes wird jedes Einzelargument des Satzes eindeutig gekennzeichnet.

```
<!ELEMENT DirBew (Arg+)>
<!ELEMENT Arg (Statementfolge+)>
<!ATTLIST Arg
ID CDATA #REQUIRED
>
```

Ein indirekter Beweis wird in drei Einzelelemente gegliedert: Invertierung, Beweis des Gegenteils und Widerspruch. Die Invertierung entspricht der Behauptung in einem direketen Beweis; der anschließende Beweisversuch des Gegenteils folgt dann bis zum Widerspruch den Regeln des direkten Beweises. Der Widerspruch ergibt sich aus dem letzten Beweisschritt und einem vorangegangenem Beweisschritt oder der „Invertierung“.

```
<!ELEMENT IndBew (Statementfolge?,Invertierung,InvertBew,Widerspruch)>
<!ELEMENT Invertierung (Statementfolge)>
<!ELEMENT InvertBew(Arg*)>
<!ELEMENT Widerspruch(Statementfolge)>
```

Eine vollständige Induktion teilt sich zunächst in eine Induktionsverankerung und einen Induktionsschluss auf. Die Induktionsverankerung besteht aus einer Statementfolge, während der Induktionsschluss sich erneut aufteilt. Dieser ist entweder wie ein direkter oder ein indirketer Beweis aufgebaut, wobei die Induktionsannahme der Voraussetzung entspricht.

<!ELEMENT VollstIndukt (Induktionsverankerung,Induktionsschluss+)>
<!ELEMENT Induktionsverankerung (Statementfolge,Statementtyp?)>
<!ELEMENT Induktionsschluss (Statementfolge?,
Induktionsannahme,Induktionsbeh,Induktionsbeweis)>
<!ELEMENT Induktionsannahme (Statementfolge+)>
<!ELEMENT Induktionsbeh (Statementfolge+)>
<!ELEMENT Induktionsbeweis (Arg*)>

Eine Statementfolge kann in beliebiger Reihenfolge mathematische Teile wie prädikatenlogische Ausdrücke kombiniert mit natürlichspachlichen Textelementen enthalten. Zu jedem mathematischem Element kann optional ein Statementtyp angegeben werden, der feststellt, ob prädikatenlogische Aussagen oder Vergleichssymbole in dem vorangegangenen mathematischen Ausdruck stehen.

<!ELEMENT Statementfolge ((TEXT|((MATH|MATHmodus|MATHreihen),Statementtyp?))+)>
<!ELEMENT TEXT (#PCDATA)>

Ein Beispiel zum Statementtyp: In Texmacs steht an einer Stelle „$0 \leq y = x^2$“.
Der zugehörige Statementtyp sieht wie folgt aus: „0;ungl,gl“
Dieser Statementtyp gibt an der erster Stelle an, ob es sich hierbei um eine prädikatenlogische Aussage handelt (ausgedrückt durch „1“) oder nicht („0“). Hinter dem Semikolon steht, ob es sich um eine Gleichung (gl) oder Ungleichung (ungl) oder beliebigen Kombinationen aus beiden handelt.

```
<!ELEMENT Statementtyp (#PCDATA)>
```

In Texmacs gibt es verschiedene mathematische Strukturen. Um diese auch in dem resultierendem XML-dokument unterscheiden zu können, wurden diese drei Elemente definiert. MATH bezieht sich auf in einer Zeile eingefügte mathematische Elemente, MATHmodus genau wie MATHreihe auf eine eigene mathematische Zeile und MATHreihen auf mehrere mathematische Zeilen. Sie können alle mathematische Ausdrücke („%Mathe“), Tags, die sich noch in andere Einzeltags gliedern („%TagsMitInhalt“), und einzelne Tags („%EinzelneTags“) enthalten.
Da im Programm der mathematische Bereich wieder in einzelne XML-Tags unterteilt wird (siehe Nebenmodule), werden die „#PCDATA“ Einträge ergänzt werden (siehe Parser). Diese DTD wird also dynamisch von den Beweisen, die im Programm bearbeitet werden, lernen.

```
<!ELEMENT MATH (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>
<!ELEMENT MATHmodus (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>
<!ELEMENT MATHreihe (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>
<!ELEMENT MATHreihen (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>
<!ELEMENT FTEXT (#PCDATA|%TagsMitInhalt;|%EinzelneTags;)*>
```

Die Entitäten, die nun folgen, sind Kurzschreibweisen für die Tags in ihren Klammern. Sie werden aufgerufen, in dem man den Namen der Entität zwischen den Zeichen „%“ und „;“ setzt. „Mathe“ bezieht sich auf alle möglichen Tag-Kombinationen, in denen mathematische Symbole dargestellt werden. Die „TagsMitInhalt“ beinhalten Tags, die wiederum weitere Tags beinhalten können (z.B. enthält „MSUP“ den Tag „mi“). In der Entität „EinzelneTags“ sind die Tags aufgelistet, die keine weiteren Tags beinhalten; diese werden vom Programm automatisch ergänzt, falls während der Validierung unbekannte Tags gefunden werden.

```
<!ENTITY % Mathe “MATH|MATHmodus|MATHreihe|MATHreihen|FTEXT“>
<!ENTITY % TagsMitInhalt “FTEXT|MSUP|MSUB|...|CELL“>
<!ENTITY % EinzelneTags “bbb-R|gtr|bbb-N|...|in|lambda“>
```

Die noch unbekannten Elemente in den Entitäten werden am Ende der Datei „Satz.dtd“ definiert. Als Beispiel für die „TagsMitInhalt“ stehen hier MSUP und MSUB; Diese bestehen aus mindestens einem „mi“, was wiederum Daten, „TagsMitInhalt“ oder „EinzelneTags“ beinhalten kann.

```
<!ELEMENT MSUP (mi+)>
<!ELEMENT MSUB (mi+)>
<!ELEMENT mi (#PCDATA|%TagsMitInhalt;|%EinzelneTags;)*>
...
```

Die „EinzelneTags“ werden vom Programm automatisch ergänzt. Es folgt ein Auszug aus diesen Tags:

```
<!ELEMENT bbb-R (#PCDATA)>
<!ELEMENT gtr (#PCDATA)>
<!ELEMENT bbb-N (#PCDATA)>
<!ELEMENT in (#PCDATA)>
<!ELEMENT lambda (#PCDATA)>
...
```

Die vollständige Datei „Satz.dtd“ befindet sich im Anhang.

6.7.2 Keywords

Die Datei „Keywords.XML“ stellt in einer XML- Struktur die Schlüsselwörter dar, die während des Programmablaufs benötigt werden. Der Vorteil dieser Datei ist, dass sie leicht pflegbar und erweiterbar ist. Die vorgegebene Struktur kann man an einem Auszug der Datei erkennen:

```
[...]
<Beweis>
        Beweis
        Bew

        <Beweistyp>
                <direkt>
                        <!-- Wenn kein anderer Fall zutrifft -->
                </direkt>

                <indirekt>
                        Widerspruch
                        contradiction
                        Annahme

                        <Invert>
                                Annahme
                        </Invert>
      [...]
    </Beweistyp>
<Beweis>

<Ungleichung>
        $gtr
        $geq
        $geqslant
        $leq
        $less
        $leqslant
</Ungleichung>
[...]
```

Der Inhalt der Tags sind die Schlüsselwörter. Nach den Namen der Tags wird in Modulen des Java- Programms gesucht, alle Schlüsselwörter des Bereichs eingelesen und mit ihnen weitergearbeitet. Die Baumstruktur der Tags in der Datei ist zwar nicht zwingend erforderlich, jedoch dient sie zur Übersicht und Zuordnung der Einzeltags im Gesamtzusammenhang.
Wenn ein Schlüsselwort mit einem “$“- Zeichen anfängt, bedeutet das, dass ein

Tag als Schlüsselwort verwendet wird (z.B. für die Zuweisung des Statementtypen). In diesem Fall wird der Inhalt zwischen zwei mathematischen Tags daraufhin analysiert, ob ein Ungleichungssymbol vorkommt, um den Statementtypen zu bestimmen. Diese Symbole werden in Tags dargestellt, die als solche ungeeignet für die „äußere“ XML- Struktur der Keywords.XML -Datei sind. Mit dem „$“- Zeichen davor ändert sich die Struktur der „Keywords.XML“- Datei nicht und die Tag-Umwandlung ist im Programm leicht zu realisieren. Die aktuelle und vollständige Version der Datei befindet sich im Anhang.

6.7.3 Bonner - DTD

Die im Unterpunkt 5.3 („DTD“) des Kapitels „Planung“ beschriebene BNF von Herrn Kolev [Kolev] wurde in diese DTD übertragen, um ein weiteres XML-Ausgabeformat mit integrierten Lemmata zu unterstützen:

```
<!ELEMENT text ([open-assumption]*, [definition]*, [proof]*)>

<!ELEMENT proof ("theorem.",[free-assumption]*,[definition]*,[statement]+,
                 "proof.", [closed-assumption | statement |lemma]+,"qed.")>

<!ELEMENT lemma ("lemma.",[assumption]*,[definition]*,[statement]+,
                 "proof.",[closed-assumption | statement]+,"qed.")>

<!ELEMENT open-assumption (assumption)>

<!ELEMENT closed-assumption (assumption,[statement]*,assumption-close)>

<!ELEMENT definition ("define",statement,["iff" | "if and only if"],statement)>

<!ELEMENT assumption (["assume that" |"assume for a contradiction that" |
                       "let" |"consider" ],statement)>

<!ELEMENT assumption-close ("thus", statement)>

<!ELEMENT free-assumption (assumption)>

<!ELEMENT statement (#PCDATA)>
```

Die wesentlichen Unterschiede der Bonner- DTD zur Satz.dtd sind die konstanten Text- Ausdrücke (z.B. „theorem.“). In beiden DTDs werden Elemente definiert, die wiederum andere Unterelemente haben. Um programmiertechnisch die Verarbeitung der „Satz.dtd“ und der Bonner-DTD zu vereinheitlichen wurde das Konzept der Trennung von grammatischen Regeln, die in der DTD abgelegt werden, und der

Schlüsselwörter, die in der Datei „Keywords.xml“ abgelegt werden auf die Bonner-DTD angewandt. Hieraus ergibt sich folgende Aufteilung:

a) Bonn.dtd
(siehe Anhang 10.7)
b) Bonn-Keywords.xml
(siehe Anhang 10.3)

Diese DTD musste im Programm „Scheme2XML“ hinzugefügt werden, um zum Beispiel die „Theory of Ordinals“ validieren zu können. Der grundliegene Unterschied des Beweises der „Theory of Ordinals“ zu den anderen Beweisen ist, dass der Beweis der „Theory of Ordinals“ mit Lemmata arbeitet. Es werden in einem Dokument also mehrere Beweise abgeschlossen, die dann anschliessend weiter verwendet werden. Das kann die Datei „Satz.dtd“ in ihrer jetzigen Form nicht.

6.7.4 Ersatzliste

In der Datei „ersatzliste.csv“ werden die Textinhalte aufgelistet, die direkt ersetzt werden können. Vor einer funktionalen Änderung im Programm war diese Liste hauptsächlich dazu gedacht, gewisse Scheme- Tags in einer SCM- Datei sofort XML- tauglich ersetzen zu können (z.B. <epsilon> in <epsilon/>). Diese Tags müssen ersetzt werden, da den öffnenden Tags kein schließendes Tag zugeordnet werden kann; dieses würde beim Parsen zu Fehlern führen. Die Liste beinhaltet zurzeit folgende Einträge:

(mid “|“);<OR/>
(cell ““);
(cell “ “);

Der Ausdruck links neben dem Semikolon wird durch den Ausdruck rechts neben den Semikolon ersetzt. Also wird aus jedem „(mid “|“)“ ein „<OR/>“ und die Zellen, die

entweder keinen oder ein Leerzeichen als Inhalt haben, werden praktisch gelöscht. Die Datei ist nicht wegen ihren drei Einträgen im Projekt geblieben, sondern um eventuelle konstante Inhalte der SCM- Datei, die im Laufe der Zeit als solche erkannt werden, leicht ersetzen zu können.

6.7.5 Satzliste

Die „Satzliste.csv“ beinhaltet eine Liste der Dateinamen, in der die bereits erfassten Beweise mit ihren zugeordneten IDs stehen:

```
1;potenzen
2;irrationaleZahlen
3;SatzVonRolle
4;Dreiecksungleichung_reell
5;SumNullfolge
6;lokalesMaximum
7;zwischenwertsatz
8;Summenformel
9;Bernoulli
10;SummegeraderZahlen
11;hauptideal
12;additionstheoreme
13;produktregel
14;Stammfunktion
```

Die Liste wird vom Java- Programm beim Erfassen eines neuen Beweises automatisch erweitert.

7 Testumgebung

7.1 Getestete TeXmacs-Beweise

Um die Software zu testen wurden in TeXmacs diverse mathematische Beweise erfasst. In diesem Test wurde untersucht, in wie weit die SCM-Dateien aus TeXmacs in die durch die Datei „Satz.dtd“ vorgegebene Struktur umgewandelt werden kann:

Mathematischer Beweis	**Beweistyp**	**validiert?**
Potenzen wachsen über alle Schranken	direkt	ja, (1)
Satz von Rolle	direkt	ja, (1)
Dreiecksungleichung für reelle Zahlen	direkt	ja, (1)
Summen von Nullfolgen	direkt	ja, (1)
lokales Maximum	direkt	ja, (1)
Summe zweier gerader Zahlen	direkt	nein*, (1)
Hauptideal	direkt	ja, (1)
Additionstheoreme	direkt	ja, (1)
Produktregel	direkt	ja, (1)
Stammfunktion	direkt	ja, (1)
$\sqrt{2}$ ist irrational	indirekt	ja, (1)
Zwischenwertsatz	vollst. Ind.	ja, (1)
Summenformel	vollst. Ind.	ja, (1)
Bernoullische Ungleichung	vollst. Ind.	ja, (1)
The Theory of Ordinals	Theory (siehe 6.7.3)	(ja), (2)

(1) Validierung nach der Datei „Satz.dtd“

(2) Validierung nach der Datei „Bonn.dtd“

* „gewollter“ Fehler durch ungünstiges SCM-Dokument

7.2 Testergebnis

Alle Beweise wurden an Hand der genannten Richtlinien erstellt. Deswegen konnten auch alle Beweise validiert werden. Die „Theory of Ordinals“ ist ein Beweis der Bonner Arbeitsgruppe des Naproche-Projektes. Er enthält im Gegensatz zu den anderen Dokumenten integrierte Lemmata (vgl. 6.7.3). Erstellt man diesen Beweis nicht nach den Regeln der Datei „Satz.dtd“ sondern nach der Datei „Bonn.dtd“, erhält man ein validiertes XML-Dokument. Diese Datei (Bonn.dtd) wurde im Unterpunkt 6.7.3 näher erläutert.

8 Fazit

8.1 Zusammenfassung

Das Ziel des Projektes ist erreicht: Mit dem entwickelten Java-Programm „Scheme2XML“ kann man mathematische Beweise unter den in Kapitel 5 und 6 ausgeführten Bedingungen parsen:
Das Programm bietet einem Anwender die Möglichkeit, einen mit dem Editor „TeXmacs“ erstellten Beweis auf seine Struktur hin zu analysieren, sofern er als SCM-Datei vorliegt. Für die Analyse benötigt man verschiedene Schlüsselwörter. Sie sind in Dateien hinterlegt, auf die im Programmablauf zugegriffen wird. Diese Dateien können unabhängig vom Quelltext gepflegt werden, wodurch eine problemlose Erweiterbarkeit der Schlüsselwörter garantiert ist. Der Anwender kann die Ausgabe eines jeden Analyseschrittes optional einstellen.
Das Programm bietet weiterhin die Möglichkeit, aus zwei DTDs auszuwählen. Je nach Auswahl wird ein anders strukturiertes XML-Ausgabeformat festgelegt. Der TeXmacs-Beweis wird in dieses XML-Format übertragen und validiert. Die dafür benötigten Methoden wurden im Rahmen dieser Bachelorarbeit entwickelt.
Zur Überprüfung der Qualität der Software wurden mehr als 15 Beweise ausgewählt und dokumentiert. Alle diese Beweise konnten unter Anwendung des Programms „Scheme2XML“ in ein wohlgeformtes XML-Format übertragen und die Validität bestätigt werden konnte.

8.2 Ausblick

Wie jede Software muss auch diese weiter gepflegt werden. Für die Umwandlung der TeXmacs-Struktur nach den Regeln der beiden DTDs sind Schlüsselwörter erforderlich. Diese stehen in einer Liste außerhalb des Quelltextes, auf die man unabhängig von dem Programm zugreifen kann. Stellt sich bei der Anwendung des Programms heraus, dass neue Schlüsselwörter erforderlich sind, lassen sich diese leicht nachträglich in die Liste aufnehmen.

Ein solches Verfahren ist auch für die Umwandlung der TeXmacs-Befehle denkbar. Diese Befehle werden zurzeit programmintern in eine XML-Struktur übersetzt. Die Zuordnung zu den Befehlen und ihren Methoden kann auch über eine Aufrufliste außerhalb des Quelltextes erfolgen.

Um mit dem Softwaremodul besser arbeiten zu können, sollte man einen Menüpunkt in TeXmacs implementieren, der das Programm „Scheme2XML“ aufruft und XML-Dateien erzeugt. Die Vorbereitungen hierzu sind schon fertig: die Methode, die auf die einzelnen Hauptmodule zugreift, wird von „Main“ mit allen dafür notwendigen Übergabeparametern aufgerufen.

9 Quellen

- Naproche:

  - [I_Naproche] Internetseite: http://www.math.uni-bonn.de/people/naproche/, letzer Besuch: 8.11.2008

  - [Material] Informationsmaterial von Prof. Dr. phil. Gregor Büchel

- XML / DTD / Parser:

  - [Erlenk.] Helmut Erlenkötter, Extensible Markup Language von Anfang an, rororo-Verlag, 1. Auflage

  - [I_selfHTML] Internetseite: http://de.selfhtml.org/, letzer Besuch: 8.11.2008

  - [I_w3XML] Internetseite: http://www.w3.org/XML/, letzer Besuch: 8.11.2008

  - [I_w3DTD] Internetseite: http://www.w3.org/QA/2002/04/valid-dtd-list.html, letzer Besuch: 8.11.2008

- Java:

  - [Java] Mäurers, Baufeld, Friedrich, Müller, Wabnitz, Mühle: Java - Das Grundlagenbuch, Data Becker, 1. Auflage

- MathML:

  - [I_MathML] Internetseite: http://www.w3.org/Math/, letzer Besuch: 8.11.2008

- TeXmacs:

  - [I_TeXmacs] Internetseite: http://www.texmacs.org/, letzer Besuch: 8.11.2008

  - [I_LaTeX] Internetseite: http://de.wikipedia.org/wiki/LaTeX, letzer Besuch: 8.11.2008

- Scheme:

  - [I_scheme] Internetseite: http://www.scheme.com/, letzer Besuch: 8.11.2008

- [I_wikiScheme] Internetseite: http://de.wikipedia.org/wiki/Scheme, letzer Besuch: 8.11.2008
- [I_bauerScheme] Internetseite: http://www.johannes-bauer.com/scheme/index.php, letzer Besuch: 8.11.2008

- mathematische Grundlagen und Beweise:
  - [Schweizer] Lambacher Schweizer: Mathematik Klasse 8, Klett-Verlag, 1. Auflage
  - [Forster] Otto Forster: Analysis 1, Vieweg-Verlag, 7. Auflage
  - [Waerden] B.L. van der Waerden: Algebra 1, Springer-Verlag, 8. Auflage
- Magisterarbeiten:
  - [Fisseni] „Entwicklung einer Annotationssprache für natürlichsprachlich formulierte mathematische Beweise“ von Bernhard Fisseni, Masterthesis, Universität Bonn, 2003
  - [Kolev] „Generating Proof Representation Structures in the Project Naproche“ von Nickolay Mitov Kolev, Masterthesis, University of Bonn, 2008

10 Anhang

10.1 Bedienungsanleitung

10.1.1 Installation des Programms

In dieser Bedienungsanleitung wird das Programm „Scheme2XML“ näher beschrieben. Zur Installation des Programms verbinden Sie die Datenquelle mit Ihrem PC und kopieren Sie den Ordner „Scheme2XML“ in ein lokales Verzeichnis Ihrer Wahl.

10.1.2 Starten des Programms

Öffnen Sie jetzt einen Konsolenfenster des Betriebssystems (Windows: „Start-> Ausführen...“ und dort „cmd“ eingeben; Linux: „Menu-> System -> Konsole). Wechseln Sie nun in das Unterverzeichnis „bin“ des angelegten Verzeichnisses („cd [name]/bin“). Das Programm wird über den Konsolenbefehl „java s2x_main“ gestartet:

```
-----------------------------------------------------
[1] Umwandeln einer bereits bearbeiteten SCM-Datei
[2] Umwandeln einer noch nicht bearbeiteten SCM-Datei
-----------------------------------------------------
[9] Einstellungen des Programms
-----------------------------------------------------
[0] Programm beenden
-----------------------------------------------------
```

Generell gilt, dass man mit dem Druck der Taste „0“ eine Programmebene zurück springt. Im Hauptfenster wird das Programm beendet.

10.1.3 Unterpunkt [1]

Wählt man nun den Unterpunkt [1] aus, folgt als nächstes die Wahl des Beweises, aus dem ein XML-Dokument erzeugt werden soll:

```
---------------------------------------
Folgende Dateien wurden bearbeitet:
[1] irrationaleZahlen
[2] SatzVonRolle
[3] Dreiecksungleichung_reell
[4] SumNullfolge
[5] lokalesMaximum
[6] zwischenwertsatz
[7] potenzen
[8] produktregel
[9] Summenformel
[10] Bernoulli
[11] Hauptideal
[12] SummegeraderZahlen
[13] UnglBruch
[14] additionstheoreme
[15] HauptsatzDiffuIntegralrechnungOK
Welcher Eintrag soll umgewandelt werden?
[0] um zurueck zu springen
---------------------------------------
```

Der Beweis wird nun erstellt und anschliessend vom Programm validiert. Die Bildschirmausgaben hängen von den Einstellungen ab, die über den Unterpunkt [9] getroffen wurden.

```
2
Datei akzeptiert: SatzVonRolle.scm
Erstellungsdatum: 15.12.2008    0:17:48

Quell-DTD: satz.dtd
Nicht zuordnungsbare Eintraege gefunden:
Eintraege verworfen

Parser:
---> Der Parser validiert.
Liste der fehlenden Tags:
keine fehlenden Tags gefunden!
```

Der Beweis wurde nun erneut erfasst.

10.1.4 Unterpunkt [2]

In diesem Unterpunkt kann man Dateien einlesen, die noch nicht erfasst wurden. Dazu gibt man den Pfad und den Namen der umzuwandelnden SCM-Datei bei den entsprechenden Eingabeaufforderungen an:

```
2
Geben Sie den Pfad ein, an dem sich die SCM-Datei befindet.
z.B. [D:/scheme2xml/scm/]
oder [/home/user/workspace/scheme2xml/scm/]

C:/workspace/scheme2xml/scm/
Pfad: C:/workspace/scheme2xml/scm/
Geben Sie nun den Namen der SCM-Datei ein:
z.B. [potenzen]

potenzen
Datei wird bearbeitet: potenzen.scm

Quell-DTD: satz.dtd
Nicht zuordnungsbare Eintraege gefunden:
Eintraege verworfen

Parser:
---> Der Parser validiert.
Liste der fehlenden Tags:
keine fehlenden Tags gefunden!
```

Der Beweis wird genau wie im Unterpunkt [1] nun erstellt und anschliessend von Programm validiert. Auch hier hängen die Bildschirmausgaben von den Einstellungen im Unterpunkt [9] ab. Der neue Beweis wurde erfasst und auch in der Liste der vorhandenen Beweise aufgenommen.

10.1.5 Unterpunkt [9]

Im Unterpunkt [9] werden die Einstellungen des Programms vorgenommen. Voreingestellt sind folgende Einstellungen:

```
9
------------------------------------------------------------
Einstellungen der Bildschirmausgaben:

[1] Ausgabe der SCM-Datei:      false
[2] Ausgabe der Vorbereitung:   false
[3] Ausgabe der Nachbreitung:   false
[4] Ausgabe der XML-Datei:      false
[5] Ausgabe des Parsers:        true
------------------------------------------------------------
Auswahl der DTD:
[6] Nach satz.dtd erstellen?    true
[7] Nach bonn.dtd erstellen?    false
------------------------------------------------------------
Zum Aendern der Einstellungen die jeweilige Nummer druecken
[0] Zurueck zur Auswahl
------------------------------------------------------------
```

Den oberen Bereich, die Bildschirmausgaben, kann man beliebig seinen Wünschen anpassen. Steht ein „false“ hinter dem Punkt, wird er nicht angezeigt, bei einem „true“ sieht man das Ergebnis dieses Schritts. Mit einem Druck auf die entsprechende Zifferntaste ändern sich die Werte (von true auf false und umgekehrt).

```
------------------------------------------------------------
Einstellungen der Bildschirmausgaben:

[1] Ausgabe der SCM-Datei:      true
[2] Ausgabe der Vorbereitung:   true
[3] Ausgabe der Nachbreitung:   true
[4] Ausgabe der XML-Datei:      true
[5] Ausgabe des Parsers:        true
------------------------------------------------------------
Auswahl der DTD:
[6] Nach satz.dtd erstellen?    true
[7] Nach bonn.dtd erstellen?    false
------------------------------------------------------------
Zum Aendern der Einstellungen die jeweilige Nummer druecken
[0] Zurueck zur Auswahl
------------------------------------------------------------
```

Hier wurden jetzt alle Ausgaben auf „true“ gesetzt; es wird also jeder einzelne Zwischenschritt ausgegeben. Sollte ein Fehler wärend des Programmablaufs auftreten, werden die entsprechenden Fehlermeldungen immer angezeigt. Die Auswahl der DTD funktioniert mit einer kleinen Einschränkung genau so: Das „true“ gibt an, nach welcher DTD validiert wird. Es kann immer nur eine DTD true sein. Wechselt man die DTD, wird der andere Wert automatisch auf false gesetzt.

