Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik, Studiengang Elektrotechnik

Institut für Nachrichtentechnik

Studiengang Bachelor Technische Informatik

Bachelor-Arbeit

Thema: Entwicklung eines KMS (Knowledge Management System) für die Erstellung von Mizar-Beweisen (KMS-MIZAR)

Student : Franck Edmond Chouaffé

Matrikelnummer : 11050907

Referent : Prof. Dr. phil. Gregor Büchel

Korreferent : Prof. Dr. Heiko Knospe

Abgabedatum : 30. Dezember 2008

Hiermit versichere ich, dass ich die Bachelor-Arbeit selbstständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

_____

Franck Edmond Chouaffé

Danksagung

Für die Betreuung, das Korrekturlesen und seine Bereitschaft das Referat zu übernehmen möchte ich Herrn Prof. Dr. phil. Gregor Büchel danken.

Ich danke Herrn Prof. Dr. Heiko Knospe für seine Bereitschaft das Korreferat zu übernehmen.

Meinen besonderen Dank möchte ich mich bei meinen Eltern und Geschwistern bedanken, die mich das ganze Studium über unterstützt haben.

Mein Dank gilt ganz herzlich meiner Schwester Doris Tchouaffé, die während des kompletten Studiums meine beste Beraterin war.

Ich möchte mich auch besonders bei Andreas Frey für seine Hilfe bedanken.

Mein weiterer Dank gilt allen Mitarbeitern des Bereichs Informatik

Inhaltverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Aufgabenstellung

Mizar ist ein Projekt, das gegen das Jahr 1973 mit dem Anspruch anfing, die Mathematik in einer computerorientierten Umgebung zu verfassen. Ein wichtiges Ziel ist hierbei, den Computer als mathematischen Beweisprüfer einzusetzen. Eine Mizar-Datei ist dementsprechend eine Datei, die mathematische Sätze ausführlich behandelt.

Hauptsächlich geht es in einer Mizar Datei darum, einen mathematischen Satz und dessen Beweis in die Mizar Sprache korrekt umzuschreiben, oder eben neue Begriffe einzuführen. Wie bei vielen Programmiersprachen (Mizar ist aber keine Programmiersprache, sondern eine formale Sprache) verfügt Mizar ebenfalls über einen Compiler namens `Verifier`, der eine Mizar-Datei auf Verstöße gegen die Mizar-Sprachregeln untersucht.

Beim Editieren des Beweises eines Satzes wird auf Hilfsbeweise, die selber Elemente[1] aus anderen Mizar-Dateien sind, verwiesen, wobei der Verweis mit dem geschriebenen Beweisschritt im Zusammenhang stehen muss.
Zusätzlich muss für alle in einer Datei angewandte Mizar-Symbole eine korrekte Umgebung[2] definiert werden. Symbole und deren Bedeutungen werden auf dieser Weise dem `Verifier` bekannt gemacht.

Aufgabenstellung

Diese Bachelor-Arbeit ist keine Anleitung darüber, wie das Umschreiben von mathematischen Sätzen in die Mizar-Sprache geschieht. Die Zielsetzung dieser Arbeit ist die Entwicklung eines KMS[3] (Knowledge Management System), das für einen mathematischen Satz einen Beweisrahmen bezüglich Mizar generiert. Außerdem kann das KMS benutzt werden, um Informationen über mathematische Begriffe im Allgemeinen und Mizar Symbole im Besonderen zu finden. Dabei wird untersucht, wie die Umgebung für die in einer Datei angewandten Symbole unter Anwendung des KMS-MIZAR korrekt definiert werden kann.

[1] Als Element wird hier entweder eine Definition, eine Notation,, eine Registration oder ein Theorem gemeint.

[2] Als Umgebung wird die Environment-Declaration gemeint. Dies wird im Kapitel 2 beschrieben.

[3] Das KMS wird im Rahmen dieser Arbeit KMS-MIZAR genannt.

Im Folgenden soll eine kurze Kapitelbeschreibung gegeben werden:

- Das folgende Kapitel 2 **„Grundlagen“** gibt grundlegende Informationen über Mizar Dateien an. Es werden nur Aspekte der Mizar Sprache eingegangen, die für die Zielsetzung dieser Arbeit relevant sind.
- Das Kapitel 3 **„Datenanalyse“** beschäftigt sich zum einen mit der Zerlegung der Mizar Dateien und zum anderen mit dem Entwurf der relationalen Datenbank des KMS-MIZAR.
- In Kapitel 4 **„Aufbau der Datenbank des KMS-MIZAR (Insert-Programm des KMS-MIZAR)** wird die Implementierung des Insert-Programms des KMS beschrieben.
- Die Implementierung des Select-Programms des KMS wird in Kapitel 5 **„Select-Programm des KMS-MIZAR“** beschrieben. Die unterschiedlichen Arten von Select-Methoden werden vorgestellt.
- In Kapitel 6 **„Erzeugung des Beweisrahmens“** werden die verschiedenen Arten von Beweisschritten sowie die Implementierung des Beweisrahmensgenerators beschrieben. Es wird im Weiteren eine Einführung in das entwickelte KMS angeboten, wobei anhand eines Beispielssatzes erläutert wird, wie die Umgebung für die verwendeten Symbole richtig definiert wird.
- Abschließend folgt eine Schlussbetrachtung.

Bei Lesern dieser Bachelor-Arbeit wird Grundlagenwissen in der Sprache Mizar empfohlen. Grundlagen zum Thema Mizar werden nur komprimiert wiedergegeben. Ein empfehlenswerter Link zum Thema Mizar ist das Dokument „Writing a Mizar article in nine easy steps“ von Freek Wiedijk [4].

Anmerkungen zur Sprache

Um Fehlübersetzungen zu vermeiden, werden englische Begriffe beibehalten. Diese Begriffe unterliegen folgenderweise der englischen Grammatik.

2 Grundlagen

Im Rahmen dieser Bachelor-Arbeit werden Daten aus Dateien aus der MML[4] extrahiert und in eine Datenbank eingelesen, um diese darauf wiederum abzufragen. Aus diesem Grund ist es relevant die Struktur der Mizar-Dateien zu analysieren.

2.1 Struktur der Mizar-Dateien

Nachdem das Mizar System installiert wurde (siehe Kapitel 6.4 „Einführung in das KMS-MIZAR“, Seite 61), stehen standardmäßig zwei Arten von Dateien zur Verfügung: die abstrakten Dateien mit der Endung *.abs*, die in dem Ordner *c:\mizar\abstr* enthalten sind und die vollständigen Mizar-Dateien mit der Endung *.miz* die in dem Ordner *c:\mizar\mml* zu finden sind. Die ersten Dateien bilden die sog. MML. Abstrakte Dateien beschreiben die `theorems` und andere Elemente ohne ihre Beweise. Die Beweise sind den vollständigen Dateien zu entnehmen. Für diese Arbeit werden die abstrakten Dateien verwendet.
Jede Mizar-Datei unterliegt einer BNF (Backus_naur Form) und besteht aus zwei Hauptteilen. Der erste davon ist die so genannte `Environment-Declaration` (Umgebung der angewandten Symbole), die seinerseits einige Direktive[5] einleitet. Eine Direktive ist ähnlich wie eine Import-Anweisung innerhalb einer höheren Programmiersprache (wie Java).
Der zweite Teil besteht aus bis zu fünf unterschiedlichen Elementen. Diese sind `definitions`, `notations`, `registrations`, `theoems` und `schemes`. Die `schemes` werden aufgrund ihrer Komplexität nicht weiter betrachtet.

2.1.1 Definition

Eine `definition` ist der Teil einer Mizar-Datei, in dem Mizar-Symbole definiert oder erneut definiert werden. Alle Mizar-Symbole, die zur Auswahl stehen, wurden bereits in einer anderen Datei definiert. Benötigte Symbole die noch nicht definiert sind, können mit einer `definition` beschrieben werden.

[4] MML(Mizar Mathematical Library) besteht aus allen Dateien, die für das Mizar System geschrieben worden sind und wird durch den Ordner *c:\mizar\abstr* abgebildet.

[5] Es können bis zu xy Direktiven in einer Mizar-Datei vorkommen, wobei jede davon eine bestimmte Bedeutung hat. Direktiven grenzen die Umgebung der in einer Datei benutzten Symbole ab.

Auszug aus der Datei SQUARE_1:

```
definition
  let x be complex number;
   func x^2 equals
   x * x;
 end;
```

Beispiel 2.1 Definition von ^2

Das obige Beispiel 2.1 definiert den Operatoren ^2 für komplexe Zahlen. Operatoren werden in Mizar als functor bezeichnet und deren definition wird mit dem Schlüsselwort func eingeführt. Mizar bietet eine zweite Art von definition an, und zwar die mit dem Schlüsselwort redefine, die einem bestimmten Ausdruck einen mehr passenden radix type[6] zuweist. Um den Begriff radix type zu verstehen, ist an dieser Stelle eine zusätzliche Erklärung notwendig.
Alle Ausdrücke in Mizar haben einen type. Bei der Verwendung eines functors ist verbindlich, dass die types der Operanden den types, die Argrumente des functor annehmen können, entsprechen. Ein Mizar type besteht aus einem mode und einer Liste von Attributen. Seine abstrakte Form ist die folgende:

$a_1 \quad a_2 \quad \ldots \quad a_n \quad \mathrm{M} \quad \texttt{of} \quad x_1 \quad x_2 \ldots \quad x_m$,

wobei $a_1 \quad a_2 \quad \ldots \quad a_n$ Adjektive bzw. Attribute sind, M ein Mode darstellt und $x_1 \quad x_2 \ldots \quad x_m$ Terme sind.
Hier ist ein Beispiel eines type in Mizar.

```
non empty finite Subset of NAT.
```

non[7] empty und finite sind hier die Attribute, Subset of ist das mode und NAT ist das Argument relativ zu Subset. Das Schlüsselwort of bindet das mode mit seinen Argumenten zusammen, falls das mode welche besitzt (ein mode muss nicht zwangsläufig Argumente haben).

[6] Type ist ähnlich wie eine Datentyp.

[7] Mit dem Schlüsselwort non kann ein Attribut negiert werden.

Die folgende Liste listet weitere öfter benutzte Mizar `types` auf:

```
set
number
Element of X
Subset of X
Nat
Integer
Real
Ordinal
Relation
Relation of X,Y
Function
Function of X,Y
FinSequence of X
```

(In dieser Auflistung ist `Nat` ein `mode`, dies ist nicht zu verwechseln mit dem Term `NAT` im obigen Bespiel.)
Hier ist auch zu beachten, dass ein `mode` als eigenständiger `type` stehen darf.
Die folgende `definition` aus der Datei `PEPIN` illustriert die Verwendung des Schlüsselworts `redefine`.

```
definition
  let n be Element of NAT;
  redefine func n^2 -> Element of NAT;
end;
```

Beispiel 2.2 Neudefinition von ^2

Was diese Neudefinition macht, ist den `type` des Ausdrucks `n^2` auf `Element of Nat` zu setzen, wobei `n` vom `type` `Element of Nat` ist. Die originale `definition` von dem `functor ^2` befindet sich in der Datei `SQUARE_1` (siehe Beispiel 2.1). Diese Neudefinition hat für Effekt, dass der `functor ^2`, der vorher lediglich Argumente vom `type complex number` zuließ, Argumente von `type Element of NAT` annehmen darf, nachdem er auf `Element of NAT` spezialisiert wurde.

2.1.2 Notation

In einem `Notation-Block` kann man für ein gegebenes `attribut`, einen `functor`, ein `mode` und ein `predicate` ein Synonym festlegen. Außerdem besteht ebenfalls die Möglichkeit für ein `attribut` und ein `predicate` ein Antonym zu definieren. Beispiel für ein Antonym aus der Datei `ABIAN`.

```
notation
  let i be number;
  antonym i is odd for i is even;
end;
```

Beispiel 2.3 Antonym von even

Dies bedeutet, dass das Schlüsselwort `odd` das Gegenteil von `even` ist.

```
notation
  let i,j be natural Ordinal;
  synonym quotient(i,j) for i/j;
end;
```

Beispiel 2.4 Synonym von i/j

Der obige Auszug aus `ARYTHM_3` zeigt, dass die Schreibweise `quotient(i,j)` eine synonyme Schreibweise von `i/j` ist.

2.1.3 Registration

`Registrations` werden am häufigsten gebraucht, um weitere Attribute für einen Ausdruck einzutragen. Dies gelingt durch das Schlüsselwort `cluster.` Hier ist ein Beispiel:

```
registration
  cluster natural -> real number
end;
```

Beispiel 2.5

Diese `registration` bedeutet, dass alle Ausdrücke, die als `type natural number` haben, auch das `attribut real` annehmen dürfen. Insofern wird `natural number` äquivalent zu `natural real number.`

Diese Möglichkeit der „Typkonvertierung“ wird in Beweisen öfter benutzt, um Ausdrücke mit einem gewollten `type` zu versehen, den sie standardmäßig nicht haben. Diese Art von `registration` basiert auf dem `type` des Ausdrucks. Eine zweite Variante basiert zusätzlich auf der Darstellung des Ausdrucks.

```
registration let X, Y be finite set;
  Cluster X \/Y -> finite;
end;
```

Beispiel 2.6

Es ist hier zu verstehen, dass alle Ausdrücke der Form `X\/`[8]`Y`, wobei `X` und `Y` den `type finite set` bekommen, ebenfalls das `attribut finite` haben. Die dritte Variante von `registration` (ohne das Zeichen `->`) hat nichts mit dem Eintragen von zusätzlichen Adjektiven zu tun. Sie erlaubt hingegen, bestimmte `type` zu benutzen. Hier ist ein Beispiel.
Um den `type non empty finite Subset of NAT` benutzen zu dürfen, wird die folgende `registration` gebraucht.

```
registration
  Cluster finite non empty Subset of X;
end;
```

Beispiel 2.7

wobei `X` durch einen Term wie `NAT` zu ersetzen ist.

2.1.4 Theorem

Ein `theorem` ist ein mathematischer Satz, der in der abstrakten Datei nicht bewiesen wird. Es wird nur die Behauptung formuliert. In einem `theorem` wird immer von einer Liste von Variablen ausgegangen, die alle entweder von einem oder unterschiedlichen `types` sind. Das folgende `theorem` aus der Datei `SQUARE_1` behauptet den binomischen Lehrsatz für ein Polynom 2-ten Grades.

```
theorem :: SQUARE_1:63
for a, b being complex number holds
```

[8] Das Zeichen \/ steht für das Vereinigungssymbol $\cup$.

```
(a + b)^2 = a^2 + 2*a*b + b^2;
```

Beispiel 2.8 Binomischer Lehrsatz eines Polynoms 2-ten Grades

Der Beweis ist dem Satz nicht gebunden. In Mizar fängt ein Beweis mit dem Schlüsselwort `proof` an und wird mit dem Schlüsselwort `end` beendet. Zum Ansehen des Beweises dieses `theorem` wird auf die Datei `SQUARE_1.miz` verwiesen.

2.2 Java und Oracle

Oracle ist ein kostenpflichtiges RDBMS (Relational Database Management System), das auf SQL[9] basiert. SQL(Structured Query Language) ist eine genormte Anfragesprache für relationale Datenbanken. Der Sprachumfang von SQL kann in 3 Teilmengen eingeteilt werden.

- DDL := Data Definition Language: Insbesondere für die Verwaltung der Metadata.
- DML := Data Manipulation Language: enthält die Kommandos zum Schreiben auf und zum Lesen von Datenbank-Tabellen.
- DCL := Data Control Language: enthält die Kommandos zur Transaktions- und Benutzerverwaltung.

Oracle Database XE (Express Edition) ist eine kostenlos erhältliche Version eines RDBMS von der Firma Oracle Corporation, die bis zu 1 GB Speicherplatz zur Verfügung stellt.
Obwohl Oracle die SQL-Norm unterstützt, gibt es einige Unterschiede in der Implementierung von Befehlen zu beachten. Ein Umstieg von Oracle auf eine andere RDBMS kann zu notwendigen Änderungen in der Implementierung der SQL Anweisungen führen. Z.B. für die Umwandlung eines Datumsformates wird in Oracle oftmals die Funktion `TO_DATE()` angewandt.

```
TO_DATE(`"newdat"`, `YYY-MM-DD`),
```

wobei `newdat` das zu formatierende Datum ist und `` `YYYY-MM-DD` `` die gewünschte Notation des Datums. Bei einem Wechsel von Oracle zu MySQL (ebenfalls ein RDBMS) müsste diese Anweisung umgeschrieben werden, da MySQL die Funktion TO_DATE() nicht implementiert.
Um die Vorteile einer höheren Programmiersprache (hier Java) und von SQL zu Nutze zu machen, hat die Firma Sun eine Datenbankschnittstelle namens JDBC (Java Database

[9] SQL: http://www.w3schools.com/sql/.

Connectivity) entwickelt, die den Zugriff aus Java-Applikationen auf Datenbanken ermöglicht. Ein solcher Zugriff erfolgt in 4 Schritten.

a. Importieren der notwendigen Pakete

Die für die Ausführung von JDBC notwendigen Klassen liegen im Package `java.sql` vor, die in einer Java-Applikation mit der folgenden Syntax importiert werden kann:

```
Import java.sql.*;
```

b. Mitteilung über den zu verwenden JDBC-Treiber erzeugen

```
OracleDataSource ods = new OracleDataSource();
String driver = "Oracle.jdbc.driver.OracleDriver";
```

c. Laden des Treibers

Zur Ausführung der JDBC-Befehle muss ein Treiber geladen werden. Am besten wird dieser Vorgang innerhalb eines `try-catch-block` realisiert, um die `java.lang.ClassNotFoundException` abzufangen, die geworfen wird, falls der Treiber nicht gefunden werden konnte.

```
try{
Class.forName(driver).newInstance();
}
Catch(ClassNotFoundException e)
{
System.out.println(„An error has occured : „+e.getMessage());
}
```

d. Erstellung einer Datenbankverbindungsinstanz

Die zu erzeugende Instanz ist eine Instanz der Klasse `Connection`. Zum Verbindungsaufbau wird die URL (Uniform Resource Locator) der Verbindungsanfrage aufgebaut.

Diese URL sieht folgendermaßen aus:

Oracle-Protokoll-Name+User+Passwort+TCP-Rechnername+Port+Oracle-DB-Schema-Name.

(siehe www.nt.fh-koeln.de/fachgebiete/inf/buechel/GesamtskriptDB0506.pdf)

```
Connection con = null;
try{
ods.setURL("jdbc:oracle:thin:user/passw@fichte.example.com:1521:orcl
");
con = ods.getConnection();
}
catch(SQLException sqlex)
{
System.out.println(„An error has occurred during the connection
attempt : „+sqlex.getMessage());
}
```

Aus Grund der Performance von Oracle und die Portabilität von JDBC wurde diese Arbeit mit Oracle als RDBMS und Java als Programmiersprache realisiert. Die Datenbankverbindung kann über JDBC für verschiedene RDBMS mit Hilfe von Treibern angepasst werden.

.