10.2 Keywords.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!-- "$" bedeutet, das es sich um einen Tag als Schlüsselwort handelt. -->

<Satz-Info>
    <Titel>
        Satz
        Titel
    </Titel>
    <Thema>
        mathematisches Teilgebiet
        Teilgebiet
        Thema
    </Thema>
    <Quelle>
        Autor
        Verfasser
        Quelle
    </Quelle>
    <Bemerkung>
        Bem
        Bemerkung
    </Bemerkung>
</Satz-Info>

<Voraussetzung>
    Voraussetzung
    Vor
</Voraussetzung>

<Behauptung>
    Behauptung
    Beh
</Behauptung>

<Beweis>
    Beweis
    Bew

    <Beweistyp>
        <direkt>
            <!-- Wenn kein anderer Fall zutrifft -->
        </direkt>

```

```
44          <indirekt>
45              Widerspruch
46              contradiction
47              Annahme
48
49              <Invert>
50                  Annahme
51              </Invert>
52
53              <InvertBew>
54                  Beweisschritt
55                  Beweisschritte
56              </InvertBew>
57
58              <Wider>
59                  Widerspruch
60              </Wider>
61          </indirekt>
62
63          <vollstInduktion>
64              Induktion
65              induction
66
67              <Anker>
68                  Induktionsverankerung
69                  anker
70                  Anker
71              </Anker>
72
73              <Schluss>
74                  schluss
75                  schluß
76
77                  <IndAn>
78                      Induktionsannahme
79                      Induktionsvor
80                  </IndAn>
81
82                  <IndBeh>
83                      Induktionsbehauptung
84                      Induktionsbeh
85                  </IndBeh>
86
87                  <IndBew>
88                      Induktionsbeweis
```

```
                    Induktionsbew
                </IndBew>
            </Schluss>
        </vollstInduktion>
    </Beweistyp>

    <Statementtyp>
        Ungleichungskette
    </Statementtyp>

    <Beweisende>
    qed
    q.e.d.
    </Beweisende>
</Beweis>

<Trennen>
    $MATHreihe
    $MATHreihen
</Trennen>

<AusnahmePunkte>
    bzw.
    z.B.
</AusnahmePunkte>



<Gleichung>
    ==
    $equal
</Gleichung>
<Aequivalent>
    $thicksim
</Aequivalent>
<Ungleichung>
    $gtr
    $geq
    $geqslant
    $leq
    $less
    $leqslant
</Ungleichung>

<Aussage>
```

```
134     $epsilon
135 </Aussage>
```

10.3 Bonn-Keywords.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
2  <!-- Alle Keywords rund um die Bestimmung der Bereiche der Bonner.dtd -->
3
4  <proof>
5      <beginproof>
6          Theorem.
7      </beginproof>
8
9      <centerproof>
10         Proof.
11     </centerproof>
12
13     <endproof>
14         qed.
15         q.e.d.
16         Qed.
17         Q.e.d.
18     </endproof>
19 </proof>
20
21 <lemma>
22     <beginlemma>
23         Lemma.
24     </beginlemma>
25
26     <centerlemma>
27         Proof.
28     </centerlemma>
29
30     <endlemma>
31         qed.
32         q.e.d.
33         Qed.
34         Q.e.d.
35     </endlemma>
36 </lemma>
37
38 <definition>
```

```
39      <begindefinition>
40          Define
41      </begindefinition>
42
43      <enddefinition>
44          iff
45          if and only if
46      </enddefinition>
47  </definition>
48
49  <assumption>
50      Assume that
51      Assume for a contradiction that
52      let
53      consider
54  </assumption>
55
56  <assumption-close>
57      thus
58  </assumption-close>
```

10.4 Ersatzliste.csv

```
1  (mid "|");<OR/>
2  (cell "");
3  (cell " ");
```

10.5 Satzliste.csv

```
1
2
3  1;irrationaleZahlen
4
5  2;SatzVonRolle
6
7  3;Dreiecksungleichung_reell
8
9  4;SumNullfolge
10
11 5;lokalesMaximum
12
```

```
6;zwischenwertsatz

7;potenzen

8;produktregel

9;Summenformel

10;Bernoulli

11;Hauptideal

12;SummegeraderZahlen

13;UnglBruch

14;additionstheoreme

15;HauptsatzDiffuIntegralrechnungOK
```

10.6 Satz.dtd (Version 3.4)

```
<!-- DTD  : Satz.DTD                                    -->
<!-- Zweck: Eine DTD f einfach strukturierte            -->
<!--        mathematische Beweise                       -->
<!-- Verf.: Sebastian Zittermann                        -->
<!-- Stand: 01.09.2008                                  -->
<!-- Version: 0.5                                       -->
<!-- ------------------------------------------------- -->
<!-- Die XML-Wurzel ist: Satz.                          -->
<!--     Ein Satz braucht eine ID.                      -->
<!--     Ein Satz hat SatzInformationen.                -->
<!--     Ein Satz kann mehrere Voraussetzungen haben.   -->
<!--     Ein Satz hat eine Behauptung.                  -->
<!--     Ein Satz hat einen Beweis.                     -->
<!-- ------------------------------------------------- -->
<!ELEMENT Satz (SatzInformationen,Voraussetzungen*,Beh,Bew)>
<!ATTLIST Satz
    ID CDATA #REQUIRED
>

<!-- ------------------------------------------------- -->
<!-- Informationen:                                     -->
```

```
22 <!-- Der Satztitel muss enthalten sein.                  -->
23 <!-- Das mathematische Teilgebeit kann angegeben werden. -->
24 <!-- Der Autor kann angegeben werden.                    -->
25 <!-- Es können mehrere Bemerkungen hinterlegt werden.    -->
26 <!-- ------------------------------------------------- -->
27 <!ELEMENT SatzInformationen (Satztitel,Mathemteilgebiet?,Quelle?,Bemerkung*)>
28 <!ELEMENT Satztitel (#PCDATA)>
29 <!ELEMENT Mathemteilgebiet (#PCDATA)>
30 <!ELEMENT Quelle (#PCDATA)>
31 <!ELEMENT Bemerkung (#PCDATA)>
32
33 <!-- ------------------------------------------------- -->
34 <!-- Voraussetzungen:                                  -->
35 <!-- Es muss mindestens eine Vor vorhanden sein.       -->
36 <!-- Jede einzelne Vor besteht aus einer Statementfolge. -->
37 <!-- Jede einzelne Vor bekommt eine ID zugewiesen.     -->
38 <!-- ------------------------------------------------- -->
39 <!ELEMENT Voraussetzungen (Vor+)>
40 <!ELEMENT Vor (Statementfolge)>
41 <!ATTLIST Vor
42     ID CDATA #REQUIRED
43 >
44
45 <!-- ------------------------------------------------- -->
46 <!-- Beh:                                              -->
47 <!-- Beh besteht aus einer Statementfolge.             -->
48 <!-- ------------------------------------------------- -->
49 <!ELEMENT Beh (Statementfolge)>
50
51 <!-- ------------------------------------------------- -->
52 <!-- Bew:                                              -->
53 <!-- Es gibt drei verschiedene Beweismodi.             -->
54 <!-- Jeder Beweismodus hat seine eigene BewArgFolge    -->
55 <!-- ------------------------------------------------- -->
56 <!ELEMENT Bew (BewModus,BewArgFolge)>
57 <!ELEMENT BewModus (#PCDATA)>
58 <!ELEMENT BewArgFolge (VollstIndukt|IndBew|DirBew)>
59
60 <!-- ------------------------------------------------- -->
61 <!-- vollst.Induktion: -->
62 <!-- ------------------------------------------------- -->
63 <!ELEMENT VollstIndukt (Induktionsverankerung,Induktionsschluss+)>
64 <!ELEMENT Induktionsverankerung(Statement,Statementtyp?)>
65 <!ELEMENT Induktionsschluss(Induktionsannahme,Induktionsbeh,Induktionsbeweis)>
66 <!ELEMENT Induktionsannahme(Voraussetzungen)>
```

```
67 <!ELEMENT Induktionsbeh(Beh)>
68 <!ELEMENT Induktionsbeweis(DirBew)>
69
70 <!-- ------------------------------------------------- -->
71 <!-- indirekter Beweis:                                -->
72 <!-- ------------------------------------------------- -->
73 <!ELEMENT IndBew (Invertierung,InvertBew,Widerspruch)>
74 <!ELEMENT Invertierung (Beh)>
75 <!ELEMENT InvertBew(DirBew)>
76 <!ELEMENT Widerspruch(Statementfolge)>
77
78 <!-- ------------------------------------------------- -->
79 <!-- direkter Beweis: -->
80 <!-- ------------------------------------------------- -->
81 <!ELEMENT DirBew (Arg*)>
82 <!ELEMENT Arg (Statementfolge+)>
83 <!ATTLIST Arg
84     ID CDATA #REQUIRED
85 >
86
87 <!-- ------------------------------------------------- -->
88 <!-- restliche Elemente: -->
89 <!-- ------------------------------------------------- -->
90
91 <!-- kann in beliebiger Reihenfolge mathematische Teile wie prädikatenlogische
       Ausdrücke enthalten-->
92 <!ELEMENT Statementfolge ((TEXT|(Mathe,Statementtyp?))+)>
93
94 <!-- Ein Textbereich -->
95 <!ELEMENT TEXT (#PCDATA)>
96
97 <!-- Statementtyp: Aufbauf wie CSV -->
98 <!-- erste Stelle (0 oder 1) gibt an, ob es eine prädeikatenlogische Aussage
       beinhaltet -->
99 <!-- hinter Semikolon steht, ob es sich um eine Gleichung (gl) oder Ungleichungen (
       ungl) handelt -->
100 <!ELEMENT Statementtyp (#PCDATA)>
101
102 <!ELEMENT Mathe (MATH|MATHmodus|MATHreihen)>
103
104 !ELEMENT MATH (#PCDATA)>
105 !ELEMENT MATHmodus (#PCDATA)>
106 !ELEMENT MATHreihen (#PCDATA)>
```

10.7 Bonn.dtd

```
<!-- DTD  : bonn.dtd                                    -->
<!-- Zweck: Eine DTD erstellt anhand der Magister-     -->
<!--        arbeit von N.Kolev                 -->
<!-- Verf.: Sebastian Zittermann              -->
<!-- Stand: 01.01.2009                                 -->
<!-- Version: 0.3 (engl.)                              -->
<!-- ################################################ -->

<!-- Die zusammengefassten Tags (EinzelneTags dynamisch) -->
<!ENTITY % Mathe "MATH|MATHmodus|MATHreihe|MATHreihen">
<!ENTITY % TagsMitInhalt "FTEXT|TEXT|MSUP|MSUB|Auflistung|KLAMMERN|BRUCH|int|uplimit|
    lowlimit|Ableitung|nicht|WURZEL|REIHE|CELL">
<!ENTITY % EinzelneTags "epsilon|bbb-R|gtr|bbb-N|forall|exists|geq|bbb-Q|OR|bbb-Z|
    wedge|alpha|less|rightarrow|xi|leq|varepsilon|leqslant|subset|searrow|nearrow|
    assign|infty|ldots|frak-a|neq|in|thicksim|lambda|neg|emptyset|leftrightarrow">

<!--eigentlich "text" statt "SATZ" -->
<!ELEMENT SATZ (assumption*, definition*, theorem*)>

<!ELEMENT assumption (#PCDATA|%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT definition (#PCDATA|%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT theorem (#PCDATA|%Mathe;|%TagsMitInhalt;|%EinzelneTags;|proof)*>

<!ELEMENT proof (#PCDATA|%Mathe;|%TagsMitInhalt;|%EinzelneTags;|lemma)*>

<!ELEMENT lemma (#PCDATA|%Mathe;|%TagsMitInhalt;|%EinzelneTags;|theorem)*>

<!-- Allgemeine DTD-Elemente die beim Programmdurchlauf erzeugt werden -->
<!ELEMENT FTEXT (#PCDATA|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT MATH (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT MATHmodus (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT MATHreihe (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!ELEMENT MATHreihen (%Mathe;|%TagsMitInhalt;|%EinzelneTags;)*>

<!-- NEU: ord, erstellt am: 17.7.2008   10:5:10 -->
<!ELEMENT TEXT (#PCDATA)>
<!ELEMENT neg (#PCDATA)>
```

```
41 <!ELEMENT exists (#PCDATA)>
42 <!ELEMENT in (#PCDATA)>
43 <!ELEMENT emptyset (#PCDATA)>
44 <!ELEMENT forall (#PCDATA)>
45 <!ELEMENT wedge (#PCDATA)>
46 <!ELEMENT rightarrow (#PCDATA)>
47 <!ELEMENT leftrightarrow (#PCDATA)>
```

10.8 OMDoc

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Die Wurzel allen Beweises -->
<!ELEMENT OMDOC (assertion|ling|metadata|proof)+>

<!-- allgemeine Attribute:

  id:
    "ID", Kennung fÃ¼r ein Element / Objekt, mindestens lokal
    eindeutig

  refid / xref:
    Verweis auf eine ID
-->


<!-- Ein paar Parameter-EntitÃ¤ten gestatten wir uns auch: -->
<!ENTITY % object "quant|expression|name">

<!-- Quasi Ã¼berall erlaubt: -->
<!ENTITY % words "ling|keyword|word">

<!-- Relationen zwischen SÃ¤tzen -->
<!ENTITY % rels "sem-relation|relation">


<!--
     Commented Mathematical Property (stammt aus OpenMath):
     Mathematischer Text; in dieser Magisterarbeit mit sematischer
     Annotation angereichert
-->
<!ELEMENT CMP (#PCDATA|arg|%object;|%rels;|formula|keyword|ling|
    metacomment|nl-method|method|nl-premise|scope)*>

<!-- Formal Mathematical Property (stammt aus OpenMath): Semantik Ã
     la OpenMath oder Content MathML; in dieser Magisterarbeit v.a.
     als Platzhalter verwandt.
-->
<!ELEMENT FMP (#PCDATA|name)*>


<!-- nicht weiter analysierte Meta-Kommentare zum Beweisverlauf -->
<!ELEMENT metacomment (#PCDATA|keyword)*>

<!-- Behauptung -->
<!ELEMENT assertion (#PCDATA|CMP)*>
<!ATTLIST assertion
    id CDATA #IMPLIED
>

<!-- Beweis -->

<!ELEMENT proof (#PCDATA|CMP|conclude|derive|hypothesis|metacomment
    |nl-method|method|proof|%rels;|ling)*>
<!ATTLIST proof
    for CDATA #IMPLIED
    id CDATA #IMPLIED
    theory CDATA #IMPLIED
>
<!--
  for:
    die Behauptung etc., die bewiesen wird;
    kann entfallen, wenn <proof> in das entspr. <derive>-Element
    eingebettet ist
  theory:
```

```
    Hintergrund-Theorie
-->

<!-- Ableitungsschritt -->
<!ELEMENT derive (CMP|FMP|proof|relation|keyword|ling|metacomment
    |nl-method|method|nl-premise|premise)*>
<!ATTLIST derive
    id CDATA #IMPLIED
    type (lacuna|really-there) "really-there"
>

<!-- letzter Ableitungsschritt; darf kein FMP enthalten -->
<!ELEMENT conclude (CMP|derive|relation|keyword|ling|metacomment
    |method|nl-premise|premise)*>
<!ATTLIST conclude
    id CDATA #IMPLIED
    type (lacuna|really-there) "really-there"
>
<!--
  type:
    entweder eine LÃ¼cke, oder ein tatsÃ¤chlich im Beweistext
    gefÃ¼hrter Beweis
-->

<!-- Hypothese -->

<!ELEMENT hypothesis (#PCDATA|CMP)*>
<!ATTLIST hypothesis
    id CDATA #IMPLIED
    discharged-in CDATA #IMPLIED
    xref CDATA #IMPLIED
>

<!--
  semantische Beziehungen zwischen Propositionen /
  Ableitungsschritten
-->
<!ELEMENT sem-relation (#PCDATA|arg|keyword|exchange)*>
<!ATTLIST sem-relation
    type (symmetry) #REQUIRED
>
<!--
  type:
    Art der Relation; bisher gefunden: Symmetrie beim Beweis durch
    Fallunterscheidung
-->

<!-- Austausch von Referenten -->
<!ELEMENT exchange (arg)*>
<!--
  erstes Argument wird gegen zweites Argument ausgetauscht;
  Kontext: Eltern-Element (sem-relation)
-->

<!-- sonstige Relationen zwischen expressions etc. -->
<!ELEMENT relation (#PCDATA|arg|keyword)*>
<!ATTLIST relation
    type (clarification|particularisation|contradiction|restatement)
                                                        #REQUIRED
>
<!--
  type:
    Art der Relation
-->
<!-- allgemeines Argument-Verweis-Element -->
<!ELEMENT arg (bridged-from|coreference|exchange|expression|formula
```

```
    |keyword|name|nl-premise)*>
<!ATTLIST arg
    refid CDATA #IMPLIED
>
<!--
  enthÃ¤lt entweder das Argument oder verweist darauf mittels
  refid
-->


<!-- Koreferenz, Split Antecedents -->
<!ELEMENT coreference (arg)*>
<!-- enthÃ¤lt Verweise auf alle Elemente, mit denen Koreferenz
     besteht; im einfachen Fall durch ein refid-Attribut zu ersetzen
-->

<!-- SchluÃŸregeln -->
<!-- ohne natÃ¼rlichsprachlichen Inhalt: -->
<!ELEMENT method (formula|arg|premise|FMP)*>
<!ATTLIST method
    xref CDATA #REQUIRED
>
<!-- mit natÃ¼rlichsprachlichem Inhalt: -->
<!ELEMENT nl-method (#PCDATA|arg|formula|FMP|CMP)*>
<!ATTLIST nl-method
    xref CDATA #REQUIRED
>
<!--
  xref:
    Verweis auf den Namen der SchluÃŸregel;
-->


<!-- Geltungsbereiche fÃ¼r Negationen und Quantoren: -->
<!ELEMENT scope (expression|formula|keyword|ling|nl-premise|quant
    |%rels;|scope-of)*>
<!ATTLIST scope
    id CDATA #IMPLIED
>

<!-- Zuordnung der Geltungsbereiche fÃ¼r Negationen und Quantoren:
-->
<!ELEMENT scope-of EMPTY>
<!ATTLIST scope-of
    refid CDATA #IMPLIED
>
<!--
  refid:
    die ID der Variablen, die durch den Quantor gebunden wird, oder
    ID der Negation
-->

<!--
  discharged-in:
    Schritt, in dem die Hypothese  verbraucht wird

  xref:
    Falls die Hypothese bereits genannt wurde, koreferente
    Proposition
-->

<!-- OMDoc-PrÃ¤misse (sozusagen PrÃ¤missen-Zeiger) -->
<!ELEMENT premise EMPTY>
<!ATTLIST premise
    xref CDATA #IMPLIED
```

```
>
<!--
  xref:
    verweist auf PrÃ¤misse
-->

<!-- natÃ¼rlichsprachliche PrÃ¤misse -->
<!ELEMENT nl-premise
  (#PCDATA|%object;|keyword|ling|metacomment|nl-premise)*>
<!ATTLIST nl-premise
    xref CDATA #IMPLIED
    form (conj|simple) "simple"
>
<!-- kann im Gegensatz zu <premise> Inhalt haben

  xref: wie <premise>

  form:
    natÃ¼rlichsprachliche PrÃ¤missen kÃ¶nnen durch eine Konjunktion
    verbunden sein ("conj"), oder einzeln auftreten ("simple")
-->


<!-- Element, das nicht semantisch Relevantes klammert -->
<!ELEMENT ling (#PCDATA|word)*>

<!-- Element, das verhindert, daÃŸ Text Â«in der Luft hÃ¤ngtÂ» -->
<!ELEMENT word (#PCDATA)>
<!ATTLIST word
    id CDATA #IMPLIED
>
<!--
  id:
    einmal gebraucht fÃ¼r Kombination aus Determinierer und
    PrÃ¤position
-->


<!-- SchlÃ¼sselwÃ¶rter, die bestimmte Kategorien anzeigen -->

<!ELEMENT keyword (#PCDATA)>
<!ATTLIST keyword
    type (arg|bridging|clarification|hypothesis|implication
        |predicate|derive|conj|quant) #IMPLIED
>

<!--
  type:
    Name des Elements etc., das angezeigt wird
-->


<!-- expression: natÃ¼rlichsprachliche AusdrÃ¼cke
    - Propositionen
    - Verben
    - Substantive
    - Adjektive
    - Adjektive ...
-->

<!ELEMENT expression
    (#PCDATA|arg|bridged-from|coreference|neg|formula
    |det|keyword|ling|%object;|%rels;|restr|scope|word)*>
<!ATTLIST expression
    form (implication|rimplication|conj|disj|equivalence|simple)
                                                    "simple"
```

```
    id    CDATA #IMPLIED
    refid CDATA #IMPLIED
    ref-type (simple|recip) "simple"
    nr (pl|sg) "sg"
    pl-type (distributive|collective) "collective"
    type CDATA #IMPLIED
>
<!--
  form:
    fÃ¼r alle <expression>-Elemente ist "conj" erlaubt (Konjunktion)
    und natÃ¼rlich "simple".  Die anderen Werte sind v.a.  fÃ¼r
    Propositionen relevant (s. Kap. 9.1.2)

  refid:
    verweist auf id eines <expression>/<name>/...-Elements, mit dem
    die Expression koreferent ist.

  nr:
    Numerus

  pl-type:
    zeigt an, ob kollektive oder distributive Konjunktionen oder
    Plurale vorliegen

  type:
    Identifikation der Art der Phrase oder des Lexikoneintrags
    eines Wortes

  ref-type:
    "recip" fÃ¼r Reziprokpronomina; "simple sonst"

  -->

<!-- Namen -->

<!ELEMENT name (#PCDATA|bridged-from|expression|name|word)*>
<!ATTLIST name
    refid CDATA #IMPLIED
    type (compound|simple) "simple"
    id CDATA #IMPLIED
    class CDATA #IMPLIED
>

<!--
  refid, id: wie expression

  class:
    Die Form des Namens weist auf die Art des Bezeichneten hin.
    (Werte, die hier mÃ¶glich sind, kÃ¶nnen auch im type-Attribut des
     expression-Elements auftreten

  type:
    zusammengesetzter oder einfacher Name
-->




<!-- Quantoren -->

<!ELEMENT quant
    (#PCDATA|arg|bridged-from|coreference|%object;|%rels;
```

```
    |formula|det|%words;|restr|
    scope)*>
<!ATTLIST quant
    id CDATA #IMPLIED
    nr (pl|sg) "sg"
    type (exists|all|gen) "exists"
    binds CDATA #REQUIRED
>

<!-- Quantoren
  binds:
    ID der durch den Quantor gebundenen Variable

  id:
    wichtig fÃ¼r die Zuordnung eines Geltungsbereiches mit <scope-of>

  nr:
    Singular oder Plural

  type:
    Existenz-, All- oder anderer generalisierter Quantor
-->


<!-- Restriktor eines generalisierten Quantors -->
<!ELEMENT restr (#PCDATA|expression|%words;)*>


<!-- Determinierer -->
<!ELEMENT det (#PCDATA)>
<!ATTLIST det
    refid CDATA #IMPLIED
    id    CDATA #IMPLIED
    type  CDATA "definite"
>

<!--
  refid, id:
    im Prinzip wie bei expression; in den Beispielen verwendet, um
    eine Kombination aus Artikel und Determinierer darzustellen

  type:
    der Typ des Determinierers; Ã¼blicherweise der definite Artikel
-->


<!-- Bridging: -->

<!ELEMENT bridged-from (arg)*>
<!ATTLIST bridged-from
    by CDATA #REQUIRED
>

<!--
  by:
    Verweis auf die entsprechende Regel im Hintergrundwissen
-->


<!-- Die Negation;
 id: wichtig fÃ¼r die Zuordnung eines Geltungsbereiches mit <scope-
of>
 -->
```

10.9 Quelltexte

10.9.1 s2x_ausgabe

```
1
2  import java.util.LinkedList;
3
4  /**
5   * Diese Klasse enthaelt Methoden für die Ausgaben
6   * diverser Daten des Projekts
7   *
8   * @author Zittermann
9   */
10 public class s2x_ausgabe {
11
12     /**
13      * Gibt eine LinkedList aus
14      * @param aus: die auszugebende Liste
15      */
16     public static void ausgabeListe(LinkedList aus){
17         int anz = aus.size();
18         for(int i=0; i < anz; i++){
19             System.out.println(aus.get(i));
20         }
21     }
22
23     /**
24      * Gibt das Klammerarray aus
25      * @param klammern: die auszugebenden Klammern
26      */
27     public static void ausgabeKlammern(int klammern[][]){
28         int k = 0;
29         while(klammern[k][0] > 0){
30             if (klammern[k][2] == -1){
31                 System.out.println("K: " + k + "\tAnfang: " + klammern[k][0] + "\tEnde
                       : " + klammern[k][1]+ "\tEbene: " + klammern[k][2]);
32             }
33             else if (klammern[k][2] == -2){
34                 System.out.println("unbekanntes Kriterium in Zeile " + k + ": "+
                       klammern[k][0] + "\t" + klammern[k][1]);
35             }
36             else {
37                 System.out.println("K: " + k + "\tAnfang: " + klammern[k][0] + "\tEnde
                       : " + klammern[k][1]+ "\tEbene: " + klammern[k][2]);
38             }
```

```
39              k++;
40          }
41      }
42
43  }
```

10.9.2 s2x_bonn

```
1
2
3   import java.util.LinkedList;
4
5   public class s2x_bonn {
6       /**
7        * (Hauptteil)
8        * Wandelt das Format ähnlich der Datei Bonn.dtd um.
9        * @param XML: die zu strukturierende Liste
10       * @param name: name der Datei
11       * @return: nach der Bonn.dtd fertige XML-Datei
12       * @throws Exception: kann beim Einlesen auftreten
13       */
14      public static LinkedList erstelleXML(LinkedList XML, String name) throws Exception
            {
15          LinkedList tmp = new LinkedList();
16          LinkedList rest = new LinkedList();
17          LinkedList keywords = s2x_einlesen.leseDatei("../Keywords/Bonn-Keywords.xml");
18          LinkedList beginproof = s2x_einlesen.leseKeywords("beginproof", keywords);
19          LinkedList endproof = s2x_einlesen.leseKeywords("endproof", keywords);
20          LinkedList centerproof = s2x_einlesen.leseKeywords("centerproof", keywords);
21          LinkedList beginlemma = s2x_einlesen.leseKeywords("beginlemma", keywords);
22          LinkedList endlemma = s2x_einlesen.leseKeywords("endlemma", keywords);
23          LinkedList assumption = s2x_einlesen.leseKeywords("assumption", keywords);
24          LinkedList definition = s2x_einlesen.leseKeywords("begindefinition", keywords)
                ;
25          LinkedList point = new LinkedList();
26          LinkedList proof = new LinkedList();
27          LinkedList lemma = new LinkedList();
28          LinkedList beh = new LinkedList();
29          LinkedList bew = new LinkedList();
30          LinkedList erg = new LinkedList();
31          int ID;
32          int anfang = 0;
33          int ende = -1;
```