3 Datenanalyse

Dieses Kapitel beschäftigt sich zum einen mit dem Vorgang, wie die Mizar-Dateien zerlegt werden, nachdem die unterschiedlichen Elemente einer Datei in Kapitel 2 „Grundlagen“ vorgestellt worden sind, und zum anderen mit dem konzeptuellen Entwurf einer relationalen Datenbank, welche die Struktur der Dateien widerspiegelt. Zuvor soll jedoch die Systemstruktur erläutert werden:

3.1 Systemstruktur des KMS-MIZAR

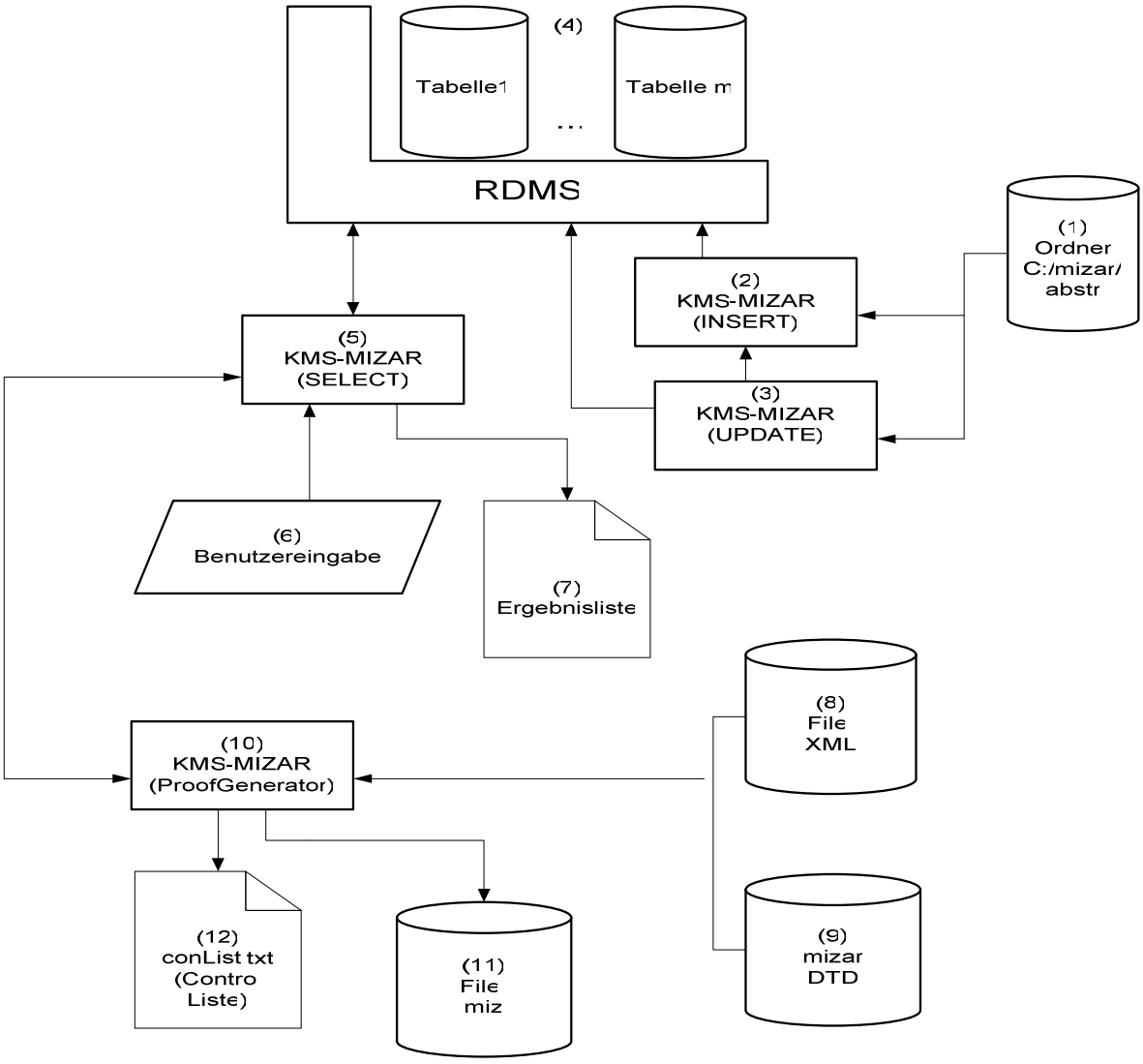


Abbildung 3.1 Komponente des KMS-MIZAR

Die Abbildung 3.1 zeigt alle Komponenten unseres Systems und wie sie aufeinander zugreifen an.
Um das KMS-Mizar anzuwenden, werden zunächst Textdateien, die sich in der MML befinden, siehe (1) in Abbildung 3.1 Komponenten des KMS-MIZAR, in die
Tabelle 1…Tabelle m, siehe (4), der Datenbank geschrieben.
(2) besteht aus allen Java Klassen, die Dateien aus der MML erstmal in Teile zerlegt und dann die einzelnen Teile in die Datenbank schreiben.
(3) besteht aus allen Java Klassen, die für die Aktualisierung der Datensätze der Datenbank zuständig sind. Das Einlesen bzw. Aktualisieren der Daten wird im Kapitel 4 „Aufbau der Datenbank des KMS-MIZAR (Insert-Programm des KMS-MIZAR) genauer beschrieben.
Die Komponente (5) implementiert die Suchfunktion des Systems. Sie nimmt die Benutzereingabe (6) entgegen, baut bezüglich der Eingabe SQL-Anfragen auf, und gibt das Resultat der Anfrage als Ergebnisliste (7) an der Konsole aus. Die Suchfunktion des KMS-Mizar ist in Kapitel 5 „Select-Programm des KMS-MIZAR“ beschrieben.
Für die Erstellung des Beweisrahmens erstellt der Benutzer ein XML-Dokument, siehe (8), das zu der DTD bzw. die Grammatik (9) der XML-Dokumente valide ist.
Die Umwandlung der XML-Datei in eine Mizar-Datei (11) wird mit Hilfe des
Proof-Generator (10) durchgeführt. Der Proof-Generator, der aus einige für diese Arbeit entwickelten Java Klassen besteht, erzeugt auch den Beweisrahmen der in die Mizar-Datei eingebunden wird. Wenn innerhalb der XML-Datei fehlerhafte Schlüsselwörter angewandt werden, werden in die Textdatei (12) Fehlermeldungen bei umwandeln eingetragen.

3.2 Speicherung der Daten

Für ein flexibles und modulares System werden die `theorems`, `notations`, `registrations`, und `definitions` jeweils in einer eigenen Tabelle mit entsprechenden Feldern gespeichert.
Abhängig davon wonach der Benutzer sucht, wird dann seine Eingabe mit dem Inhalt aus der passenden Spalte in der zugehörigen Tabelle verglichen. Es wird dann die Elemente zurückgegeben, bei denen eine Übereinstimmung mit der Eingabe vorliegt. Die Datenbankablage von `definitions, notations` und `registrations` ist mit der von `theorems` unterschiedlich, dies soll im Folgenden behandelt werden. Es wird die folgende `definition` aus der Datei `XCMPLX_0` betrachtet.

```
definition
  let x,y be complex number;
  func x+y means
  :: XCMPLX_0:def 4

  ex x1,x2,y1,y2 being Element of REAL st
  x = [*x1,x2*] & y = [*y1,y2*] & it =[*+(x1,y1),+(x2,y2)*];
  commutativity;
  func x*y means
  :: XCMPLX_0:def 5

  ex x1,x2,y1,y2 being Element of REAL st
  x = [*x1,x2*] & y = [*y1,y2*] &
  it = [* +(*(x1,y1),opp*(x2,y2)), +(*(x1,y2),*(x2,y1)) *];
  commutativity;
end;
```

Beispiel 3.1

Innerhalb dieses `Definition-Block` befinden sich zwei `definitions`. Die erste für den Mizar-`functor` ‚+' und die letzte für den ebenfalls Mizar-`functor` ‚*'. Diese beiden `definitions` werden separat gespeichert, um den Fall zu vermeiden, dass der gesamte Block ausgegeben wird, falls sich der Benutzer nur für die `definition` eines von den beiden Mizar `functor` interessiert.

Eine Besonderheit von `definitions` ist, dass bei deren Variante, in der das Schlüsselwort `redefine` verwendet wird, auch mehrere Zeilen mit diesem Schlüsselwort auftauchen dürfen, wobei jede Zeile ein Mizar-Symbol erneut definiert. In einem solchen Fall werden die Zeilen separat in die Datenbank eingetragen. Hier ein Beispiel aus der Datei SQUARE_1:

```
definition
  let x,y be Element of REAL;
  redefine func min(x,y) -> Element of REAL;
  redefine func max(x,y) -> Element of REAL;
end;
```

Beispiel 3.2

Somit können überflüssige Informationen in der Ausgabe vermieden werden. Bei den beiden Variante von `registrations`, die im Kapitel 1.1.3 „Registration" erwähnt wurde, ist diese

Besonderheit ebenfalls zu bemerken, wobei jede Zeile diesmal mit dem Schlüsselwort `cluster` eine eingenständige `registration` darstellt. Die folgenden Beispiele wurden aus der Datei `ABIAN` entnommen.

```
registration
  cluster even Element of NAT;
  cluster odd Element of NAT;
  cluster even Integer;
  cluster odd Integer;
end;

registration
  let i be even Integer, j be Integer;
  cluster i*j -> even;
  cluster j*i -> even;
end;
```

Beispiel 3.3

Es wird dadurch deutlich, dass eine separate Speicherung der Zeilen notwendig ist.
Der folgende Ausschnitt aus der Mizar-Datei `XXREAL_0` lässt ebenfalls diese Charakteristik für `notations` feststellen. D.h. getrennte Speicherung jedes Synonym- bzw. Antonym-Eintrags.

```
notation
  let a,b;
  synonym b >= a for a <= b;
  antonym b < a for a <= b;
  antonym a > b for a <= b;
end;
```

Beispiel 3.4

`Theorems` im Gegenteil haben keine besonderen Merkmale. Jedoch hat jedes `theorem` genauso wie jede `definition` einen Bezeichner, der eine eigene Spalte in der Datenbank bekommt. Für das achte `theorem` aus der Datei `XCMPLX_1` sieht der Bezeichner wie folgt aus: `XCMPLX_1:8`

3.3 Datenbankstruktur des KMS-MIZAR

3.3.1 Entitäten der Datenbank

Das für diese Arbeit entwickelte System liefert Informationen über Mizar Begriffe zurück. Diese Informationen, die an der Konsole ausgegeben und gegebenenfalls in eine Datei geschrieben werden, stammen ausschließlich aus Mizar-Dateien.

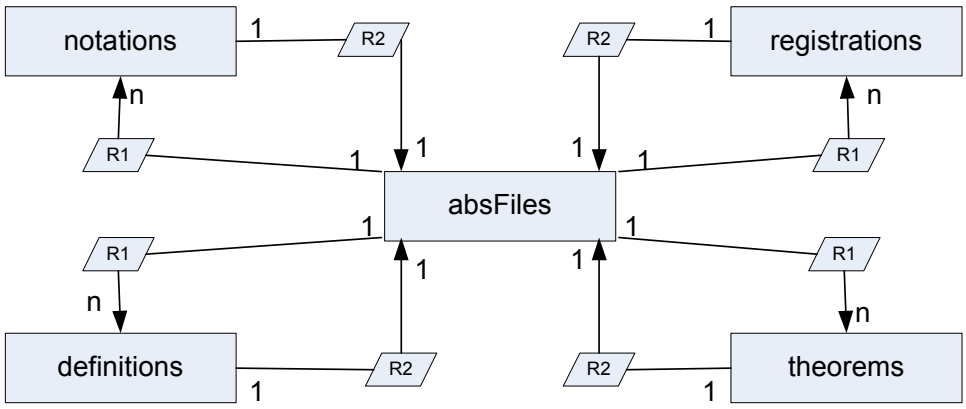
Da eine Datei mehr als ein der oben genannten Elemente beinhalten kann, wird eine Tabelle für die Speicherung der Dateien in der Datenbank angelegt. Des Weiteren ist es ebenfalls notwendig für jedes diese Elemente jeweils eine Tabelle zu erstellen.

Aus dem Grund, dass eine Datei mehr als ein Exemplar eines Dateielements haben darf, lässt sich eine (1-n)-Beziehung zwischen einer Datei und ihren Elementen abbilden, wobei sowohl eine Datei als auch ihre Elemente eigenständige Entitäten repräsentieren.

3.3.2 Entity-Relationship Diagram der Tabellen

Ein Entity-Relationship Diagram (ERD) ist ein Graph, der die Relationen der einzelnen Tabellen bzw. Entitäten einer Datenbank zueinander abbildet. Ein ERD[10] verwendet Symbole, um drei Typen von Informationen darzustellen. Rechtecke werden benutzt, um Entitäten darzustellen. Die Raute wird eingesetzt für die Darstellung von Relationen, und die Bus- bzw. Ellipsen-Notation stellt die Attribute dar.

Aus dem ERD kann, sofern es in 1. Normalform vorliegt, unmittelbar die Struktur der Datenbank erstellt werden.



R1:= enthält
R2:= gehört zu n>=0

Abbildung 3.2 ERD der Datenbank

[10] Prof. Dr. phil. G. Büchel: www.nt.fh-koeln.de/fachgebiete/inf/buechel/GesamtskriptDB0506.pdf

3.3.3 Relationenschemata der Tabellen

In diesem Kapitel werden alle Entitäten des Entity-Relationship Diagram auf ein Relationenschema abgebildet. In der Zeile spi stehen die Namen der Attribute, die jede Entität besitzt. In der Zeile dti ist für jedes Attribut einen korrespondierenden Oracle Datentyp eingetragen und in der letzten Zeile jeder Tabelle ist die Integritätsbedingungen, die jedes Attributwert erfüllen muss, aufgelistet. Jeder Eintrag in einer Tabelle ist eindeutig identifizierbar durch einen Primärschlüssel. Der Fremdschlüssel einer Tabelle bezieht sich immer auf den Primärschlüssel einer anderen Tabelle.
(Übernommene Notation siehe
www.nt.fh-koeln.de/fachgebiete/inf/buechel/GesamtskriptDB0506.pdf)

RS(absFiles) = A

spi	fileID	fileName	description	defNo	notNo	regNo	theoNo	locked
dti	int	varchar2(30)	varchar2(1000)	int	int	int	int	int
wi	PRIK							{0,1}

Tabelle 3.1Relationenschema für die abstrakten Dateien aus der MML

defNo, notNo, regNo und theoNo sind respektiv die Anzahl der `definitions`, `notations`, `registrations` und `theorems`, die sich in einer Datei befinden. Die Spalte fileID ist der PRIK[11] der Tabelle absFiles und fileName steht für den Namen der Dateien aus der MML, die mittels des KMS-MIZAR in der Datenbank eingetragen wurden. Die Spalte description gibt eine kurze Information über die Datei. Diese Information ist der Datei *C:\mizar\doc\mml* zu entnehmen. Die Rolle der Spalte locked wird im Kapitel
4.3.2 „Die Insert-Befehle“, Seite 36, erläutert.

RS(definitions) = B

spi	defID	fileID	text	redefine	pattern	defName
dti	int	int	varchar2(4000)	int	varchar2(1000)	varchar2(100)
wi	PRIK	FKEY(A.fileID)		{0,1}		

Tabelle 3.2 Relationenschema für die gesamten definitions

[11] PRIK steht für Primärschlüssel.

Alle `definitions` werden in der Tabelle definitions abgespeichert. Wie für die unten erstellten Tabellen besitzt die Tabelle definitions ein FKEY[12], das auf das PRIK der Tabelle absFiles verweist. Dies ist eine Folge der (1-n)- Relation aus dem ERD. Die Spalte Redefine ermöglicht, zu ermitteln, ob es sich um eine Neudefinition eines Mizar-Symbols handelt. Die Spalte Text, die ebenfalls in anderen Tabellen zu finden ist, ist die, in der der eigentliche Inhalt gespeichert ist. Die Spalte defName enthält die Namen der `definitions`, sofern sie vorhanden sind. Die Spalte pattern ist von großer Relevanz während der Suchanfrage, da sie nicht die gesamte `definition`, sondern lediglich den definierten Ausdruck innerhalb einer `definition` enthält.

RS(notations) = C

spi	notID	fileID	text
dti	int	int	varchar2(4000)
wi	PRIK	FKEY(A.file_ID)	

Tabelle 3.3 Relationenschema für die gesamten notations

RS(registrations) = D

spi	regID	fileID	text	existential
dti	int	int	varchar2(4000)	int
wi	PRIK	FKEY(A.file_ID)		{0,1}

Tabelle 3.4 Relationenschema für die gesamten registrations

Es wurde in dem Abschnitt 1.1.3 „Registration“ erklärt, dass es eine Variante von `registration` gibt, die nichts mit dem Hinzufügen von Attributen zu tun hat. In Mizar werden sie `existential Cluster` genannt. Die Spalte Existential gibt also die Möglichkeit zu ermitteln, ob es so eine `registration` ist oder nicht.

[12] FKEY steht für Fremdschlüssel.

RS(theorems) = E

spi	theo_ID	file_ID	text	theoName
dti	int	int	varchar2(4000)	varchar2(30)
wi	PRIK	FKEY(A.file_ID)		

Tabelle 3.5 Relationenschema für die gesamten theorems

Die Spalte theoName speichert den Namen der Theorems.

3.3.4 CREATE-TABLE-Statements der Tabellen

Dieser Schritt kann sehr hilfreich sein und den Aufwand drastisch reduzieren, wenn das System auf einem anderen Rechner abgespielt werden sollte. Dann werden die Tabellen nach Ausführung der jeweiligen CREATE TABLE Kommandos automatisch angelegt.

```
create  table absFiles(
fileID integer not null constraint fileID_a primary key,
fileName varchar2(1000) not null,
description varchar2(4000) not null,
defNo integer not null,
notNo integer not null,
regNo integer not null,
theoNo integer not null,
locked integer not null check(locked = 0 or locked = 1));



create table definitions(
defID integer not null constraint defID primary key,
fileID   integer   not   null   constraint   fileID_b   references
absFiles(fileID),
text varchar2(4000) not null,
redefine integer not null check(redefine = 0 or redefine = 1)
pattern varchar2(4000) not null,
defName varchar2(100) not null);



create table notations(
```

```
notID integer not null constraint notID primary key,
fileID integer not null constraint fileID_c references
absFiles(fileID),
text varchar2(4000) not null
);


create table Registrations(
regID integer not null constraint regID primary key,
fileID integer not null constraint fileID_d references
absFiles(fileID),
text varchar2(4000) not null,
existential integer not null check(existential = 0 or existential =
1)
);


create table theorems(
theoID integer not null constraint theoID primary key,
fileID integer not null constraint fileID_e references
absFiles(fileID),
text varchar2(4000) not null,
theoName varchar2(4000)
)
```

4 Aufbau der Datenbank des KMS-MIZAR (Insert-Programm des KMS-MIZAR)

In den kommenden Abschnitten dieses Kapitels werden die Algorithmen, die zum einen zum Aufsplittern der Dateien und zum anderen für das Ablegen der Daten in die Datenbank implementiert wurden, vorgestellt.

4.1 Erstes Übersichtdiagramm

Das hier aufgeführte Modul-Übersichtsdiagramm dokumentiert das Aufrufverhalten der Module des Insert-Programms des KMS-MIZAR. Das Insert-Programm stellt die Komponente (2) aus der Systemstruktur, Seite 16, dar.

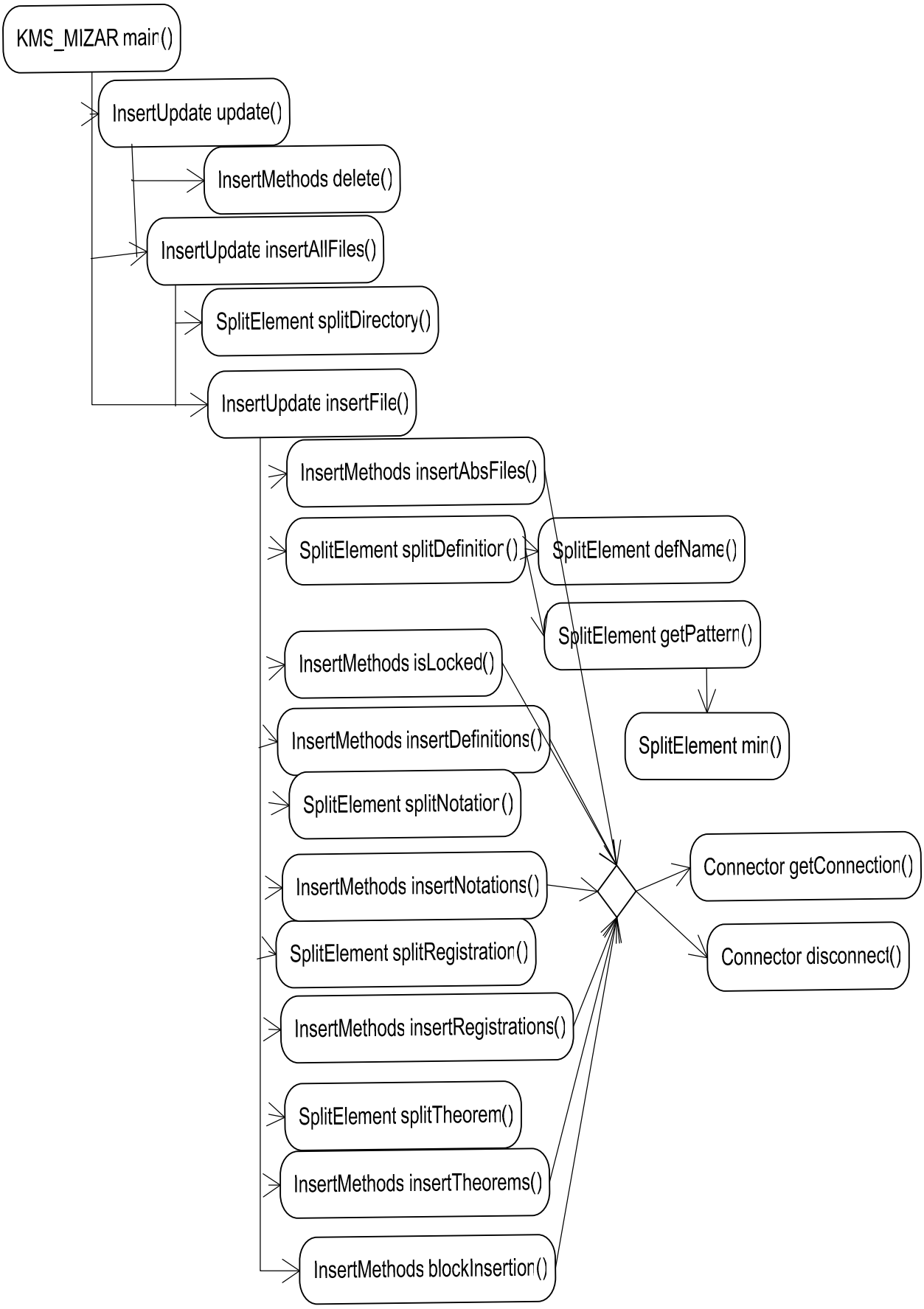


Abbildung 4.1 Übersichtsdiagramm des Insert-Programms

4.2 Zerlegung einer gegebenen Mizar-Datei

Für ein gut funktionierendes System muss eine Datei, die in der Datenbank einzutragen ist, gezielt zerlegt werden und daraus die gewünschten Teile herausnehmen, die dann später in die entsprechenden Tabellen der Datenbank persistent gespeichert werden. Die Java Klassen `InsertUpdate` und `SplitElement` implementieren die erforderlichen Algorithmen.

4.2.1 Allgemeine Information über eine Mizar-Datei

Die Tabelle `absFiles` besitzt die Spalte description zur Angabe einer kurzen Beschreibung davon, welches mathematische Thema in der Datei behandelt ist. Diese Beschreibung steht am Anfang einer jeden Mizar-Datei. Obwohl die Beschreibung nicht der BNF unterliegt, kann festgestellt werden, dass es zwischen dieser Beschreibung und dem restlichen Teil der Datei eine Zeile mit lediglich dem Zeichen `'::'` steht. Der folgende Programmausschnitt aus der Methode `insertFile()` der Java-Klasse `InsertUpdate` nutzt diese Tatsache, um den erforderlichen Datensatz für die Spalte description zu bestimmen,

```
[…]
try {
            raf = new RandomAccessFile(filename, "r");
            raf.seek(0);
      while ((line = raf.readLine()) != null
&& !line.endsWith("::")) {
                  des.append(line + "\n");
                  }
            description = des.toString();
            raf.close();
            }[…]
```

wobei `des` eine Instanz der Klasse `StringBuffer` ist.

4.2.2 Extraktion der unterschiedlichen Dateielemente

Es soll im Folgenden die BNF für weitere Elemente einer Datei betrachtet werden.

```
Definitional-Block = "definition" { Definition-Item | Definition } {
Redefinition-Block } "end" .
```

```
Notation-Block = "notation" { Loci-Declaration | Notation-
Declaration } "end" .

Registration-Block = "registration" { Loci-Declaration | Cluster-
Registration | Identify-Registration } "end" .

Theorem = "theorem" Compact-Statement .
```

Alle diese Elemente beginnen mit entsprechenden Schlüsselwörtern und enden mit dem Schlüsselwort `end`. Mit der Ausnahme von dem Element `Theorem`.

```
[…]
offsetd = new ArrayList<Long>();
while ((line = raf.readLine()) != null) {
     if (line.startsWith("definition")) {
          defno++;
          offsetd.add(raf.getFilePointer());
          }
          }[…]
```

Der oben liegende Code, der ebenfalls aus der Methode `insertFile()` entnommen wurde, ermöglicht es, die Anzahl an `definitions` in einer Datei zu ermitteln. Der Aufbau dieses Programmcodes sieht für die anderen Elemente ähnlich aus. Diese Anzahl ist später Inhalt der Spalte defno in der Tabelle absFiles. Nachdem die Tabelle absFiles mit Werten ausgefüllt wurde, sollen die Werte für die anderen Tabellen gebildet werden. Um dies zu erreichen, wurde jeweils für jedes Element aus einer Mizar-Datei eine Liste `ArrayList<Long>` aus Elementen vom Datentyp `Long` erzeugt, die die Position des Satzzeigers speichert, sobald das Wort `definition` gelesen wird. Es ist zu merken, dass bis auf Hinweis gilt für die anderen Elemente sämtliche Erläuterungen die für `definitions` gelten. In dem obigen Code heißt diese Liste `offsetd`. Das abschließende `d` steht für `definition`. Um einen gesamten `Definitional-Block` zu speichern, wird den Satzzeiger nach und nach an die Positionen, die von `offsetd` geliefert sind gesetzt und die Datei wird ab diesen Stellen bis auf das nächste Wort `end`, wie es in dem folgenden Ausschnitt aus `insertFile()` zu sehen ist, gelesen.

```
[…]
ArrayList<Object> containerd = new ArrayList<Object>();
for (int i = 0; i < offsetd.size(); i++) {
      raf.seek(offsetd.get(i));
      bufferd = new StringBuffer();
      while ((line = raf.readLine()) != null
            && !line.endsWith("end;")) {
            bufferd.append(line + "\n");
            }
            containerd.add(bufferd);
            }[…]
```

Anschließend werden alle diese `Definitional-Blocks` in der Variablen `containerd` gespeichert. Weiterhin steht das abschließende `d` für `definition`.

Bei `theorems` sieht der Algorithmus anders aus. Anders als bei den anderen Elementen, enden `theorems` nicht mit dem eindeutigen Schlüsselwort `end`, sondern lediglich mit einem Semikolon. Mehrere Semikolons können aber in einem `theorem` vorkommen.

Jedoch darf nach einem `theorem` eine bestimmte Menge an Schlüsselwörtern auftauchen, die auf dessen Ende andeuten. Diese Wörter werden in dem unmittelbar unten stehenden Code in blau gekennzeichnet.

Ausschnitt aus `insertFile()`:

```
[…]
String[] theonames = new String[offsett.size()];
String[] theorems = new String[offsett.size()];
for (int i = 0; i < offsett.size(); i++) {
      raf.seek(offsett.get(i));
      buffert = new StringBuffer();
      while ((line = raf.readLine()) != null
            && !line.startsWith("canceled")
            && !line.startsWith("definition")
            && !line.startsWith("notation")
            && !line.startsWith("registration")
            && !line.startsWith("theorem")
            &&!line.startsWith("scheme")&& line.startsWith("::")
            && !line.startsWith("begin")
```

```
            && !line.startsWith("reserve")) {
            buffert.append(line + "\n");
      }
      theonames[i] = Insert.SplitElement.splitTheorem((buffert
      .toString()).replace("'", "''"), file)[0];
      theorems[i] = Insert.SplitElement.splitTheorem((buffert
      .toString()).replace("'", "''"), file)[1];
                  }[…]
```

Die Arrays `theorems` und `theonames` werden für das Ablegen von allen `theorems` und deren Namen der gerade verarbeiteten Datei verwendet. Die Methode `splitTheorem`() aus der Klasse `splitElement` liefert ein Feld, dessen erstes Element den Namen und zweites Element den Inhalt eines gegebenen `theorems` enthält, zurück.

Im Folgenden wird zusätzlich die Zerlegung für die Spalte existential und redefine beschrieben, die jeweils in den Tabellen definitions und registrations vorkommen. Wie im Kapitel 2.1 „Speicherung der Daten“ erwähnt wurde, kommt es vor, dass einen `Definition-`, `Notation-` oder `Registration-Block` wiederum mehrere `definitions`, `notations` oder `registrations` enthält, wie das Beispiel 3.1, Seite 18, zeigt.

Der `Definition-Block` aus Beispiel 3.1 beinhaltet zwei `definitions`, die mit dem Wort `func` eingeleitet werden. Damit das KMS-MIZAR präziser funktioniert, ist es relevant, dass diese beiden `definitions` separat in der Tabelle definitions eingetragen werden. Dafür wird die ArrayList `defNo` aus Elementen vom Datentyp `Integer` angelegt und die folgende `for-Schleife` aus der Methode `splitDefinition`() der Java Klasse `SplitElement` schreibt in diese Liste eine Zeilennummer, sobald das Wort `func` gelesen wird.

```
[…]
for (int i = 0; i < buffer.length; i++) {
      if(buffer[i].contains("attr")
|| buffer[i].contains("func")
||buffer[i].contains("mode")|| buffer[i].contains("pred")
            || buffer[i].contains("struct")) {
            defNo.add(i);
}[…]
```

`Buffer` ist hierbei ein Array aus Elementen von Datentyp `String`, die eine `definition` (eine `notation` oder eine `registration`) zeilenweise beinhaltet. Anschließend werden mit Hilfe der Nummern aus `defNo`, die Zeilen ab dem ersten bis zu dem zweiten Wort `func` als erste `definition` gespeichert.

```
[…]
if (k < defNo.size() - 1) {
      for (int m = defNo.get(k); m < defNo.get(k + 1); m++) {
            if (buffer[m].contains("let")
                  || buffer[m].contains("assume")
                  || buffer[m].contains("given"))
                  break;
            def[k] += buffer[m] + "\n";
[…]
```

Dann werden die Zeilen ab dem zweiten Wort `func` bis zu der letzten Zeile des Blocks als zweite `definition` gespeichert.

```
[…]
if (k == defNo.size() - 1) {
      for (int m = defNo.get(k); m < buffer.length; m++) {
            def[k] += buffer[m] + "\n";
            }[…]
```

Im Folgenden wird die BNF des Elements `Definitional-Block` betrachtet.

```
Definitional-Block = "definition" { Definition-Item | Definition } {
Redefinition-Block } "end" .
```

```
Definition-Item = Loci-Declaration | Permissive-Assumption |
Auxiliary-Item.
```

Die Produktionsregel von dem Element `Definition-Item` sorgt dafür, dass die Schlüsselwörter `let, assume und given` innerhalb einer `definition` vorkommen können. `Let` ist in dem Element `Loci-Declaration` zu finden und `assume` und `given` in dem Element `Permissive-Assumption`. Diese Wörter leiten `definitions` ein. In dem Beispiel 3.1, Seite 18, ist es aber nicht der Fall. Bei `notations` und

registrations ist lediglich das Schlüsselwort let von Bedeutung. Beim Eintreten eines solchen Ereignises wird das Hinzufügen von Zeilen abgebrochen.

4.2.3 Loci-Declaration

Eine weitere Problemstellung, ist die Zuordnung von Loci-Declaration. Eine Loci-Declaration entspricht in gewisser Hinsicht einer Deklaration in einer höheren Programmiersprache und deklariert Variablen, die anzuwenden sind.

```
let x,y be complex number;
```

ist die Loci-Declaration des Beispiels 3.1 (siehe Seite 18) und steht lediglich am Anfang des gesamten Definition-Block. Mit dem Algorithmus, der bisher vorgestellt wurde, wäre diese Deklaration ohne weiteres für die beiden Definition-Blocks abwesend.
Um dieses Problem umzugehen, wurde eine Liste von Intergerwerte letNo angelegt, die diesmal die Zeilennummern speichert, in denen das Wort let zu finden ist. Die Schlüsselwörter given und assume sind ebenfalls zu beachten.

```
[…]
for (int i = 0; i < buffer.length; i++) {
if(buffer[i].contains("let")|| buffer[i].contains("assume")
      || buffer[i].contains("given")) {
      for (int j = i; j < buffer.length; j++) {
      letNo.add(j);
      if (buffer[j].endsWith(";"))
      break;
[…]
```

Anschließend sind die Zeilenummern aus letNo mit denen aus defNo zu vergleichen und die Loci-Declaration ist Teil einer definition, falls die Zeilennummern aus letNo kleiner als die aus defNo sind.

```
[…]
for (int j = 0; j < letNo.size(); j++) {
      if (letNo.get(j) < defNo.get(k)) {
            def[k] += buffer[letNo.get(j)] + "\n";}}[…]
```

4.2.4 Existential und Redefine

Für die Spalten existential und redefine, wird ein identischer Algorithmus angewandt. Beim Lesen einer Zeile innerhalb einer `definition` bzw. `registration`, wird geprüft, ob diese Zeile das Wort `redefine` bzw. das Zeichen `->` beinhaltet und falls es der Fall ist, wird die Variable `redefine` auf 1 bzw. die Variable `existential` auf `0` gesetzt. Die Variable `existential` wird auf `0` gesetzt, denn eine `registration`, die das Zeichen `->` besitzt, ist keine `existential cluster`, siehe Kapitel 2.1.3 „Registration“.

Programmausschnitte respektiv aus den Methoden `splitDefinition()` und `splitRegistration()`.

```
•          if (buffer[m].contains("redefine")) {
redefine = 1;[…]
```

```
•          if (buffer[m].contains("->")) {
existential = 0;[…]
```

4.2.5 Ermittlung des definierten Ausdrucks

Zur Ermittlung des innerhalb einer `definition` definierten Ausdrucks bzw. Symbols, bietet Mizar einige Sprachkonstrukte an, die umgesetzt werden können, um dies zu ermitteln. Die BNF der Syntax von Mizar besagt, dass jedes Symbol bzw. jeder Ausdruck, das zu definieren ist, verbindlich vor bestimmten Schlüsselwörtern vorkommen muss, die hier als erste Gruppe bezeichnet werden. Diese Wörter sind die folgenden:
`means, equals, ->, existence, uniqueness, coherence, compatibility, consistency, correctness, commutativity, idempotence, involutiveness, projectivity, symmetry, asymmetry, connectedness, reflexivity, irreflexivity.`
Des Weiteren muss der zu definierende Ausdruck nach den weiteren Schlüsselwörtern `mode`, `func`, `pred`, `attr` (hier als zweite Gruppe bezeichnet) stehen, wenn es jeweils um ein `mode`, einen `functor`, ein Prädikat und ein Attribut geht. Die BNF selbst wird aus Grund ihrer Komplexität und ihres Umfangs hier nicht mehr behandelt. Die Methode `getPattern()` aus der Klasse `SplitElement()` bekommt eine `definition` als Parameter und gibt die Zeichenkette, die zwischen dem Index vom dem Schlüsselwort aus der

zweiten Gruppe und dem kleinsten Index von den Schlüsselwörtern aus der ersten Gruppe, die in der `definition` vorkommen, zurück.

```
[…]
firstIndex = min(indexes);
pattern = definition.substring(kindIndex, firstIndex);
return pattern;
[…]
```

Die Liste `indexes` beinhaltet die Indizes von den Schlüsselwörtern aus der ersten Gruppe, die in der `definition` vorhanden sind, und `kindIndex` besitzt den Index des Schlüsselworts aus der zweiten Gruppe. Die Methode `min()` gibt die kleinste Zahl einer Liste von Integerzahlen zurück.
Der definierte Ausdruck in Beispiel 2.1, Seite 9, ist `func x^2.`

4.3 Einfügen von Daten in die Tabellen

Im Kapitel 4.2 „Zerlegung einer gegebenen Mizar-Datei“ wurde gezeigt, wie die Dateien aus der MML zerlegt werden. In diesem Kapitel werden die Methoden, die die einzelnen Dateielemente in die Datenbank schreiben, vorgestellt.

4.3.1 Auto-Increment und die Klasse Connector

In einem Projekt mit relationalen Datenbanken ist das AUTO-INCREMENT von großer Bedeutung.
Das RDBMS Oracle implementiert kein Attribut AUTO_INCREMENT wie es beispielsweise bei dem RDBMS MySQL der Fall ist. Die Instruktion AUTO_INCREMENT ermöglicht es, die Werte einer Spalte, deren Datentyp ganzzahligen Zahlen entspricht, automatisch hoch zu zählen. Diese Eigenschaft ist besonders wichtig, wenn die Werte für eine Spalte, die den Primärschlüssel bildet, automatisch generiert werden. Darum wurde diese Eigenschaft erstmal programmiert.
Innerhalb der RDBMS von Oracle ist es notwendig für jede Tabelle unserer Datenbank eine eigene Sequenz und einen Trigger anzulegen, die diese AUTO_INCREMENT Eigenschaft implementieren.

Das folgende Skript ist für die Tabelle absFiles

```
create sequence file_seq
start with 1
increment by 1
nomaxvalue;
```

```
create trigger file_trig
before insert on absFiles
for each row
begin
select file_seq.nextval into :new.fileID from dual;
end;
```

`file_trig` ist der Trigger und `file_seq` die Sequenz für die Tabelle absFiles.

Ein ebenfalls relevanter Punkt sind die Datenbankverbindungen.

Ein häufiger Aufbau und Abbau von Datenbankverbindungen sollte aus Performance-Gründen vermieden werden. Obwohl das Insert-Programm des KMS-MIZAR so implementiert worden ist, dass alle `definitions` (`notations`, `registrations`, `theorems`) einer Datei zusammengefasst und darauf durch eine einzelne Datenbankverbindung in die Datenbank geschrieben werden, kann das Programm, dadurch dass für jede Insert-Anweisung ein Verbindungsaufbau- und Abbau notwendig ist, in der Praxis zum Stillstand kommen.. Dies kommt aus dem Grund vor, dass der Treiber nicht mehr in der Lage ist, alle vorliegenden Datenbankverbindungen zu verwalten.

Es ist aber zu gewährleisten, dass das System vor dem Absturz bewahrt wird.

Um dieses Ziel zu erreichen, wurde die Klasse `Connector`, die den benötigten Algorithmus über das Singleton-Pattern[13] implementiert.

```
public synchronized static  Connection getConnection()
 throws SQLException {
          if (instance == null) {
```

[13] Ein Singleton ist ein Entwurfmuster, das sicherstellt, dass von einer Klasse zu jeder gegebenen Zeit nur ein Exemplar existieren kann.

```
                new Connector();
            }
            return instance;
      }
```

Die Methode `getConnection()` sorgt dafür, dass eine Datenbankverbindung aufgebaut werden kann, wenn keine andere bereits existiert, wobei `instance`, eine Referenz der Klasse `Connection` ist. Andererseits gibt es die Methode `disconnect()`, die die Verbindung beendet und `instance` auf `null` setzt.

```
public static void disconnect() {
            try {
                  instance.close();
                  instance = null;
            } catch (SQLException e) {

      System.out.println("The attempt to connect to the database
failed!");
      System.out.println(e.getMessage());
                  }
      }
```

4.3.2 Die Insert-Befehle

Die Methode `insertAbsFiles` füllt für eine gegebene Mizar-Datei die Tabelle absFiles mit den erforderlichen Werten auf. Vorher muss sichergestellt werden, dass die Datei noch nicht in der Datenbank vorhanden ist. Erst wenn die Datei nicht vorhanden ist, wird die Tabelle aufgefüllt. Dies ist in dem folgenden Programmabschnitt zu sehen. `result.next()` gibt `true` zurück, falls die Datei bereits vorhanden ist.

```
[…]
sql1 = "insert into absFiles (filename, description, defno, notno,
regno,      theono, locked) values('"
                        + filename
                        + "','"
                        + description
                        + "',"
```

```
                        + defno
                        + ","
                        + notno
                        + "," + regno + "," + theono + ", " + 0 + ")";
sql2 = "select * from absFiles where filename = '" + filename + "'";
try {
      Connection con = Connector.getConnection();
      Statement st;
      ResultSet result;
      st = con.createStatement();
      result = st.executeQuery(sql2);
      if (!result.next()) {
            st.executeUpdate(sql1);
            } else {
[…]
```

Die Algorithmen für den Eintrag einer Zeile in die Tabellen definitions, notations und registrations sind im größten Teil ähnlich zueinander, mit dem Unterschied, dass in der Tabelle notations die zusätzliche Spalte redefine bzw. existential nicht existiert.
Da eine Definition aus einer Datei stammt, muss der Primärschlüssel dieser Datei (PRIK aus der Tabelle absFiles) als Fremdschlüssel in der Spalte fileID der Tabelle definitions stehen. Um die restlichen Werte der Tabellen definitions zu erhalten, soll folgender Codeausschnitt aus der Methode `insertFile()` betrachtet werden.

```
[…]
String[] definitions = new String[counterd];
// For the entire file
int[] redefine = new int[counterd];
String[] patterns = new String[counterd];
String[] defNames = new String[counterd];
int countd = 0;
for (int i = 0; i < containerd.size(); i++) {
      // def only for a definition of the file
      String[] def = ((String[]) (SplitElement.splitDefinition(
      containerd.get(i).toString(), file).get(0)));
      int[] red = ((int[]) (SplitElement.splitDefinition(containerd
      .get(i).toString(), file).get(1)));
```

```
String[] pattern = ((String[]) (SplitElement.splitDefinition(
containerd.get(i).toString(), file).get(2)));
String[] defName = ((String[]) (SplitElement.splitDefinition(
containerd.get(i).toString(), file).get(3)));
for (int k = 0; k < def.length; k++) {
      definitions[countd] = def[k];
      redefine[countd] = red[k];
      patterns[countd] = pattern[k];
      defNames[countd] = defName[k];
      countd++;
      }[…]
```

counterd ist eine Variable, die die tatsächliche Anzahl von definitions aus einer Datei enthält. containerd ist eine Liste vom Typ ArrayList<Object> mit den gesamten Definition-Blocks einer Datei. Die Methode splitDefinition() besitzt als Rückgabewert eine Liste vom Typ ArrayList<Object>, die vier Felder enthält, die jeweils die definitions, die Werte für die Spalte redefine, die definierten Ausdrücke, und die Definitionsnamen beinhalten. Diese Methode splittert einen gegebenen Definition-Block gegebenenfalls in kleineren definitions, die in der Hilfsvariablen def gespeichert werden. Die Werte für die Spalten redefine, pattern und defName werden respektiv in den Hilfsvariablen red, pattern und defName gespeichert. Um jetzt die gesamte Datei zu bearbeiten, werden die Werte die in den Hilfsvariablen vorliegen nach und nach in den Variablen definitions, redefine, patterns und defNames gespeichert.
Für einen Schreibvorgang werden die Variablen definitions, redefine, patterns und defNames der Methode insertDefinitions() aus der Klasse InsertMethods übergeben.