```
34
35          point.add(".");
36          int beginProof = s2x_keywords.postionErstesKeyword(beginproof, XML, anfang);
37
38          tmp = findeBereich(beginProof, XML, assumption, "assumption");
39          erg.addAll(tmp);
40
41          tmp = findeBereich(beginProof, XML, definition, "definition");
42          erg.addAll(tmp);
43
44
45          // ist dazeischen noch ein lemma, suche lemmaende, dann nochmal weiter
                dahinter theoremende
46          //vorgeplenkel speichern
47          // suche ab ende erstes theorem nächstes theorem mit ende
48
49          anfang = 0;
50          ende = -1;
51
52          // solange theoreme vorkommen
53          while (anfang > -1){
54              //suche erstes Theorem, dann suche erstes theoremende
55              if (ende > anfang)
56                  anfang = ende;
57              anfang = s2x_keywords.postionErstesKeyword(beginproof, XML, anfang);
58
59              if (anfang != -1){
60                  int start = -1, stop = -1;
61                  start = s2x_keywords.postionErstesKeyword(beginlemma, XML, anfang);
62                  ende = s2x_keywords.postionErstesKeyword(endproof, XML, anfang);
63
64                  // wenn ein Lemma im Proof vorhanden ist
65                  if (start > -1 && start < ende){
66
67                      // von Proof bis Lemma
68                      tmp = s2x_keywords.erzeugeListe(anfang, start, XML, false);
69                      stop = s2x_keywords.postionErstesKeyword(endlemma, XML, anfang);
70
71                      // Bestimmung des Lemmas:
72                      rest = s2x_keywords.erzeugeListe(start, stop, XML, true);
73
74                      rest = s2x_keywords.tabdazu(0, rest.size(), rest);
75                      rest.addFirst("<lemma>");
76                      rest.addLast("</lemma>");
77                      tmp.addAll(rest);
```

```
78
79                      // Bestimmung des Rests:
80                      stop = stop + 1;
81                      rest.clear();
82                      // Position von QED dacnach!
83                      ende = s2x_keywords.postionErstesKeyword(endproof, XML, stop);
84                      rest = s2x_keywords.erzeugeListe(stop, ende ,XML, true);
85                      //s2x_ausgabe.ausgabeListe(rest);
86
87                      // fertige Liste mit Lemma
88                      tmp.addAll(rest);
89
90                      // füge in fertige Liste Proof ein
91                      tmp = einfuegenProof(tmp, centerproof);
92
93                  }
94                  else {
95
96                      tmp = s2x_keywords.erzeugeListe(anfang, ende, XML, true);
97                      tmp = einfuegenProof(tmp, centerproof);
98
99                  }
100                 tmp = s2x_keywords.tabdazu(0, tmp.size(), tmp);
101                 tmp.addFirst("<theorem>");
102                 tmp.addLast("</theorem>");
103                 erg.addAll(tmp);
104             }
105             else {
106                 //System.out.println("--->Keine Theoreme mehr!! \n\n\n");
107             }
108
109
110         }
111         //erg.addFirst("<text>");
112         //erg.addLast("</text>");
113         erg.addFirst("<SATZ>");
114         erg.addLast("</SATZ>");
115
116         s2x_ausgabe.ausgabeListe(erg);
117
118         return erg;
119     }
120
121
122     /**
```

```
123      * Methode zur Gliederung in Bereiche
124      * @param pos: Position, bis zu der Untersucht werden soll
125      * @param XML: Die Liste, die Untersucht werden soll
126      * @param bereich: Die Schlüsselwörter, nach denen untersucht werden soll
127      * @param bereichtag: Bezeichnung der Gliederung
128      * @return: Liste, die den gesuchten Bereich beinhaltet
129      */
130     public static LinkedList findeBereich(int pos, LinkedList XML, LinkedList bereich,
              String bereichtag){
131         LinkedList erg = new LinkedList();
132         LinkedList point = new LinkedList();
133         LinkedList tmp = new LinkedList();
134 
135         point.add(".");
136         int anfang = s2x_keywords.postionErstesKeyword(bereich, XML, 0);
137         while ((anfang > -1) && (anfang < pos)){
138             tmp = null;
139             anfang = s2x_keywords.postionErstesKeyword(bereich, XML, anfang);
140             if ((anfang > -1) && (anfang < pos)) {
141                 int ende = s2x_keywords.postionErstesKeyword(point, XML, anfang);
142 
143                 tmp = s2x_keywords.erzeugeListe(anfang, ende, XML, true);
144                 tmp = s2x_keywords.tabdazu(0, tmp.size(), tmp);
145                 tmp.addFirst("<" + bereichtag +">");
146                 tmp.addLast("</" + bereichtag +">");
147                 //s2x_ausgabe.ausgabeListe(tmp);
148 
149                 erg.addAll(tmp);
150                 anfang = ende+1;
151             }
152         } // ende Definition-While
153         return erg;
154     }
155 
156     /**
157      * Fuegt den Proofbereich anhand der Schluesselwoerter in tmp ein
158      * @param tmp: Beweisliste
159      * @param centerproof: Liste mit Schluesselwoertern
160      * @return: angepasste Liste
161      */
162     public static LinkedList einfuegenProof(LinkedList tmp, LinkedList centerproof) {
163         // füge in fertige Liste Proof ein
164         int start = 0;
165         start = s2x_keywords.postionErstesKeyword(centerproof, tmp, start);
166         int stop = tmp.size();
```

```
167         tmp = s2x_keywords.tagBereich(start, stop, tmp, "proof");
168         //s2x_ausgabe.ausgabeListe(tmp);
169         // BIS HIER ALLES OK - nur noch PROOF einrücken!!!!
170 
171         LinkedList aaa = new LinkedList();
172         LinkedList rest = new LinkedList();
173         rest.clear();
174         rest = s2x_keywords.erzeugeListe(0, start ,tmp, true);
175         //s2x_ausgabe.ausgabeListe(rest);
176         aaa.addAll(rest);
177         rest.clear();
178         rest = s2x_keywords.tabdazu(start+1, stop+1, tmp);
179         //s2x_ausgabe.ausgabeListe(rest);
180         aaa.addAll(rest);
181         rest.clear();
182         rest = s2x_keywords.erzeugeListe(stop+1, tmp.size()-1 ,tmp, true);
183         //s2x_ausgabe.ausgabeListe(rest);
184         aaa.addAll(rest);
185 
186         return aaa;
187     }
188 }
```

10.9.3 s2x_einlesen

```
1 
2 import java.io.BufferedReader;
3 import java.io.FileReader;
4 import java.util.LinkedList;
5 
6 /**
7  * Diese Klasse enthaelt die Methoden zum Einlesen von Dateien
8  *
9  * @author Zittermann
10  */
11 public class s2x_einlesen {
12 
13     /**
14      * Liesst eine komplette Datei in eine LinkedList ein
15      * @param fname: Name der Datei
16      * @return: die Liste
17      */
18     public static LinkedList leseDatei(String fname) throws Exception{
```

```
19          String tmp = new String();
20          LinkedList erg = new LinkedList();
21          FileReader file_read = new FileReader(fname);
22          BufferedReader file_in = new BufferedReader(file_read);
23
24          do {
25              tmp = file_in.readLine();
26              if (tmp != null){
27                  erg.add(tmp);
28              }
29          }while(tmp!= null);
30
31          //System.out.println("Die Daten der Datei " + fname + " wurden eingelesen\n");
32          file_in.close();
33
34          return erg;
35      }
36
37      /**
38       * Liesst eine Datei nach Kriterien einer DTD in eine LinkedList ein
39       * @param fname: Name der Datei
40       * @return: die gefilterte Liste
41       */
42      public static LinkedList leseDTD(String fname) throws Exception{
43          LinkedList DTD = new LinkedList();
44          LinkedList erg = new LinkedList();
45
46          String tmpstr = new String();
47          String zeile = "dtd/" + fname;
48
49          char tmpchar;
50          int pos;
51
52          DTD = leseDatei(zeile);
53
54          System.out.println("Uebertragung der relevanten Daten in temporaere XML-Liste
                :\n");
55          for (int i = 0; i < DTD.size(); i++){
56              zeile = DTD.get(i).toString();
57              if (zeile.length() > 2) {
58                  tmpchar = zeile.charAt(2);
59                  if (tmpchar == '-'){
60                      //Mach nix
61                  }
62                  else if (tmpchar == 'A'){
```

```
63                      // Mach auch nix
64                  }
65                  else if (tmpchar == 'E'){
66                      // "End"-Elemente werden nicht aufgelistet
67                      pos = zeile.indexOf("#PCDATA");
68                      if (pos < 1) {
69                          erg.add(zeile.substring(10));
70                      }
71                  }
72                  else {
73                      // Packe die Attribute hinten dran
74                      pos = zeile.indexOf("CDATA");
75                      if (pos > 0) {
76                          tmpstr = zeile.substring(0,pos);
77                          tmpstr.trim();
78                          tmpstr = erg.getLast() + tmpstr;
79                          erg.removeLast();
80                          erg.add(tmpstr);
81                      }
82                  }
83
84              }
85          }
86          return erg;
87      }
88
89
90      /**
91       * Filtert die relevanten Keywords heraus
92       * @param bereich: Tag-Bereich in der XML-Datei
93       * @param Keywords: Die komplette Keyword-Datei als Liste
94       * @return: die gefilterte Liste
95       */
96      public static LinkedList leseKeywords(String bereich, LinkedList Keywords) {
97          LinkedList tmp = new LinkedList();
98          //LinkedList erg = new LinkedList();
99          String word = new String();
100         boolean aktiv = false;
101         boolean da = false;
102
103         // Sammeln der SchlÃ¼sselwÃ¶rter
104         for (int i = 0; i < Keywords.size(); i++) {
105             word = Keywords.get(i).toString().trim();
106             if ((word.indexOf("<" + bereich + ">") > -1) || aktiv == true){
107                 aktiv = true;
```

```
108                 //System.out.println("erg:" + Keywords.get(i).toString());
109
110                 if (word.indexOf("</" + bereich + ">") > -1){
111                     aktiv = false;
112                 }
113                 else {
114                     if (da == true) {
115                         // rausschneiden von der Untergruppierungen von
                              SchlÃ¼sselwÃ¶rtern
116                         if ((word.indexOf("<") > -1 && word.indexOf(">") > -1))
117                             i = Keywords.size();
118                         else
119                             if (word.length() > 1)
120                                 tmp.add(word);
121                     }
122                 }
123                 da = true;
124
125             }
126         }
127         if(da == false)
128             tmp = null;
129
130         return tmp;
131     }
132
133     /**
134      * sucht in einem String nach Statementtypen
135      * Aufbau der Statementtypen:
136      * x;gl,ungl, ...
137      * wobei x=1 fuer ein praedikatenlogischer Ausdruck
138      * und x=0 fuer kein praedikatenlogischer Ausdruck
139      * steht.
140      * Hinter dem Semikolon steht die Reihenfolge
141      * der Gleichungs-/Ungleichungstypen
142      * (liesst pro aufruf die datei Keywords ein)
143      * @param mathe: den zu untersuchenden mathematischen String
144      * @return:
145      * @throws Exception: kuennte bei Aufruf des Einlesend der Datei auftreten
146      */
147     public static String findeStatementtyp(String mathe) throws Exception{
148         LinkedList Keywords = s2x_einlesen.leseDatei("../Keywords/Keywords.xml");
149         LinkedList gl = s2x_einlesen.leseKeywords("Gleichung", Keywords);
150         LinkedList un = s2x_einlesen.leseKeywords("Ungleichung", Keywords);
151         LinkedList aequi = s2x_einlesen.leseKeywords("Aequivalent", Keywords);
```

```
152         LinkedList aus = s2x_einlesen.leseKeywords("Aussage", Keywords);
153         LinkedList aaa = new LinkedList();
154         String tmp = mathe;
155         String erg = new String();
156         int[] x;
157         int[] anzgl;
158         int[] anzun;
159         int anzaus = 0;
160 
161         for (int i = 0; i < aus.size(); i++){
162             tmp = aus.get(i).toString();
163             if (mathe.indexOf(tmp)> -1){
164                 anzaus = 1;
165             }
166         }
167         for (int i = 0; i < gl.size(); i++){
168             tmp = gl.get(i).toString();
169 
170             // Ueberprueft, ob es sich um ein XML-aehnliches Keyword handelt
171             tmp = s2x_keywords.checkKeyword(tmp);
172 
173             String tmpp = tmp + "Â§";
174 
175             if (mathe.indexOf(tmp)> -1){
176 
177                 x = s2x_suchen.zaehleElemente(mathe, tmp);
178                 int[] y = s2x_suchen.zaehleElemente(mathe, tmpp);
179 
180                 // gehe y durch und streiche die inhalte in x
181                 int anzy = 0;
182                 while (y[anzy] > -1){
183                     anzy++;
184                 }
185                 // anzy fertig;
186                 int anzx = 0;
187                 while (x[anzx] > -1){
188                     anzx++;
189                 }
190 
191                 // wenn Ueberhaupt erst kritische Eintraege vorhanden sind...
192                 if (anzy > 0){
193                     int[] relevante = s2x_keywords.findeUngleiche(x, y);
194                     x = new int[relevante.length];
195                     x = relevante;
196                     int z = 0;
```

```
197                     while (z < relevante.length){
198                         aaa.add(x[z] + ",gl");
199                         z++;
200                     }
201 
202                 }
203 
204                 // Wenn alles ist, wie es sein sollte!!!
205                 else {
206                     int z = 0;
207                     z = 0;
208                     while (x[z] > -1){
209                         aaa.add(x[z] + ",gl");
210                         z++;
211                     }
212                 }
213 
214 
215             }
216         }
217         for (int i = 0; i < un.size(); i++){
218             tmp = un.get(i).toString();
219             tmp = s2x_keywords.checkKeyword(tmp);
220             if (mathe.indexOf(tmp)> -1){
221                 x = s2x_suchen.zaehleElemente(mathe, tmp); // fuer jedes Wort ne
                        eigene Liste???
222                 int z = 0;
223                 while (x[z] > -1){
224                     aaa.add(x[z] + ",ungl");
225                     z++;
226                 }
227             }
228         }
229 
230         String gleichung = s2x_keywords.erstelleGleichungsstring(aaa);
231         if (gleichung.length() > 0) {
232             gleichung = gleichung.substring(0,gleichung.length()-1);
233 
234             erg = "<Statementtyp>";
235             erg = erg + anzaus + ";" + gleichung;
236             erg = erg + "</Statementtyp>";
237             return erg;
238         }
239         else {
240             return null;
```

```
241         }
242     }
243 
244     /**
245      * Liesst die Daten der Datei ersatzliste.csv in ein Stringarray ein
246      * @return ein Stringarray, in dem
247      * @throws Exception
248      */
249     public static String[][] tauschen() throws Exception{
250         LinkedList ersatz = new LinkedList();
251 
252         ersatz = s2x_einlesen.leseDatei("../CSV/Ersatzliste.csv");
253         if (ersatz != null){
254             int anz = ersatz.size();
255             //System.out.println("Anzahl:" + anz);
256             String erg[][] = new String[anz][2];
257             for (int x = 0; x < anz; x++){
258                 String tmp = new String();
259                 tmp = ersatz.get(x).toString();
260                 if (tmp.indexOf(";") > 1){
261                 erg[x][0] = tmp.substring(0,tmp.indexOf(";"));
262                 erg[x][1] = tmp.substring(tmp.indexOf(";")+1);
263                 //System.out.println(x + ".tmp" + tmp + "\tinhalt: " + erg[x][0]+ "\
                        tersetzen: " + erg[x][1]);
264                 }
265                 else{
266                     System.out.println("Fertig!");
267                     erg[x][0] = null;
268                     erg[x][1] = null;
269                 }
270             }
271             return erg;
272         }
273 
274         else {
275             return null;
276         }
277 
278     }
279 
280 
281 
282 }
```

10.9.4 s2x_errorHandler

```
1
2  import java.io.File;
3  import java.io.IOException;
4  import javax.xml.parsers.SAXParserFactory;
5  import javax.xml.parsers.SAXParser;
6  import javax.xml.parsers.ParserConfigurationException;
7  import org.xml.sax.Attributes;
8  import org.xml.sax.SAXException;
9  import org.xml.sax.SAXParseException;
10 import org.xml.sax.helpers.DefaultHandler;
11 import java.util.LinkedList;
12
13 //public class s2x_errorHandler implements ErrorHandler{
14 public class s2x_errorHandler extends DefaultHandler{
15     public LinkedList fehlendeTags = new LinkedList();
16
17     public s2x_errorHandler(){
18         super();
19         //System.out.println("Konstruktur");
20     }
21
22     public void warning(SAXParseException ep) throws SAXException
23      {System.out.println("Parser meldet WARNUNG: "+ep.toString());
24       System.out.println("an der Entity        : "+ep.getPublicId());
25       System.out.println("Zeile,Spalte         : "+ep.getLineNumber()+","+ep.
             getColumnNumber());
26      }
27
28      public void error(SAXParseException ep) throws SAXException{
29          String tmp = new String();
30          System.out.println("Parser meldet FEHLER : "+ep.toString());
31          System.out.println("an der Entity        : "+ep.getPublicId());
32          System.out.println("Zeile,Spalte         : "+ep.getLineNumber()+","+ep.
                getColumnNumber());
33          tmp = ep.toString();
34          // wenn Tag fehlt:
35          if (tmp.indexOf("Element type") > -1 && tmp.indexOf("must be declared.") >
                -1){
36              // Finde die Tag-Bezeichnung
37              String x = new String();
38              int pos = tmp.indexOf("\"");
39              x = tmp.substring(pos+1);
40              pos = x.indexOf("\"");
```

```
41              x = x.substring(0,pos);
42              System.out.println("!!!-->Tag: " + x);
43
44              // Ueberpruefe, ob Tag bereits in Liste vorhanden ist
45
46              fehlendeTags.add(x);
47          }
48
49      }
50
51      public void fatalError(SAXParseException ep) throws SAXException
52      {System.out.println("Fataler FEHLER !!!   : "+ep.toString());
53       System.out.println("an der Entity        : "+ep.getPublicId());
54       System.out.println("Zeile,Spalte         : "+ep.getLineNumber()+","+ep.
             getColumnNumber());
55       System.exit(3);
56      }
57
58      public LinkedList getFehlendeTags(){
59          return fehlendeTags;
60      }
61
62
63     }
```

10.9.5 s2x_keywords

```
1
2  import java.util.LinkedList;
3  import java.io.*;
4
5  public class s2x_keywords {
6
7      /**
8       * gibt die Position des ersten Keywords in einer Liste ab einer bestimmen
            Position zurueck
9       * @param Keywords: Liste mit den zu untersuchenden Keywords
10      * @param XML: Liste in der gesucht wird
11      * @param pos: position, ab der gesucht wird
12      * @return: position in der XML-Liste
13      */
14     public static int postionErstesKeyword(LinkedList Keywords, LinkedList XML, int
           pos){
```

```
15          String zeile = new String();
16          String wort = new String();
17          int erg = -1;
18          for (int i = pos; i < XML.size(); i++){
19              zeile = XML.get(i).toString();
20              for (int k = 0; k < Keywords.size(); k++){
21                  wort = Keywords.get(k).toString();
22                  if (zeile.indexOf(wort) > -1 && erg == -1){
23                      erg = i;
24                  }
25              }
26          }
27          return erg;
28      }
29
30      /**
31       * erzeugt eine Liste aus einem bestimmten Bereich einer anderen Liste
32       * @param anfang: startposition des Bereichs
33       * @param ende: endposition des Bereichs
34       * @param XML: die aufzuteilende Liste
35       * @return: Teil der Liste
36       */
37      //public static LinkedList erzeugeListe(int anfang, int ende, LinkedList XML){
38      public static LinkedList erzeugeListe(int anfang, int ende, LinkedList XML,
            boolean mitende){
39          LinkedList erg = new LinkedList();
40          int i = 0;
41          if (anfang > -1 && (anfang < ende) && XML.size() > ende-1){
42
43              for(i = anfang; i < ende; i++){
44                  erg.add(XML.get(i).toString());
45              }
46
47              if (mitende == true){
48                  erg.add(XML.get(i).toString());
49              }
50
51          }
52          else if (anfang == ende){
53              erg.add(XML.get(anfang).toString());
54          }
55          else {
56              //System.out.println("Falsche Uebergabeparameter:\nanfang: " + anfang + "\
                    tende: " + ende + "\tGroesse: " + XML.size());
57              erg = null;
```

```
58          }
59
60          return erg;
61      }
62
63      /**
64       * Bestimmt aus der info-Liste die Info-Struktur
65       * @param info: beinhaltet den kompletten Satz-Informationsbereich
66       * @param Keywords: Liste mit allen Keywords
67       * @return: die fertige Satzinfo-Liste
68       */
69      public static LinkedList aktualisiereInfo(LinkedList info, LinkedList Keywords,
            boolean vor){
70          LinkedList erg = new LinkedList();
71          LinkedList tmp = new LinkedList();
72          LinkedList optvor = new LinkedList();
73          LinkedList titel = s2x_einlesen.leseKeywords("Titel", Keywords);
74          LinkedList thema = s2x_einlesen.leseKeywords("Thema", Keywords);
75          LinkedList quelle = s2x_einlesen.leseKeywords("Quelle", Keywords);
76          LinkedList bem = s2x_einlesen.leseKeywords("Bemerkung", Keywords);
77
78          int postitel = -1;
79          int posthema = -1;
80          int posquelle = -1;
81          int posbem = -1;
82
83          int anfang = 0;
84          int ende = s2x_keywords.postionErstesKeyword(titel, info, anfang);
85          postitel = ende;
86          if (postitel < 0 ) return null; // Abbruch weil Pflichtfeld
87
88          // Gibt es ein Thema?
89          posthema = s2x_keywords.postionErstesKeyword(thema, info, anfang);
90          posquelle = s2x_keywords.postionErstesKeyword(quelle, info, anfang);
91          posbem = s2x_keywords.postionErstesKeyword(bem, info, anfang);
92
93
94          // wenn nix vorkommt
95          if (posthema < 0 && posquelle < 0 && posbem < 0){
96              tmp = erzeugeListe(anfang, info.size(), info, false);
97              if (tmp == null) return null;
98              tmp = tagBereich(0, tmp.size(), tmp, "Satztitel");
99              erg.addAll(tmp);
100         }
101         // nur Thema
```

```
102         else if (posthema > -1 && posquelle < 0 && posbem < 0){
103             tmp = erzeugeListe(anfang, posquelle, info, false);
104             if (tmp == null) return null;
105             tmp = tagBereich(0, tmp.size(), tmp, "Satztitel");
106             erg.addAll(tmp);
107 
108             tmp = erzeugeListe(posquelle, info.size(), info, false);
109             if (tmp == null) return null;
110             tmp = tagBereich(0, tmp.size(), tmp, "Mathemteilgebiet");
111             erg.addAll(tmp);
112         }
113 
114         // nur quelle
115         else if (posthema < 0 && posquelle > -1 && posbem < 0){
116             tmp = erzeugeListe(anfang, posquelle, info, false);
117             if (tmp == null) return null;
118             tmp = tagBereich(0, tmp.size(), tmp, "Satztitel");
119             erg.addAll(tmp);
120 
121             tmp = erzeugeListe(posquelle, info.size(), info, false);
122             if (tmp == null) return null;
123             tmp = tagBereich(0, tmp.size(), tmp, "Quelle");
124             erg.addAll(tmp);
125         }
126 
127         // nur bemerkung
128         else if (posthema < 0 && posquelle < 0 && posbem > -1){
129             tmp = erzeugeListe(anfang, posbem, info, false);
130             if (tmp == null) return null;
131             tmp = tagBereich(0, tmp.size(), tmp, "Satztitel");
132             erg.addAll(tmp);
133 
134             tmp = erzeugeListe(posbem, info.size(), info, false);
135             if (tmp == null) return null;
136             tmp = tagBereich(0, tmp.size(), tmp, "Bemerkung");
137             erg.addAll(tmp);
138         }
139 
140         // Wenn ein thema vorkommt
141         else if (posthema > -1) {
142             ende = posthema;
143             tmp = erzeugeListe(postitel, ende, info, false);
144             if (tmp == null) return null;
145             tmp = tagBereich(0, tmp.size(), tmp, "Satztitel");
146             erg.addAll(tmp);
```

```
147
148             // Gibt es eine Quelle?
149             posquelle = s2x_keywords.postionErstesKeyword(quelle, info, anfang);
150             if (posquelle > -1) {
151                 ende = posquelle;
152                 tmp = erzeugeListe(posquelle, ende, info, false);
153                 if (tmp == null) return null;
154                 tmp = tagBereich(0, tmp.size(), tmp, "Mathemteilgebiet");
155                 erg.addAll(tmp);
156
157                 // Gibt es eine Bemerkung?
158                 posbem = s2x_keywords.postionErstesKeyword(bem, info, anfang);
159                 if (posbem > -1){
160                     ende = posbem;
161                     tmp = erzeugeListe(posbem, ende, info, false);
162                     if (tmp == null) return null;
163                     tmp = tagBereich(0, tmp.size(), tmp, "Quelle");
164                     erg.addAll(tmp);
165
166                     tmp = erzeugeListe(ende, info.size(), info, false);
167                     if (tmp == null) return null;
168                     tmp = tagBereich(0, tmp.size(), tmp, "Bemerkung");
169                     erg.addAll(tmp);
170                 }
171
172             }
173
174
175         }
176         else {
177             System.out.println("Fall noch nicht implementiert:");
178             s2x_ausgabe.ausgabeListe(info);
179         }
180
181         erg = tabdazu(0, erg.size(), erg);
182         erg.addFirst("<SatzInformationen>");
183         erg.addLast("</SatzInformationen>");
184
185         return erg;
186     }
187
188     /**
189      * Bestimmt aus der Vor-Liste mit Hilfe der Keywords die Vor-Struktur
190      * @param vor: beinhaltet den kompletten Voraussetzungsbereich
191      * @param Keywords: Liste mit allen Keywords
```