Ausschnitt von insertDefinitions():

```
[…]
sql1 = "select fileid from absFiles where filename = '" + filename+
"'";
try {
```

```
      // building of the connection instance using the method
connect()
      Connection con = Connector.getConnection();
      Statement st;
      ResultSet result;
      st = con.createStatement();
      result = st.executeQuery(sql1);
      if (result.next()) {
            fileID = result.getInt("fileid");// gets the primary key
of the file
      }
      for (int i = 0; i < definitions.length; i++) {
            definitions[i] = "definition\n" + definitions[i] +
"end;";
            sql2 = "insert into definitions (fileid, text, redefine,
pattern, defName) values("+ fileID+ ",'"
+ definitions[i]+ "',"+ redefine[i]+ ",'" + patterns[i] + "','" +
defNames[i] + "')";
            st.executeUpdate(sql2);
      }
      Connector.disconnect();[…]
```

Es ist weiterhin zwei Methoden zu besprechen. Es handelt sich um die Methoden `blockInsertion()` und `isLocked()` aus der Java Klasse `InsertMethods`.
Sobald eine Datei in das System eingetragen wurde, soll ein weiteres Einfügen derselben Datei gesperrt werden. Um dies zu erreichen, setzt die Methode `blockInsertion()`, jedes Mal wenn eine Datei eingefügt wird, den Wert der Spalte locked der Tabelle absFiles auf `1`.

Ausschnitt von `blockInsertion ()`:

```
[…]
sql = "update absFiles set locked = " + 1 + " where filename = '"+
filename + "'";
try {
      Connection con = Connector.getConnection();
      Statement st;
```

```
      st = con.createStatement();
      st.executeUpdate(sql);[…]
```

Im Anschluss wird vor dem jedem Einfügen einer `definition` (einer `notation`, einer `registration` oder eines `theorem`) die Methode `isLocked()` aufgerufen, die die Zeichenkette `yes` zurückgibt, falls ein Einfügen nicht mehr möglich ist bzw. wenn die Spalte `locked` den Wert `1` aufweist.

Ausschnitt von `isLocked()`:

```
[…]
sql = "select locked from absFiles where filename = '" + filename +
"'";

try {
      Connection con = Connector.getConnection();
      Statement st;
      ResultSet result;
      st = con.createStatement();
      result = st.executeQuery(sql);
      if (result.next()) {
            if ((result.getInt("locked") == 0)) {
                  isLocked = "no";
            } else {
                  isLocked = "yes";
[…]
```

4.4 Das UPDATE des KMS-MIZAR

Das UPDATE des KMS-MIZAR ist nicht mit einem herkömmlichen SQL UPDATE Statement zu verwechseln, in dem das Kommando UPDATE benutzt wird. Da da System abhängig von den Änderungen an der MML aktualisiert werden soll, muss genau betrachtet werden, wie diese Änderungen zustande kommen. Die aktuelle Version der MML ist 4.110.1033. Diese Version ist auf der Webseite ftp://mizar.uwb.edu.pl/pub/version/mml.ini zu entnehmen. Eine Änderung der linken Versionsnummer (1033) weist darauf hin, dass eine Datei zu der MML hinzugefügt wurde. In diesem Fall kann diese mit der implementierten Insert-Methode in die DB-Tabellen geschrieben werden. Wenn sich hingegen die mittlere

Versionsnummer ändert, bedeutet dies, dass einige Dateien in der MML umgeschrieben wurden.

Da viele Dateien in der MML aufeinander bauen und um sicher zu stellen, dass diese Abhängigkeit nicht verloren geht, sollte die gesamte MML (hier der gesamte Inhalt des Ordners *c:\mizar\abstr*) neu in die Datenbank eingefügt werden. Wie oben bereits erwähnt, kann eine Datei lediglich einmal eingespielt werden.

Für ein erfolgreiches UPDATE müsste vorher die gesamte Datenbank geleert werden. Damit die Primärschlüssel wieder von 1 ausgehend hoch gezählt werden, müssen ebenfalls die schon angelegten Sequenzen gelöscht und neu angelegt werden. Dies wird in der Methode `delete()` aus `InsertMethods` implementiert.

Die Methode `delete()` wird von der Methode `update()` aus `InsertUpdate` aufgerufen.

```
public static void update() {
            InsertMethods.delete();
            insertAllFiles();
      }
```

Die Methode `update()` ruft ebenfalls die Methode `insertAllFiles()` auf.

```
public static void insertAllFiles() {
            String directoryPath = "C:\\mizar\\abstr";
            File[] subFiles = null;
            subFiles = SplitElement.splitDirectory(directoryPath);
            for (int i = 0; i < subFiles.length; i++) {
                  insertFile(subFiles[i].getName().substring(0,
                              subFiles[i].getName().length() - 4));
            }
```

Die Methode `insertAllFiles()` ruft seinerseits die Methode `splitDirectory()` aus `SplitElement` auf, die alle Dateien eines Ordners als Rückgabewert hat, und darauf wird für jede dieser Dateien die Methode `insertFile()`, die die Datei letztendlich in die Datenbank schreibt, aufgerufen.

5 Select-Programm des KMS-MIZAR

Das Select-Programm des KMS-MIZAR ist eine Suchfunktion, die den Benutzern des Systems dabei unterstützt, unterschiedliche Arten von Informationen über Mizar Begriffe zu finden. Diese Informationen können während des Editierens einer Mizar-Datei genutzt werden.

5.1 Zweites Übersichtsdiagramm

Das hier aufgeführte Modul-Übersichtsdiagramm dokumentiert das Aufrufverhalten der Module des Select-Programms des KMS-MIZAR. Das Select-Programm stellt die Komponente (5) aus der Systemstruktur, Seite 16, dar.

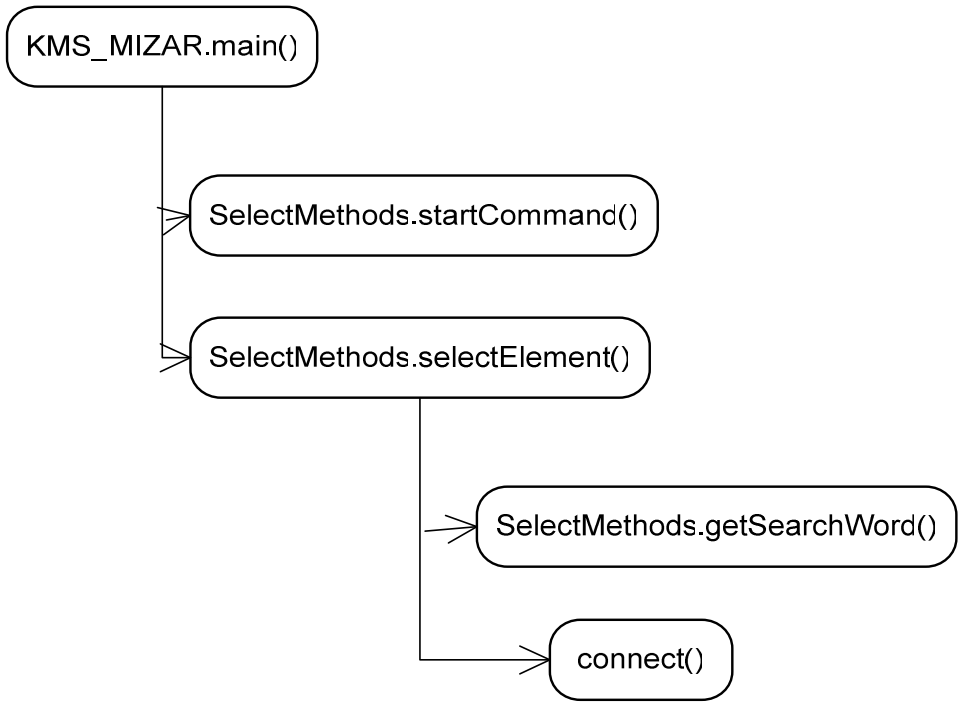


Abbildung 5.1 Übersichtsdiagramm des Select-Programms

5.2 Das findVoc Kommando des Mizar Systems

Jedes Mal wenn eine Mizar-Datei verfasst wird, gibt es eine Menge von Symbolen (`predicate`, `functor`, `attribute` oder `mode`), die benutzt werden. Für jedes dieser Symbole muss eine relevante Referenz in der Direktive `vocabularies` angeben werden. Mit der Referenz wird der Name einer Datei aus der MML gemeint. Das `findVoc` Kommando ermöglicht zu bestimmen, welche Dateinamen in der Direktive `vocabularies` einzutragen sind. Die Methode `startCommand`() aus der Klasse `selectMethods` ruft

das `findVoc` Kommando auf. Es soll als Beispiel für die Benutzung dieses Kommandos eine Datei geschrieben werden, in der der Operator[14]
`,-'` (minus) vorkommt. Die Eingabe der Zeichenkette `-w -` in das Programmsystem, liefert die folgenden Zeilen zurück:

```
FindVoc, Mizar Ver. 7.9.03 (Win32/FPC)
Copyright (c) 1990-2008 Association of Mizar Users
vocabulary: ARYTM_1
O-  32
```

Die ersten beiden Zeilen beinhalten die Systeminformationen, Versions- und Copyrighthinweise der Software Mizar. Diese sind für die Arbeit des Benutzers nicht von Relevanz. Die dritte Zeile gibt Information darüber, welcher Dateiname (in diesem Fall `ARYTM_1)` in der Direktive `vocabularies` zu schreiben ist. Das `O` in der letzten Zeile ist ein `Symbol Qualifier` und steht für `functor`. Hier sind die weiteren `Symbol Qualifiers`:

```
R-Predicate
M-Mode
G-Structure
U-Selector
V-Attribute
K-Left Functor Bracket
L-Right Functor Bracket
```

Das `findVoc` Kommando kann aber deutlich mehr leisten und von großer Hilfe für Einsteiger in Mizar sein. Unter Benutzung von `findVoc` kann ein beliebiger Begriff aus der Mathematik eingegeben und abgefragt werden, ob dieser Begriff in Mizar existiert. In dieser Art der Benutzung des `findVoc` sollte die Zeichenkette `-w` ausgelassen werden. Die Ergebnisliste wird alle Treffer beinhalten, die ausschließlich oder teilweise aus dem eingegebenen Begriff bestehen.

Es soll als Beispiel die Ergebnisliste für den Begriff `vector` betrachtet werden.

```
FindVoc, Mizar Ver. 7.9.03 (Win32/FPC)
```

[14] Ein Operator bekommt den Namen `functor` in Mizar.

```
Copyright (c) 1990-2008 Association of Mizar Users
vocabulary: POLYFORM
Oalternating-f-vector
Oproper-f-vector
Oalternating-semi-proper-f-vector
Oalternating-proper-f-vector

vocabulary: PRGCOR_2
Rvector_minus_prg
Rvector_add_prg
Rvector_sub_prg

vocabulary: RLVECT_2
Ovector  128

vocabulary: VECTSP11
Meigenvector.
```

5.3 Verarbeitung und Bewertung der Benutzereingabe

In dem Kapitel 3 „Aufbau der Datenbank des KMS-MIZAR“ wurde das Datenbank-Schema und die Vorbelegung mit allen notwendigen Daten beschrieben. Damit wurde einen entscheidenden Meilenstein in der Realisierung des Systems erreicht. Das KMS-MIZAR ist jetzt in der Lage, Benutzereingabe entgegenzunehmen, zu verarbeiten und konsequenterweise mit einer Liste von Treffern zu reagieren.
Im Folgenden wird die Notation einer gültigen Eingabe erklärt.

5.3.1 Allgemeine Eingabe

Die Methode `selectElement()` aus der Klasse `selectMethods` implementiert den Algorithmus für eine effektive Ergebnisliste. Mit dem folgenden Programmcode aus der Methode `selectElement()` wird den Benutzer dazu aufgefordert, ein Element auszuwählen.

```
[…]
choice = IO1.einint();
if (choice == 1) {
```

```
      table = "definitions";
} else if (choice == 2) {
      table = "notations";
} else if (choice == 3) {
      table = "registrations";
} else if (choice == 4) {
      table = "theorems";
} else if (choice == 5) {
      table = "absfiles";
} else {
      System.out.println("Please, choose a number from 1, 2, 3, 4
before going on.");
      System.out.println("1. Definition");
      System.out.println("2. Notation");
      System.out.println("3. Registration");
      System.out.println("4. Theorem");
      System.out.println("5. Article");
      }
} while (choice < 1 || choice > 5);[…]
```

Nachdem eine Zahl zwischen 1[15] und 5 (1und 5 inklusiv) eingegeben worden ist, soll der Benutzer im Anschluss mögliche Teile des Textes eingeben, wonach er sucht. Um eine mehr effektive und damit kürzere Ergebnisliste zu bekommen, sollte der eingegebene Textabschnitt möglichst gut mit dem Text, der vermutet wird, in irgendeiner Datei der MML vorhanden zu sein, übereinstimmen. Da es lediglich mit als vorhanden vermuteten Textabschnitten gearbeitet wird, muss dafür gesorgt werden, dass der restliche Teil des Textes mitberücksichtigt wird. Dies kann mit dem folgenden Ausschnitt aus `selectElement()` ermöglicht werden.

```
text = text.replace("xy", "%");
```

Wobei die Variable `text` der insgesamt angegebener Text darstellt. Die Zeichenkette `xy` steht für alle Textabschnitte, die man nicht kennt und diese unbekannte Textabschnitte werden

[15] 1 steht für eine Definition; 2steht für eine Notation; 3 steht für eine Registration; 4 steht für ein Theorem; 5 für allgemeine Informationen über eine Mizar-Datei.

anschließend mit dem Prozentzeichen ersetzt, das in der SQL-Syntax eine beliebige Zeichenkette repräsentiert.
Im Folgenden wird nach einer `registration` gesucht, die die Zeichenketten ‚`complex`' und ‚`REAL`' beinhaltet, dann wird der einzugebende Befehl wie folgt aussehen:

```
complexxyREAL
```

Nach der Eingabe des gesuchten Textes, wird der Select-Befehl gebildet.

```
do {
      System.out
      .println("Type 1 if you want the registration to be
existential, otherwise type 0");
      existential = IO1.einint();
       while (existential!= 1 && existential!= 0);
sql = "select fileID, text from " + table + " where text like '"+
"%" + text + "%" + "' and existential = " + existential;
[…]
```

Der oben stehende Code gilt für den Fall einer `registration`. Die String-Variable `sql` bildet eine Anfrage, die alle `registrations`, die alle eingegebenen Textabschnitte enthalten, zurückliefert. Des Weiteren, soll der Benutzer entscheiden, ob die zu ausgebenden `registrations` von der Art `existential` sein sollen oder nicht. Diese Angabe ist relevant, weil es konsequent die Ergebnisliste minimiert, was andererseits eine bessere Übersicht über die Ergebnisliste ermöglicht. Je mehr Begriffe eingegeben werden, desto geringer wird die Ergebnisliste, sofern die gewählte Kombination der Begriffe in Mizar vorkommt. Im Fall einer `definition` wird bei der Bildung der Select-Anfrage `existential` durch `redefine` ersetzt. Diese Angabe (`existential` bzw. `redefine`) entfällt sowohl für `notations` als auch für `theorems`.
Eine wichtige Aufgabe der Klasse `SelectMethods` ist es, den Namen der Mizar-Datei, in der das gesuchte Element (`definition`, `notation`, `registration`, `theorem`) enthalten ist, auszugeben. Dieser Name könnte beispielsweise in der entsprechenden Direktive in dem `Environment-Declaration` der geschriebenen Datei eingetragen werden. Der Dateiname kann ermittelt werden, nachdem der folgende Codeausschnitt ausgeführt wird und steht unmittelbar oberhalb eines jeden Treffers.

```
[…]
result = st.executeQuery(sql);
while (result.next()) {
      Statement st1;
      st1 = con.createStatement();
      ResultSet res;
      int fileID = result.getInt("fileID");
      String sql1 = "select fileName from absfiles where fileID = "
      + fileID;
      res = st1.executeQuery(sql1);[…]
```

In dem Fall das ein bestimmtes `theorem` angezeigt werden soll, ist es ausreichend, nachdem das entsprechende Unterprogramm gewählt wurde, den Namen des `theorem`s in das System einzugeben. Dies wird durch die folgende SQL-Anfrage ermöglicht:

```
text = IO1.einstring();
text = text.replace("xy", "%");
sql = "select fileID, text, theoname from " + table
            + " where text like '" + "%" + text + "%" + "'"; […]
```

Wobei die Variable `table` den Wert `theorems` zugewiesen bekommt und `text` der eingegebene Theoremname ist. Der Name eines `theorem`s wird folgendermaßen gebildet.
`,Dateiname' + ,:' + ,Zahl'` (mit Zahl ∈ IN\{0}).
Beispiel: `ABIAN:5`
Mizar behandelt einige weit bekannte Themen der Mathematik. Eine Möglichkeit herauszufinden, welche Themen in Mizar behandelt sind und welche nicht, ist es sich die Datei *c:\mizar\doc* des Mizar Systems durchzulesen. Die andere Möglichkeit besteht darin das Select-Programm des KMS-MIZAR zu nutzen. Wird beispielsweise das Wort `,Riemann'` eingegeben, werden im Anschluss die folgenden drei Treffer ausgegeben:

- `The following match is in the file INTEGRA1`

```
:: The Definition of Riemann Definite Integral and some :: Related
Lemmas
:: by Noboru Endou and Artur Korni{\l}owicz
```

- `The following match is in the file INTEGRA2`

```
:: Scalar Multiple of Riemann Definite Integral
::  by Noboru Endou , Katsumi Wasaki and Yasunari Shidama
```

- The following match is in the file INTEGRA7

:: Riemann Indefinite Integral of Functions of Real
:: Variable
:: by Yasunari Shidama , Noboru Endou , Katsumi Wasaki ::and Katuhiko Kanazashi

5.3.2 Spezialfall einer Definition

Im Fall einer definition ist die Vorgehensweise für eine sinnvolle Eingabe anders. In der Tat wird eine definition durch einen Ausdruck charakterisiert. Dieser Ausdruck ist der, dessen definition gemacht wird und liegt in der Spalte pattern der Tabelle definitions vor.

Anders als bei allen anderen Elementen wird der eingegebene Text, wenn er ohne spezielle Vorgabe eingegeben ist, sowohl mit dem Wert der Spalte text als auch dem Wert der Spalte pattern verglichen. Das heißt, man sollte im Kopf behalten, dass der eingegebene Text ein Begriff bzw. ein Ausdruck ist, nach dessen definition gesucht wird.

Hier ist die SQL-Anfrage:

```
sql = "select fileID, text, pattern , defName from " + table
          + " where text like '" + "%" + text + "%"
          + "' and pattern like '" + "%"+searchWord+"%" + "' and
redefine = "
          + redefine;
```

Sollen zu dem gesuchten definierten Begriff zusätzliche Angabe gemacht werden, um die Ergebnisliste einzuschränken, dann soll den definierten Begriff mit der Zeichenkette ,ww'(wwgesucheterBegriffww) eingeschlossen werden. Somit kann durch den Aufruf der Methode getSearchWord() den definierten Begriff ermittelt werden. Dann wird der mit ,ww' eingeschlossene Begriff mit dem Inhalt der Spalte pattern und nochmals die gesamte Eingabe mit dem Inhalt der Spalte text verglichen.

Wird nach einer definition des Operatoren ,+' für Komplexe Zahlen gesucht, dann müsste die Eingabe wie folgt aussehen:

```
complex numberxyww+ww
```

wobei `complex number` für komplexe Zahlen steht. Die Angabe von zusätzlichen Begriffen ist immer empfohlen, da sie konsequent die Trefferanzahl reduziert. Es sollte jedoch, eine derartige Kombination der Begriffe in Mizar vorkommen. Es ist ebenfalls darauf aufmerksam zu machen, dass die Reihenfolge in der Angabe der Begriffe nicht willkürlich ist. So könnte es sein, dass die Eingabe von
`complex numberxyww+ww` unterschiedliche Treffer generieren als die Eingabe von `number complexxyww+ww.`
Bisher wurde hauptsächlich darüber dokumentiert, wie das KMS-MIZAR dabei helfen kann, bestimmte Elemente in der MML zu finden. Eine andere Funktionalität des KMS-MIZAR ist, einen Beweisrahmen für ein gegebenes `theorem` zu erzeugen.

6 Erzeugung eines Beweisrahmens

In diesem Kapitel werden zunächst einige Grundlagen über XML angegeben, da `theorems`, für die einen Beweisrahmen zu erzeugen ist, aus XML-Dokumenten hervorgehen. Im Anschluss darauf werden sowohl die unterschiedlichen Arten von Beweisschritte als auch die Algorithmen zum Erzeugen von Beweisrahmen vorgestellt.

6.1 DTD und der SAX-Parser

6.1.1 Document Type Definition DTD

Eine DTD (Document Type Definition) legt für eine Klasse strukturgleicher XML-Dokumente den Aufbau fest. Dies heißt eine Grammatik, die es ermöglicht, die Validität eines XML-Dokuments zu prüfen. Die Benutzung einer DTD ist für ein XML-Dokument jedoch optional, solange es nur auf die Wohlgeformtheit des Dokuments ankommt. Folgenderweise wird man von einem wohlgeformten Dokument sprechen, wenn dieses Dokument bezüglich der XML-Syntax korrekt aufgebaut ist. Eine Regel dieser Syntax ist zum Beispiel, dass jeder öffnender Tag einen entsprechenden schließenden Tag besitzen muss. Ein Dokument ist zusätzlich valide, wenn sich dies an die in der DTD enthaltene Grammatik hält.

Für die Erzeugung von Beweisrahmen werden die `theorems` als XML-Dokumente verfasst. Die Grammatik dieser Dateien ist in der Datei mizarDTD.dtd (siehe Seite 67) enthalten. Diese DTD wurde aus der Backus-Naur Form von Mizar hergeleitet. Damit der Endbenutzer des KMS-MIZAR weiterhin eine Übersicht über die von ihm geschriebenen XML-Dokumente behält, wurden einige Produktionsregeln aus der BNF ausgelassen. Im Folgenden werden einige von den sechs und zwanzig restlichen Produktionsregeln beschrieben.

Das Wurzelelement eines XML-Dokument ist `section` mit der folgenden Produktionsregel

```
<!ELEMENT section (article,text-item)>,
```

wobei `article` der Name der später erstellten Mizar-Datei ist. Eine externe DTD wird am Anfang eines XML-Dokuments in der sog Document Type Declaration wie folgt untergebracht:

```
<!DOCTYPE section SYSTEM "mizarDTD.dtd">.
```

Hinter dem Schlüsselwort `SYSTEM` ist die URL[16] der DTD anzugeben. Ein anderes Element ist `theorem`:

```
<!ELEMENT theorem (compact-statement)>.
```

[16] URL:= Uniform Resource Locator

Das Element `theorem` umschließt das eigentliche `theorem`, für das ein Beweisrahmen zu erzeugen ist. Des Weiteren gibt es die Elemente `formula-expression`, `atomic-formula-expression` und `quantified-formula-expression`. Die Tatsache, dass es eine gewisse Rekursivität in der Produktionsregel dieser drei Elemente vorliegt, erschwert die Erzeugung eines Beweisrahmens für ein Dokument aus beliebig verschachtelten Elementen. Um also zu gewährleisten, dass der erzeugte Beweisrahmen korrekt ist, wurde der Grad der Komplexität des Aufbaus einer Datei beschränkt, worauf in den folgenden Unterkapitel(n) eingegangen wird.
Alle Blätter haben als Inhalt eine Zeichenkette und besitzen außerdem ein Attribut, das ermöglicht während der Implementierung des SAX-Parsers, Blätter von Nicht-Blattelementen zu unterscheiden.

6.1.2 Der SAX-Parser

Der SAX[17]-Parser ist ein XML-Parser. Anders als bei dem DOM[18]-Parser, arbeitet der SAX-Parser ereignisorientiert. Bei der Erstellung eines SAX-Parsers sind die beiden Java Interfaces `ContentHandler` und `ErrorHandler` zu implementieren. Der SAX-Parser des KMS ist in der Klasse `MIzarParser` implementiert. Es sind unter anderem die folgenden fünf Ereignisse, die bei der Verarbeitung eines XML-Dokuments eintreten und für jedes dieser Ereignisse definiert das Interface `ContentHandler` eine Methode. Das erste Ereignis ist der Dokumentanfang mit der zugehörigen Methode `startDocument`(). Hier sind alle Aktivitäten auszuführen, die zu Beginn eines XML-Dokuments notwendig sind. Dann folgt das Ereignis Elementanfang mit der Methode `startElement`(). Die Methode `endElement`() ist aufzurufen, wenn ein öffnender Tag geschlossen wird. Die Methode `characters`() gibt die Zeichenfolge des Nutzdaten eines XML-Elements von dem Typ (`#PCDATA`[19]) zurück. Zu aller letzte wird die Methode `endDocument`() aufgerufen, wenn das Ende des Dokuments erreicht wird.
Das Interface `ErrorHandler` definiert einige Schnittstellen für die Behandlung von Ausnahmen wie `ParserConfigurationException`, `SAXException`,

[17] SAX:=Simple Api for Xml

[18] DOM:= Dokument Object Model. Der DOM-Parser ist auch ein XML-Parser.

[19] Ein XML-Element ist vom Typ (#PCDATA), wenn es als Inhalt eine Zeichenkette hat.

`SAXParserException`, die während der Verarbeitung eines XML-Dokuments eintreten können. Wenn der Parser das Ende des XML-Dokuments erreicht hat, wird die Methode `genProof()` aufgerufen, die den Beweisrahmen für das verfasste `theorem` generiert.

Ausschnitt von der Klasse `MyContentHandler`:

```
proof = mizHelpGen.genProof(theorem);
theorem += "\r\nproof\r\n" + proof + "end;";
try {
            raf = new RandomAccessFile(dsn, "rw");
            raf.setLength(0);
            raf.writeBytes("environ\r\n");
if (vocabularies.length() > 15)
                  raf.writeBytes(vocabularies + "\r\n");
            raf.writeBytes(vocabularies + "\r\n");
            raf.writeBytes(requirements + "\r\n");
            raf.writeBytes("begin\r\n");
            raf.writeBytes(reservation);
            raf.writeBytes(theorem);
            raf.close();
      } catch (IOException ex1) {
System.out.println("An error has occured when opening/writing "+ dsn
+ " : " + ex1.toString());
      }[…]
```

Anschließend wird eine Datei mit der Endung *.miz* erzeugt, in dem der Beweisrahmen eingebunden wird.
Während des Parsens der XML-Datei ist eine Überprüfung der Schlüsselwörter relevant, da diese Aufgabe von dem Benutzer zu erledigen ist. Um diese Funktionalität anzubieten, überprüft der SAX-Parser des KMS-MIZAR bei Aufruf der Methode `startElement()`, ob das aktive Element ein Schlüsselwort ist. Wenn das aktive Element ein Schlüsselwort ist, wird die Variable `key` auf `1` gesetzt.

Ausschnitt aus `startElement()`:

```
for (int i = 0; i < keywords.length; i++) {
```

```
      if (qName.equals(keywords[i].trim()))
            key = 1;
}[…]
```

keywords ist ein Array von String, das alle mögliche Schlüsselwörter beinhaltet.

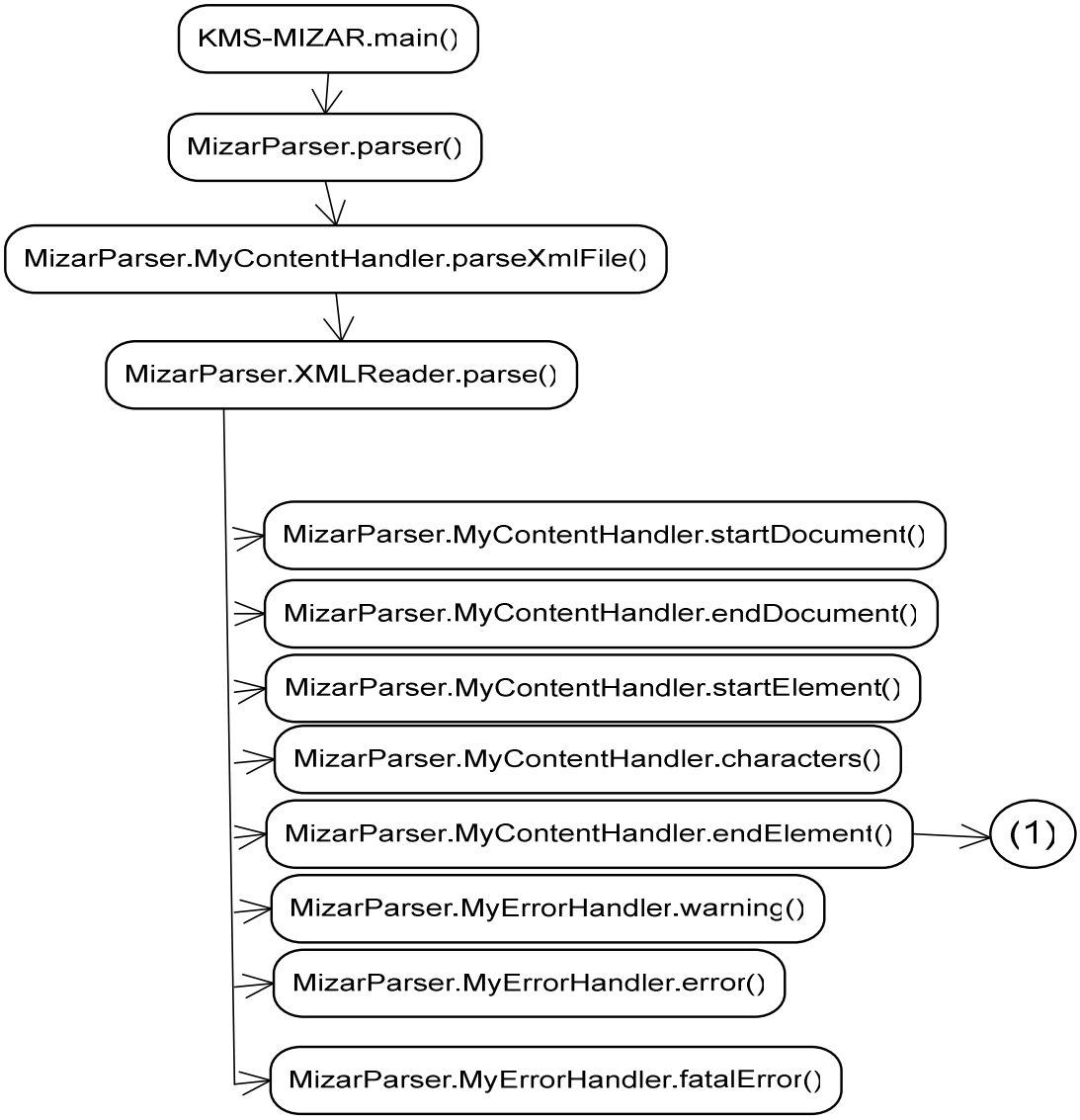
Beim Schließen des XML-Tags bzw. dem Aufruf der Methode endElement() wird überprüft, ob das eingetippte Schlüsselwort zulässig ist. Im Falle eines ungültigen Schlüsselworts wird eine Fehlermeldung in die Datei conList.txt geschrieben.

```
if (key == 1) {
      if (!qName.equals(actValue.trim())) {
            errorMessages += "The keyword " + actValue
                  + " is not correctly spelled\r\n";
      }
      key = 0;
}[…]
```

6.2 Der Beweisrahmen

6.2.1 Übersichtsdiagramm 3

Das hier aufgeführte Modul-Übersichtsdiagramm dokumentiert das Aufrufverhalten der Module, die zuständig für die Erzeugung von Beweisrahmen sind. Die Module in diesem Diagramm bilden die Komponente (10) aus der Systemstruktur, Seite 16.


KMS-MIZAR.main()
MizarParser.parser()
MizarParser.MyContentHandler.parseXmlFile()
MizarParser.XMLReader.parse()
MizarParser.MyContentHandler.startDocument()
MizarParser.MyContentHandler.endDocument()
MizarParser.MyContentHandler.startElement()
MizarParser.MyContentHandler.characters()
MizarParser.MyContentHandler.endElement()
(1)
MizarParser.MyErrorHandler.warning()
MizarParser.MyErrorHandler.error()
MizarParser.MyErrorHandler.fatalError()

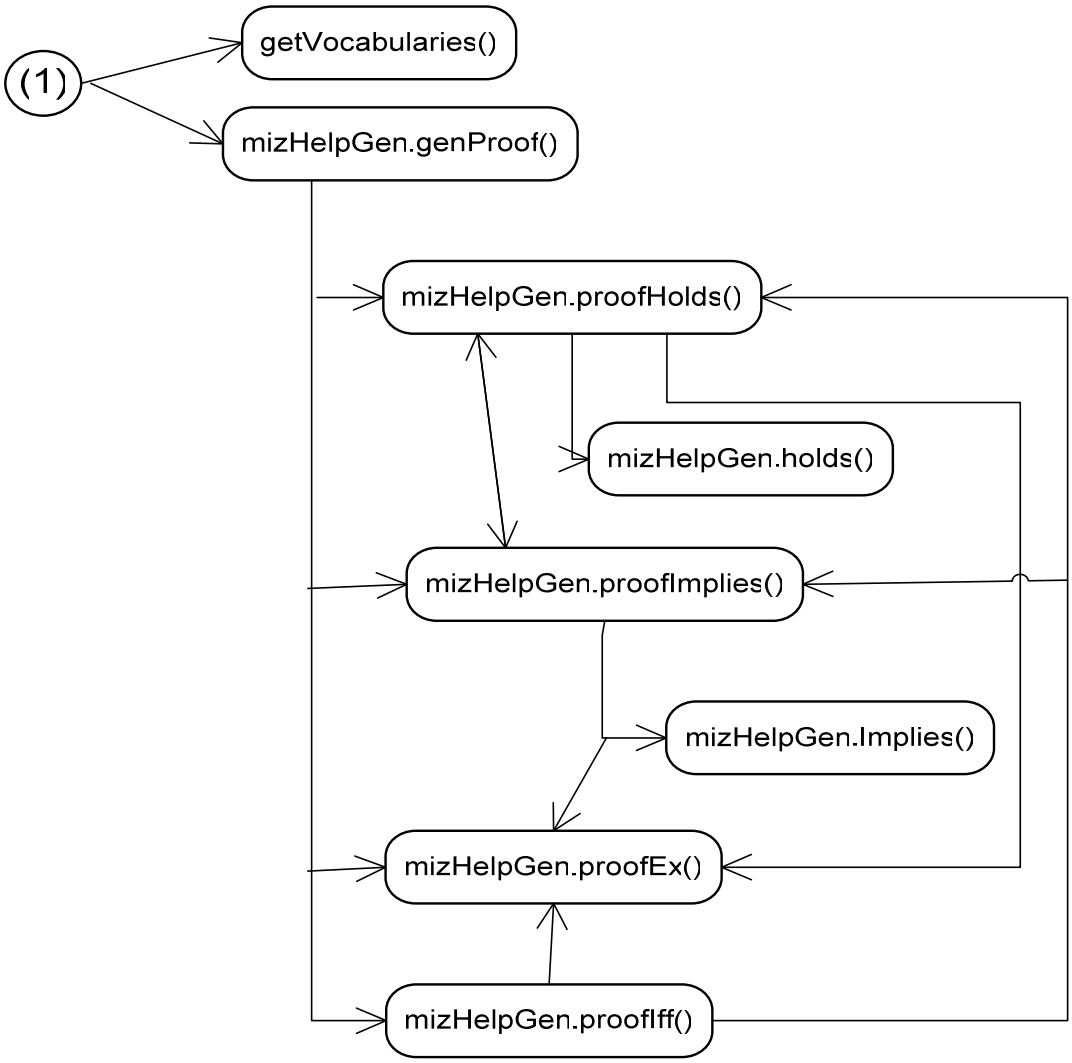


Abbildung 6.1 Übersichtsdiagramm des Beweisrahmensgenerators

6.2.2 Die Erzeugung des Beweisrahmens

In dem Codeausschnitt auf Seite 52 wird die Methode `genProof`() aufgerufen, in der je nach Aufbau des `theorems` ein Beweisrahmen erstellt wird. Ein Beweis besteht aus Beweisschritten, sog. `Steps`. Mizar verfügt über eine Reihe von vordefinierten Beweisschritten, die eingesetzt werden können. Da es in dieser Arbeit lediglich um die Erstellung eines Beweisrahmens geht, wird der Umfang der Arbeit auf vier wesentliche Beweisschritte eingeschränkt. Diese sind `assume`, `thus`, `let` und `take`. Die sind wie folgt zu interpretieren:

Zu beweisender Behauptung(vorher)	Der Beweisschritt	Zu beweisender Behauptung(nachher)
A) ϕ implies Ψ	assume ϕ	Ψ
B) ϕ & Ψ	thus ϕ	Ψ
C) for x being ζ holds Ψ	let x be ζ	Ψ
D) ex x being ζ st Ψ	take t	Ψ [x:=t]

Tabelle 6.1 Vier wesentliche Beweisschritte

Da die Produktionsregel für manche Mizar-Elemente rekursiv ist, war die Frage zu wissen, ob diese Beweisschritte, im Falle einer automatisierten Generierung des Beweisrahmens entsprechend oft eingesetzt werden können. Die Antwort ist nein. In einem `theorem`, in dem das Schlüsselwort `implies` mehrfach auftaucht, reicht es nicht aus, dem Ausdruck links von `implies` das Wort `assume` voranzustellen und dem Ausdruck rechts von `implies` das Wort `thus` voranzustellen.
Es soll das folgende Beispiel betrachtet werden:

```
a implies b implies c implies d.
```

Intuitiv könnte diese Behauptung folgendermaßen bewiesen werden:

```
assume a
  thus assume b
  thus assume c
thus d.
```

Dies ist aber nicht richtig. Daher wurde festgelegt, dass für eine Behauptung, die mit dem KMS-MIZAR bewiesen werden soll, die Schlüsselwörter `implies`, `holds` und `iff` nur einmal auftauchen dürfen. Somit kann versichert werden, dass der erzeugte Beweisrahmen richtig ist. Die Bestimmung der möglichen Beweisschritte für eine gegebene Behauptung wurde mit Hilfe des Dokuments[20] „Writing a Mizar article in nine easy Stepps“[4] durchgeführt.

[20] Freek Wiedijk “Writing a Mizar article in nine easy steps”: http://www.cs.ru.nl/~freek/mizar/

Für die Generierung der Beweisrahmen, wurde die Java Klasse `mizHelpGen` erstellt, die unter anderem die Methoden `proofHolds()`, `proofImplies()`, `proofIff()` und `proofEx()` implementiert.

Die Methode `proofHolds()` erzeugt einen Beweisrahmen für ein `theorem` vom Typ C) aus der Tabelle 6.1. Die Methode `proofImplies()` generiert einen Beweisrahmen für eine Behauptung vom Typ A). Die Methode `proofEx()` wird ihrerseits in den Methoden `proofHolds()`, `proofImplies()` und `proofIff()` aufgerufen.

Die Methode `proofImplies()` betrachtet die Ausdrücke links und rechts des Worts `implies`. Der linke Ausdruck wird mit `assume` angenommen und der rechte Ausdruck wird mit `thus` als richtige Folgerung akzeptiert. Bei der Methode `proofHolds()` ist die Vorgehensweise anders. Eine Behauptung vom Typ C) aus der Tabelle 6.1 hat drei wesentliche Bestandteile. Diese sind die Variablendeklaration, die mit `for` eingeleitet wird, der Ausdruck rechts des Worts `holds` und ein optionaler Ausdruck, der mit der Zeichenkette `st`[21] eingeführt wird und zwischen der Variablendeklaration und das Wort `holds` steht. Hier ein Beispiel:

```
for x,y being Element of NAT st x>y holds x-y>0.
```

In dem Beweisrahmen des Beispiels werden die Variablen in einer `Loci-Declaration` deklariert. Der optionale Ausdruck wird mit `assume` angenommen und der Ausdruck rechts von `holds` mit `thus` als richtig interpretiert. Der Aufruf von `proofHolds()` gibt die folgenden Zeilen als Beweisrahmen aus:

```
let x,y be Element of NAT;
  assume x>y;
thus  x-y>0;
```

Die `theorems` müssen aus einer XML-Datei hervorgehen. Die Methode `proofEx()` erzeugt einen Beweisrahmen für eine Behauptung vom Typ D). Eine Behauptung des Typs D) hat zwei relevante Bestandteile. Diese sind die Variablendeklaration, die diesmal etwas anders ist als die Variablendeklaration innerhalb einer Behauptung des Typs C) und liegt zwischen den Wörtern `ex` und `st`. Dann folgt der Ausdruck rechts von `st`. Diese Variablendeklaration wird in dem Beweisrahmen mit dem Konstrukt `consider` und `take` ersetzt und zum Schluss wird der Ausdruck rechts von `st` mit `thus` als richtig angenommen. Es soll die

[21] Das Wort `st` könnte in Deutsch als ‚so dass' übersetzt werden.

Ausgabe des KMS-MIZAR für ein `theorem`, das Behauptungen von Typ D) und Typ A) enthält, betrachtet werden:

```
a+b =a+c & b=k implies ex k,b,c being Nat st k=c & c=b.
```

Ausgabe:

```
assume a+b =a+c;
assume b=k;
  consider k,b,c being Nat;
  take k,b,c;
thus k=c;
thus c=b;
```

(Dies soll nur als Beipiel für einen Beweisrahmen gelten.)