```
192      * @return: die fertige Voraussetzungsliste
193      * @throws Exception: kann beim "trennen" auftreten
194      */
195     public static LinkedList aktualisiereVor(LinkedList vor, LinkedList Keywords)
             throws Exception{
196         LinkedList erg = new LinkedList();
197         LinkedList tmp = new LinkedList();
198         LinkedList words = s2x_einlesen.leseKeywords("Voraussetzung", Keywords);
199         LinkedList trennen = s2x_einlesen.leseKeywords("Trennen", Keywords);
200
201         erg = trennen(vor,trennen, "Vor", words);
202
203         erg = tabdazu(0, erg.size(), erg);
204         erg.addFirst("<Voraussetzungen>");
205         erg.addLast("</Voraussetzungen>");
206         return erg;
207     }
208
209     /**
210      * Bestimmt aus der Vor-Liste mit Hilfe der Keywords die Vor-Struktur
211      * @param beh: beinhaltet den kompletten Behauptungsbereich
212      * @param Keywords: Liste mit allen Keywords
213      * @return: die fertige Behauptungsliste
214      * @throws Exception: kann beim "trennen" auftreten
215      */
216     public static LinkedList aktualisiereBeh(LinkedList beh, LinkedList Keywords)
             throws Exception{
217         LinkedList erg = new LinkedList();
218         LinkedList tmp = new LinkedList();
219         LinkedList trennen = s2x_einlesen.leseKeywords("Trennen", Keywords);
220
221         erg = trennen(beh, trennen, null, null);
222         erg = tabdazu(0, erg.size(), erg);
223         erg.addFirst("<Behauptung>");
224         erg.addLast("</Behauptung>");
225         return erg;
226     }
227
228     /**
229      * Die Hauptmethode der vollstaendigen Induktion.
230      * Hier wird die Induktion in bereiche aufgeteilt,
231      * an die entsprechenden Methode weitergeleitet
232      * und letzendlich zusammengesetzt.
233      * @param bew: beinhaltet den komplette Beweis
234      * @param Keywords: Liste mit allen Keywords
```

```
235      * @return: vollstaendige Induktionsliste
236      * @throws Exception: kann beim "trennen" auftreten
237      */
238     public static LinkedList induktionBearbeiten(LinkedList bew, LinkedList Keywords)
             throws Exception{
239         LinkedList erg = new LinkedList();
240         LinkedList kwschluss = s2x_einlesen.leseKeywords("Schluss", Keywords);
241         LinkedList zwischen = new LinkedList();
242         LinkedList ablage = new LinkedList();
243         String tmp = new String();
244         int anfang = 0;
245         int ende = -1;
246 
247         anfang = 0;
248         ende = s2x_keywords.postionErstesKeyword(kwschluss, bew, anfang);
249         ablage = erzeugeListe(anfang, ende, bew, false);
250         if (ablage == null) return null;
251         tmp = LinkedListToString(ablage);
252 
253         ablage.clear();
254         ablage.add("<Induktionsverankerung>");
255         ablage.add("\t<Statementfolge>");
256         ablage.add("\t\t" + tmp);
257         ablage.add("\t</Statementfolge>");
258         ablage.add("</Induktionsverankerung>");
259 
260         anfang = ende;
261         ende = bew.size();
262         zwischen = erzeugeListe(anfang, ende, bew, false);
263         if (zwischen == null) return null;
264 
265         erg.clear();
266         erg.addAll(ablage);
267         erg.add("<Induktionsschluss>");
268         ablage.clear();
269         // aufteilung in andere Methode
270         ablage = induktionsSchluss(zwischen, Keywords);
271         ablage = tabdazu(0, ablage.size(), ablage);
272         erg.addAll(ablage);
273         erg.add("</Induktionsschluss>");
274 
275         return erg;
276     }
277 
278     /**
```

```
279      * Teilt den Induktionsschluss in seine Unterelemente auf
280      * und gibt den zusammengesetzten Induktionsschluss zurueck
281      * @param induktion: kompletter Induktionsschluss des Beweises
282      * @param Keywords: Liste mit allen Keywords
283      * @return: vollstaendiger Induktionsschluss in einer Liste
284      * @throws Exception: kann beim "trennen" auftreten
285      */
286     public static LinkedList induktionsSchluss(LinkedList induktion, LinkedList
             Keywords) throws Exception{
287         LinkedList erg = new LinkedList();
288         LinkedList iannahme = new LinkedList();
289         LinkedList ibeh = new LinkedList();
290         LinkedList ibew = new LinkedList();
291         LinkedList kwannahme = s2x_einlesen.leseKeywords("IndAn", Keywords);
292         LinkedList kwbeh = s2x_einlesen.leseKeywords("IndBeh", Keywords);
293         LinkedList kwbew = s2x_einlesen.leseKeywords("IndBew", Keywords);
294         LinkedList bew = s2x_einlesen.leseKeywords("Trennen", Keywords);
295         LinkedList ablage = new LinkedList();
296         String tmp = new String();
297         String indschluss = new String();
298         int anfang = 0;
299         int ende = -1;
300
301         anfang = 0;
302         ende = s2x_keywords.postionErstesKeyword(kwannahme, induktion, anfang);
303         ablage = erzeugeListe(anfang, ende, induktion, false);
304         if (ablage == null) return null;
305         tmp = LinkedListToString(ablage);
306
307         erg.add(tmp);
308         erg = tabdazu(0, erg.size(), erg);
309         erg.addFirst("<Statementfolge>");
310         erg.addLast("</Statementfolge>");
311
312         // Voraussetzung
313         anfang = ende;
314         ende = s2x_keywords.postionErstesKeyword(kwbeh, induktion, anfang);
315         ablage.clear();
316         ablage = erzeugeListe(anfang, ende, induktion, false);
317         if (ablage == null) return null;
318         tmp = LinkedListToString(ablage);
319
320         ablage.clear();
321         ablage.add(tmp);
322         ablage = tabdazu(0, ablage.size(), ablage);
```

```
323         ablage.addFirst("<Statementfolge>");
324         ablage.addLast("</Statementfolge>");
325         ablage = tabdazu(0, ablage.size(), ablage);
326         ablage.addFirst("<Induktionsannahme>");
327         ablage.addLast("</Induktionsannahme>");
328         erg.addAll(ablage);
329 
330         // Behauptung
331         anfang = ende;
332         ende = s2x_keywords.postionErstesKeyword(kwbew, induktion, anfang);
333         ablage.clear();
334         ablage = erzeugeListe(anfang, ende, induktion, false);
335         if (ablage == null) return null;
336         tmp = LinkedListToString(ablage);
337 
338         ablage.clear();
339         ablage.add(tmp);
340         ablage = tabdazu(0, ablage.size(), ablage);
341         ablage.addFirst("<Statementfolge>");
342         ablage.addLast("</Statementfolge>");
343         ablage = tabdazu(0, ablage.size(), ablage);
344         ablage.addFirst("<Induktionsbeh>");
345         ablage.addLast("</Induktionsbeh>");
346         erg.addAll(ablage);
347 
348         // Beweis
349         anfang = ende;
350         ende = induktion.size();
351         ablage.clear();
352         ablage = erzeugeListe(anfang, ende, induktion, false);
353         if (ablage == null) return null;
354         tmp = LinkedListToString(ablage);
355 
356         ablage.clear();
357         ablage.add(tmp);
358         ablage = trennen(ablage, bew, "Arg", null); // trennt auf, nach kriterien des
                Beweises
359         ablage = tabdazu(0, ablage.size(), ablage);
360         ablage.addFirst("<Induktionsbeweis>");
361         ablage.addLast("</Induktionsbeweis>");
362         erg.addAll(ablage);
363 
364         return erg;
365     }
366 
```

```
367
368     /**
369      * Teilt den indirketen Beweis auf, ordnet die Bereiche des Beweises zu
370      * und gibt den fertigen Beweisteil als LinkedList zurueck
371      * @param bew: kompletter Beweisteil
372      * @param Keywords: Liste mit allen Keywords
373      * @param ID: Bezeichung der ID
374      * @return: fertigen indirekten Beweis
375      * @throws Exception: kann beim "trennen" auftreten
376      */
377     public static LinkedList indirektBew(LinkedList bew, LinkedList Keywords, String
         ID) throws Exception{
378         LinkedList erg = new LinkedList();
379         LinkedList trennen = s2x_einlesen.leseKeywords("Trennen", Keywords);
380         LinkedList kwinvert = s2x_einlesen.leseKeywords("Invert", Keywords);
381         LinkedList kwbew = s2x_einlesen.leseKeywords("InvertBew", Keywords);
382         LinkedList kwwider = s2x_einlesen.leseKeywords("Wider", Keywords);
383         LinkedList beweisfolge = new LinkedList();
384         LinkedList ablage = new LinkedList();
385         String tmp = new String();
386         int anfang = 0;
387         int ende = -1;
388
389         // "Bereich" kennzeichnen
390         anfang = 0;
391         ende = s2x_keywords.postionErstesKeyword(kwinvert, bew, anfang);
392         ablage = erzeugeListe(anfang, ende, bew, false);
393         if (ablage == null) return null;
394         tmp = LinkedListToString(ablage);
395
396         ablage.clear();
397         ablage.add("<Statementfolge>");
398         ablage.add("\t" + tmp);
399         ablage.add("</Statementfolge>");
400
401         erg.addAll(ablage);
402
403         // Invertierung des Beweises
404         anfang = ende;
405         ende = s2x_keywords.postionErstesKeyword(kwbew, bew, anfang);
406         ablage = erzeugeListe(anfang, ende, bew, false);
407         if (ablage == null) return null;
408         tmp = LinkedListToString(ablage);
409         ablage.clear();
410         ablage.add("<Invertierung>");
```

```
411         ablage.add("\t<Statementfolge>");
412         ablage.add("\t\t" + tmp);
413         ablage.add("\t</Statementfolge>");
414         ablage.add("</Invertierung>");
415         erg.addAll(ablage);
416 
417         // Beweisschritte nach Invertierung
418         anfang = ende;
419         ende = s2x_keywords.postionErstesKeyword(kwwider, bew, anfang);
420         ablage.clear();
421         ablage = erzeugeListe(anfang, ende, bew, false);
422         if (ablage == null) return null;
423         tmp = LinkedListToString(ablage);
424 
425         ablage.clear();
426         ablage.add(tmp);
427         beweisfolge = trennen(ablage, bew, ID, null); // trennt auf, nach kriterien
                des Beweises
428         ablage = tabdazu(0, beweisfolge.size(), beweisfolge);
429         ablage.addFirst("<InvertBew>");
430         ablage.addLast("</InvertBew>");
431         erg.addAll(ablage);
432 
433         // Ermittlung des Widerspruchs
434         anfang = ende;
435         ende = bew.size();
436         ablage = erzeugeListe(anfang, ende, bew, false);
437         if (ablage == null) return null;
438         tmp = LinkedListToString(ablage);
439         ablage.clear();
440         ablage.add("<Widerspruch>");
441         ablage.add("\t<Statementfolge>");
442         ablage.add("\t\t" + tmp);
443         ablage.add("\t</Statementfolge>");
444         ablage.add("</Widerspruch>");
445         erg.addAll(ablage);
446 
447         return erg;
448     }
449 
450     /**
451      * Bestimmt aus der Beweisliste die Beweisstruktur.
452      * Sorgt fï¿½r die richtige Bearbeitung des Beweisteiles.
453      * @param bew: kompletter Beweisteil
454      * @param Keywords: Liste mit allen Keywords
```

```
455      * @return: den kompletten ueberarbeiteten Beweis
456      * @throws Exception: kann beim "trennen" auftreten
457      */
458     public static LinkedList aktualisiereBew(LinkedList bew, LinkedList Keywords)
             throws Exception{
459         LinkedList erg = new LinkedList();
460         LinkedList beweis = s2x_einlesen.leseKeywords("Beweis", Keywords);
461 
462         // Beweismodus finden!
463         boolean b = istdrin("vollstInduktion", bew, Keywords);
464         int modus = 0;
465         if (b == true) {
466             //System.out.println("vollstaendige Induktion");
467             erg = induktionBearbeiten(bew, Keywords);
468             erg = tabdazu(0, erg.size(), erg);
469             erg = tabdazu(0, erg.size(), erg);
470             erg.addLast("\t</VollstIndukt>");
471             erg.addFirst("\t<VollstIndukt>");
472             modus = 2;
473         }
474         else {
475             b = istdrin("indirekt", bew, Keywords);
476             if (b == true) {
477                 //System.out.println("indirekter Beweis");
478                 erg = indirektBew(bew, Keywords, "Arg");
479                 erg = tabdazu(0, erg.size(), erg);
480                 erg = tabdazu(0, erg.size(), erg);
481                 erg.addLast("\t</IndBew>");
482                 erg.addFirst("\t<IndBew>");
483                 modus = 1;
484             }
485             else {
486                 //System.out.println("direkter Beweis!!!");
487                 erg = trennen(bew, Keywords, "Arg", beweis); // zur Zeit direkter
                        Beweis
488                 erg = tabdazu(0, erg.size(), erg);
489                 erg = tabdazu(0, erg.size(), erg);
490                 erg.addLast("\t</DirBew>");
491                 erg.addFirst("\t<DirBew>");
492                 modus = 0;
493             }
494         }
495 
496         erg.addFirst("<BewArgFolge>");
497         erg.addFirst("</BewModus>");
```

```
498         if (modus == 0) erg.addFirst("\tDirBew");
499         if (modus == 1) erg.addFirst("\tIndBew");
500         if (modus == 2) erg.addFirst("\tVollstIndukt");
501         erg.addFirst("<BewModus>");
502         erg.addLast("</BewArgFolge>");
503         erg = tabdazu(0, erg.size(), erg);
504         erg.addFirst("<Beweis>");
505         erg.addLast("</Beweis>");
506 
507         return erg;
508     }
509 
510     /**
511      * Untersuche eine Zeile, ob sie aufgeteilt werden muss:
512      * Wenn das der Fall ist, fuege an der richtigen Stelle ein Symbol ein.
513      * @param zeile: die zu untersuchende Zeile
514      * @param symbol: das einzufuegende Symbol
515      * @return: die angepasste Zeile
516      * @throws Exception: kann beim einlesen der Datei auftreten
517      */
518     public static String trennePunkte(String zeile, String symbol) throws Exception{
519         LinkedList Keywords = s2x_einlesen.leseDatei("../Keywords/Keywords.xml");
520         LinkedList ausnahme = s2x_einlesen.leseKeywords("AusnahmePunkte", Keywords);
521         boolean ok = true;
522 
523         int max = zeile.length()-1;
524         for (int i = max; i > 0; i--){
525             if (zeile.charAt(i) == '.'){
526                 // ueberpruefe Liste
527                 String tmp = new String();
528                 tmp = zeile.substring(0,i+1);
529                 ok = true;
530                 for(int k = 0; k < ausnahme.size(); k++){
531                     String an = new String();
532                     an = ausnahme.get(k).toString();
533 
534                     int pos = tmp.lastIndexOf(an);
535                     if (tmp.lastIndexOf(an) == (i - an.length()+1)){
536                         ok = false;
537                     }
538                 }
539 
540                 // wenn nicht in Liste, mache Rest
541                 if (zeile.charAt(i+1) == ' ' && ok == true){
542                     //System.out.println("Satzende -> trennen!");
```

```
543                     zeile = pasteString(zeile, "</TEXT>"+symbol+"<TEXT>", i+1);
544                 }
545                 if (zeile.charAt(i+1) == ':' && ok == true){
546                     //System.out.println(".: => trennen!");
547                     zeile = pasteString(zeile, "</TEXT>"+symbol+"<TEXT>", i+2);
548                 }
549                 if (zeile.charAt(i+1) == '\n' && ok == true){
550                     //System.out.println("Zeilenumbruch => trennen!");
551                     zeile = pasteString(zeile, "</TEXT>"+symbol+"<TEXT>", i+1);
552                 }
553 
554             }
555         }
556         return zeile;
557     }
558 
559     /**
560      * Sucht in einem String nach Keywords.
561      * werden diese gefunden, wird ein trennsymbol eingefuegt.
562      * @param zeile: die zu untersuchende Zeile
563      * @param symbol: das einzufuegende Symbol
564      * @param Keywords: die Liste mit den zu trennenden Keywords
565      * @return: die angepasste Zeile
566      */
567     public static String trenneKeywords(String zeile, String symbol, LinkedList
         Keywords){
568         String word = new String();
569         LinkedList kword = s2x_einlesen.leseKeywords("Trennen", Keywords);
570 
571         for (int i = 0; i < kword.size(); i++){
572             int pos = 0;
573             word = kword.get(i).toString();
574             if (word.indexOf("$") > -1){
575 
576                 pos = 1 + word.indexOf("$");
577                 word = "</" + word.substring(pos) + ">";
578 
579             }
580 
581             pos = zeile.indexOf(word);
582             String anfang = new String();
583             String tmp = new String();
584             String ende = zeile;
585             while (pos > -1) {
586                 tmp = ende.substring(0,pos+word.length());
```

```
587                 anfang = anfang + tmp + "$";
588                 ende = ende.substring(pos+word.length(),ende.length());
589                 //System.out.println("Anfang:" + anfang + "\t ende: " + ende);
590 
591                 pos = ende.indexOf(word);
592             }
593             zeile = anfang + ende;
594 
595             //System.out.println("trennwort:" + word + "\t zeile: " + zeile);
596         }
597         return zeile;
598     }
599 
600     /**
601      * Gibt eine Liste, die anhand der inhaltlichen (Doppel)-Punkte und Keywords die
            Liste aufteilt
602      * @param list: die zu trennende Liste
603      * @param keywords: die Liste mit den zu trennenden Keywords
604      * @param ID: Name des Bereichs
605      * @return: getrennte Liste
606      * @throws Exception
607      */
608     public static LinkedList trennen(LinkedList list, LinkedList keywords, String ID,
            LinkedList bereich)throws Exception{
609         LinkedList erg = new LinkedList();
610         LinkedList tmpwords = new LinkedList();
611         String zeile = new String();
612         String tmp = new String();
613         String symbol = "$";
614         String word = new String(); // Keywords
615         int pos = -1;
616         int ind = -1;
617         boolean rest = false;
618 
619         zeile = LinkedListToString(list);
620 
621 
622 
623         // Keywords
624         if (ID != null && zeile != null){
625             // Loesche das Keyword selber aus der Zeile raus - in der voraussetzung (
                vor) und dem beweis (arg)
626 
627             //if((ID.indexOf("Vor") > -1 || ID.indexOf("Arg") > -1) && bereich != null
                )){
```

```
628             if((ID.indexOf("Vor") > -1) && bereich != null){
629                 boolean ersetzt = false;
630                 //s2x_ausgabe.ausgabeListe(bereich);
631                 for (int h = 0; h < bereich.size(); h++){
632                     tmp = bereich.get(h).toString();
633                     ind = zeile.indexOf(tmp);
634                     // Nullpointer exception -> zeile wird null. warum?
635                     if (ind > -1 && ersetzt == false){
636                         int rechts = 0;
637                         //System.out.println("Testzeile:\n" +zeile);
638                         String fff = zeile.substring(ind+tmp.length());
639                         rechts = fff.indexOf(":") +1;
640                         fff = zeile.substring(ind+tmp.length()+rechts);
641                         fff.trim();
642                         zeile = zeile.substring(0,ind);
643                         //System.out.println("Anfang:\n" +zeile);
644                         zeile = zeile.trim() + fff;
645                         //System.out.println("Danach zeile:\n" +zeile);
646                         // Falls es in jetzt leeren textelementen stand.
647                         //zeile = s2x_verarbeiten.ersetzen(zeile, "<TEXT></TEXT>", "
                                ");
648                         ersetzt = true;
649                     }
650                 }
651
652             }//*/
653             //System.out.println("Test:\n" +zeile);
654             // fuegt ein Symbol als trennzeichen nach . : \n ein
655             if (zeile != null){
656                 zeile = trennePunkte(zeile, symbol);
657                 try {
658                      // fuegt ein Symbol als trennzeichen nach jedem gefundenen
                            Schluesselwort ein
659                     zeile = trenneKeywords(zeile, symbol, keywords);
660                 }catch (Exception e){};
661             }
662
663         }
664
665         if (zeile != null){
666         while (!(zeile.indexOf("</TEXT><TEXT>") < 0)) {
667             zeile = s2x_verarbeiten.ersetzen(zeile, "</TEXT><TEXT>", "</TEXT>"+symbol+
                    "<TEXT>");
668         }
669         while (!(zeile.indexOf("<TEXT></TEXT>") < 0)) {
```

```
670             zeile = s2x_verarbeiten.ersetzen(zeile, "<TEXT></TEXT>", "");
671         }
672         erg = StringToLinkedList(zeile, symbol, ID);
673 
674         while (!(zeile.indexOf(symbol) < 0)) {
675             zeile = s2x_verarbeiten.ersetzen(zeile, symbol, "\n");
676         }
677         }
678         return erg;
679     }
680 
681     /**
682      * (Hauptteil) erstellt zuerst 4 Listen aus der XML-Liste
683      * diese werden entsprechend durch Methoden bearbeitet
684      * und letzendlich zusammengefasst und zurueckgegeben
685      * @param XML: die zu strukturierende Liste
686      * @param name: name der Datei
687      * @return: nach der Satz.dtd fertige XML-Datei
688      * @throws Exception: kann beim Einlesen auftreten
689      */
690     public static LinkedList erstelleXML(LinkedList XML, String name, boolean optvor)
         throws Exception{
691         LinkedList tmp = new LinkedList();
692         LinkedList Keywords = s2x_einlesen.leseDatei("../Keywords/Keywords.xml");
693         LinkedList vorkey = s2x_einlesen.leseKeywords("Voraussetzung", Keywords);
694         LinkedList behkey = s2x_einlesen.leseKeywords("Behauptung", Keywords);
695         LinkedList bewkey = s2x_einlesen.leseKeywords("Beweis", Keywords);
696         LinkedList qedkey = s2x_einlesen.leseKeywords("Beweisende", Keywords);
697         LinkedList info = new LinkedList();
698         LinkedList vor = new LinkedList();
699         LinkedList beh = new LinkedList();
700         LinkedList bew = new LinkedList();
701         int ID;
702         int anfang = -1;
703         int ende = -1;
704 
705         // Einruecken der Liste zur besseren Darstellung
706         XML = tabdazu(0,XML.size(),XML);
707 
708         anfang = 0;
709         ende = s2x_keywords.postionErstesKeyword(vorkey, XML, anfang);
710 
711         //wenn ende komisch ist, gibts keine vorausetzung
712         // dann muss man bei Behauptung gucken...
713         if (ende <= anfang){
```

```
714             ende = s2x_keywords.postionErstesKeyword(behkey, XML, anfang);
715             info = erzeugeListe(anfang, ende, XML, false);
716             if (info == null) {
717                 System.out.println("Keine Behauptung gefunden.");
718                 return null;
719             }
720             // testausgabe:
721             //System.out.println("Infoliste vor Verarbeitung: "); s2x_ausgabe.
                    ausgabeListe(info);
722             info = aktualisiereInfo(info, Keywords, optvor);
723 
724         }
725         else {
726             info = erzeugeListe(anfang, ende, XML, false);
727             if (info == null){
728                 System.out.println("Keinen Infobereich gefunden.");
729                 return null;
730             }
731             info = aktualisiereInfo(info, Keywords, optvor);
732 
733             anfang = ende;
734             ende = s2x_keywords.postionErstesKeyword(behkey, XML, anfang);
735             vor = erzeugeListe(anfang, ende, XML, false);
736             if (vor == null) {
737                 System.out.println("Keine Behauptung gefunden.");
738                 return null;
739             }
740             vor = aktualisiereVor(vor, Keywords);
741         }
742 
743         anfang = ende;
744         ende = s2x_keywords.postionErstesKeyword(bewkey, XML, anfang);
745         beh = erzeugeListe(anfang, ende, XML, false);
746         if (beh == null) {
747             System.out.println("Keinen Beweis gefunden.");
748             return null;
749         }
750         beh = aktualisiereBeh(beh, Keywords);
751 
752         anfang = ende;
753         ende = s2x_keywords.postionErstesKeyword(qedkey, XML, anfang) +1;// da qed
                noch mit rein muss +1
754         bew = erzeugeListe(anfang, ende, XML, false);
755         if (bew == null) {
756             System.out.println("Kein Beweisende gefunden.");
```

```
757             return null;
758         }
759 
760         bew = aktualisiereBew(bew, Keywords);
761 
762         tmp = fusioniereListen(info, vor, beh, bew);
763         tmp = tabdazu(0, tmp.size(), tmp);
764 
765         ID = sucheSatzID(name); //ist "name" bereits erfasst?
766 
767         // Testausgabe:
768         //System.out.println("Liste: "); s2x_ausgabe.ausgabeListe(tmp);
769 
770 
771         if (ID > -1){
772             tmp.addFirst("<SATZ ID = \"" + ID + "\">");
773             tmp.addLast("</SATZ>");
774         }
775 
776         else {
777             System.out.println("Katastrophaler FEHLER!");
778             tmp = null;
779         }
780         return tmp;
781     }
782 
783     /**
784      * Sucht in der Datei "Satzliste.csv" ob es bereits eine Datei mit dem "namen"
            gibt
785      * @param name: zu untersuchender Name
786      * @return: ID des Namen / letzter +1 fuer neuen Namen
787      * @throws Exception: kann beim Einlesen der Datei auftreten
788      */
789     public static int sucheSatzID(String name) throws Exception{
790         int ID = -1;
791         String zeile = new String();
792         String tmp = new String();
793         boolean vorhanden = false;
794         LinkedList daten = s2x_einlesen.leseDatei("../CSV/Satzliste.csv");
795 
796         // Wenn Daten eingelesen wurden
797         if (daten.size() > 1){
798 
799             for(int i = 0; i < daten.size(); i++){
800                 zeile = daten.get(i).toString();
```

```
801                 if (zeile.indexOf(name) > -1){
802                     vorhanden = true;
803                     tmp = zeile.substring(0,zeile.indexOf(";"));
804                     ID = Integer.valueOf(tmp).intValue();
805                 }
806             }
807             if (vorhanden == false) {
808                 zeile = daten.getLast().toString();
809                 tmp = zeile.substring(0,zeile.indexOf(";"));
810                 ID = Integer.valueOf(tmp).intValue() + 1;
811                 schreibeSatzliste(name, ID);
812             }
813 
814         }
815 
816         // Wenn die Liste leer ist, fange mit ID = 1 an und schreibe den ersten
                Eintrag
817         else
818             schreibeSatzliste(name, 1);
819 
820         return ID;
821     }
822 
823     /**
824      * Fuege in der Satzliste den neuen Namen mit der zugehoerigen ID ein!
825      * @param name: name der Datei
826      * @param ID: ID der Datei
827      */
828     public static void schreibeSatzliste(String name, int ID){
829         String fname = "../CSV/Satzliste.csv";
830         String zeile = ID + ";" + name;
831         File ausgabedatei;
832         FileWriter fw;
833         BufferedWriter bw;
834         try {
835             ausgabedatei = new File(fname);
836             fw = new FileWriter(ausgabedatei,true);
837             bw = new BufferedWriter(fw);
838             bw.write("\n\n" + zeile);
839             bw.close();
840           }
841           catch (Exception e) {
842             System.out.println("Habe gefangen: " + e);
843           }
844     }
```