Für ein `theorem`, in dem das Schlüsselwort `iff` auftaucht, wird die Methode `proofIff`() aufgerufen. Diese Methode ruft die anderen drei Methoden `proofEx`(), `proofHolds`() und `proofImplies`() auf, abhängig davon, ob das eingegebene `theorem` die Schlüsselwörter `ex`, `holds` oder `implies` beinhaltet. Wenn ein `theorem` die Zeichenkette `iff` enthält, dann sind zwei Schritte zu gehen. Der erste Schritt besteht daraus den Ausdruck links von `iff` mit `assume` anzunehmen und für den Ausdruck rechts von `iff` eine der Methoden `proofImplies()`, proofEx`()` und `proofHolds()` aufzurufen. In dem zweiten Schritt muss der Ausdruck rechts von `iff` mit `assume` angenommen werden und für den Ausdruck links von `iff` ist eine der Methoden `proofImplies()`, `proofEx()` und `proofHolds()` aufzurufen ES ist noch wichtig darauf hinzuweisen, dass ein `theorem` von Typ A), B), C), D) eine Behauptung von einem anderem Typ beinhalten darf, solange in der zusammen gebildeten Behauptung die Wörter `implies`, `holds`, `iff` lediglich einmal vorkommen.

Die Methode `genProoF`() ist die Hauptmethode der Klassen `menHelpGen` und ruft entsprechend die vier Methoden, die bisher erläutert worden sind, auf.

```
public static String genProof(String theorem) {
            String output = "";
            if (theorem.contains(" iff ")) {
                  output = proofIff(theorem);
} else if (theorem.contains(" implies ") && theorem.contains(" holds
")) {
                  int posIm = 0;// index of implies
                  int posHol = 0;// index of holds
```

```
                posIm = theorem.indexOf(" implies ");
                posHol = theorem.indexOf(" holds ");
                if (posIm > posHol) {
                      output = proofHolds(theorem);
                } else {
                      output = proofImplies(theorem);
                }
          } else if (theorem.contains(" implies ")) {
                output = proofImplies(theorem);
          } else if (theorem.contains(" holds ")) {
                output = proofHolds(theorem);
          }
          return output;
     }
```

Das Schlüsselwort `proof`, das einen Beweis einleitet, wird zu dem Beweisrahmen hinzugefügt, bevor der Beweisrahmen in die *.miz* Datei eingebunden wird.
Um die *.miz* Datei für das eingegebene `theorem`, den Anforderungen nach, vollständig zu erstellen, müssen noch die relevanten Dateinamen für die unterschiedlichen Direktiven bestimmt werden.

6.3 Belegung der unterschiedlichen Direktiven

Für die Bestimmung von Vorschlägen für die unterschiedlichen Direktiven aus der `Environment-Declaration` sollte der Benutzer von dem KMS-MIZAR unterstützt werden.

6.3.1 Die Direktiven vocabularies und requirements

Die Methode `getVocabularies()` aus der Klasse `mizHelpGen` gibt für ein gegebenes Mizar-Symbol einen Dateinamen zurück, der in der Direktive `vocabularies` einzutragen ist.

```
public static String getVocabularies(String symbol) {
     String voc = SelectMethods.startCommand("-w " + symbol);
     String[] output = voc.split("\n");
```

```
      String article = "";
      String[] out;
      String line = "no";
      for (int i = 0; i < output.length; i++) {
            if (output[i].trim().startsWith("vocabulary"))
                  line = output[i];
      }
      if (line != "no") {
            out = line.split(" ");
            article = out[1];
            }
            return article;
      }.
```

Im Gegensatz zu der Methode `startCommand()`, gibt `getVocabularies()` lediglich einen Dateinamen aus und alle weiteren Informationen über das eingegebene Symbol werden ausgeblendet. Der Aufruft der Methode `getVocabularies()` findet in der Methode `endElement()` des SAX-Parsers statt, jedoch nur für XML-Elemente vom Typ `(#PCDATA)`. Da sich unter einer Zeichenkette, die ein XML-Element beinhalten kann, nicht Mizar spezifische Symbole befinden können, gibt `getVocabularies ()` für diese Symbole ein leeres String zurück. Des Weiteren ist der Dateiname `HIDDEN(hidden:= verborgen)` auszublenden.

Ein Symbol, das mehrfach in einer Datei auftaucht, wird es verursachen, dass der zugehörige Dateiname genau so oft in der Direktive `vocabularies` vorkommt. Da ein Dateiname höchsten einmal eingetragen werden darf, sollen die übrigen Einträge gelöscht werden. Der folgende Codeausschnitt von der Methode `endElement()` speichert die gefundenen Dateinamen genau einmal in der Variable `vocab`.

```
String voc = "";
String[] out = actValue.split(" ");
for (int i = 0; i < out.length; i++) {
      voc = mizHelpGen.getVocabularies(out[i]);
            if (!voc.trim().equals("HIDDEN") && voc.trim() != "") {
            store1.add(voc.trim());
            }
}
```

```
for (int i = 0; i < store1.size(); i++) {
      if (!vocab.contains(store1.get(i))) {
            vocab += store1.get(i) + "\n";
      }
}[…]
```

Anschließend werden diese Einträge so formatiert, dass die Komas an den richtigen Stellen stehen und die resultierende Zeichenkette wird dem String `vocabularies` zugewiesen.

```
String[] buffer = vocab.split("\n");
for (int i = 0; i < buffer.length; i++) {
      if (i < buffer.length - 1) {
            vocabularies += buffer[i] + " ,";
      } else {
            vocabularies += buffer[i] + ";";
            }
}
```

Die Generierung der Einträge für die Direktive `requirements` ist relativ simple.
Folgende Einträge sollten bevorzugt angewandt werden:
`requirements BOOLE, SUBSET, NUMERALS, ARITHM, REAL;.`
Dies kann immer benutzt werden und gelten für alle Fälle.

6.3.2 Die Direktiven notations, constructors und registrations

Analog wie es im Kapitel 5.2 „Das `findVoc` Kommando des MIZAR-Systems“, Seite 42, erklärt ist, muss ebenfalls für jedes Symbol, das in einem `theorem` verwendet wird, eine relevante Referenz jeweils in den Direktiven `notations` und `constructors`. eingetragen werden. Die Einträge für diese beiden Direktiven sollen also identisch sein.
Die Direktive `notations` ist für die Syntax der Ausdrücke, die mit einem Mizar- Symbol gebildet werden können und die Direktive `constructors` ist für die Bedeutung dieser Ausdrücke.
Die Direktive `registrations` dient dazu, den `type` von Ausdrücken richtig zu erweitern.
Wie Einträge für diese Direktiven bestimmt werden, wird im Schritt 5 „Vervollständigung des Beweisrahmens“, Seite 64, erläutert.

6.4 Einführung in das Mizar KMS

Diese Einführung gibt Informationen darüber, wie das System zu benutzen ist, von der Installation des Mizar Systems bis zur Erstellung einer Mizar-Datei.
In dieser Einführung werden folgende Schritte durchgegangen.

- Installation des Mizar Systems
- Aufbau des KMS-MIZAR
- Erstellung eines XML-Dokuments
- Erstellung der Mizar-Datei
- Vervollständigung des Beweisrahmens.

Schritt 1: Installation des Mizar Systems

1. Gehen Sie auf die Seite http://mizar.org/system/ unter Downloads und laden Sie das Mizar System unter den Ordner C:\mizar herunter.
2. Folgen Sie die Anweisungen in der Datei readme.txt, die sich in dem Ordner C:\mizar befindet, um das System korrekt zu installieren.

Schritt 2: Aufbau des KMS-MIZAR

1. Öffnen Sie den SQL-Editor Ihres RDBMS und führen Sie das Script `createTables.sql` aus der beiliegenden CD aus.
2. Führen Sie ebenfalls die Skripte `sequence.sql` und `trigger.sql` aus, um die Sequenzen und Trigger für die Tabellen anzulegen.
3. Kompilieren Sie alle Java Dateien immer aus der beiliegenden CD und dann führen Sie die Hauptklasse `KMS-MIZAR` aus.
4. Nachdem Sie `KMS-MIZAR` ausgeführt haben, bekommen Sie die folgende Ausgabe auf Ihrer Konsole:

```
-----------------MIZAR-KMS-----------------
To ensure that this system works properly you should have the abstr
directory located
at C:\mizar. In additional, make sure that an appropriate database
has already been designed as well.
-------------------------------------------------
1. Type 1 to enter the Mizar insert program.
2. Type 2 to enter the Mizar select program.
3. Type 3 to run the Mizar Proof Generator.
```

```
0. Type 0 to leave the application.
```

Geben Sie 1 ein, um das Insert-Programm auszuwählen, dann geben Sie nochmals 1 und dann y ein, um alle Dateien aus der MML in die Datenbank einzufügen.
An dieser Stelle ist das KMS-MIZAR vollständig aufgebaut, wenn Sie Schritt 1 bis Schritt 2 korrekt gefolgt haben.

Schritt 3: Erstellung eines XML-Dokuments

In dieser Einführung wird eine Mizar-Datei für das folgende `theorem` erstellt:

```
Satz 1:
Sei x, y, u, v ∈ IR.
x > y ∧ u > v → x + u > y + v
```

Orientieren Sie sich nach der DTD, die auf Seite 67 zu finden ist, um das XML-Dokument für den Satz 1 zu editieren. Ein bereits erstelltes Dokument ist auf Seite 68 zu finden. Wichtig ist das XML-Dokument in dem Ordner, wo sich die Datei `mizarDTD.dtd` befinden, unterzubringen. Beim Editieren eines XML-Dokuments sollen nebeneinander stehende Symbole mit einem Leerzeichen getrennt werden. Es soll beispielsweise `a ^2` anstatt `a^2` geschrieben werden. Es wird so gemacht, damit die Methode `getVocabularies()`, die das `findVoc` Kommando aufruft, Werte zurückgibt (`a^2` als ganzes ist kein Mizar spezifischer Begriff und würde keine nützliche Rückgabe bewirken).

Schritt 4: Erstellung der Mizar-Datei

1. Führen Sie die Klasse KMS-MIZAR nochmals aus, falls Sie die Applikation beendet hatten. Andernfalls geben Sie 3 ein, um den „Proof-Generator“ aufzurufen.

```
1. Type 1 to enter the Mizar insert program.
2. Type 2 to enter the Mizar select program.
3. Type 3 to run the Mizar Proof Generator.
0. Type 0 to leave the application.
```

2. Geben Sie den Namen des XML-Dokuments z.B. `satz1.xml` ein und drücken Sie die Enter Taste. Falls das XML-Dokument valid ist, wird eine Mizar-Datei mit dem Namen `satz1.miz` erzeugt.

3. Prüfen Sie nach, ob die Datei `conList.txt` erzeugt wurde. Diese Datei gibt Informationen über mögliche Tippfehler, die Sie begangen haben. Der Tippfehler bezieht sich ausschließlich auf Schlüsselwörter wie `implies`, `and` etc...

Die Datei `satz1.miz` sieht wie folgt aus:

```
environ
requirements BOOLE, SUBSET, NUMERALS, ARITHM, REAL;
begin
reserve x, y, u, v for Real;
 x > y & u > v implies x + u > y + v
proof
assume x > y;
assume u > v;
thus x + u > y + v;
end;
```

Schritt 5: Vervollständigung des Beweisrahmens

1. Von der Konsole aus checken Sie die Datei `satz_3.miz` mittels des Kommandos `mizf` des Mizar Systems, Nachdem Sie auf den Pfad *c:\mizar* gewechselt haben.

```
mizf satz_1.miz
```

Die folgenden Fehlermeldungen werden **in der Datei** ausgegeben:

```
::> 825: Cannot find constructors name on constructor list
::> 856: Inaccessible requirements directive.
```

Die Fehlermeldungsnummer 825 deutet an, dass die Direktiven `notations` und `constructors` fehlen, und die Fehlermeldungsnummer 856 ist eine Folge des ersten Fehlers.

Um diese Fehler zu beheben müssen wir relevante Referenzen bezüglich der Symbole `Real` und > in den beiden Direktiven `notations` und `constructors` eintragen.

2. Wählen Sie das Menu `1. Definition` aus dem Select-Programm und geben Sie folgendes ein:

```
ww Real ww
```

3. Geben Sie `0` ein.

4. Entnehmen Sie der Ausgabe den Dateinamen. Der Dateiname ist der Wert hinter Article. In diesem Fall ist dieser Wert `Real_1.`

5. Analog geben Sie folgendes ein: <=

Hier geben wir <= und nicht >, weil Mizar das Symbol <= und nicht das Symbol > definiert. Um auf das Symbol > zurückzuführen, wird in Mizar festgelegt, dass > und <= synonyme Symbole sind.

Für die Eingabe von <=, kriegen wir 37 Treffer. Der Treffer, der für den Satz 1 relevant ist, ist der aus der Datei `XXREAL_0`.

6. Fügen Sie die folgenden Zeilen in die Datei `Satz_1.miz` hinzu:

```
notations REAL_1, XXREAL_0;
constructors REAL_1, XXREAL_0;
```

Dann checken wir noch mal die Datei mit dem Befehl aus Schritt 5: 1.

Es werden die folgenden Meldungen ausgegeben:

```
::> 102: Unknown predicate
::> 103: Unknown functor
```

Die Nummern 102 und 103 deuten darauf hin, dass die Benutzung des Prädikats `,>'` und des Operators `,+'` für den `type Real` nicht zulässig ist. In Fakt, der `type Real` ist nicht automatisch kompatibel weder zu dem `type ,ext-real number'`, der in der `definition` von `,>'` bzw. `,<='` benutzt wird, noch zu `,real number'`, der in der Neudefinition von `,+'` benutzt wird. Daher werden die Direktive `registrations` mit den Einträgen `REAL_1` und `XREAL_0` in die Datei `satz_3.miz` eingefügt.

Der erste Eintrag dient dazu, `Real` auf `real number` zu erweitern, und der letzte Eintrag sorgt dafür, dass `real number` auf `ext-real number` spezialisiert wird. Nachdem wir die Zeile

```
registrations REAL_1, XREAL_0;
```

in die Datei `satz_1.miz` eingefügt und die Datei noch einmal gescheckt haben, kriegen wir die folgende Meldung:

```
::> 4: This inference is not accepted
```

Diese Fehlermeldung deutet an, dass die Schlussfolgerung (`thus x + u > y + v;`) in dem Satz 1 nicht akzeptiert wird. Um den Fehler zu beheben, soll für den Satz einen vollständigen Beweis geschrieben werden. Jedoch ist der Beweisrahmen soweit richtig.

Die endgültige Datei sieht dann wie folgt aus:

```
environ
notations REAL_1, XXREAL_0;
constructors REAL_1, XXREAL_0;
registrations REAL_1, XREAL_0;
requirements BOOLE, SUBSET, NUMERALS, ARITHM, REAL;
begin
reserve x, y, u, v for Real;
 x > y & u > v implies x + u > y + v
proof
assume x > y;
assume u > v;
thus x + u > y + v;
::>              *4
end;

::>
::> 4: This inference is not accepted
```

7 Schlussbetrachtung

Das KMS-MIZAR kann benutzt werden, um Informationen über Mizar Begriffe zu recherchieren. Dadurch wird die Einarbeitung in die Sprache Mizar erleichtert.

Beweisrahmen können für 4 Typen von `theorems` erzeugt werden (wie es in Kapitel 6.2.2 „Die Erzeugung des Beweisrahmens“ beschrieben ist) und es konnte anhand eines Beispiels gezeigt werden, wie unter Benutzung des KMS-MIZAR die `Environment-Declaration` korrekt deklariert wird, um den Beweisrahmen zu vervollständigen.
Somit wurden alle Anforderungen an diese Bachelor-Arbeit erfüllt.

Neben dem KMS-Mizar existiert eine weitere Suchhilfe für die Mizar-Sprache, welche die Bezeichnung MML Query[22] trägt. Im Gegensatz zu KMS-MIZAR wird für die Benutzung von MML Query eine Internetverbindung vorausgesetzt.

Eine Erweiterungsmöglichkeit an KMS-MIZAR wäre die Reduzierung der Auswahlliste.
Denn eine lange Ergebnisliste erschwert die Suche nach dem passenden Eintrag.
Das Select-Programm des KMS-MIZAR vergleicht die Benutzereingabe mit den Datensätzen aus den DB-Tabellen und zeigt die Ergebnisliste an der Konsole an. Diese Liste kann je nach Benutzereingabe noch relativ lang sein. Die Existenz der Spalte pattern in der Tabelle definitions brachte jedoch schon eine erste Einschränkung der Auswahlliste.
Es wäre also denkbar, eine Herangehensweise zu definieren, die auf diese Bachelor-Arbeit aufbaut, zur weiteren Verfeinerung der Auswahlliste. Dies könnte beispielsweise das Anlegen von zusätzlichen Spalten sein, die dann in der WHERE-Klausel der SQL-Anfrage Verwendung finden.

[22] Home page der MML Query: http://merak.pb.bialystok.pl/mmlquery/three.html

Anhang A:

A.1: DTD der XML-Dokumente (mizarDTD.dtd)

```
<!ELEMENT section (article,text-item)>

<!ELEMENT article (#PCDATA)>
<!ELEMENT text-item (reservation*, theorem)>

<!ELEMENT reservation (reservation-segment+)>

<!ELEMENT reservation-segment (reserved-identifiers, type-
expression)>
<!ELEMENT reserved-identifiers (identifiers+)>

<!ELEMENT identifiers (#PCDATA)>
<!ATTLIST identifiers type CDATA "">

<!ELEMENT type-expression (#PCDATA)>
<!ATTLIST type-expression type CDATA "">

<!ELEMENT theorem (compact-statement)>

<!ELEMENT compact-statement (formula-expression)>

<!ELEMENT formula-expression (atomic-formula-expression|quantified-
formula-expression|(formula-expression, and, formula-expression)|
  (formula-expression, or, formula-expression)|(formula-expression,
implies, formula-expression)|(formula-expression, iff, formula-
expression)
  |(not, formula-expression)|contradiction|thesis)
>

<!ELEMENT atomic-formula-expression ((term-expression-
list?,predicate-symbol,term-expression-list?)|(term-
expression,is,adjective)
  |(term-expression,is,type-expression))
>

<!ELEMENT quantified-formula-expression ((for,qualified-
variables,(st,formula-expression)?,((holds, formula-
expression)|quantified-formula-expression))|
  (ex, qualified-variables, st, formula-expression))
>

<!ELEMENT and (#PCDATA)>
<!ATTLIST and type CDATA "">
<!ELEMENT contradiction (#PCDATA)>
<!ATTLIST contradiction type CDATA "">
<!ELEMENT ex (#PCDATA)>
<!ATTLIST ex type CDATA "">
<!ELEMENT for (#PCDATA)>
<!ATTLIST for type CDATA "">
<!ELEMENT holds (#PCDATA)>
<!ATTLIST holds type CDATA "">
```

```
<!ELEMENT iff (#PCDATA)>
<!ATTLIST iff type CDATA "">
<!ELEMENT implies (#PCDATA)>
<!ATTLIST implies type CDATA "">
<!ELEMENT is (#PCDATA)>
<!ATTLIST is type CDATA "">
<!ELEMENT not (#PCDATA)>
<!ATTLIST not type CDATA "">
<!ELEMENT or (#PCDATA)>
<!ATTLIST or type CDATA "">
<!ELEMENT st (#PCDATA)>
<!ATTLIST st type CDATA "">
<!ELEMENT thesis (#PCDATA)>
<!ATTLIST thesis type CDATA "">
<!ELEMENT adjective (#PCDATA)>
<!ATTLIST adjective type CDATA "">
<!ELEMENT qualified-variables (#PCDATA)>
<!ATTLIST qualified-variables type CDATA "">
<!ELEMENT term-expression-list (#PCDATA)>
<!ATTLIST term-expression-list type CDATA "">
<!ELEMENT predicate-symbol (#PCDATA)>
<!ATTLIST predicate-symbol type CDATA "">
<!ELEMENT term-expression (#PCDATA)>
<!ATTLIST term-expression type CDATA "">
```

A.2: XML-Dokument für Satz 1

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE section SYSTEM 'mizarDTD.dtd'>

<section>
  <article>satz_3</article>
  <text-item>
    <reservation>
      <reservation-segment>
        <reserved-identifiers>
          <identifiers>x</identifiers>
          <identifiers>y</identifiers>
          <identifiers>u</identifiers>
          <identifiers>v</identifiers>
        </reserved-identifiers>
        <type-expression>Real</type-expression>
      </reservation-segment>
    </reservation>
    <theorem>
      <compact-statement>
        <formula-expression>
          <formula-expression>
          <formula-expression>
            <atomic-formula-expression>
              <term-expression-list>x</term-expression-list>
              <predicate-symbol>&gt;</predicate-symbol>
              <term-expression-list>y</term-expression-list>
            </atomic-formula-expression>
          </formula-expression>
          <and>&amp;</and>
          <formula-expression>
          <atomic-formula-expression>
```

```
            <term-expression-list>u</term-expression-list>
            <predicate-symbol>&gt;</predicate-symbol>
            <term-expression-list>v</term-expression-list>
          </atomic-formula-expression>
          </formula-expression>
          </formula-expression>
          <implies>implieshh</implies>
          <formula-expression>
            <atomic-formula-expression>
              <term-expression-list>x + u</term-expression-list>
              <predicate-symbol>&gt;</predicate-symbol>
              <term-expression-list>y + v</term-expression-list>
            </atomic-formula-expression>
          </formula-expression>
        </formula-expression>
      </compact-statement>
    </theorem>
  </text-item>
</section>
```

Anhang B: Java Source-Code

Quelltext der Klasse `Connector.java`

```
import java.sql.*;

import oracle.jdbc.pool.*;

public class Connector {
	private static Connection instance = null;

	private Connector() throws SQLException {
		connect();
	}

	private void connect() throws SQLException {
		String driver;
		OracleDataSource ods = new OracleDataSource();

		driver = "oracle.jdbc.driver.OracleDriver";

		// loading of the driver
		try {
			Class.forName(driver).newInstance();
		} catch (Exception e) {
			System.out.println("An error occured when loading
the driver: "
					+ e.getMessage());
		}

		// creation of the connection instance named instance
		try {
			ods

	.setURL("jdbc:oracle:thin:bachelor/maximm@localhost:1521/XE");
			instance = ods.getConnection();
		} catch (SQLException e) {

			System.out
					.println("The attempt to connect to the
database failed!");
			System.out.println(e.getMessage());
		}

	}

	public static void disconnect() {
		try {
			instance.close();
			instance = null;
		} catch (SQLException e) {

			System.out
					.println("The attempt to connect to the
database failed!");
```

```
                System.out.println(e.getMessage());
            }
        }

        public synchronized static Connection getConnection() throws
SQLException {
            if (instance == null) {
                new Connector();
            }
            return instance;
        }
}
```

Quelltext der Klasse `SplitElement.java`

```
import java.util.*;
import java.io.File;

public class SplitElement {
        /**
         * @param definition
         *              : a definition of the file whose name you just
typed
         * */
        public    static    ArrayList<Object>    splitDefinition(String
definition,
                String fileName) {
            String[] buffer = definition.split("\\n");
            int redefine = 0; // the role of redefine is explained in
the
            // thesis
            ArrayList<Integer> defNo = new ArrayList<Integer>();
            // to count the exact number of definitions contained in
a block
            ArrayList<Integer> letNo = new ArrayList<Integer>();
            // line number where a definition-item starts and ends
            ArrayList<Object> container = new ArrayList<Object>();
            // the second element in the list container represents
the variable
            // 'redefine'

            for (int i = 0; i < buffer.length; i++) {
                if      (buffer[i].contains("      attr      ")      ||
buffer[i].contains(" func ")
                        || buffer[i].contains(" mode ")
                        || buffer[i].contains(" pred ")
                        || buffer[i].contains(" struct ")) {
                    defNo.add(i);// this operation help to count
the exact
                    // number of definitions in a definition-block
and to save their
                    // position
                }
            }
```

```
            // the first loop is to recognize the line number which a
let stands at.
            // the second loop is to save the lines that form this
let.
            // this means starting from let up to the next semicolon.
            for (int i = 0; i < buffer.length; i++) {
                  if      (buffer[i].contains("      let      ")      ||
buffer[i].contains(" assume ")
                              || buffer[i].contains(" given ")) {
                        for (int j = i; j < buffer.length; j++) {
                              letNo.add(j);
                              if (buffer[j].endsWith(";"))
                                    break;
                        }
                  }
            }

            String[] def = new String[defNo.size()];// defNo.size()
returns the
            // number of definitions
            int[] isRedefine = new int[defNo.size()];
            String[] patterns = new String[defNo.size()];
            String[] defNames = new String[defNo.size()];

            for (int k = 0; k < defNo.size(); k++) {
                  def[k] = "";
                  // this for-loop serves to add the definition-item
to
                  // the real definition
                  for (int j = 0; j < letNo.size(); j++) {
                        if (letNo.get(j) < defNo.get(k)) {
                              def[k] += buffer[letNo.get(j)] + "\n ";
                        }
                  }

                  if (k < defNo.size() - 1) {

                        for (int m = defNo.get(k); m < defNo.get(k +
1); m++) {
                              if (buffer[m].contains(" let ")
                                          || buffer[m].contains(" assume
")
                                          || buffer[m].contains(" given
"))
                                    break;
                              // to stop adding when either the word
let or assume or
                              // given appears
                              def[k] += buffer[m] + "\n ";
                              if (buffer[m].contains(" redefine ")) {
                                    redefine = 1;
                              }
                        }
                        isRedefine[k] = redefine;
                  }
                  // defNo.size() - 1 represents the last definition
                  if (k == defNo.size() - 1) {
```

```
                        for (int m = defNo.get(k); m < buffer.length;
m++) {
                              def[k] += buffer[m] + "\n ";
                              if (buffer[m].contains(" redefine ")) {
                                    redefine = 1;
                              }
                        }
                        isRedefine[k] = redefine;
                  }
                  def[k] = def[k].replace("'", "''");// The proper
definition
                  patterns[k] = getPattern(def[k]);// The defined term
                  defNames[k] = defName(def[k], fileName);// The name
of the
                  // definition

            }
            container.add(0, def);
            container.add(1, isRedefine);
            container.add(2, patterns);
            container.add(3, defNames);

            return container;
      }

      /**
       * @param notation
       *                : a notation of the file whose name you just
typed
       * */
      public static String[] splitNotation(String notation) {
            String[] buffer = notation.split("\\n");
            ArrayList<Integer> notNo = new ArrayList<Integer>();//
not number
            ArrayList<Integer> letNo = new ArrayList<Integer>();
            // line number where a loci-declaration starts and ends
            for (int i = 0; i < buffer.length; i++) {
                  if (buffer[i].contains(" synonym ")
                              || buffer[i].contains(" antonym ")) {
                        notNo.add(i);// this operation help to count
the exact
                        // number of notation-blocks and to save their
position
                  }
            }
            // the first loop is to recognize the line number which a
let stands at.
            // the second loop is to save the lines that form this
loci-declaration.
            // this means starting from let up to the next semicolon.
            for (int i = 0; i < buffer.length; i++) {
                  if (buffer[i].contains(" let ")) {
                        for (int j = i; j < buffer.length; j++) {
                              letNo.add(j);
                              if (buffer[j].endsWith(";"))
                                    break;
                        }
```

```
                }
            }
            String[] not = new String[notNo.size()];
            // notNo.size() returns the
            // number of notation-blocks
            for (int k = 0; k < notNo.size(); k++) {
                not[k] = "";
                // this for-loop serves to add the loci-declaration
to the
                // definition
                for (int j = 0; j < letNo.size(); j++) {
                    if (letNo.get(j) < notNo.get(k)) {
                        not[k] += buffer[letNo.get(j)] + "\n";
                    }
                }
                if (k < notNo.size() - 1) {
                    for (int m = notNo.get(k); m < notNo.get(k +
1); m++) {
                        if (buffer[m].contains("let"))
                            break;
                        // to stop adding when either the word
let appears
                        not[k] += buffer[m] + "\n";
                    }
                }
                // notNo.size() - 1 represents the last notation
                if (k == notNo.size() - 1) {
                    for (int m = notNo.get(k); m < buffer.length;
m++) {
                        not[k] += buffer[m] + "\n";
                    }
                }
                not[k] = not[k].replace("'", "''");
            }
            return not;
        }

        /**
         * @param registration
         *             : a registration of the file whose name you just
typed
         * */
        public static ArrayList<Object> splitRegistration(String
registration) {
            String[] buffer = registration.split("\\n");
            int existential = 1; // the role of existential is
explained in the
            // thesis
            ArrayList<Integer> regNo = new ArrayList<Integer>();//
line number
            ArrayList<Integer> letNo = new ArrayList<Integer>();
            // line number where a definition-item starts and ends
            ArrayList<Object> container = new ArrayList<Object>();
            // the first element in the list container represents the
variable
            // 'existential'
            for (int i = 0; i < buffer.length; i++) {
```

```
                if (buffer[i].contains(" cluster ")) {
                      regNo.add(i);// this operation help to count
the exact
                      // number of cluster-registrations and to save
their position
                }
           }
           // the first loop is to recognize the line number where a
let stands .
           // the second loop is to save the lines that form the
loci-declaration.
           // This means starting from let up to the next semicolon.
           for (int i = 0; i < buffer.length; i++) {
                if (buffer[i].contains(" let ")) {
                      for (int j = i; j < buffer.length; j++) {
                            letNo.add(j);
                            if (buffer[j].endsWith(";"))
                                  break;
                      }
                }
           }

           String[] reg = new String[regNo.size()];// regNo.size()
returns the
           // number of cluster-registrations
           int[] isExistential = new int[regNo.size()]; // to save
the variable
           // existential
           for (int k = 0; k < regNo.size(); k++) {
                reg[k] = "";
                // this for-loop serves to add the definition-item
to the
                // definition
                for (int j = 0; j < letNo.size(); j++) {
                      if (letNo.get(j) < regNo.get(k)) {
                            reg[k] += buffer[letNo.get(j)] + "\n";
                      }
                }
                if (k < regNo.size() - 1) {
                      for (int m = regNo.get(k); m < regNo.get(k +
1); m++) {
                            if (buffer[m].contains(" let "))
                                  break;
                            // to stop adding when either the word
let appears
                            reg[k] += buffer[m] + "\n";
                            if (buffer[m].contains("->")) {
                                  existential = 0;
                            }
                      }
                      isExistential[k] = existential;
                }
                // regNo.size() - 1 represents the last registration
                if (k == regNo.size() - 1) {
                      for (int m = regNo.get(k); m < buffer.length;
m++) {
                            reg[k] += buffer[m] + "\n";
```

```
                        if (buffer[m].contains(" -> ")) {
                              existential = 0;
                        }
                  }
                  isExistential[k] = existential;
            }
            reg[k] = reg[k].replace("'", "''");
      }
      container.add(0, reg);
      container.add(1, isExistential);

      return container;
}

/**
 * @param theorem
 *            : a registration of the file whose name you just
typed
 * @param file
 *            is the name of the file entered
 * */

public static String[] splitTheorem(String theorem, String
file)

{
      String[] output = new String[2];
      String[] buffer = theorem.split("\\n");
      String theo = "";
      String name = "";
      int index = 0;
      if (buffer[0].contains(file)) {
            index = buffer[0].indexOf(file);
            name          =          buffer[0].substring(index,
buffer[0].length());
      }
      for (int i = 1; i < buffer.length; i++) {
            theo += buffer[i] + "\n";
      }

      output[0] = name;
      output[1] = theo;
      return output;

}

/**
 * This method returns all of the files contained in the
directory
 * directoryPath. In our case it should be the directory that
contains the
 * mizar abstract files.
 *
 * @param directoryPath
 *            : name of the directory
 * @return File[]: an array containing all the files of the
directory
```

```
	 */
	public static File[] splitDirectory(String directoryPath) {
		File[] mizarArticles = null;
		File directory = new File(directoryPath);
		if (!directory.exists()) {
			System.out.println("The directory '" + directoryPath
					+ "' does not exist");
		} else if (!directory.isDirectory()) {
			System.out.println("The path '" + directoryPath
					+ "' fits a file and not a directory");
		} else {
			mizarArticles = directory.listFiles();
		}
		return mizarArticles;
	}

	/**
	 * This method returns the smallest index of a list of integer
values
	 *
	 * @param a
	 *            list of integer values
	 * @return the smallest index
	 */
	public static int min(int[] array) {
		int temp = 0;
		for (int j = 0; j < array.length - 1; j++) {
			if ((array[j] < array[j + 1]) && array[j] != 0) {
				temp = array[j + 1];
				array[j + 1] = array[j];
				array[j] = temp;
			}
		}
		return array[array.length - 1];
	}

	/**
	 * This method returns the defined term
	 *
	 * @param a
	 *            definition from the MML
	 * @return the defined term
	 */

	public static String getPattern(String definition) {
		int kindIndex = 0;
		String pattern = "";
		int[] indexes;
		ArrayList<Integer> indexs = new ArrayList<Integer>();//
contains the
		// indexes of
		// the words
		// below
		// These words could appear before a functor pattern,
predicate pattern
		// etc...
```

```
String buffer = "";
int k = 0;
indexs.add(definition.length() - 1);
do {
      buffer += definition.charAt(k);
      if (definition.charAt(k) == 32) {
            buffer = "";
      }

      if (buffer.equals("means")) {
            indexs.add(k - 5);
      } else if (buffer.equals("equals")) {
            indexs.add(k - 6);
      } else if (buffer.equals("->")) {
            indexs.add(k - 2);
      } else if (buffer.equals("existence")) {
            indexs.add(k - 9);
      } else if (buffer.equals("uniqueness")) {
            indexs.add(k - 10);
      } else if (buffer.equals("coherence")) {
            indexs.add(k - 9);
      } else if (buffer.equals("compatibility")) {
            indexs.add(k - 13);
      } else if (buffer.equals("consistency")) {
            indexs.add(k - 11);
      } else if (buffer.equals("correctness")) {
            indexs.add(k - 11);
      } else if (buffer.equals("commutativity")) {
            indexs.add(k - 13);
      } else if (buffer.equals("idempotence")) {
            indexs.add(k - 11);
      } else if (buffer.equals("involutiveness")) {
            indexs.add(k - 14);
      } else if (buffer.equals("projectivity")) {
            indexs.add(k - 12);
      } else if (buffer.equals("symmetry")) {
            indexs.add(k - 8);
      } else if (buffer.equals("asymmetry")) {
            indexs.add(k - 9);
      } else if (buffer.equals("connectedness")) {
            indexs.add(k - 13);
      } else if (buffer.equals("reflexivity")) {
            indexs.add(k - 11);
      } else if (buffer.equals("irreflexivity")) {
            indexs.add(k - 13);
      }
      k++;
} while (k < definition.length());
indexes = new int[indexs.size()];
for (int i = 0; i < indexs.size(); i++) {
      indexes[i] = indexs.get(i);
      // System.out.println(i + " hier " + indexes[i]);
}
if (definition.contains(" mode ")) {
      kindIndex = definition.indexOf(" mode ");
} else if (definition.contains(" func ")) {
      kindIndex = definition.indexOf(" func ");
```

```
            } else if (definition.contains(" pred ")) {
                  kindIndex = definition.indexOf(" pred ");
            } else if (definition.contains(" attr ")) {
                  kindIndex = definition.indexOf(" attr ");
            }

            int firstIndex = 0;// The smallest index among all of
them
            firstIndex = min(indexes);
            pattern = definition.substring(kindIndex, firstIndex);
            return pattern;
      }

      /**
       * This method returns the name of the definition if it has one
       *
       * @param a
       *            definition
       * @param the
       *            file where the definition comes from
       * @return the definition's name
       */
      public static String defName(String definition, String
filename) {
            String[] def = definition.split("\n");
            String name = "No name!";
            for (int i = 0; i < def.length; i++) {
                  if (def[i].contains(filename))
                        name = def[i].substring(3);
            }
            return name;
      }

}
```

Quelltext der Klasse `InsertUpdate.java`

```
import java.io.*;
import java.util.*;

public class InsertUpdate {
      /**
       * @param file
       *            name of a given file from the MML
       */
      public static void insertFile(String file) {
            String filename = "C:\\mizar\\abstr\\";// Path of the
abstract articles
            // under windows
            String description = "";
            StringBuffer des = new StringBuffer();
            file = file.toUpperCase();
            String line = "";
            ArrayList<Long> offsetd, offsetn, offsetr, offsett; //
needed to
```

```
            // navigate in the
            // file
            StringBuffer bufferd, buffern, bufferr, buffert;
            int defno = 0; // number of definitions contained in the
given file
            int notno = 0; // number of notations contained in the
given file
            int regno = 0; // number of registrations contained in
the given file
            int theono = 0; // number of theorems contained in the
given file
            filename = filename + file + ".abs";
            RandomAccessFile raf;// raf is reserved for the given
file,
            // ran
            // for the output file and ra for the
            // file where the description is to get

            try {
                  raf = new RandomAccessFile(filename, "r");
                  raf.seek(0);
                  while ((line = raf.readLine()) != null
                              && !line.trim().endsWith("::")) {
                        // the file description is separated from the
rest of the file
                        // by an empty line

                        des.append(line + "\n");
                  }
                  description = des.toString();
                  raf.close();
            } catch (IOException e) {
                  System.out.println("A problem  occured when reading
the file "
                              + filename);
                  System.out.println(e.getMessage());
            }// end of the first try-catch-block

            try {
                  raf = new RandomAccessFile(filename, "r");

                  // start of definition
                  offsetd = new ArrayList<Long>();
                  while ((line = raf.readLine()) != null) {
                        if (line.startsWith("definition")) {
                              defno++;
                              offsetd.add(raf.getFilePointer());
                        }
                  }// end of definition

                  // start of notation
                  offsetn = new ArrayList<Long>();
                  raf.seek(0);
                  while ((line = raf.readLine()) != null) {
                        if (line.startsWith("notation")) {
                              notno++;
                              offsetn.add(raf.getFilePointer());
```

```
                    }
                }// end of notation

                // start of registration
                offsetr = new ArrayList<Long>();
                raf.seek(0);
                while ((line = raf.readLine()) != null) {
                    if (line.startsWith("registration")) {
                        regno++;
                        offsetr.add(raf.getFilePointer());
                    }
                }// end of registration

                // start of theorem
                offsett = new ArrayList<Long>();
                raf.seek(0);
                while ((line = raf.readLine()) != null) {
                    if (line.startsWith("theorem")) {
                        theono++;
                        long len = line.trim().length();
                        offsett.add(raf.getFilePointer() - len);
                        // we add 2 because of the line break
                    }
                }// end of theorem

                // insertAbsFiles invocation
                description = description.replace("'", "''");
                // the character ' makes it not possible to insert
the variable
                // description as a literal string
                InsertMethods.insertAbsFiles(file,      description,
defno, notno,
                        regno, theono);

    /*****************************************************************
****************************************/
                // start of definition
                ArrayList<Object>       containerd       =       new
ArrayList<Object>();
                for (int i = 0; i < offsetd.size(); i++) {
                    raf.seek(offsetd.get(i));
                    bufferd = new StringBuffer();
                    while ((line = raf.readLine()) != null
                            && !line.endsWith("end;")) {

                        bufferd.append(line + "\n");
                    }
                    containerd.add(bufferd);

                }
                // container.get(0) contains an array with all of
the definitions
                //  container.get(1)  contains  the  values  of  the
variable
                // redefine
                int counterd = 0;
                for (int i = 0; i < containerd.size(); i++) {
```

```
                    for (int j = 0; j < ((String[])
(SplitElement.splitDefinition(
                                containerd.get(i).toString(),
file).get(0))).length; j++) {
                          counterd++;
                    }
               }
               String[] definitions = new String[counterd];
               // For the entire file
               int[] redefine = new int[counterd];
               String[] patterns = new String[counterd];
               String[] defNames = new String[counterd];
               int countd = 0;
               for (int i = 0; i < containerd.size(); i++) {
                    // def only for a definition of the file
                    String[] def = ((String[])
(SplitElement.splitDefinition(
                                containerd.get(i).toString(),
file).get(0)));
                    int[] red = ((int[])
(SplitElement.splitDefinition(containerd
                                .get(i).toString(), file).get(1)));
                    String[] pattern = ((String[])
(SplitElement.splitDefinition(
                                containerd.get(i).toString(),
file).get(2)));
                    String[] defName = ((String[])
(SplitElement.splitDefinition(
                                containerd.get(i).toString(),
file).get(3)));
                    for (int k = 0; k < def.length; k++) {
                          definitions[countd] = def[k];
                          redefine[countd] = red[k];
                          patterns[countd] = pattern[k];
                          defNames[countd] = defName[k];
                          countd++;
                    }

               }

               if (InsertMethods.isLocked(file).equals("no")) {
                    InsertMethods.insertDefinitions(file,
definitions, redefine, patterns, defNames);
               }// end of definition

     /*****************************************************************
*****************************************/
               // start of notation
               ArrayList<Object> containern = new
ArrayList<Object>();
               for (int i = 0; i < offsetn.size(); i++) {
                    raf.seek(offsetn.get(i));
                    buffern = new StringBuffer();
                    while ((line = raf.readLine()) != null
                              && !line.endsWith("end;")) {

                         buffern.append(line + "\n");
```

```
                    }
                    containern.add(buffern);
                }
                int countern = 0;
                for (int i = 0; i < containern.size(); i++) {
                    for (int j = 0; j <
(SplitElement.splitNotation(containern.get(
                            i).toString())).length; j++) {
                        countern++;
                    }
                }

                String[] notations = new String[countern];
                int countn = 0;
                for (int i = 0; i < containern.size(); i++) {
                    String[] not = (String[])
SplitElement.splitNotation(containern
                            .get(i).toString());
                    for (int k = 0; k < not.length; k++) {
                        notations[countn] = not[k];
                        countn++;
                    }

                }
                if (InsertMethods.isLocked(file).equals("no")) {
                    InsertMethods.insertNotations(file, notations);
                }// end of notation


    /*****************************************************************
*****************************************/

                // start of registration
                ArrayList<Object> containerr = new
ArrayList<Object>();
                for (int i = 0; i < offsetr.size(); i++) {
                    raf.seek(offsetr.get(i));
                    bufferr = new StringBuffer();
                    while ((line = raf.readLine()) != null
                            && !line.endsWith("end;")) {

                        bufferr.append(line + "\n");
                    }
                    containerr.add(bufferr);
                }
                int counterr = 0;
                for (int i = 0; i < containerr.size(); i++) {
                    for (int j = 0; j < ((String[]) (SplitElement