```
845
846     /**
847      * Umwandlung einer LinkedList in einen String
848      * @param x: die LinkedList
849      * @return: der String
850      */
851     public static String LinkedListToString(LinkedList x){
852         String erg = new String();
853         for (int i = 0; i < x.size(); i++){
854             erg = erg + x.get(i).toString().trim();
855         }
856         return erg;
857     }
858
859     /**
860      * Umwandlung von String in LinkedList.
861      * Kirterium fuer die neuen Listeneintraege ist das trennsymbol
862      *
863      * @param zeile: String der umgewandelt werden soll
864      * @param symbol: trennsymbol innerhalb des Strings
865      * @param bereich: Name des Bereichs, in den die Liste aufgeteilt wird
866      *                 (bei null wird dieser weggelassen)
867      * @return: die resultierende LinkedList
868      */
869     public static LinkedList StringToLinkedList(String zeile, String symbol, String
         bereich){
870         LinkedList erg = new LinkedList();
871         LinkedList ablage = new LinkedList();
872         String tmp = new String();
873         int max = zeile.length()-1;
874         int anfang = 0;
875         int id = 1;
876         tmp = zeile;
877         int weiter = 0;
878
879         // Unterteile, wenns drankommt
880         while (weiter < 1){
881
882             int pos = tmp.indexOf(symbol);
883             if (pos > -2) {
884                 if (pos == -1){
885                     pos = tmp.length();
886                     weiter++;
887                 }
888
```

```
889                 ablage.clear();
890                 ablage.add(tmp.substring(anfang,pos));
891                 ablage = tabdazu(0, ablage.size(), ablage);
892                 ablage.addFirst("<Statementfolge>");
893                 ablage.addLast("</Statementfolge>");
894 
895                 // Uebergebenes Attribut fuer den Bereich
896                 if (bereich != null){
897                     //erg.addFirst("<" + bereich + " ID=" + id + ">");
898                     ablage = tabdazu(0, ablage.size(), ablage);
899                     ablage.addFirst("<" + bereich + " ID= \"" + id + "\">");
900                     ablage.addLast("</" + bereich + ">");
901                     id++;
902                 }
903 
904                 erg.addAll(ablage);
905                 if (pos != tmp.length()){
906                     tmp = tmp.substring(pos+1,tmp.length());
907                 }
908             }
909 
910             else {
911                 weiter++;
912             }
913         }
914         return erg;
915     }
916 
917     /**
918      * Erstelle aus vier listen eine
919      * @param info: erste Liste
920      * @param vor: zweite Liste
921      * @param beh: dritte Liste
922      * @param bew: vierte Liste
923      * @return: fusionierte Liste
924      */
925     public static LinkedList fusioniereListen(LinkedList info, LinkedList vor,
         LinkedList beh, LinkedList bew){
926         LinkedList tmp = new LinkedList();
927         tmp.addAll(info);
928         tmp.addAll(vor);
929         tmp.addAll(beh);
930         tmp.addAll(bew);
931         return tmp;
932     }
```

```
933
934     /**
935      * Fuege einen Bereichstag in einer Liste ein
936      * @param anfang: Anfang des Bereichs
937      * @param ende: Ende des Bereichs
938      * @param XML: zu ergaenzende Liste
939      * @param bereich: Name des Bereichs
940      * @return: neue Liste
941      */
942     public static LinkedList tagBereich(int anfang, int ende, LinkedList XML, String
         bereich){
943         LinkedList erg = XML;
944
945         erg.add(ende, "</"+ bereich +">");
946         erg.add(anfang,"<"+ bereich +">");
947
948         return erg;
949     }
950
951     /**
952      * ruecke die Listeninhalte in einem Bereich ein
953      * @param anfang: Anfang des Bereichs
954      * @param ende: Ende des Bereichs
955      * @param XML: zu ergaenzende Liste
956      * @return: neue Liste
957      */
958     public static LinkedList tabdazu(int anfang, int ende, LinkedList XML){
959         LinkedList erg = new LinkedList();
960         for(int h = anfang; h < ende; h++){
961             String tab = "\t" + XML.get(h).toString();
962             erg.add(tab);
963         }
964         return erg;
965     }
966
967     /**
968      * Zur Ueberpruefug, ob  "bereich" in "liste" vorhanden ist
969      * @param bereich: Bereich in der Keyword-Datei
970      * @param liste: zu ueberpruefende Liste
971      * @param Keywords: alle Keywords
972      * @return: ob es gefunden wurde
973      * @throws Exception: beim Einlesen der Datei
974      */
975     public static boolean istdrin(String bereich, LinkedList liste, LinkedList
         Keywords) throws Exception{
```

```
976         boolean erg = false;
977         LinkedList words = s2x_einlesen.leseKeywords(bereich, Keywords);
978 
979         for (int k = 0; k < liste.size(); k++){
980             String zeile = liste.get(k).toString();
981             for (int i = 0; i < words.size(); i++){
982                 String w = words.get(i).toString();
983                 if (zeile.indexOf(w) > -1) {
984                     erg = true;
985                     break;
986                 }
987 
988             }
989         }
990         return erg;
991     }
992 
993     /**
994      * Fuege einen String an eine bestimmte Postition ein
995      * @param zeile: orginalzeile
996      * @param paste: der einzufuegende String
997      * @param pos: die postion
998      * @return: neuer String
999      */
1000    public static String pasteString(String zeile, String paste, int pos){
1001        String erg = new String();
1002        erg = zeile.substring(0,pos);
1003        erg = erg + paste;
1004        erg = erg + zeile.substring(pos,zeile.length());
1005        return erg;
1006    }
1007 
1008    /**
1009     * Ueberpruefe, ob es sich um ein XML-aehnliches oder ein einziges Zeichen ist
1010     * @param word: zu ueberpruefendes Wort
1011     * @return: korrektes Wort
1012     */
1013    public static String checkKeyword(String word){
1014        String erg = word;
1015        if(word.indexOf("$") > -1){
1016            erg = "<" + word.substring(1) + "/>";
1017        }
1018        if(word.indexOf("=") > -1){
1019            erg = word.substring(1);
1020        }
```

```
1021
1022            return erg;
1023        }
1024
1025        /**
1026         * untersucht zwei arrays, in wie weit ihre Inhalte identisch sind.
1027         * gleiche Elemente werden ausgelassen.
1028         * Das Ergebniss enthaelt also alle eintraege aus beiden genau einmal
1029         * @param x: erstes Array
1030         * @param y: zweites Array
1031         * @return: kombiniertes Array
1032         */
1033        public static int[] findeUngleiche(int[] x, int[] y){
1034            LinkedList tmp = new LinkedList();
1035            int anzx = 0;
1036            while (x[anzx] > -1){
1037                anzx++;
1038            }
1039            int anzy = 0;
1040            while (y[anzy] > -1){
1041                anzy++;
1042            }
1043
1044            for(int i = 0; i < anzx; i++){
1045                for(int k = 0; k < anzy; k++){
1046                    if (y[k]-1 != x[i]){
1047                        tmp.add("" + x[i]);
1048                    }
1049                }
1050            }
1051
1052            int zahlneu;
1053            int zahlalt = -1;
1054            int z = 0;
1055            int[] erg = new int[anzx-anzy];
1056            for (int i = 0; i < (anzx-anzy); i++){
1057                zahlneu = Integer.valueOf(tmp.get(z).toString()).intValue();
1058                if (zahlneu != zahlalt){
1059                    erg[i] = zahlneu;
1060                    z++;
1061                }
1062                else {
1063                    if (z < tmp.size()){
1064                        z++;
1065                        zahlneu = Integer.valueOf(tmp.get(z).toString()).intValue();
```

```
1066                }
1067            }
1068            zahlalt = zahlneu;
1069        }
1070        return erg;
1071    }
1072
1073    /**
1074     * erstellt fuer den Statementtyp den Gleichungsstring
1075     * @param arg: Liste mit den Argumenten
1076     * @return: Gleichungsstring
1077     */
1078    public static String erstelleGleichungsstring(LinkedList arg){
1079        String erg = new String();
1080        String tmp = new String();
1081        int pos[][] = new int[arg.size()][2];
1082        int ablage[][];
1083
1084        for (int i = 0; i < arg.size(); i++){
1085            tmp = arg.get(i).toString();
1086            tmp = tmp.substring(0,tmp.indexOf(","));
1087            pos[i][0] = Integer.valueOf(tmp).intValue();
1088            pos[i][1] = i;
1089
1090        }
1091        ablage = sortierePos(pos, arg.size(), 0);
1092        for (int i = 0; i < ablage.length; i++){
1093            tmp = arg.get(ablage[i][1]).toString();
1094            tmp = tmp.substring(tmp.indexOf(",")+1, tmp.length());
1095            erg = erg + tmp + ",";
1096        }
1097        return erg;
1098    }
1099
1100    /**
1101     * sortiere absteigend nach bubblesort ein int - array
1102     * @param klammern: zu sortierendes Array
1103     * @param gr: groesse des ersten Feldes
1104     * @param pos: position, nach der soritert werden soll
1105     * @return: sortiertes Feld
1106     */
1107    public static int[][] sortierePos(int klammern[][], int gr, int pos){
1108        int erg[][] = new int[gr][2];
1109        erg = klammern;
1110
```

```
1111         for (int i=0; i < gr-1; i=i+1)
1112               for (int j=gr-1; j > i; j=j-1)
1113                 if (erg[j-1][pos] > erg[j][pos])
1114                  {
1115                      int l = erg[j-1][0];
1116                      int r = erg[j-1][1];
1117                      erg[j-1][0] = erg[j][0];
1118                      erg[j-1][1] = erg[j][1];
1119                      erg[j][0] = l;
1120                      erg[j][1] = r;
1121
1122                  }
1123         return erg;
1124     }
1125
1126 }
```

10.9.6 s2x_main

```
1
2  import java.util.LinkedList;
3  import java.io.*;
4  import org.xml.sax.*;
5  import org.xml.sax.helpers.*;
6  import javax.xml.parsers.*;
7  import java.text.DateFormat;
8  import java.util.Date;
9  import java.text.*;
10 import java.util.Calendar;
11
12 public class s2x_main {
13
14     //-----------------------------------------------------------------
15     // Main
16     //-----------------------------------------------------------------
17     public static void main(String[] args)throws Exception {
18         LinkedList XML = new LinkedList();
19         LinkedList SCM = new LinkedList();
20
21         LinkedList pfadname = new LinkedList();
22         boolean scmdatei = false;
23         boolean vorbereitung = false;
24         boolean nachbereitung = false;
```

```
25          boolean xmldatei = false;
26          boolean parsen = true;
27          boolean satz = true;
28          boolean optvor = true;
29
30          String fname[] = new String[50];
31          String einstellungen = new String();
32          //String pfad = "texmacs/";
33          String pfad = "../scm/";
34          String tmp = new String();
35
36          fname[1] = "irrationaleZahlen";// indirekt
37          fname[2] = "SatzVonRolle"; // direkt -> OK?
38          fname[3] = "Dreiecksungleichung_reell"; // direkt -> BEISPIEL!
39          fname[4] = "SumNullfolge"; // direkt ! keine Vor!
40          fname[5] = "lokalesMaximum"; // direkt
41          fname[6] = "zwischenwertsatz"; // vollstaendige Induktion
42          fname[7] = "potenzen"; // direkt
43          fname[8] = "produktregel";  // dirket
44          fname[9] = "Summenformel";  // vollstaendige Induktion
45          fname[10] = "Bernoulli";  // vollstaendige Induktion
46          fname[11] = "Hauptideal";  // dirket
47          fname[12] = "SummegeraderZahlen";  // dirket
48          fname[13] = "UnglBruch";  // dirket
49          fname[14] = "additionstheoreme";  // dirket
50          fname[15] = "HauptsatzDiffuIntegralrechnungOK";  // dirket
51
52
53          //fname[30] = "symtest"; // ...
54          //fname[31] = "ord_meins"; //Bonn -> viele
55          //fname[32] = "ord"; // ord
56          //fname[33] = "HauptsatzDiffuIntegralrechnung";  // dirket
57          //fname[34] = "HauptsatzDiffuIntegralrechnungBeh";  // dirket
58
59          int x = 2;
60          int ausgabe = 0;
61
62          tmp = fname[x];
63
64          //"normal":
65          //programm(false, false, false, true, true, true, pfad, tmp, false);
66
67          // Bonn:
68          //tmp = "ord"; programm(false, false, false, true, false, true, pfad, tmp,
                false);
```

```
69
70
71          // Schleife für alle Beweise
72          /*
73          for (int iii = 1; iii < 16; iii++){
74              tmp = fname[iii];
75              programm(false, false, false, true, true, true, pfad, tmp, false);
76          }
77          //*/
78
79          //programm(true, true, true, true, true, parsen, "texmacs/", tmp, true);
80
81
82          // bis hier ALT
83
84          BufferedReader einlesen = new BufferedReader(new InputStreamReader(System.in))
                ;
85          int menu, auswahl = 0;
86
87
88          do{
89
90              menu = 0;
91              menu = hauptmenu();
92
93              switch(menu){
94
95                  case(0):
96
97                      System.out.println("Programm wurde beendet.");
98                      System.exit(0);
99
100                 break;
101
102                 case(1):
103
104                     do {
105                         System.out.println("---------------------------------------");
106                         System.out.println("Folgende Dateien wurden bearbeitet:");
107                         LinkedList dateien = s2x_einlesen.leseDatei("../CSV/Satzliste.
                                csv");
108                         pfadname.clear();
109                         for (int i = 0; i < dateien.size(); i++){
110                             tmp = dateien.get(i).toString();
111                             if (tmp.indexOf(";") > 0 ){
```

```
112                                 String id = tmp.substring(0,tmp.indexOf(";"));
113                                 String name = tmp.substring(tmp.indexOf(";")+1,tmp.
                                        length());
114                                 pfadname.add(name);
115                                 if (id.compareTo("0") >0){
116                                     System.out.println("[" + id + "] " + name);
117                                 }
118                             }
119                         }
120                         System.out.println("Welcher Eintrag soll umgewandelt werden?")
                                ;
121                         System.out.println("[0] um zurueck zu springen");
122                         System.out.println("---------------------------------------");
123 
124                         // Ausgabe aller in der Satzliste.csv-Datei vorhandenen Daten
125                         try {
126                             ausgabe = Integer.parseInt(einlesen.readLine());
127                             ausgabe = ausgabe - 1; // wegen der 0
128                             if (ausgabe < pfadname.size() && ausgabe > -1){
129                                 tmp = pfadname.get(ausgabe).toString()+"";
130                                 String time = Zeitstempel(tmp);
131                                 System.out.println("Datei akzeptiert: " + tmp + ".scm
                                        \nErstellungsdatum: " + time);
132                                 ausgabe = 0;
133 
134                                 programm(scmdatei, vorbereitung, nachbereitung,
                                        xmldatei, satz, parsen, pfad, tmp, optvor);
135                             }
136                             else {
137                                 if (ausgabe != -1){ // wegen des 0. Eintrags
138                                     System.out.println("...Fehlerhafte Eingabe: Bitte
                                            einen Menupunkt auswaehlen...\n");
139                                 }
140                                 else ausgabe = 0;
141                             }
142 
143 
144                         }
145                         catch (Exception e) {
146                             System.out.println("...Fehlerhafte Eingabe: Bitte einen
                                    Menupunkt auswaehlen...\n");
147                         }
148                     } while (ausgabe != 0);
149 
150                 break;
```

```
151
152                 case(2):
153
154                     System.out.println("Geben Sie den Pfad ein, an dem sich die SCM-
                            Datei befindet.");
155                     System.out.println("z.B. [D:/scheme2xml/scm/]");
156                     System.out.println("oder [/home/user/workspace/scheme2xml/scm/] \n
                            ");
157
158                     // Ausgabe aller in der Satzliste.csv-Datei vorhandenen Daten
159                     try {
160                         pfad = einlesen.readLine();
161                         System.out.println("Pfad: " + pfad);
162
163                         System.out.println("Geben Sie nun den Namen der SCM-Datei ein:
                                ");
164                         System.out.println("z.B. [potenzen] \n");
165
166                         tmp = einlesen.readLine();
167                         if (tmp.indexOf(".scm") > 0)
168                             tmp = tmp.substring(0,tmp.indexOf(".scm"));
169                         if (tmp.indexOf(".SCM") > 0)
170                             tmp = tmp.substring(0,tmp.indexOf(".SCM"));
171
172                         if (Dateida(pfad + tmp + ".scm") == true){
173                             System.out.println("Datei wird bearbeitet: " + tmp + ".scm
                                    ");
174                             programm(scmdatei, vorbereitung, nachbereitung, xmldatei,
                                    satz, parsen, pfad, tmp, optvor);
175
176                         }
177                         else {
178                             System.out.println("Diese Datei ist nicht vorhanden.");
179                         }
180
181                     }
182                     catch (Exception e) {
183                         System.out.println("Das XML-Dokument konnte nicht erstellt
                                werden. ");
184                     }
185                 break;
186
187                 case(9):
188
189                     do {
```

```
190                     System.out.println("
                            ------------------------------------------------------------
                            ");
191                     System.out.println("Einstellungen der Bildschirmausgaben:\n");
192                     System.out.println("[1] Ausgabe der SCM-Datei: \t" + scmdatei)
                            ;
193                     System.out.println("[2] Ausgabe der Vorbereitung: \t" +
                            vorbereitung);
194                     System.out.println("[3] Ausgabe der Nachbreitung: \t" +
                            nachbereitung);
195                     System.out.println("[4] Ausgabe der XML-Datei: \t" + xmldatei)
                            ;
196                     System.out.println("[5] Ausgabe des Parsers: \t" + parsen);
197                     System.out.println("
                            ------------------------------------------------------------
                            ");
198                     System.out.println("Auswahl der DTD:");
199                     System.out.println("[6] Nach satz.dtd erstellen? \t" + satz);
200                     System.out.println("[7] Nach bonn.dtd erstellen? \t" + !satz);
201                     System.out.println("
                            ------------------------------------------------------------
                            ");
202                     //System.out.println("[8] Optionale Voraussetzungen \n" +
203                     //                   "    automatisch zuordnen? \t" + optvor);
204                     //System.out.println
                            ("------------------------------------------------------------")
                            ;
205                     System.out.println("Zum Aendern der Einstellungen die
                            jeweilige Nummer druecken");
206                     System.out.println("[0] Zurueck zur Auswahl");
207                     System.out.println("
                            ------------------------------------------------------------
                            ");
208
209                     // Ausgabe aller in der Satzliste.csv-Datei vorhandenen Daten
210                     try {
211                         ausgabe = Integer.parseInt(einlesen.readLine());
212                         if (ausgabe == 1){
213                             scmdatei = !scmdatei;
214                         }
215                         if (ausgabe == 2){
216                             vorbereitung = !vorbereitung;
217                         }
218                         if (ausgabe == 3){
219                             nachbereitung = !nachbereitung;
```

```
220                             }
221                             if (ausgabe == 4){
222                                 xmldatei = !xmldatei;
223                             }
224                             if (ausgabe == 5){
225                                 parsen = !parsen;
226                             }
227                             if (ausgabe == 6){
228                                 satz = true;
229                             }
230                             if (ausgabe == 7){
231                                 satz = false;
232                             }
233                             //if (ausgabe == 8){optvor = !optvor;   }
234                         }
235                         catch (Exception e) {
236                             System.out.println("Es ist etwas schiefgelaufen: " + e);
237                         }
238 
239                     }while(ausgabe != 0);
240                 break;
241             }
242 
243         }while(menu != 0);
244 
245     }
246 
247     public static int hauptmenu(){
248         int erg = 0;
249         BufferedReader einlesen = new BufferedReader(new InputStreamReader(System.in))
                ;
250 
251         System.out.println("-----------------------------------------------------");
252         System.out.println("[1] Umwandeln einer bereits bearbeiteten SCM-Datei");
253         System.out.println("[2] Umwandeln einer noch nicht bearbeiteten SCM-Datei");
254         System.out.println("-----------------------------------------------------");
255         System.out.println("[9] Einstellungen des Programms");
256         System.out.println("-----------------------------------------------------");
257         System.out.println("[0] Programm beenden");
258         System.out.println("-----------------------------------------------------");
259 
260         try {
261 
262             erg = Integer.parseInt(einlesen.readLine());
263         }
```

```
264
265         catch(Exception e){
266
267             erg = -1;
268         }
269
270         return erg;
271     }
272
273 /**
274  * Fï¿½hrt das eigentliche Programm aus:
275  * - einlesen
276  * - Vorbereitung
277  * - Nachbereitung
278  * - XML-erstellen
279  * - parsen
280  * @param s: bei true: scm ausgeben
281  * @param v: bei true: vorbereitung ausgeben
282  * @param n: bei true: nachbereitung ausgeben
283  * @param x: bei true: xml ausgeben
284  * @param satz: ob es nach Satz / Bonn-dtd umgewandelt wird
285  * @param p: bei true: parsen ausgeben
286  * @param pfad
287  * @param name
288  * @throws Exception
289  */
290     public static void programm(boolean s, boolean v, boolean n, boolean x, boolean
         satz, boolean p, String pfad, String name, boolean optvor) throws Exception{
291         LinkedList XML = new LinkedList();
292         LinkedList SCM = new LinkedList();
293         String einstellungen = new String();
294         boolean fehler = false;
295         String time = Zeitstempel(name);
296
297         SCM = s2x_einlesen.leseDatei(pfad + name + ".scm");
298         //if (pfad.compareTo("scm/")> 0){
299             System.out.println("Datei muss geschrieben werden: " + pfad);
300             System.out.println("Ziel: " + "scm/" + name + ".scm");
301             s2x_schreibeDatei.datenSchreiben("../scm/" + name + ".scm", SCM ); //
                 schreibe SCM-Datei in SCM-Ordner
302             System.out.println("Datei geschrieben!!!");
303         //}
304
305         if (s == true) {
306             System.out.println("\nDie SCM-Datei:");
```

```
307             s2x_ausgabe.ausgabeListe(SCM);
308         }
309
310         //System.out.println(".vor.");
311
312         XML = s2x_verarbeiten.vorbereitungScm(SCM);
313         if (v == true) {
314             System.out.println("\nVorbereitung:");
315             s2x_ausgabe.ausgabeListe(XML);
316         }
317
318         //System.out.println("..nach..");
319
320         XML = s2x_verarbeiten.nachbereitungScm(XML);
321         if (n == true) {
322             System.out.println("\nNachbereitung:");
323             s2x_ausgabe.ausgabeListe(XML);
324         }
325
326         //System.out.println("...fertig...");
327
328         if (satz == false){
329             System.out.println("\nQuell-DTD:  bonn.dtd:");
330             XML = s2x_bonn.erstelleXML(XML, name);
331             //System.out.println("\nGrï¿½ï¿½e der Datei: " + XML.size());
332             if (XML.size() > 2){
333                 if (x == true){
334
335                     s2x_ausgabe.ausgabeListe(XML);
336                     fehler = false;
337                 }
338             }
339             else {
340                     System.out.println("Die XML-Datei konnte nicht erstellt werden!");
341                     fehler = true;
342             }
343
344         }
345
346         else {
347             System.out.println("\nQuell-DTD: satz.dtd");
348             XML = s2x_keywords.erstelleXML(XML,name,optvor);
349
350             int gr = 0;
351             if (XML != null){
```

```
352                 gr = XML.size();
353             }
354
355             //System.out.println("\nGroesse der Liste: " + gr);
356             if (gr > 2){
357                 if (x == true){
358
359                     System.out.println("\nListe:\n"); s2x_ausgabe.ausgabeListe(XML);
360                     fehler = false;
361                 }
362
363             }
364             else {
365                     System.out.println("Die XML-Datei konnte nicht erstellt werden!");
366                     fehler = true;
367             }
368
369         }
370
371         if (fehler == false){
372             //einstellungen  = "<?xml version='1.0' encoding='ISO-8859-1' standalone='
                yes'?>";
373             einstellungen  = "<?xml version='1.0' encoding='ISO-8859-1' standalone='no
                '?>";
374             if (satz == true){
375                 einstellungen  = einstellungen +"\n<!DOCTYPE SATZ SYSTEM \"../dtd/
                    satzx.dtd\">";//!!TBC!!
376                 //einstellungen  = einstellungen +"\n<!DOCTYPE Beweis SYSTEM \"../dtd/
                    Satz.dtd\">";
377             }
378
379             else{
380                 einstellungen  = einstellungen +"\n<!DOCTYPE SATZ SYSTEM \"../dtd/Bonn
                    .dtd\">";
381             }
382
383
384             XML.addFirst(einstellungen);
385             s2x_schreibeDatei.datenSchreiben("../XML/" + name + ".XML", XML);
386
387             // Parsen
388             if (p == true){
389
390                 try {
391                     System.out.println("\nParser:");
```

```
392                     // Neuen SAX-Parser erzeugen
393                     SAXParserFactory factory   = SAXParserFactory.newInstance();
394
395                     factory.setValidating(true); // es soll validiert werden!
396                     SAXParser        saxParser = factory.newSAXParser();
397                     //DefaultHandler ehandler = new s2x_errorHandler();
398                     s2x_errorHandler ehandler = new s2x_errorHandler();
399
400                     // XML Datei parsen, die entsprechenden Methoden des
                            DefaultHandler
401                     // werden Ã¼ber den Errorhandler aufgerufen.
402                     saxParser.parse("../XML/" + name + ".XML",(DefaultHandler)ehandler
                            );
403
404                     boolean w1=saxParser.isValidating();
405                     if(w1) System.out.println("---> Der Parser validiert.");
406                     else System.out.println("...");
407
408
409                     System.out.println("Liste der fehlenden Tags:");
410                     LinkedList Tags = ehandler.getFehlendeTags();
411
412                     if (Tags.size() > 0){
413                         // einlesen der DTD
414                         LinkedList DTD = new LinkedList();
415                         if (satz == true){
416                             //System.out.println("-------SATZ");
417                             DTD = s2x_einlesen.leseDatei("../dtd/satzx.dtd");
418                         }
419
420                         else {
421                             System.out.println("-------Bonn");
422                             DTD = s2x_einlesen.leseDatei("../dtd/Bonn.dtd");
423                         }
424                         //System.out.println("\nDTD:");s2x_ausgabe.ausgabeListe(DTD);
425
426                         //System.out.println("Vorher:"); s2x_ausgabe.ausgabeListe(Tags
                                );
427
428                         Tags = komprimieren(Tags);
429                         System.out.println("\nkomprimiert:");s2x_ausgabe.ausgabeListe(
                                Tags);
430
431                         // bastele String
432                         String aaa = LL2TagString(Tags);
```

```
433                         System.out.println("\nals String: " + aaa);
434
435                         // DTD um Tags ergaenzen
436                         LinkedList DTDtmp = new LinkedList();
437                         DTD.add("<!-- NEU: " + name + ", erstellt am: " + time + " -->
                                ");
438                         for (int k = 0; k < Tags.size(); k++){
439                             String stmp = Tags.get(k).toString();
440                             stmp = "<!ELEMENT " + stmp + " (#PCDATA)>";
441
442                             DTD.add(stmp);
443                         }
444
445
446                         // finde Maths und ersetze sie
447                         DTD = ersetzeMath(DTD, aaa);
448                         //System.out.println("\ntmp-DTD:"); s2x_ausgabe.ausgabeListe(
                                DTD);
449                         if (satz == true){
450                             s2x_schreibeDatei.datenSchreiben("../DTD/satzX.dtd", DTD);
                                    // SatzX
451                         }
452                         else {
453                             s2x_schreibeDatei.datenSchreiben("../DTD/Bonn.dtd", DTD);
                                   // Bonn
454                         }
455                         //System.out.println("Zeit: " + time);
456                     }
457                     else System.out.println("keine fehlenden Tags gefunden!");
458
459
460                 }
461                 catch (SAXParseException ep)
462                 {// A parsing error occurred; the xml input is not valid
463                  System.out.println("SAX-Parser-Ausnahme in "+name+" :\n"+ep);
464                  System.out.println("Parser meldet FEHLER : "+ep.toString());
465                  System.out.println("an der Entity        : "+ep.getPublicId());
466                  System.out.println("Zeile,Spalte         : "+ep.getLineNumber()+","+
                         ep.getColumnNumber());
467                 }
468                 catch (SAXException e)
469                 {// A parsing error occurred; the xml input is not valid
470                  System.out.println("Da ist eine XML-Invaliditaet in "+name+" :\n"+e);
471                 }
472                 catch (ParserConfigurationException e)
```

```
473                 {System.out.println("Ein Parser-Konfigurationsproblem.");
474                 }
475                 catch (IOException e)
476                 {System.out.println("XML-File = "+name+" konnte nicht geoeffnet werden
                        ");
477                 }
478             }// parsen
479         }// fehler
480     }// methode
481 
482     /**
483      * Es wird ueberprueft, ob eine Datei "fname" exisitert
484      * @param fname: Pfad + Dateiname
485      * @return: true ist da / false nicht da
486      */
487     public static boolean Dateida(String fname){
488         boolean da = (new File(fname)).exists();
489 
490         return da;
491 
492     }
493 
494     public static LinkedList komprimieren(LinkedList x){
495         LinkedList erg = new LinkedList();
496         int size = x.size();
497         boolean check = false;
498         for (int i = 0; i < size; i++){
499             check = pruefeTag(erg,x.get(i).toString());
500             if (check == false){
501                 erg.add(x.get(i));
502             }
503 
504         }
505         return erg;
506     }
507 
508     public static boolean pruefeTag(LinkedList x, String s){
509         boolean da = false;
510         if (x != null){
511             for (int i = 0; i < x.size(); i++){
512                 if (x.get(i).toString().compareTo(s) == 0){
513                     da = true;
514                     break;
515                 }
516             }
```

```
517         }
518         return da;
519     }
520 
521     /**
522      * LinkedList in String umwandeln
523      * @param x
524      * @return
525      */
526     public static String LL2TagString(LinkedList x){
527         String erg = new String();
528         if (x != null){
529             erg = x.get(0).toString();
530             for (int i = 1; i < x.size(); i++){
531                 erg = erg + "|" + x.get(i);
532             }
533         }
534         return erg;
535     }
536 
537     public static LinkedList ersetzeMath(LinkedList x, String dazu){
538         LinkedList erg = new LinkedList();
539         /*
540         String math = "<!ELEMENT MATH (";
541         String mathmodus = "<!ELEMENT MATHmodus (";
542         String mathreihe = "<!ELEMENT MATHreihe (";
543         String mathreihen = "<!ELEMENT MATHreihen (";
544         String ftext = "<!ELEMENT FTEXT (";
545         //*/
546 
547         String ftext = "<!ENTITY % EinzelneTags ";
548         String tmp = new String();
549         String ablage = new String();
550         String zeile = new String();
551         int size = x.size();
552 
553         System.out.println("size: " + size);
554 
555         for (int i = 0; i < size; i++){
556             tmp = x.get(i).toString();
557 
558             /*
559             //System.out.println("TMP: " + tmp);
560             if (tmp.indexOf(math) == 0){
561                 ablage = tmp.substring(tmp.indexOf("(")+1,tmp.indexOf(")"));
```

```
562                 zeile = math + ablage.trim() + "|"+ dazu.trim() + ")*>";
563                 //System.out.println("Zeile: " + zeile);
564                 x.remove(i);
565                 x.add(i, zeile);
566             }
567             if (tmp.indexOf(mathmodus) == 0){
568                 ablage = tmp.substring(tmp.indexOf("(")+1,tmp.indexOf(")"));
569                 zeile = mathmodus + ablage.trim() + "|"+ dazu.trim() + ")*>";
570                 //System.out.println("Zeile: " + zeile);
571                 x.remove(i);
572                 x.add(i, zeile);
573             }
574             if (tmp.indexOf(mathreihe) == 0){
575                 ablage = tmp.substring(tmp.indexOf("(")+1,tmp.indexOf(")"));
576                 zeile = mathreihe + ablage.trim() + "|"+ dazu.trim() + ")*>";
577                 //System.out.println("Zeile: " + zeile);
578                 x.remove(i);
579                 x.add(i, zeile);
580             }
581             if (tmp.indexOf(mathreihen) == 0){
582                 ablage = tmp.substring(tmp.indexOf("(")+1,tmp.indexOf(")"));
583                 zeile = mathreihen + ablage.trim() + "|"+ dazu.trim() + ")*>";
584                 //System.out.println("Zeile: " + zeile);
585                 x.remove(i);
586                 x.add(i, zeile);
587             }
588             //*/
589             if (tmp.indexOf(ftext) == 0){
590                 ablage = tmp.substring(tmp.indexOf("\""),tmp.indexOf("\">"));
591                 zeile = ftext + ablage.trim() + "|"+ dazu.trim() + "\">";
592                 //System.out.println("Zeile: " + zeile);
593                 x.remove(i);
594                 x.add(i, zeile);
595             }
596 
597         }
598         return x;
599     }
600 
601     public static String Zeitstempel(String name){
602         String time = "scm/" + name + ".scm";
603         File f1 = new File(time);
604 
605         Calendar cal = Calendar.getInstance();
606         cal.setTimeInMillis( f1.lastModified() );
```

```
607
608         //System.out.println( cal.get(Calendar.DAY_OF_MONTH) + "." + (cal.get(Calendar
                .MONTH)+1) + "." + cal.get(Calendar.YEAR));
609         //System.out.println( cal.get(Calendar.HOUR) + ":" + cal.get(Calendar.MINUTE)
                + ":" + cal.get(Calendar.SECOND));
610
611         time = cal.get(Calendar.DAY_OF_MONTH) + "." + (cal.get(Calendar.MONTH)+1) + ".
                " + cal.get(Calendar.YEAR);
612         time = time + "\t" + cal.get(Calendar.HOUR) + ":" + cal.get(Calendar.MINUTE) +
                 ":" + cal.get(Calendar.SECOND);
613
614         return time;
615
616     }
617
618 }
```

10.9.7 s2x_Parser

```
1
2  import java.io.*;
3  import org.xml.sax.*;
4  import org.xml.sax.helpers.*;
5  import javax.xml.parsers.*;
6
7  /**
8   * Quelle: http://blog.planetxml.de/archives/2-XML-in-Java-mit-dem-SAX-Parser.html
9   * Minimales Beispiel fuer die Verwendung des SAX-Parsers in Java. Die XML Datei wird
10  * eingelesen und ordentlich formatiert auf der Konsole ausgegeben.
11  * DefaultHandler ist eine abstrakte Klasse, die die Interfaces ContentHandler,
12  * DTDHandler, EntityResolver und ErrorHandler implementiert.
13  */
14 public class s2x_Parser extends DefaultHandler
15 {
16     /**
17      * Verschachtelungstiefe der Tags, wird verwendet, um das XML-Dokument
18      * formatiert auszugeben.
19      */
20     private int level = 0;
21
22     /**
23      * Leerer Konstruktor, die Initialisierung des Parsers erfolgt in der
24      * main-Methode.
```

```
25       */
26      public s2x_Parser()
27      {
28      }
29
30      /**
31       * Gibt <code>level</code> Tabs auf der Konsole aus.
32       */
33      public void indent()
34      {
35          // Mit Tabs einruecken
36          for (int i=0;i<level;i++)
37              System.out.print("\t");
38      }
39
40      /**
41       * Wird am Anfang des Dokuments aufgerufen, definiert im Interface ContentHandler.
42       */
43      public void startDocument() throws SAXException
44      {
45          //System.out.println("Start des Dokuments");
46      }
47
48      /**
49       * Wird bei jedem oeffnenden Tag aufgerufen, definiert im Interface ContentHandler
             .
50       * Bei leeren Tags wie zum Beispiel &lt;img /&gt; wird startElement und
51       * endElement direkt hintereinander aufgerufen. Mit J2SE 1.4.2 scheint nur
52       * qName gefuellt zu sein.
53       *
54       * @param namespaceURI URI des Namespaces fuer dieses Element, kann auch ein
             leerer String sein.
55       * @param localName Lokaler Name des Elements, kann auch ein leerer String sein.
56       * @param qName Qualifizierter Name (mit Namespace-Prefix) des Elements.
57       * @param atts Liste der Attribute.
58       */
59      public void startElement(String namespaceURI, String localName,
60              String qName, Attributes atts) throws SAXException
61      {
62          indent();
63
64          System.out.print("<" + qName);
65
66          // Test-Code um zu sehen, was in namespaceURI und localName steht
67          // System.out.print(" " + namespaceURI);
```

```
68          // System.out.print(" " + localName);
69
70          // Attribute ausgeben
71          for( int i = 0; i < atts.getLength(); i++ )
72              System.out.print(" " + atts.getQName(i) + "=\"" + atts.getValue(i) + "\"")
                    ;
73
74          System.out.println(">");
75
76          level++;
77      }
78
79      /**
80       * Wird bei jedem schliessenden Tag aufgerufen, definiert im Interface
             ContentHandler.
81       *
82       * @param namespaceURI URI des Namespaces fï¿½r dieses Element, kann auch ein
             leerer String sein.
83       * @param localName Lokaler Name des Elements.
84       * @param qName Qualifizierter Name des Elements.
85       */
86      public void endElement(String namespaceURI, String localName, String qName)
87      {
88          level--;
89
90          indent();
91
92          System.out.println("</" + qName + ">");
93      }
94
95      /**
96       * Wird immer aufgerufen, wenn Zeichen im Dokument auftauchen.
97       *
98       * @param ch Character Array
99       * @param start Startindex der Zeichen in ch
100      * @param length Laenge der Zeichenkette
101      */
102     public void characters(char ch[], int start, int length)
103     {
104         String s = new String(ch,start,length).trim();
105         if (s.length() > 0) {
106             indent();
107             System.out.println(s);
108         }
109     }
```

```
110
111     /**
112      * Wird aufgerufen, wenn Leerraum (" ", "\t", "\n", "\r") im Dokument
113      * auftaucht, der fuer die Struktur des Dokuments nicht von Bedeutung ist.
114      *
115      * @param ch Character Array
116      * @param start Startindex der Zeichen in ch
117      * @param length Laenge der Zeichenkette
118      */
119     public void ignorableWhitespace(char[] ch, int start, int length)
120     {
121     }
122
123 }
```

10.9.8 s2x_schreibeDatei

```
1
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.util.LinkedList;
6
7  /**
8   *  Die Klasse enthaelt alle schreibenden Methoden des Projektes
9   * @author Zittermann
10   */
11 public class s2x_schreibeDatei {
12
13     /**
14      * Die Daten der LinkedList werden in eine Datei mit dem namen "fname" geschrieben
15      * @param fname: Name der zu schreibenden Datei
16      * @param list: zu schreibender Inhalt
17      * @throws Exception
18      */
19     public static void datenSchreiben (String fname, LinkedList list) throws Exception
20     {
21         FileWriter fw = new FileWriter(fname);
22         BufferedWriter pw = new BufferedWriter(fw);
23
24         for (int v=0; v<list.size();v++)
25         {
26             pw.write((list.get(v)).toString());
```

```
27              pw.newLine();
28          }
29          pw.close();
30      }
31
32  }
```

10.9.9 s2x_suchen

```
1
2
3   import java.util.LinkedList;
4
5   public class s2x_suchen {
6
7       /**
8        * suche die tiefste Ebene, gib die Position in dem array zurueck
9        * @param klammern: zu untersuchende Klammerstruktur
10       * @return: maximale Ebene
11       */
12      public static int sucheMaxEbene(int [][] klammern){
13          int pos = 0;
14          int i = 0;
15              while (klammern[i][2] > 0){
16                  //System.out.println(i + ".Klammer / Ebene: " + klammern[i][2]);
17                  System.out.println(i + " =>  " + klammern[i][2]);
18                  if (klammern[i][2] > klammern[pos][2]){
19                      pos = i;
20                      System.out.println("wechsel zu "+ pos);
21                  }
22                  i++;
23              }
24          return pos;
25      }
26
27
28      /**
29       *  Soll die positionen aller runden Klammern mitzaehlen
30       *  und diese mit ihrer ebene speichern
31       * @param eingabe: der komplette String
32       * @param anfang: Startstring
33       * @param ende: Endstring
34       * @return: positionen der runden klammerpaare mit ihren Ebenen
```

```
35      */
36     public static int[][] findeStrukturen(String eingabe, String anfang, String ende){
37         int anz = 0;
38         int fehler = 0;
39         int auf[], zu[],ebene[][];
40         int level=0;
41         int x = 0, y = 0;
42         // erstes array fuer Anzahl der Klammer, zweites fuer Anfang, Ende und Ebene
43         int pos[][] = new int[1000][3];
44 
45         eingabe = eingabe + "()";
46         auf = zaehleElemente(eingabe, anfang);
47         zu = zaehleElemente(eingabe, ende);
48 
49         // Testausgaben +
50         for (int i = 0; i < 1000; i++){
51             if ((auf[i] == -1 && zu[i] > -1) ||(auf[i] > -1 && zu[i] == -1)) {
52                 System.out.println("Ungleiche Anzahl von oeffnenden und schliessenden
                        Symbolen " +anfang+ " " +ende+ ":\n" + eingabe);
53                 fehler = 1;
54                 break;
55             }
56             else if ((auf[i] == -1 && zu[i] == -1)) {
57                 anz = i;
58                 break;
59             }
60 
61         }
62         // Zaehle Ebenen:
63         ebene = new int[eingabe.length()][2];
64         level = 0;
65         int h = 0;
66         for (int i = 0; i < eingabe.length(); i++){
67             if (eingabe.charAt(i) == '('){
68                 level++;
69                 ebene[h][1] = level;
70                 ebene[h][0] = i;
71                 h++;
72             }
73             else if (eingabe.charAt(i) == ')'){
74                 level--;
75                 ebene[h][1] = level;
76                 ebene[h][0] = i;
77                 h++;
78             }
```

```
79          }
80
81          if (fehler == 1) {
82              System.out.println("Es ist ein Fehler aufgetreten");
83              return null;
84          }
85
86          // Voraussetzungen, um eine Struktur zu finden, sind erfuellt.
87          // Untersuche entsprechend!
88
89          else {
90              for (int i = 0; i < anz; i++){
91                  for (int j = 0; j < anz; j++){
92
93                      // Wenn zu zwischen auf und auf+1 liegt
94
95                      if ((auf[j] < zu[i]) && (zu[i] < auf[j+1])){
96                          pos[i][0] = auf[j];
97                          pos[i][1] = zu[i];
98
99                          // grundsaetzliche Zuordnung der Ebenen:
100                         do {
101                             x = ebene[y][0]-1;
102                             y++;
103                         }while (auf[j] > x);
104                         pos[i][2] = ebene[y-2][1]-1;
105                         zu[i] = -1;
106                         // loescht das schon gefundene Element aus "auf"
107                         auf = (s2x_verarbeiten.update(auf,j,anz));
108                     }
109                 }
110             }
111
112         }// ende else
113
114         return pos;
115     }
116
117     /**
118      * Zaehlt die Elemente in einem String und gibt die positionen zurueck
119      * @param eingabe: der komplette String
120      * @param suchen: der zu suchende String
121      * @return positionen des Strings
122      */
123     public static int[] zaehleElemente(String eingabe, String suchen){
```

```
124         String tmp = eingabe;
125         int erg[] = new int [2000];
126         int i=0;
127         for (i = 0; i < 1000; i ++){
128             erg[i]=-1;
129         }
130         int pos;
131         int max;
132         int z = 0;
133         i = 0;
134 
135         // falls es oefter als einmal vorkommt
136         do {
137             pos = -1;
138             pos = tmp.indexOf(suchen);
139             max = tmp.length();
140 
141             if (pos > -1){
142                 tmp = tmp.substring(pos + suchen.length(), max);
143                 z  = z + pos + suchen.length();
144                 erg[i]= z;
145                 i++;
146             }
147 
148             else {
149                 //System.out.println("Nichts zu ersetzen");
150                 return erg;
151             }
152 
153 
154         } while(pos > -1);
155 
156         return erg;
157 
158     }
159 
160     /**
161      *  Suche mit Hilfe von Schluesselwoertern Start und Ende in einem String
162      * @param inhalt: der zu untersuchende String
163      * @param start: startschluesselwort
164      * @param ende: Beendendes Schluesselwort
165      * @param abschneiden: ob die schluesselwoerter mit drinsein sollen, oder nicht.
166      * @return: der String dazwischen
167      */
```

```
168     public static String SucheString(String inhalt, String start, String ende, boolean
            abschneiden){
169         String erg = new String();
170         String tmp = new String();
171         int i = 0,pos,max;
172         erg = "";
173 
174         tmp = inhalt;
175         max = tmp.length();
176         pos = tmp.indexOf(start);
177         i++;
178 
179         // Anfang wurde gefunden
180         if (pos > 0){
181             if (abschneiden == false)
182                 erg = tmp.substring(pos,max);
183             else
184                 erg = tmp.substring(pos+start.length(),max);
185 
186             // Wenn alles in einer Zeile steht
187             if (erg.indexOf(ende) > 0){
188                 if (abschneiden == true){
189                     erg = erg.substring(0,erg.length()-ende.length());// - 2 wegen 0
                            und position???
190                 }
191             }
192             // wenn das Ende nicht in der Zeile steht
193             else {
194                 System.out.println("Das Endkriterum konnte nicht zugeordnet werden. ")
                        ;
195                 erg = null;
196             }
197 
198             return erg;
199 
200         }// Ende von Postion gefunden
201         else
202             return null;
203 
204 
205     }
206 
207     /**
208      * Suche mit Hilfe von Schluesselwoertern Start und Ende Inhalte einer Liste
209      * @param list: die zu untersuchende Loste
```

```
210      * @param start: startschluesselwort
211      * @param ende: Beendendes Schluesselwort
212      * @param abschneiden: ob die schluesselwoerter mit drinsein sollen, oder nicht.
213      * @return: der String dazwischen
214      */
215     public static String sucheStruktur(LinkedList list, String start, String ende,
             boolean abschneiden){
216         String erg = new String();
217         String tmp = new String();
218         boolean aktiv = false;
219         int i = 0,pos,max;
220         erg = "";
221         do {
222 
223             tmp = (list.get(i).toString());
224             max = tmp.length();
225             pos = tmp.indexOf(start);
226             i++;
227 
228             // Anfang wurde gefunden
229             if (pos > 0){
230                 if (abschneiden == false)
231                     erg = tmp.substring(pos,max);
232                 else
233                     erg = tmp.substring(pos+start.length(),max);
234                 aktiv = true;
235 
236                 // Wenn alles in einer Zeile steht
237                 if (erg.indexOf(ende) > 0){
238                     if (abschneiden == true){
239                         // - 2 wegen 0 und position
240                         erg = erg.substring(0,erg.length()-ende.length()-2);
241                     }
242                 }
243 
244             }
245             // Ende wurde gefunden
246             else if (tmp.indexOf(ende) > 0){
247                 // Wenn es abgeschnitten werden soll, schneide das Ende raus
248 
249                 if (abschneiden == true){
250                     // - 2 wegen 0 und position
251                     erg = tmp.substring(0,erg.length()-ende.length()-2);
252                 }
253                 // Wenn nicht gib alles zurueck
```

```
254                 aktiv = false;
255                 return erg;
256             }
257 
258             // Falls es nicht vorkommt
259             else if((pos < 0) && (aktiv == false)){
260                 erg="";
261             }
262 
263             // Mittelteil
264             else{
265                 aktiv = true;
266                 erg = erg + tmp;
267             }
268 
269         }while (list.size() != i);
270 
271         return erg;
272 
273     }
274 
275     /**
276      * Ueberprueft, ob schluesselwoerter vorkommen
277      * => 0 entspricht inhaltlicher Klammer
278      * => 1 entspricht Texmacs-Klammer
279      * => 5 wird spï¿½ter konkret zugeordnet
280      * @param klammern: Klammerstruktur der Zeile
281      * @param zeile: zu ueberpruefender Inhalt
282      * @param pos: ab wo es losgehen soll
283      * @return:
284      */
285     public static int pruefeWoerter(int [][] klammern, String zeile, int pos){
286 
287         int erg = 0;
288         int ebene = 0;
289         int x = pos;
290         String tmp = new String();
291         String []liste  = new String[20];
292 
293         liste[0] = "document ";
294         liste[1] = "style ";
295         liste[2] = "body ";
296         liste[3] = "equation* ";
297         liste[4] = "concat ";
298         liste[5] = "math ";
```

```
299         liste[6] = "initial ";
300         liste[7] = "collection ";
301         liste[8] = "eqnarray* ";
302         liste[9] = "frac ";
303         liste[10] = "table ";
304         liste[11] = "row ";
305         liste[12] = "cell ";
306         liste[13] = "rsub ";
307         liste[14] = "rsup ";
308         liste[15] = "tformat ";
309 
310         ebene = klammern[pos][2];
311 
312         // ermittlung der aufgehenden Klammer vor der Klammer
313         while (klammern[x][2] == ebene){
314             x++;
315         }
316         if ((zeile.charAt(klammern[pos][0]-2)) == ' ') {
317 
318             if (   (zeile.charAt(klammern[pos][0]-3) == '\"')
319                 || (zeile.charAt(klammern[pos][0]-3) == ')')
320                 || (zeile.charAt(klammern[pos][0]-3) == ' ')
321                 || (zeile.charAt(klammern[pos][0]-3) == '\n')
322                 || (zeile.charAt(klammern[pos][0]-3) == '>')){
323                 tmp = zeile.substring(0,klammern[pos][0]);
324                 return 1;
325             }
326 
327             else if (klammern[x][0] < klammern[pos][0]){
328                 tmp = zeile.substring(klammern[x][0],klammern[pos][0]);
329                 for (int i = 0; i < 16; i++){
330                     if (tmp.indexOf(liste[i])> -1){
331                         if ((tmp.length()- 1 - liste[i].length()) == 0){
332                             return 1;
333                         }
334                         else {
335                             erg = 0;
336                         }
337                     }
338                     // Klammer
339                     else if (zeile.charAt(klammern[pos][1]+1) == ')'){
340                         return 1;
341                     }
342 
343                     else {
```

```
344                         erg = 0;
345                     }
346                 }
347                 return erg;
348             }
349             else if (klammern[pos][2] < 2){
350                 return 1;
351             }
352             else if ((klammern[pos][1]-klammern[pos][0])< 2){
353                 return 0;
354             }
355 
356             else {
357                 return 5; // was anderes
358             }
359         }
360         else {
361             return 0;
362         }
363     }
364 
365     /**
366      * gibt die Klammern und deren Ebenen (inhaltliche im Dokument => -1, rest Texmacs
            ) zurueck
367      * @param klammern
368      * @param zeile
369      * @return
370      */
371     public static int[][] pruefeKlammerStruktur(int[][] klammern, String zeile){
372         int p = 0;
373         int x = 0;
374         while (klammern[p][2] > 0){
375             x = s2x_suchen.pruefeWoerter(klammern, zeile, p);
376             if (x == 0){
377                 klammern[p][2] = -1;
378             }
379             if (x == 5){
380                 //System.out.println("!!!unvorhergesehenes!!!");
381             }
382             p++;
383         }
384         return klammern;
385 
386     }
387 
```

```
388     /**
389      * gibt hoechsten Wert des Feldes todo zurueck
390      * @param todo: zu untersuchendes Feld
391      * @return: hoechster Wert
392      */
393     public static int sucheMaxTodo(int todo[]){
394         int erg = 0;
395         int i = 0;
396         while (todo[i] != 0){
397             if (todo[i] > todo[erg]){
398                 erg = i;
399             }
400             i++;
401         }
402         return erg;
403     }
404 
405     /**
406      * Sucht Anzahl der Todos, also anzahl der relevanten Daten
407      * @param todo: zu untersuchendes Feld
408      * @return: Anzahl der Daten
409      */
410     public static int sucheAnzTodo(int todo[]){
411         int i = 0;
412         while (todo[i] != 0){
413             i++;
414         }
415         return i;
416     }
417 
418 
419     /**
420      * gibt die Anzahl der relevanten Klammern zurÃ¼ck
421      * @param klammern: zu untersuchendes Feld
422      * @return: Anzahl der Klammern
423      */
424     public static int sucheAnzKlammern(int klammern[][]){
425         int i = 0;
426         while (klammern[i][0] != 0){
427             i++;
428         }
429         return i;
430     }
431 
432     /**
```

```
433      * Zaehlt, wir oft ein String in einem anderen vorkommt
434      * @param zeile: zu analysierender String
435      * @param inhalt: zu zaehlendes Element
436      * @return: Anzahl der Elemente
437      */
438     public static int zaehleString(String zeile, String inhalt){
439         int anz = 0;
440         int pos = 0;
441         int ende = zeile.lastIndexOf(inhalt);
442         String tmp = zeile;
443 
444         do{
445             pos = tmp.indexOf(inhalt);
446             if (pos != -1) {
447                 anz++;
448                 tmp = tmp.substring(pos+1,tmp.length());
449                 ende = tmp.lastIndexOf(inhalt);
450             }
451             else {
452                 //System.out.println("\n\nHier ist eigentlich nix mehr drin: \n"+tmp);
453             }
454 
455         }while (ende > 0);
456         return anz;
457     }
458 
459 }
```

10.9.10 s2x_verarbeiten

```
1 
2 
3 import java.util.LinkedList;
4 
5 /**
6  * Stellt die Methode zur Verarbeitung der SCM-Struktur
7  * in ein XML-ï¿½hnlichem Format zur Verfï¿½gung
8  *
9  */
10 public class s2x_verarbeiten {
11 
12     /**
13      * aktualisierung des Arrays - kuerzen um die uebergebene anzahl
```

```
14      * @param auf: zu kuerzendes Array
15      * @param j: position, ab der gekuerzt wird
16      * @param anz: groeï¿½e des zu kuerzendes array
17      * @return: gekuerztes array
18      */
19     public static int[] update(int[] auf, int j, int anz){
20         int erg[] = new int [anz];
21         erg = auf;
22 
23         for (int i = j; i < anz-1; i++){
24             erg[i] = erg[i+1];
25         }
26 
27         return erg;
28     }
29 
30     /**
31      * aktualisiere die Klammern
32      * @param zeile: zeile mit der klammerstruktur
33      * @param anfang: startbereich
34      * @param ende: endbereich
35      * @return aktuelle klammerstruktur
36      */
37     public static int[][] aktualisiereKlammern(String zeile, String anfang, String
           ende){
38         int klammern[][];
39         klammern = s2x_suchen.findeStrukturen(zeile, anfang, ende);
40         klammern = s2x_suchen.pruefeKlammerStruktur(klammern, zeile);
41         //s2x_ausgabe.ausgabeKlammern(klammern);
42         klammern = s2x_verarbeiten.ordneKlammern(klammern);
43         return klammern;
44     }
45 
46     /**
47      * ersetzt einen String in einer zeile durch einen anderen
48      * @param zeile: die komplette Zeile
49      * @param zuersetzen: was ersetzt werden soll
50      * @param ersetzen: das was nachher da stehen soll
51      * @return: aktueller String
52      */
53     public static String ersetzen(String zeile, String zuersetzen, String ersetzen){
54         String tmp = new String();
55         String erg = new String();
56         int pos;
57         int max;
```

```
58          erg = zeile;
59
60          pos = erg.indexOf(zuersetzen);
61          max = erg.length();
62
63          if (pos > -1){
64              tmp = erg.substring(0,pos);
65              erg = erg.substring(pos + zuersetzen.length(), max);
66              erg =  tmp + ersetzen + erg;
67          }
68
69          else {
70              //System.out.println("Nichts zu ersetzen");
71              return null;
72          }
73          return erg;
74
75      }
76
77      /**
78       * ordnet die klammern aufsteigend nach ebenen (bubblesort)
79       * -> die hoechste Ebene an Pos 0
80       * @param klammern: die zu sortierenden Klammern
81       * @return: sortierte Klammern
82       */
83      public static int[][] ordneKlammern(int klammern[][]){
84          int n = s2x_suchen.sucheAnzKlammern(klammern);
85          int erg[][] = new int[n][2];
86          erg = klammern;
87
88          for (int i=0; i < n-1; i=i+1)
89                for (int j=n-1; j > i; j=j-1)
90                  if (erg[j-1][2] < erg[j][2])
91                   {
92                      int l = erg[j-1][0];
93                      int r = erg[j-1][1];
94                      int e = erg[j-1][2];
95                      erg[j-1][0] = erg[j][0];
96                      erg[j-1][1] = erg[j][1];
97                      erg[j-1][2] = erg[j][2];
98                      erg[j][0] = l;
99                      erg[j][1] = r;
100                     erg[j][2] = e;
101                  }
102         return erg;
```

```
103     }
104
105
106     /**
107      * untersucht die Schachtelung in einem Bereich
108      * @param suchen: was gesucht wird
109      * @param zeile: die komplette Zeile
110      * @param anfang: anfang des Bereichs
111      * @param ende: Ende des Bereichs
112      * @return: aktuelle Klammerstruktur
113      */
114     public static int[][] ersetzeStrukturen(String suchen, String zeile, String anfang
           , String ende){
115         int erg[][] = new int[1000][1000];
116         int klammern[][] = s2x_suchen.findeStrukturen(zeile, anfang, ende);
117         int posSuchen[] = s2x_suchen.zaehleElemente(zeile, suchen);
118         int k = 0;
119         int p = 0;
120         int q = 0;
121
122         // Ermittlung der Position der Klammer vor dem Suchstring
123         while (posSuchen[p] > -1){
124             posSuchen[p] = posSuchen[p] - suchen.length();
125             p++;
126         }
127         p = 0;
128         q = 0;
129
130         // Zuweisung der Positionen der Klammern
131         while(klammern[k][0] > 0){
132             while (posSuchen[p] > -1){
133                 if((posSuchen[p]) == klammern[k][0]){
134                     erg[q][0] = klammern[k][0]+suchen.length();
135                     erg[q][1] = klammern[k][1];
136                     q++;
137                 }
138                 p++;
139             }
140             p = 0;
141             k++;
142
143         }
144         return erg;
145     }
146
```

```
147     /**
148      *
149      * Vorbereitung der SCM-Liste
150      * Filtert die relevanten Inhalte der SCM-Liste und
151      * ersetzt die Elemente, die direkt erkannt und bearbeitet werden koennen:
152      * - Ableitung
153      * - Kursivschrift
154      * - numerische Aufzaehlung
155      * - "Grosse" oeffnende Zeichen
156      * - Klammern
157      * - Mathematische Zeilen
158      * - Brueche
159      * - Wurzeln
160      * - hoehergestellte und untergestellte Ausdruecke
161      * - Texmacs-Cellen
162      * @param SCM: Die SCM-Datei als Liste
163      * @return: Die vorbereitete SCM-Datei
164      * @throws Exception: kann beim tauschen auftreten
165      */
166     public static LinkedList vorbereitungScm(LinkedList SCM) throws Exception{
167         LinkedList XML = new LinkedList();
168         int klammern[][];
169 
170         int k = 0;
171         int p = 0;
172 
173         //int anzklammern = 0;
174         String zeile = new String();
175         String tmp = new String();
176         String suchen = new String();
177         String tauschen[][] = s2x_einlesen.tauschen();
178 
179         String anfang = "(";
180         String ende = ")";
181 
182         for (int i = 0; i < SCM.size(); i++){
183             zeile = SCM.get(i).toString();
184             klammern = aktualisiereKlammern(zeile, anfang, ende);
185 
186             // Nur die relevanten Daten im "body"
187             do {
188                 tmp = zeile.substring(klammern[k][0]-1,klammern[k][1]);
189                 k++;
190             }while(tmp.indexOf("(body") < 0);
191             k--;
```

```
192
193             //  +5 laenge vom bodystring | -1 anzahl der schliessenden klammern
194             tmp = zeile.substring(klammern[k][0]+4,klammern[k][1]-1);
195             zeile = tmp;
196
197             // ersetze alle ">" durch "\>"
198             while(zeile.indexOf(">") > -1){
199                 zeile = ersetzen(zeile, ">", "###ende###");
200             }
201             while(zeile.indexOf("###ende###") > -1){
202                 zeile = ersetzen(zeile, "###ende###", "/>");
203             }
204
205
206             // Ersetze die "Konstanten Werte"
207             for (i = 0; i < tauschen.length; i++){
208                 while(zeile.indexOf(tauschen[i][0]) > -1){
209                     zeile = ersetzen(zeile, tauschen[i][0], tauschen[i][1]);
210                 }
211             }
212
213             // aktualisiere die Klammerstruktur
214             klammern = aktualisiereKlammern(zeile, anfang, ende);
215             //System.out.println("Bis hier ok!");
216
217             // AB HIER: Bekannte Klammerstruktur!!!
218             k = 0;
219             p = 0;
220
221             // ersetze die schluesselwoerter der Vorbereitung
222             do {
223                 tmp = zeile.substring(klammern[p][0]-1,klammern[p][1]);
224
225                 // Ableitung
226                 if (tmp.indexOf("(hide-preamble ") > -1){
227                     suchen = s2x_wortmethoden.formate(zeile, klammern, p);
228
229                     if (suchen != null){
230                         zeile = suchen;
231                         klammern = aktualisiereKlammern(zeile, anfang, ende);
232                         //durch den Ersatz wurde die Zeile geaendert und muss wieder
                                ueberprueft werden!
233                         p = 0;
234                     }
235                 }// ende Ableitungs-IF
```

```
236
237                 // Ableitung
238                 if (tmp.indexOf("(rprime ") > -1){
239                     suchen = s2x_wortmethoden.rprime(zeile, klammern, p);
240
241                     if (suchen != null){
242                         zeile = suchen;
243                         klammern = aktualisiereKlammern(zeile, anfang, ende);
244                         //durch den Ersatz wurde die Zeile geaendert und muss wieder
                                ueberprueft werden!
245                         p = 0;
246                     }
247                 }// ende Ableitungs-IF
248
249                 // Kursive Schrift: em
250                 else if (tmp.indexOf("(em ") > -1){
251                     suchen = s2x_wortmethoden.em(zeile, klammern, p);
252
253                     if (suchen != null){
254                         zeile = suchen;
255                         klammern = aktualisiereKlammern(zeile, anfang, ende);
256                         p = 0;
257                     }
258
259                 }// ende EM-IF
260
261                 // Numerische Aufzaehlung: enumerate-numeric
262                 else if (tmp.indexOf("(enumerate-numeric ") > -1){
263                     suchen = s2x_wortmethoden.numerate(zeile, klammern, p);
264
265                     if (suchen != null){
266                         zeile = suchen;
267                         klammern = aktualisiereKlammern(zeile, anfang, ende);
268                         p = 0;
269                     }
270
271                 }// ende EM-IF
272
273                 // Grosse oeffende Zeichen (Summe, Integral, ...)
274
275                 else if (tmp.indexOf("(big") > -1){
276                     suchen = s2x_wortmethoden.big(zeile, klammern, p);
277
278                     if (suchen != null){
279                         zeile = suchen;
```

```
280
281                     klammern = aktualisiereKlammern(zeile, anfang, ende);
282                     //System.out.println("BIG OK");
283                     p = 0;
284                 }
285             }// ende BIG-IF
286
287             // Klammerungen
288             else if ((tmp.indexOf("(left") > -1) && (tmp.indexOf("(right") > -1)){
289                 suchen = s2x_wortmethoden.klammer(zeile, klammern, p);
290
291                 if (suchen != null){
292                     zeile = suchen;
293                     klammern = aktualisiereKlammern(zeile, anfang, ende);
294                     p = 0;
295                 }
296             }// ende Klammer-IF
297
298             // Mathereihen
299             else if ((tmp.indexOf("(equation*") > -1) || (tmp.indexOf("(eqnarray*"
                ) > -1)){
300                 suchen = s2x_wortmethoden.eq(zeile, klammern, p);
301
302                 if (suchen != null){
303                     zeile = suchen;
304                     klammern = aktualisiereKlammern(zeile, anfang, ende);
305                     p = 0;
306                 }
307             }// ende Reihen-IF
308
309             // Bruch
310             else if (tmp.indexOf("(frac") > -1){
311                 suchen =  s2x_wortmethoden.bruch(zeile, klammern, p);
312
313                 if (suchen != null){
314                     zeile = suchen;
315                     klammern = aktualisiereKlammern(zeile, anfang, ende);
316                     p = 0;
317                 }
318             }// ende BRUCH-IF
319
320             // Wurzel
321             else if (tmp.indexOf("(sqrt") > -1){
322                 suchen =  s2x_wortmethoden.wurzel(zeile, klammern, p);
323
```

```
324                     if (suchen != null){
325                         zeile = suchen;
326                         klammern = aktualisiereKlammern(zeile, anfang, ende);
327                         p = 0;
328                     }
329                 }// ende Wurzel-IF
330 
331 
332                 // rsu
333                 else if ((tmp.indexOf("(rsup") > -1) || (tmp.indexOf("(rsub") > -1)){
334                     String z = new String();
335                     //System.out.println("--->rsu" + tmp);
336                     z = tmp.substring(tmp.indexOf("rsu")+3, tmp.indexOf("rsu")+4);
337                     //suchen =  s2x_wortmethoden.rsualt(zeile, klammern, p, z);
338                     suchen =  s2x_wortmethoden.rsu(zeile, klammern, p, z);
339 
340                     if (suchen != null){
341                         zeile = suchen;
342                         klammern = aktualisiereKlammern(zeile, anfang, ende);
343                         p = 0;
344                     }
345                 }// ende Wurzel-IF
346 
347                 // einzelne Zellen
348                 else if ((tmp.indexOf("(cell ") > -1)){
349                     suchen = s2x_wortmethoden.codocell(zeile, klammern, p);
350 
351                     if (suchen != null){
352                         zeile = suchen;
353                         klammern = aktualisiereKlammern(zeile, anfang, ende);
354                         p = 0;
355                     }
356 
357                 }// ende CELL-IF
358 
359                 // negierte Elemente
360                 else if ((tmp.indexOf("(neg ") > -1)){
361                     suchen = s2x_wortmethoden.neg(zeile, klammern, p);
362 
363                     if (suchen != null){
364                         zeile = suchen;
365                         klammern = aktualisiereKlammern(zeile, anfang, ende);
366                         p = 0;
367                     }
368 
```

```
369                 }// ende CELL-IF
370
371                 // wenn kein Texmacs-Schluesselwort gefunden wurde
372                 else {
373                     p++;
374                 }
375
376                 //aktualisiere tmp
377                 tmp = zeile;
378
379             }while (klammern[p][2] != 0);
380
381             // ersetze die runden Klammern im Bereich Klammern
382             while(zeile.indexOf("###rundeklammern###") > -1){
383                 zeile = ersetzen(zeile, "###rundeklammern###", "(,)");
384             }
385
386             // uebertrage die zeile zur Liste
387             XML.add(zeile);
388         }
389
390         return XML;
391     }
392
393     /**
394      * Nachbereitung der SCM-Datei
395      * ersetzt die Elemente, die "vorbereitet" werden mussten
396      * oder in kombination mit anderen Elementen haetten auftreten koennen:
397      * - concat, row & document
398      * - with und math
399      * - Texte
400      * - runde Klammersymbole
401      * @param SCM: die vorbereitete SCM-Liste
402      * @return: die nachbereitete Liste
403      * @throws Exception: kann beim trennen auftreten
404      */
405     public static LinkedList nachbereitungScm(LinkedList SCM) throws Exception{
406         LinkedList XML = new LinkedList();
407         int klammern[][];
408         int p = 0;
409
410         String zeile = new String();
411         String tmp = new String();
412         String suchen = new String();
413
```

```
414         String anfang = "(";
415         String ende = ")";
416 
417         for (int i = 0; i < SCM.size(); i++){
418             zeile = SCM.get(i).toString();
419             klammern = aktualisiereKlammern(zeile, anfang, ende);
420             p = 0;
421 
422             do {
423 
424                 tmp = zeile.substring(klammern[p][0]-1,klammern[p][1]); //!!!!!
425 
426                 // concat, row & document
427                 if ((tmp.indexOf("(concat ") > -1)
428                         || (tmp.indexOf("(row ") > -1)
429                         || (tmp.indexOf("(document ") > -1)){
430                     suchen = s2x_wortmethoden.codocell(zeile, klammern, p);
431 
432                     if (suchen != null){
433                         zeile = suchen;
434                         klammern = aktualisiereKlammern(zeile, anfang, ende);
435 
436                         p = 0;
437                     }
438                     else p++;
439 
440                 }// ende codocell-IF
441 
442                 //  with und math...
443                 else if ((tmp.indexOf("\"math\"") > -1) || (tmp.indexOf("(math") > -1)
                        ){
444                     suchen = s2x_wortmethoden.mathMode(zeile, klammern, p);
445 
446                     if (suchen != null){
447                         zeile = suchen;
448                         klammern = aktualisiereKlammern(zeile, anfang, ende);
449                         p = 0;
450                     } // ende suchen-if
451                     else p++;
452 
453                 }// ende Klammer-IF
454 
455                 // wenn kein Texmacs-Schluesselwort gefunden wurde
456                 else{
457                     p++;
```

```
458                 } // ende ELSE
459                 tmp = zeile;
460 
461             }while (klammern[p][2] != 0);
462 
463             if (klammern[p][2] == 0){
464                 suchen = "(document";
465                 if (tmp.indexOf(suchen) > -1){
466                     tmp = zeile.substring(klammern[p][0]+suchen.length(),klammern[p
                        ][1]-1);
467                 }
468 
469                 zeile = tmp;
470             }
471 
472             // ersetze die "" durch <Text>
473 
474             int anz = s2x_suchen.zaehleString(zeile, "\"");
475             tmp = zeile;
476             for (int f = 0; f < anz/2+1; f++){
477                 tmp = s2x_wortmethoden.TextTag(tmp,0);
478                 if (tmp != null){
479                     zeile = tmp;
480                 }
481             }
482             //*/
483             // Ersetzt alle ###anfzeichen### durch " --> Tritt beim Klammertyp auf
484             while (zeile.indexOf("###anfzeichen###") > -1){
485                 zeile = ersetzen(zeile, "###anfzeichen###", "\"");
486             }
487 
488             // wandelt den "fertigen" String in eine Liste um
489             XML = stringToList(zeile);
490 
491         }
492 
493         return XML;
494     }
495 
496     /**
497      * erstellt aus einem durch \n getrennten String eine LinkedList
498      * @param zeile
499      * @return
500      */
501     public static LinkedList stringToList(String zeile){
```

```
502         LinkedList erg = new LinkedList();
503         String inhalt = new String();
504         String trennen = "\n";
505         int anz = 0;
506         int pos = 0;
507         int ende = zeile.lastIndexOf(trennen);
508         String tmp = zeile;
509 
510         do{
511             pos = tmp.indexOf(trennen);
512             if (pos != -1) {
513                 anz++;
514                 inhalt = tmp.substring(0,pos);
515                 inhalt = inhalt.trim();
516 
517                 // Ueberpruefung des Inhalts:
518                 if (!inhalt.isEmpty())
519                     erg.add(inhalt);
520 
521                 tmp = tmp.substring(pos+1,tmp.length());
522                 ende = tmp.lastIndexOf(trennen);
523             }
524             else {
525                 //System.out.println("\n\nHier ist eigentlich nix mehr drin: \n"+tmp);
526             }
527 
528         }while (ende > 0);
529 
530         return erg;
531     }
532 }
```

10.9.11 s2x_wortmethoden

```
1 
2 import java.util.LinkedList;
3 
4 /**
5  * diese Klasse enthaelt alle Methoden fuer die Schluesselwoerter in Texmacs
6  */
7 public class s2x_wortmethoden {
8 
9     /**
```

```
10      * ersetzt die Anfuehrungszeichen in einem String
11      * @param tmp: zu bearbeitender String
12      * @return: bearbeiteter String
13      */
14     public static String ersetzeAnfuehrungszeichen(String tmp){
15         String erg = tmp;
16 
17         // ersetze die "
18         while (erg.indexOf("\"") > -1){
19             erg = s2x_verarbeiten.ersetzen(erg, "\"", "");
20         }
21 
22         return erg;
23     }
24 
25     public static String formate(String zeile, int[][] klammern, int pos){
26         String erg = new String();
27         String inhalt = new String();
28         String ersatz = new String();
29         String em = "(hide-preamble ";
30         int pl = 0;
31 
32         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
33         pl = inhalt.indexOf(em);
34 
35         int pr = inhalt.length();
36 
37         if (pl == 0){
38             pr = pr - 1; // 1 schliessende klammern
39             //ersatz = inhalt.substring(pl + em.length(),pr);
40             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, " ");
41         }
42         else {
43             erg = null;
44         }
45         return erg;
46     }
47 
48     /**
49      * ersetzt die in dem String vorhandenen "(em"s einer konkreten position
50      * @param zeile: die zu bearbeitete Zeile
51      * @param klammern: die Klammerstruktur der Zeile
52      * @param pos: Position in der Klammer
53      * @return: bearbeiteter String
54      */
```

```
55      public static String em(String zeile, int[][] klammern, int pos){
56          String erg = new String();
57          String inhalt = new String();
58          String ersatz = new String();
59          String em = "(em ";
60          int pl = 0;
61
62          inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
63          pl = inhalt.indexOf(em);
64
65          int pr = inhalt.length();
66
67          if (pl == 0){
68              pr = pr - 1; // 1 schliessende klammern
69              ersatz = inhalt.substring(pl + em.length(),pr);
70              erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
71          }
72          else {
73              erg = null;
74          }
75          return erg;
76      }
77
78      /**
79       * ersetzt die in dem String vorhandene numerische Aufzaehlung an einer konkreten
             position
80       * @param zeile: die zu bearbeitete Zeile
81       * @param klammern: die Klammerstruktur der Zeile
82       * @param pos: Position in der Klammer
83       * @return: bearbeiteter String
84       */
85      public static String numerate(String zeile, int[][] klammern, int pos){
86          String erg = new String();
87          String inhalt = new String();
88          String ersatz = new String();
89          String tmp = new String();
90          String item = "(item)";
91          String num = "(enumerate-numeric ";
92          int index = 0;
93          int pl = 0;
94          int pli = 0;
95
96          inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
97          pl = inhalt.indexOf(num);
98
```

```
99          int pr = inhalt.length();
100
101         if (pl == 0){
102             pr = pr - 1; // 1 schliessende klammern
103             tmp = inhalt.substring(pl + num.length(),pr);
104
105             while (tmp.indexOf("(item)") > 0){
106                 index++;
107                 pli = tmp.indexOf("(item)");
108                 String ind = tmp.substring(pli,tmp.length());
109                 ersatz = ind.substring(ind.indexOf(item)+item.length());
110
111                 if (ersatz.indexOf("(item)") > 0){
112
113                     ersatz = ersatz.substring(0,ersatz.indexOf("(item)"));
114                     ersatz = "\n<Item ID=" + index + ">"+ ersatz + "</Item>";
115                 }
116                 else {
117                     ersatz = "\n<Item ID=" + index + ">"+ ersatz + "</Item>";
118                 }
119
120                 ind = ind.substring(0,ersatz.length()-13);  // 13 zeichen wurden
                        eingefuegt
121                 tmp = s2x_verarbeiten.ersetzen(tmp, ind, ersatz);
122             }
123
124             tmp = "\n<Auflistung> " + tmp + "\n</Auflistung>\n";
125             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, tmp);
126         }
127         else {
128             erg = null;
129         }
130         return erg;
131
132     }
133
134     /**
135      * Fuer alle mathematischen Zeilen in Texmacs
136      * @param zeile: die zu bearbeitete Zeile
137      * @param klammern: die Klammerstruktur der Zeile
138      * @param pos: Position in der Klammer
139      * @return: bearbeiteter String
140      */
141     public static String eq(String zeile, int[][] klammern, int pos){
142         String erg = new String();
```

```
143         String inhalt = new String();
144         String math = new String();
145         String equation = "(equation* (document ";
146         //String equation_c = "(equation* (document (concat "; // wird so nicht
                gebraucht-> codocell
147         String eqnarray = "(eqnarray* (document (tformat (table ";// (row?
148         String eqnarray_nd = "(eqnarray* (tformat (table "; // (row?
149         int pl = 0;
150
151         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
152         pl = inhalt.indexOf(equation);
153
154         int pr = inhalt.length();
155
156         if (pl == 0){
157             pr = pr - 2; // 3 schliessende klammern
158             math = inhalt.substring(pl + equation.length(),pr);
159             math = "\n<MATHreihe>\n " + math + "\n</MATHreihe>\n ";
160             // ersetze die "" durch <FText>
161             math = ftext(math);
162             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, math);
163         }
164         else {
165             pl = inhalt.indexOf(eqnarray);
166             if (pl == 0){
167
168                 pr = pr - 4; // 5 schliessende klammern mit row
169                 math = inhalt.substring(pl + eqnarray.length(),pr);
170
171                 // loesche leere zellen
172                 while (math.indexOf("(cell \"\")") > -1){
173                     math = s2x_verarbeiten.ersetzen(math, "(cell \"\")", "");
174                 }
175
176                 math = "\n<MATHreihen>\n " + math + "\n</MATHreihen>\n ";
177                 // ersetze die "" durch <FText>
178                 math = ftext(math);
179
180                 erg = s2x_verarbeiten.ersetzen(zeile, inhalt, math);
181             }
182             else {
183                 pl = inhalt.indexOf(eqnarray_nd);
184                 if (pl == 0){
185
186                     pr = pr - 3; // 4 schliessende klammern mit row
```

```
187                     math = inhalt.substring(pl + eqnarray.length(),pr);
188
189                     // loesche leere zellen
190                     while (math.indexOf("(cell \"\")") > -1){
191                         math = s2x_verarbeiten.ersetzen(math, "(cell \"\")", "");
192                     }
193                     // ersetze die "" durch <FText>
194                     math = ftext(math);
195                     math = "\n<MATHreihen>\n " + math + "\n</MATHreihen>\n ";
196
197                     erg = s2x_verarbeiten.ersetzen(zeile, inhalt, math);
198                 }
199                 else {
200                     erg = null;
201                 }
202             }
203         }
204         return erg;
205     }
206
207
208     /**
209      * Ersetzt die Texmacs-Klammerstrukturen
210      * @param zeile: die zu bearbeitete Zeile
211      * @param klammern: die Klammerstruktur der Zeile
212      * @param pos: Position in der Klammer
213      * @return: bearbeiteter String
214      */
215     public static String klammer(String zeile, int[][] klammern, int pos){
216         String klammer = new String();
217         String orignal = new String();
218         String ersatz = new String();
219         String up = new String();
220         String down = new String();
221         String tmp = new String();
222         String typ = new String();
223         String dahinter = new String();
224         String erg = new String();
225         int lp= 0, rp = 0;
226         int lk= 0, rk = 0;
227         int p = 0;
228
229         up = null;
230         down = null;
231
```

```
232         lp = klammern[pos][0];
233         rp = klammern[pos][1];
234         orignal = zeile.substring(lp-1,rp);
235
236         lk = orignal.indexOf("(left \"");
237         rk = orignal.indexOf("(right \"");
238
239         klammer = "(left \"(\")";
240         klammer = orignal.substring(lk + klammer.length(),rk);
241
242         typ = orignal.substring(lk,rk);
243         typ = typ.substring(typ.indexOf("\"")+1);
244         typ = typ.substring(0,typ.indexOf("\""));
245
246         tmp = orignal.substring(rk);
247         tmp = tmp.substring(tmp.indexOf("\"")+1);
248         tmp = tmp.substring(0,tmp.indexOf("\""));
249
250
251         // Ersetze temporaer runde Klammern in ruecktransformierbare Symbole
252         if (typ.indexOf("(") == 0  && tmp.indexOf(")") == 0){
253             typ = "###rundeklammern###";
254         }
255         else {
256             typ = typ +"," + tmp;
257         }
258
259         // ###anfzeichen### wird nach Texterkennung durch " ersetzt.
260         klammer = "\n<KLAMMERN typ=###anfzeichen###" + typ + "###anfzeichen###>\n " +
                klammer + "\n</KLAMMERN>\n ";
261
262         ersatz = orignal.substring(0,orignal.indexOf("left") -1);
263         ersatz = ersatz + klammer;
264
265         tmp = "(right \"(\")";
266         tmp = orignal.substring(rk+tmp.length(),orignal.length());
267         ersatz = ersatz + tmp;
268
269         klammer = s2x_verarbeiten.ersetzen(zeile, orignal, ersatz);
270         // Fertig ohne SUPB
271
272         tmp = klammer;
273
274         // ueberpruefung, ob an schliessenden symbolen exponenten oder indizes stehen
275         p = tmp.indexOf("</KLAMMERN>  (rsup"); //???
```

```
276         if (p > -1){
277             int i = 0;
278             klammern = s2x_suchen.findeStrukturen(klammer,"(", ")");
279             klammern = s2x_suchen.pruefeKlammerStruktur(klammern, klammer);
280             while (klammern[i][0] < p){
281                 i++;
282             }
283             tmp = klammer.substring(klammern[i][0]-1,klammern[i][1]);
284 
285             i = i - 1;
286             lp = 0;
287             rp = klammer.length();
288 
289             if  (tmp.indexOf("rsub") > -1){
290 
291                 down = klammer.substring(klammern[i+1][0] + 6 ,klammern[i+1][1]-2);
292                 erg = erg + "\n<KlammerInd>\n " + down + "\n</KlammerInd>\n ";
293                 rp = klammern[i+1][1];
294 
295                 tmp = zeile.substring(klammern[i+2][0],klammern[i+2][1]);
296                 if  (tmp.indexOf("rsup") > -1){
297                     up = klammer.substring(klammern[i][0] + 6 ,klammern[i+2][1]-2);
298                     erg = erg + "\n<KlammerExp>\n " + up + "\n</KlammerExp>\n ";
299                     rp = klammern[i+2][1];
300                 }
301             }
302             else if  (tmp.indexOf("rsup") > -1){
303                 up = klammer.substring(klammern[i+1][0] + 6 ,klammern[i+1][1]-2);
304                 erg = erg + "\n<KlammerExp>\n " + up + "\n</KlammerExp>\n ";
305                 rp = klammern[i+1][1];
306             }
307 
308             // erg in klammer einfuegen
309             dahinter = klammer.substring(0,klammern[i+1][0]-1);
310             dahinter = dahinter + erg;
311             dahinter = dahinter + klammer.substring(klammern[i+1][1]);
312             klammer = s2x_verarbeiten.ersetzen(klammer, klammer, dahinter);
313         }
314 
315         return klammer;
316     }
317 
318     /**
319      * Ersetzt den Bruch
320      * @param zeile: die zu bearbeitete Zeile
```

```
321      * @param klammern: die Klammerstruktur der Zeile
322      * @param pos: Position in der Klammer
323      * @return: bearbeiteter String
324      */
325     public static String bruch(String zeile, int[][] klammern, int pos){
326         String bruch = "(frac ";
327         String inhalt = new String();
328         String zaehler = new String();
329         String nenner = new String();
330         String erg = new String();
331         String con = "(concat ";
332         int p = 0;
333         int pzaehler = 0;
334 
335         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
336         //System.out.println("Inhalt:" + inhalt);
337 
338         int pl = inhalt.indexOf(bruch);
339         int pr = inhalt.length() -1;
340 
341         if (pl == 0){
342             // Zaehler bestimmen
343             zaehler = inhalt.substring(pl+bruch.length(),pr);
344 
345             // Wenn der aus "" besteht...
346             p = zaehler.indexOf("\"");
347 
348             if (p < 2 && p > -1){
349                 zaehler = zaehler.substring(p+1);
350                 p = zaehler.indexOf("\"");
351                 zaehler = zaehler.substring(0,p);
352                 pzaehler = (bruch.length() + zaehler.length()+3);  //laenge bruch/
                        Zaehler + 3 zeichen:>"" <
353             }
354             else {
355                 p = zaehler.indexOf(con);
356 
357                 // Wenn eine Klammerstruktur "(concat" vorkommt
358                 if (p < 2 && p > -1){
359                     String tmp;
360                     int x = -1;
361                     int anz = 0;
362                     int [][] innen = s2x_suchen.findeStrukturen(zaehler, "(", ")");
363                     innen = s2x_suchen.pruefeKlammerStruktur(innen, zaehler);
364                     innen = s2x_verarbeiten.ordneKlammern(innen);
```

```
365                     anz = s2x_suchen.sucheAnzKlammern(innen);
366                     //s2x_ausgabe.ausgabeKlammern(innen);
367 
368                     // finde die erste Ebene, die "0" ist
369                     for (int z = anz; z > -1; z--){
370                         if (innen[z][2] == 0){
371                             x = z;
372                         }
373                     }
374                     tmp = zaehler.substring(innen[x][0],innen[x][1]-1);
375                     //System.out.println(x + ". tmp:" + tmp);
376 
377                     zaehler = zaehler.substring(innen[x][0] + con.length()-1, innen[x
                            ][1] -1); // - 2 zufall?
378                     pzaehler = con.length() + innen[x][1]-1;
379 
380                     //System.out.println("Zaehler:" + zaehler);
381                 }
382                 else pzaehler = 0;
383             }// ELSE - Zaehler fertig!
384 
385             nenner = inhalt.substring(pzaehler,inhalt.length() - 1);
386             //System.out.println("Nenner:" + nenner);
387 
388             p = nenner.indexOf("\"");
389             if (p < 2 && p > -1){
390                 nenner = nenner.substring(p+1,nenner.length());
391                 p = nenner.indexOf("\"");
392                 nenner = nenner.substring(0,p);
393             }
394             else {
395                 p = nenner.indexOf("(");
396                 // Wenn eine Klammerstruktur vorkommt
397                 if (p < 2 && p > -1){
398                     String tmp;
399                     int x = 0;
400                     int anz = 0;
401                     int [][] innen = s2x_suchen.findeStrukturen(nenner, "(", ")");
402                     innen = s2x_suchen.pruefeKlammerStruktur(innen, nenner);
403                     anz = s2x_suchen.sucheAnzKlammern(innen);
404 
405                     p = p + 1;
406                     for (x = 0; x < anz; x++){
407                         if (innen[x][0] == p){
408                             break;
```

```
409                         }
410                     }
411                     tmp = codocell(nenner, innen, x);
412                     if (tmp != null){
413                         nenner = tmp.substring(0,tmp.length());
414                     }
415                     else {
416                         nenner = null;
417                     }
418                 }
419 
420             }// Ende ELSE
421 
422             // setze Teilergebnisse zusammen
423             erg = "\n<BRUCH>\n<Zaehler>\n "+zaehler+"\n</Zaehler>\n<Nenner>\n "+nenner
                    +"\n</Nenner>\n</BRUCH>\n";
424             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, erg);
425 
426         }
427         else erg = null;
428 
429         return erg;
430     }
431 
432     /**
433      * fuer concat, row, document und cell
434      * @param zeile: die zu bearbeitete Zeile
435      * @param klammern: die Klammerstruktur der Zeile
436      * @param pos: Position in der Klammer
437      * @return: bearbeiteter String
438      */
439     public static String codocell(String zeile, int[][] klammern, int pos){
440         String erg = new String();
441         String inhalt = new String();
442         String ersatz = new String();
443         String con = "(concat ";
444         String doc = "(document ";
445         String cel = "(cell ";
446         String row = "(row ";
447 
448         int pl = 0;
449 
450         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
451         pl = inhalt.indexOf(con);
452 
```

```
453         int pr = inhalt.length();
454 
455         if (pl == 0){
456             pr = pr - 1; // 1 schliessende klammern
457             ersatz = inhalt.substring(pl + con.length(),pr);
458             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
459         }
460         else {
461             pl = inhalt.indexOf(cel);
462             if (pl == 0){
463                 pr = pr - 1; // 1 schliessende klammern
464                 ersatz = inhalt.substring(pl + cel.length(),pr);
465 
466                 // loesche leere zellen
467                 while (ersatz.indexOf("(cell \"\")") > -1){
468                     ersatz = s2x_verarbeiten.ersetzen(ersatz, "(cell \"\")", "");
469                 }
470 
471                 ersatz = "\n<CELL>\n " + ersatz + "\n</CELL>\n ";
472                 erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
473 
474             }
475             else {
476                 pl = inhalt.indexOf(doc);
477                 if (pl == 0){
478                     pr = pr - 1; // 1 schliessende klammern
479                     ersatz = inhalt.substring(pl + doc.length(),pr);
480                     erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
481 
482                 }
483                 else {
484                     pl = inhalt.indexOf(row);
485                     if (pl == 0){
486                         pr = pr - 1; // 1 schliessende klammern
487                         ersatz = inhalt.substring(pl + row.length(),pr);
488                         ersatz = "\n<REIHE> " + ersatz + "</REIHE>\n ";
489                         erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
490                     }
491                     else erg = null;
492                 }
493             }
494         }
495         return erg;
496     }
497 
```

```
498     /**
499      * ersetze den mathematischen Bereich in einer Textzeile
500      * @param zeile: die zu bearbeitete Zeile
501      * @param klammern: die Klammerstruktur der Zeile
502      * @param pos: Position in der Klammer
503      * @return: bearbeiteter String
504      * @throws Exception: kann beim finden des Statementtyps auftreten
505      */
506     public static String mathMode(String zeile, int[][] klammern, int pos) throws
         Exception{
507         String erg = new String();
508         String inhalt = new String();
509         String ersatz = new String();
510         String typ = new String();
511         String math = "(math ";
512         String mathm = "(with \"mode\" \"math\" ";
513         int pl = 0;
514 
515         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
516         pl = inhalt.indexOf(math);
517 
518         int pr = inhalt.length();
519 
520         if (pl == 0){
521             pr = pr - 1; // 1 schliessende klammern
522             ersatz = inhalt.substring(pl + math.length(),pr);
523 
524             typ = s2x_einlesen.findeStatementtyp(ersatz);
525             ersatz = "\n<MATH>\n " + ersatz + "\n</MATH>\n";
526             if (!(typ == null)) {
527                 ersatz = ersatz  + typ;
528             }
529 
530             // ersetze die "" durch <FText>
531             ersatz = ftext(ersatz);
532 
533             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
534         }
535         else {
536             pl = inhalt.indexOf(mathm);
537 
538             if (pl == 0){
539                 pr = pr - 1; // 1 schliessende klammern
540                 ersatz = inhalt.substring(pl + mathm.length(),pr);
541 
```

```
542                 // ersetze die "" durch <FText>
543                 ersatz = ftext(ersatz);
544 
545                 typ = s2x_einlesen.findeStatementtyp(ersatz);
546                 ersatz = "\n<MATHmodus>\n " + ersatz + "\n</MATHmodus>\n";
547 
548                 if (!(typ == null)) {
549                     ersatz = ersatz  + typ;
550                 }
551                 else{
552                     //System.out.println("KEIN Statement!!!");
553                 }
554 
555                 erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
556             }
557             else erg = null;
558 
559         }
560         return erg;
561     }
562 
563     /**
564      * ersetze die Wurzeln
565      * @param zeile: die zu bearbeitete Zeile
566      * @param klammern: die Klammerstruktur der Zeile
567      * @param pos: Position in der Klammer
568      * @return: bearbeiteter String
569      */
570     public static String wurzel(String zeile, int[][] klammern, int pos){
571         String erg = new String();
572         String inhalt = new String();
573         String ersatz = new String();
574         String wurzel = "(sqrt ";
575 
576         int pl = 0;
577 
578         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
579         pl = inhalt.indexOf(wurzel);
580 
581         int pr = inhalt.length();
582 
583         if (pl == 0){
584             pr = pr - 1; // 1 schliessende klammern
585             ersatz = inhalt.substring(pl + wurzel.length(),pr);
586             int p = ersatz.indexOf("(");
```

```
587
588             // Wenn eine Klammerstruktur vorkommt
589             if (p < 2 && p > -1){
590                 String tmp;
591                 int x = 0;
592                 int [][] innen = s2x_suchen.findeStrukturen(ersatz, "(", ")");
593                 innen = s2x_suchen.pruefeKlammerStruktur(innen, ersatz);
594                 while (innen[x][0] != 0 ){
595                     x++;
596                 }
597                 tmp = codocell(ersatz, innen, x-1);
598                 if (tmp != null){
599                     ersatz = tmp;
600                 }
601                 else {
602                     ersatz = null;
603                 }
604             }
605             else {
606                 ersatz = inhalt.substring(pl + wurzel.length(),pr);
607             }
608             ersatz = "\n<WURZEL>\n " + ersatz + "\n</WURZEL>\n ";
609             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
610         }
611
612         else erg = null;
613         return erg;
614     }
615
616     /**
617      * ersetze die negationen
618      * @param zeile: die zu bearbeitete Zeile
619      * @param klammern: die Klammerstruktur der Zeile
620      * @param pos: Position in der Klammer
621      * @return: bearbeiteter String
622      */
623     public static String neg(String zeile, int[][] klammern, int pos){
624         String erg = new String();
625         String inhalt = new String();
626         String ersatz = new String();
627         String wurzel = "(neg ";
628
629         int pl = 0;
630
631         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
```

```
632         pl = inhalt.indexOf(wurzel);
633 
634         int pr = inhalt.length();
635 
636         if (pl == 0){
637             pr = pr - 1; // 1 schliessende klammern
638             ersatz = inhalt.substring(pl + wurzel.length(),pr);
639             int p = ersatz.indexOf("(");
640 
641             // Wenn eine Klammerstruktur vorkommt
642             if (p < 2 && p > -1){
643                 String tmp;
644                 int x = 0;
645                 int [][] innen = s2x_suchen.findeStrukturen(ersatz, "(", ")");
646                 innen = s2x_suchen.pruefeKlammerStruktur(innen, ersatz);
647                 while (innen[x][0] != 0 ){
648                     x++;
649                 }
650                 tmp = codocell(ersatz, innen, x-1);
651                 if (tmp != null){
652                     ersatz = tmp;
653                 }
654                 else {
655                     ersatz = null;
656                 }
657             }
658             else {
659                 ersatz = inhalt.substring(pl + wurzel.length(),pr);
660             }
661             ersatz = "\n<nicht>\n " + ersatz + "\n</nicht>\n ";
662             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
663         }
664 
665         else erg = null;
666         return erg;
667     }
668 
669     /**
670      * ersetze die Ableitungen
671      * @param zeile: die zu bearbeitete Zeile
672      * @param klammern: die Klammerstruktur der Zeile
673      * @param pos: Position in der Klammer
674      * @return: bearbeiteter String
675      */
676     public static String rprime(String zeile, int[][] klammern, int pos){
```

```
677         String erg = new String();
678         String inhalt = new String();
679         String ersatz = new String();
680         String rprime = "(rprime ";
681         int pl = 0;
682 
683         inhalt = zeile.substring(klammern[pos][0]-1,klammern[pos][1]);
684         pl = inhalt.indexOf(rprime);
685 
686         int pr = inhalt.length();
687 
688         if (pl == 0){
689             pr = pr - 1; // 1 schliessendes "
690             ersatz = inhalt.substring(pl + rprime.length()+1,pr-1); // ?
691             ersatz = "\n<Ableitung>\n " + ersatz.length() + "\n</Ableitung>\n "; //
                    ???
692             erg = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
693         }
694         else erg = null;
695 
696         return erg;
697     }
698 
699     /**
700      * wenn ein oeffnendes Symbol vorkommt (BIG)
701      * @param zeile: die zu bearbeitete Zeile
702      * @param klammern: die Klammerstruktur der Zeile
703      * @param pos: Position in der Klammer
704      * @return: bearbeiteter String
705      */
706     public static String big(String zeile, int[][] klammern, int pos){
707         String big = new String();
708         String name = new String();
709         String up = new String();
710         String down = new String();
711         String tmp = new String();
712         int lp= 0, rp = 0;
713 
714         up = null;
715         down = null;
716          // bis zum inhalt sinds 5 zeiche, es endet zwei davor
717         name = zeile.substring(klammern[pos][0] + 5 ,klammern[pos][1]-2);
718         tmp = zeile.substring(klammern[pos+1][0],klammern[pos+1][1]);
719         big = "\n<" + name + "/>\n ";
720         lp = klammern[pos][0];
```

```
721         rp = klammern[pos][1];
722 
723         if  (tmp.indexOf("rsub") > -1){
724             down = zeile.substring(klammern[pos+1][0] + 6 ,klammern[pos+1][1]-2);
725             big = big + "\n<lowlimit>\n " + down + "\n<lowlimit>\n ";
726             rp = klammern[pos+1][1];
727 
728             tmp = zeile.substring(klammern[pos+2][0],klammern[pos+2][1]);
729             if  (tmp.indexOf("rsup") > -1){
730                 up = zeile.substring(klammern[pos+2][0] + 6 ,klammern[pos+2][1]-2);
731                 big = big + "\n<uplimit>\n " + up + "\n</uplimit>\n ";
732                 rp = klammern[pos+2][1];
733             }
734         }
735         else if  (tmp.indexOf("rsup") > -1){
736             up = zeile.substring(klammern[pos+1][0] + 6 ,klammern[pos+1][1]-2);
737             big = big + "\n<uplimit>\n " + up + "\n</uplimit>\n ";
738             rp = klammern[pos+1][1];
739 
740             if  (tmp.indexOf("rsup") > -1){
741                 up = zeile.substring(klammern[pos+2][0] + 6 ,klammern[pos+2][1]-2);
742                 big = big + "\n<lowlimit>\n " + up + "\n</lowlimit>\n ";
743                 rp = klammern[pos+2][1];
744             }
745         }
746 
747         tmp = zeile.substring(lp-1,rp);
748         big = s2x_verarbeiten.ersetzen(zeile, tmp, big);
749         return big;
750     }
751 
752     /**
753      * fuer rsuP/rsuB
754      * @param zeile: die zu bearbeitete Zeile
755      * @param klammern: die Klammerstruktur der Zeile
756      * @param pos: Position in der Klammer
757      * @param rsu: b oder p?
758      * @return: bearbeiteter String
759      */
760     public static String rsu(String zeile, int[][] klammern, int pos, String rsu){
761         String tmp = new String();
762         int lp= 0, rp = 0;
763         String erg = new String();
764         String exp = new String();
765         String basis = new String();
```

```
766         boolean klammerndrin = false;
767         int []anz;
768         int i;
769 
770         if (zeile.charAt(klammern[pos][0] - 3) != ')' && (zeile.charAt(klammern[pos
                ][0] - 2) != ')')){
771 
772             exp = zeile.substring(klammern[pos][0]+5, klammern[pos][1]);
773 
774             // Ermittlung des Exponent: mit Trennzeichen " bzw (
775             lp = exp.indexOf("\"");
776             if (lp > -1 && lp < 2){ // so das das wort concat nicht dazwischen passt
777                 tmp = exp.substring(lp+1,exp.length()-1);
778                 rp = tmp.indexOf("\"");
779                 exp = tmp.substring(0,rp);
780 
781             }
782             else {
783                 lp = exp.indexOf("(");
784                 if (lp < 2 && lp > -1){
785                     exp = exp.substring(0,exp.length()-1);
786                     int x = 0;
787                     int anzkl = 0;
788                     int [][] innen = s2x_suchen.findeStrukturen(exp, "(", ")");
789                     innen = s2x_suchen.pruefeKlammerStruktur(innen, exp);
790                     anzkl = s2x_suchen.sucheAnzKlammern(innen);
791 
792                     lp = lp + 1;
793                     for (x = 0; x < anzkl; x++){
794                         if (innen[x][0] == lp){
795                             break;
796                         }
797                     }
798                     tmp = codocell(exp, innen, x);
799                     if (tmp != null){
800                         exp = tmp.substring(0,tmp.length());
801 
802                     }
803                     else {
804                         exp = null;
805                     }
806                 }
807                 else exp = null;
808             }
809             //System.out.println("Exponent:" + exp); // OK!
```

```
810
811 //            Ermittlung der Basis
812             tmp = zeile.substring(0,klammern[pos][0]-1);
813             anz = s2x_suchen.zaehleElemente(tmp, "\"");
814             i = 0;
815             while (anz[i] > 1){
816                 i++;
817             }
818             i = i - 1;
819             lp = anz[i-1];
820             rp = anz[i];
821             basis = zeile.substring(lp,rp-1);
822
823             tmp = new String();
824             int trennpos = -90;
825             String textrest = new String();
826             String basisrest = new String();
827             int k = basis.length()-1;
828             boolean getauscht = false;
829             while (k >= 0 && getauscht == false){
830             //for (int k = basis.length()-1; k >= 0; k--){
831
832                 // Wenn nur ein Element in der Basis steht
833                 if (
834                     basis.charAt(k) == '(' ||
835                     basis.charAt(k) == '=' ||
836                     basis.charAt(k) == ' ' )
837                     {
838                     if (trennpos < k) {
839                         trennpos = k;
840                         textrest = basis.substring(0,trennpos+1);
841                         basisrest = basis.substring(trennpos+1);
842                         //tmp = zeile.substring(lp-1,klammern[pos][1]);
843                         getauscht = true;
844                         //System.out.println("Text: " + textrest + "\tBasis: " +
                                basisrest);
845                     }
846                     //tmp = tmp + basis.substring(0,k+1);
847                     //System.out.println("Nur Index:" + tmp);
848                 }
849                 // fuer die Klammerausdruecke
850
851                 else if (basis.charAt(k) == ')' ){
852                     //System.out.println("Klammer zu!");
853                     klammerndrin = true;
```

```
854
855                     int m = rp;
856                     while (zeile.charAt(m) != '(' && m > 0){
857                         trennpos = m;
858                         m--;
859                     }
860
861                     // trennpos bekannt!
862                     if (trennpos > -1) {
863                         String wirdersetzt = new String();
864                         String ersatzbasis = new String();
865                         wirdersetzt = zeile.substring(trennpos-1,klammern[pos][1]);
866                         //System.out.println("()Zu ersetzen:\n" + zeile.substring(
                                trennpos-1,klammern[pos][1]));
867                         ersatzbasis = "<FTEXT>" + zeile.substring(trennpos-1,rp);
868                         textrest = "</FTEXT>";
869                         basis = ersatzbasis;
870                         tmp = wirdersetzt;
871                         //System.out.println("Basis davon:\n" + ersatzbasis);
872                         //System.out.println("-------");
873                         getauscht = true;
874
875                     }
876                 } // ende runde Klammer
877                 //*/
878                 k--;
879             }// ENDE while
880             //System.out.println("Alles: " + basis);
881             if (textrest.length() > 0){
882                 //System.out.println("Textrest da!");
883                 textrest = "<FTEXT>" + textrest + "</FTEXT>";
884             }
885             if (basisrest.length() > 0){
886                 //System.out.println("Basis da!");
887                 basis = basisrest;
888             }
889
890             //System.out.println("Text: " + textrest + "\nBasis: " + basisrest);
891
892             if (klammerndrin == false)
893                 tmp = zeile.substring(lp-1,klammern[pos][1]);
894             else
895                 tmp = zeile.substring(trennpos-1,klammern[pos][1]);
896
897             if (rsu.indexOf("b") > -1){
```

```
898                 //erg = textrest + "<MSUB><mi> " + basis + "</mi><mi> " + exp + "</mi
                        ></MSUB>";
899                 erg = "\n" + textrest + "\n<MSUB>\n<mi>\n " + basis + "\n</mi>\n<mi> "
                         + exp + "\n</mi>\n</MSUB>\n ";
900             }
901
902             else if (rsu.indexOf("p") > -1){
903                 //erg = textrest + "<MSUP><mi> " + basis + "</mi><mi> " + exp + "</mi
                        ></MSUP>";
904                 erg = "\n" + textrest + "\n<MSUP>\n<mi>\n " + basis + "\n</mi>\n<mi> "
                         + exp + "\n</mi>\n</MSUP> ";
905             }
906
907             if (tmp != null){
908             //if (getauscht == true){
909                 //System.out.println("Ersetze:" + tmp + "\ndurch:" + erg);
910                 erg = s2x_verarbeiten.ersetzen(zeile, tmp, erg);
911             }
912             else {
913                 //System.out.println("Abbruch:" + tmp + "\ndurch:" + erg);
914                 erg = null;
915             }
916
917             return erg;
918
919         }
920         else {
921             System.out.println("Es ist ein Fehler im RSU aufgetreten!");
922             return null;
923         }
924     }
925
926     /**
927      * fuegt fuer alle "" den Text-tag ein
928      * @param zeile: die zu bearbeitete Zeile
929      * @return: bearbeiteter String
930      */
931     public static String TextTag(String zeile, int art){
932         String erg = new String();
933         String tmp = new String();
934         String inhalt = new String();
935         String ersatz = new String();
936
937         erg = zeile;
938
```

```
939         if (erg.indexOf("\"") > -1){
940 
941             int anz = 1;
942             int [] pos = s2x_suchen.zaehleElemente(erg, "\"");
943             anz = 0;
944             while(pos[anz] > 0){
945                 anz++;
946             }
947 
948             inhalt = erg.substring(pos[0]-1,pos[1]); // alles mit "
949             ersatz = inhalt.substring(1,inhalt.length()-1);
950 
951             if (art == 0){
952                 ersatz = "<TEXT>"+ersatz.trim()+"</TEXT>\n";
953             }
954             else if (art == 1){
955                 ersatz = "<FTEXT>"+ersatz.trim()+"</FTEXT>\n";
956             }
957 
958             // wenn was leeres eingefuegt werden soll, fuege nix ein
959             if (ersatz.length() < 6+7+2){
960                     ersatz = "";
961             }
962 
963             tmp = s2x_verarbeiten.ersetzen(zeile, inhalt, ersatz);
964             if (tmp != null){
965                 erg = tmp;
966             }
967             else {
968                 System.out.println("Irgendwas ist schiefgelaufen");
969             }
970 
971         }
972         else {
973             erg = null;
974         }
975 
976         return erg;
977     }
978 
979     public static String ftext(String zeile){
980         // ersetze die "" durch <FText>
981         int anz = s2x_suchen.zaehleString(zeile, "\"");
982         String tmp = zeile;
983         for (int f = 0; f < anz/2+1; f++){
```

```
984             tmp = s2x_wortmethoden.TextTag(tmp,1);
985             if (tmp != null){
986                 zeile = tmp;
987             }
988         }
989         return zeile;
990     }
991
992 }
```