    .splitRegistration(containerr.get(i).toString()).get(0))).lengt
h; j++) {
                        counterr++;
                    }
                }
                String[] registrations = new String[counterr];
                int[] existential = new int[counterr];
                int countr = 0;
```

```
                for (int i = 0; i < containerr.size(); i++) {
                      String[] reg = ((String[]) (SplitElement

      .splitRegistration(containerr.get(i).toString()).get(0)));
                      int[]          exi          =          ((int[])
(SplitElement.splitRegistration(containerr
                                  .get(i).toString()).get(1)));
                      for (int k = 0; k < reg.length; k++) {
                            registrations[countr] = reg[k];
                            existential[countr] = exi[k];
                            countr++;
                      }
                }
                if (InsertMethods.isLocked(file).equals("no")) {
                      InsertMethods.insertRegistrations(file,
registrations,
                                  existential);
                }// end of registration

      /*****************************************************************
****************************************/
                // start of theorem
                // theonames stands for the names
                // theorems stands for the theorems
                String[] theonames = new String[offsett.size()];
                String[] theorems = new String[offsett.size()];
                for (int i = 0; i < offsett.size(); i++) {
                      raf.seek(offsett.get(i));
                      buffert = new StringBuffer();
                      while ((line = raf.readLine()) != null
                                  && !line.startsWith("canceled")
                                  && !line.startsWith("definition")
                                  && !line.startsWith("notation")
                                  && !line.startsWith("registration")
                                  && !line.startsWith("theorem")
                                  &&   !line.startsWith("scheme")   &&
!line.startsWith("::")
                                  && !line.startsWith("begin")
                                  && !line.startsWith("reserve")) {
                            //  all  these  words  can  appear  after  a
theorem

                            buffert.append(line + "\n");
                      }
                      theonames[i]                                      =
SplitElement.splitTheorem((buffert.toString())
                                  .replace("'", "''"), file)[0];
                      theorems[i]                                       =
SplitElement.splitTheorem((buffert.toString())
                                  .replace("'", "''"), file)[1];
                }
                if (InsertMethods.isLocked(file).equals("no")) {
                      InsertMethods.insertTheorems(file,    theorems,
theonames);
                }// end of theorem

                raf.close();
```

```
            } catch (IOException e) {
                  System.err.println("A problem has occurred when
reading the file "
                              + filename);
                  System.out.println(e.getMessage());
            }// end of the second try-catch-block
            InsertMethods.blockInsertion(file);
      }

      public static void insertAllFiles() {
            String      directoryPath      =      "C:\\mizar\\abstr";//
"C:\\mizar\\abstr\\test"
            File[] subFiles = null;
            subFiles = SplitElement.splitDirectory(directoryPath);
            for (int i = 0; i < subFiles.length; i++) {
                  insertFile(subFiles[i].getName().substring(0,
                              subFiles[i].getName().length() - 4));
            }

      }

      public static void update() {
            InsertMethods.delete();
            insertAllFiles();
      }
}
```

Quelltext der Klasse `InsertMethods.java`

```
import java.sql.*;

public class InsertMethods {

      /**
       * Insert information related to a file as defined in the
thesis
       *
       * @param filename
       * @param description
       *            : brief description of what the file does
       * @param defno
       *            : number of definitions contained in the file
       * @param notno
       *            : number of notations contained in the file
       * @param regno
       *            : number of registrations contained in the file
       * @param theono
       *            : number of theorems contained in the file
       */
      public static void insertAbsFiles(String filename, String
description,
                  int defno, int notno, int regno, int theono) {
```

```
            String sql1, sql2;// sql1 and sql2 build the sql-
expressions to be used
            sql1 = "insert into absFiles (filename, description,
defno, notno, regno, theono, locked) values('"
                        + filename
                        + "','"
                        + description
                        + "',"
                        + defno
                        + ","
                        + notno
                        + "," + regno + "," + theono + ", " + 0 + ")";
            sql2 = "select * from absFiles where filename = '" +
filename + "'";
            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();
                  Statement st;
                  ResultSet result;
                  st = con.createStatement();
                  result = st.executeQuery(sql2);
                  if (!result.next()) {
                        // checks whether the entered file already
exists
                        st.executeUpdate(sql1);
                  } else {
                        System.out
                                    .println("The file "
                                                + filename
                                                + " probably already
exists in the database und could not be inserted.\nTry to enter a
new file.");

                  }

                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
                  System.out.println("the file " + filename
                              + " could not be inserted");
            }
      }

      /**
       * insert all of the definitions that are included in a given
Mizar file
       *
       * @param filename
       *              : name of the file where this actual definition
comes from
       * @text: the proper text of the definition
       * @redefine: this role variable is already explained in the
thesis
       */
```

```
     public static void insertDefinitions(String filename, String[]
definitions,
                int[]   redefine,   String[]   patterns,   String[]
defNames) {
          int fileID = 0;// fileID represents a primary key in the
table absFiles
          // , and is foreign key in the table notations
          String  sql1,  sql2;//  sql1  and  sql2  build  the  sql-
expressions to be used
          sql1 = "select fileid from absFiles where filename = '" +
filename
                    + "'";
          try {
               // building of the connection instance using the
method connect()
               Connection con = Connector.getConnection();
               Statement st;
               ResultSet result;
               st = con.createStatement();
               result = st.executeQuery(sql1);
               if (result.next()) {
                    fileID  =  result.getInt("fileid");//  gets  the
primary key of the
                    // file
               }
               for (int i = 0; i < definitions.length; i++) {
                    definitions[i]     =     "definition\n"     +
definitions[i] + "end;";
                    sql2 = "insert into definitions (fileid, text,
redefine, pattern, defName) values("
                              + fileID
                              + ",'"
                              + definitions[i]
                              + "','"
                              + redefine[i]
                              +  "','"  +  patterns[i]  +  "','"  +
defNames[i] + "')";
                    st.executeUpdate(sql2);
               }

               Connector.disconnect();
          } catch (SQLException e) {
               System.out.println(e.getMessage());
               System.out.println("SQL Exception has been thrown");
          }
     }

     /**
      * insert all of the notations that are included in a given
Mizar file
      *
      * @param filename
      *             : name of the file where this actual definition
comes from
      * @text: the proper text of the notation
      */
```

```
	public static void insertNotations(String filename, String[]
notations) {
		int fileID = 0; // fileID represents a primary key in the
table absFiles
		// , and is foreign key in the table notations
		String sql1, sql2;// sql1 and sql2 build the sql-
expressions to be used
		sql1 = "select fileid from absFiles where filename = '" +
filename
				+ "'";
		try {
			// building of the connection instance using the
method connect()
			Connection con = Connector.getConnection();
			Statement st;
			ResultSet result;
			st = con.createStatement();
			result = st.executeQuery(sql1);
			if (result.next()) {
				fileID = result.getInt("fileid");// gets the
primary key of the
				// file
			}
			for (int i = 0; i < notations.length; i++) {
				notations[i] = "notation\n" + notations[i] +
"end;";
				sql2 = "insert into notations (fileid, text)
values(" + fileID
						+ ",'" + notations[i] + "')";
				st.executeUpdate(sql2);
			}

			Connector.disconnect();
		} catch (SQLException e) {
			System.out.println(e.getMessage());
			System.out.println("SQL Exception has been thrown");
		}
	}

	/**
	 * insert all of the registrations that are included in a given
Mizar file
	 *
	 * @param filename
	 *            : name of the file where this actual definition
comes from
	 * @text: the proper text of the definition
	 * @existential: this role variable is already explained in the
thesis
	 */

	public static void insertRegistrations(String filename,
			String[] registrations, int[] existential) {
		int fileID = 0;// fileID represents a primary key in the
table absFiles
		// , and is foreign key in the table registrations
```

```
            String sql1, sql2;// sql1 and sql2 build the sql-
expressions to be used
            sql1 = "select fileid from absFiles where filename = '" +
filename
                        + "'";
            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();
                  Statement st;
                  ResultSet result;
                  st = con.createStatement();
                  result = st.executeQuery(sql1);
                  if (result.next()) {
                        fileID = result.getInt("fileid");// gets the
primary key of the
                        // file
                  }
                  for (int i = 0; i < registrations.length; i++) {
                        registrations[i] = "registration\n" +
registrations[i] + "end;";
                        sql2 = "insert into registrations (fileid,
text, existential) values("
                                    + fileID
                                    + ",'"
                                    + registrations[i]
                                    + "',"
                                    + existential[i] + ")";
                        st.executeUpdate(sql2);
                  }

                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
            }
      }

      /**
       * insert all of the theorems that are included in a given
Mizar file
       *
       * @param filename
       *              : name of the file where this actual definition
comes from
       * @text: the proper text of the definition
       */
      public static void insertTheorems(String filename, String[]
theorems,
                  String[] theonames) {
            int fileID = 0;// fileID represents a primary key in the
table absFiles
            // , and is foreign key in the table theorems
            String sql1, sql2;// sql1 and sql2 build the sql-
expressions to be used
            sql1 = "select fileid from absFiles where filename = '" +
filename
```

```
                    + "'";
            Statement st = null;
            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();

                  ResultSet result;
                  st = con.createStatement();
                  result = st.executeQuery(sql1);
                  if (result.next()) {
                        fileID = result.getInt("fileid");// gets the
primary key of the
                        // file
                  }
                  for (int i = 0; i < theorems.length; i++) {
                        theorems[i] = "theorem :: " + theonames[i] +
"\n" + theorems[i];
                        sql2 = "insert into theorems (fileid, text,
theoname) values("
                                    + fileID + ",'" + theorems[i] +
"','" + theonames[i]
                                    + "')";
                        st.executeUpdate(sql2);
                  }

                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
            }
      }

      /**
       * this method makes it no longer possible to insert a file and
its elements
       * more than once.
       *
       * @filename: name of a given Mizar file
       */
      public static void blockInsertion(String filename) {
            String sql;
            sql = "update absFiles set locked = " + 1 + " where
filename = '"
                        + filename + "'";

            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();
                  Statement st;
                  st = con.createStatement();
                  st.executeUpdate(sql);
                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
```

```
            }
      }

      /**
       * This method helps us know whether the entered file has been
inserted
       * before
       *
       * @param filename
       *            : name of a given Mizar file
       * @return true if the file already exists in the database
       */
      public static String isLocked(String filename) {
            String isLocked = "";
            String sql;
            sql = "select locked from absFiles where filename = '" +
filename + "'";

            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();
                  Statement st;
                  ResultSet result;
                  st = con.createStatement();
                  result = st.executeQuery(sql);
                  if (result.next()) {
                        if ((result.getInt("locked") == 0)) {
                              //   checks   whether   the   entered   file
already exists
                              isLocked = "no";
                        } else {
                              isLocked = "yes";
                        }
                  }

                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
            }
            return isLocked;
      }

      /**
       * This method delete the whole content of the database.
       */

      public static void delete() {
            String[]   tables   =   {   "definitions",   "notations",
"registrations",
                        "theorems", "absFiles" };
            String[] sequences = { "file_seq", "def_seq", "not_seq",
"reg_seq",
                        "theo_seq" };
            String[] sql1, sql2, sql3;
            sql1 = new String[5];
```

```
            sql2 = new String[5];
            sql3 = new String[5];
            for (int i = 0; i < 5; i++) {
                  sql1[i] = "delete from " + tables[i];
                  sql2[i] = "drop sequence " + sequences[i];
                  sql3[i] = "create sequence " + sequences[i]
                              +   "   start   with   1   increment   by   1
nomaxvalue";
            }

            try {
                  // building of the connection instance using the
method connect()
                  Connection con = Connector.getConnection();
                  Statement st;
                  st = con.createStatement();
                  for (int i = 0; i < 5; i++) {
                        st.executeUpdate(sql1[i]);
                  }
                  for (int i = 0; i < 5; i++) {
                        st.executeUpdate(sql2[i]);
                  }
                  for (int i = 0; i < 5; i++) {
                        st.executeUpdate(sql3[i]);
                  }

                  Connector.disconnect();
            } catch (SQLException e) {
                  System.out.println(e.getMessage());
                  System.out.println("SQL Exception has been thrown");
            }

      }

}
```

Quelltext der Klasse `IO1.java`

```
import java.io.*;

class IO1
{public static int einint()
 { int w=-999999, rc;
   BufferedReader in
        = new BufferedReader(new InputStreamReader(System.in));
   do
   { rc=0;      //Fehlervermutung
     try
     { w=Integer.valueOf(in.readLine().trim()).intValue();
       rc=1;    //kein Fehler <=> rc=1
     }
     catch(IOException e)
     { System.out.println("Schwerer Inputfehler: "+e.getMessage());
       System.out.println("Bitte korrekten Integerwert eingeben:");
     }
     catch(NumberFormatException g)
     { System.out.println("Inputfehler: "+g.getMessage());
```

```
      System.out.println("Bitte korrekten Integerwert eingeben:");
    }
  } while (rc==0);
  return w;
}

public static double eindouble()
{ int rc;
  double w=-99999999999999.99999;
  BufferedReader in
       = new BufferedReader(new InputStreamReader(System.in));
  do
  { rc=0;     //Fehlervermutung
    try
    { w=Double.valueOf(in.readLine().trim()).doubleValue();
      rc=1;   //kein Fehler <=> rc=1
    }
    catch(IOException e)
    { System.out.println("Schwerer Inputfehler: "+e.getMessage());
      System.out.println("Bitte korrekten Doublewert eingeben:");
    }
    catch(NumberFormatException g)
    { System.out.println("Inputfehler: "+g.getMessage());
      System.out.println("Bitte korrekten Doublewert eingeben:");
    }
  } while (rc==0);
  return w;
}

public static String einstring()
{ int rc;
  String st1= new String();
  BufferedReader in
       = new BufferedReader(new InputStreamReader(System.in));
  do
  { rc=0;     //Fehlervermutung
    try
    { st1=in.readLine().trim();
      rc=1;   //kein Fehler <=> rc=1
    }
    catch(IOException e)
    { System.out.println("Schwerer Inputfehler: "+e.getMessage());
      System.out.println("Bitte korrekten Stringwert eingeben:");
    }

  } while (rc==0);
  return st1;
}

public static char einchar()
{ char x='#';  /* Initialisierung */
  int rc;
  String st1= new String();
  BufferedReader in
       = new BufferedReader(new InputStreamReader(System.in));
  do
  { rc=0;     //Fehlervermutung
```

```
      try
      { st1=in.readLine();
        if (st1.length()>0)
        { x=st1.toCharArray()[0];
          rc=1;
        }
        else
        {                                    System.out.println("Achtung:
Laenge("+st1+")="+st1.length());
          System.out.println("Bitte  nur  LOWER-CASE-Zeichen  von  der
Tastatur eingeben!");
        }
      }
      catch(IOException e)
      { System.out.println("Schwerer Inputfehler: "+e.getMessage());
        System.out.println("Bitte korrekten Stringwert eingeben:");
      }

    } while (rc==0);
    return x;
  }


  public static String estrOt()
  { /* wie Methode einstring() , aber ohne trim() des Eingabestrings
*/
    int rc;
    String st1= new String();
    BufferedReader in
         = new BufferedReader(new InputStreamReader(System.in));
    do
    { rc=0;      //Fehlervermutung
      try
      { st1=in.readLine();
        rc=1;    //kein Fehler <=> rc=1
      }
      catch(IOException e)
      { System.out.println("Schwerer Inputfehler: "+e.getMessage());
        System.out.println("Bitte korrekten Doublewert eingeben:");
      }

    } while (rc==0);
    return st1;
  }
}
```

Quelltext der Klasse `SelectMethods.java`

```
import java.io.*;
import java.util.*;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
import oracle.jdbc.pool.OracleDataSource;

public class SelectMethods {

	/**
	 * @return Connection : instance of the class Connection
	 */
	public static Connection connect() throws SQLException {
		String driver;
		OracleDataSource ods = new OracleDataSource();

		driver = "oracle.jdbc.driver.OracleDriver";
		Connection dbConnection = null;

		// loading of the driver
		try {
			Class.forName(driver).newInstance();
		} catch (Exception e) {
			System.out.println("An error occured when loading
the driver: "
					+ e.getMessage());
		}

		// creation of the connection instance named dbConnection
		try {

	ods.setURL("jdbc:oracle:thin:bachelor/maximm@localhost:1521/XE"
);
			dbConnection = ods.getConnection();
		} catch (SQLException e) {

			System.out
					.println("The attempt to connect to the
database failed!");
			System.out.println(e.getMessage());
		}
		return dbConnection;
	}

	/**
	 * @param command
	 *            : the command we want to run
	 */
	public static String startCommand(String symbol) {
		String response = "";
		try {

			// creation of the process
			Process p = Runtime.getRuntime().exec(
					"c:\\mizar\\findVoc " + symbol);
			InputStream in = p.getInputStream();

			// we get the output flow of the command
			StringBuffer build = new StringBuffer();
			char c = (char) in.read();

			while (c != (char) -1) {
```

```
                    build.append(c);
                    c = (char) in.read();
                }

                response = build.toString();

                // we display the response
                String[] buffer = response.split("\n");
                int matches = buffer.length - 2;
                // minus 2 to remove the first two lines that do not
contain any
                // relevant information
                for (int i = 0; i < buffer.length; i++) {
                    if (buffer[i].length() < 2
                            ||
buffer[i].startsWith("vocabulary")) {
                        matches--;
                    }
                }

            } catch (Exception e) {
                System.out.println("\n An unknown problem has
occured ");
            }
            return response;
        }

        public static void selectElement() {
            // StringBuffer out = new StringBuffer();
            // String output = "";
            // int matches = 0;
            ArrayList<String> resultSet = new ArrayList<String>();
            int choice = 0;
            String column = "text";
            String searchWord = "";

            // choose whether you result should be either a
definition, a
            // registration, a theorem or a notation
            String text = "";
            // text is the piece of text you are looking for in the
database

            String table = "";
            String sql = "";// sql builds the select statement
            int redefine = -1;
            // redefine enables us to choose between a redefine-block
and a simple
            // definition

            int existential = -1;
            // existential enables us to choose between an
existential and a
            // non-existential registration

            System.out
```

```
                        .println("Please, choose a number from 1, 2, 3,
4, 5 before going on.");
            System.out.println("1. Definition");
            System.out.println("2. Notation");
            System.out.println("3. Registration");
            System.out.println("4. Theorem");
            System.out.println("5. Article");
            do {
                  choice = IO1.einint();
                  if (choice == 1) {
                        table = "definitions";
                  } else if (choice == 2) {
                        table = "notations";
                  } else if (choice == 3) {
                        table = "registrations";

                  } else if (choice == 4) {
                        table = "theorems";
                  } else if (choice == 5) {
                        table = "absfiles";
                  } else {
                        System.out
                                    .println("Please, choose a number
from 1, 2, 3, 4 before going on.");
                        System.out.println("1. Definition");
                        System.out.println("2. Notation");
                        System.out.println("3. Registration");
                        System.out.println("4. Theorem");
                        System.out.println("5. Article");
                  }
            } while (choice < 1 || choice > 5);
            if (choice == 5) {
                  column = "description";
            }

            if (choice == 1 || choice == 2 || choice == 3) {
                  System.out.println(table.toUpperCase());
                  System.out
                              .println("Please, enter the piece of text
you are looking for.");
                  // There should'nt be any whitespace between xy and
the text around
                  // it

                  text = IO1.einstring();
                  text = text.replace("xy", "%");
            }
            // xy stands for any expression we don't exactly know how
it could
            // look like. So we replace xy with % because of the fact
that
            // % also stands for any expression.
            if (choice == 1) {
                  searchWord = getSearchWord(text);
                  text = text.replace("ww", "");
                  do {
                        System.out
```

```
                                        .println("Type 1 if you want the
definitions to be redefine, otherwise type 0");
                              redefine = IO1.einint();
                        } while (redefine != 1 && redefine != 0);
                        sql = "select fileID, text, pattern , defName from "
+ table
                                    + " where text like '" + "%" + text + "%"
                                    + "' and pattern like '" + "%" +
searchWord + "%"
                                    + "' and redefine = " + redefine;
                  }

                  else if (choice == 3) {

                        do {
                              System.out
                                        .println("Type 1 if you want the
registrations to be existential, otherwise type 0");
                              existential = IO1.einint();
                        } while (existential != 1 && existential != 0);
                        sql = "select fileID, text from " + table + " where
text like '"
                                    + "%" + text + "%" + "' and existential
=" + existential;
                  }

                  else if (choice == 4) {
                        System.out
                                    .println("Please, enter the piece of text
or\n the name of the theorem that you are looking for.");
                        text = IO1.einstring();
                        text = text.replace("xy", "%");
                        sql = "select fileID, text, theoname from " + table
                                    + " where text like '" + "%" + text + "%"
+ "'";

                  } else if (choice == 5) {
                        System.out.println(table.toUpperCase());
                        System.out
                                    .println("Enter a mathematical expression
which you suppose it might be treated in a Mizar article.");
                        text = IO1.einstring();
                        text = text.replace("xy", "%");
                        sql = "select fileID, description from " + table
                                    + " where description like '" + "%" +
text + "%" + "'";

                  } else {

                        sql = "select fileID, text from " + table + " where
text like '"
                                    + "%" + text + "%" + "'";
                  }
                  System.out.println(sql);
                  try {
                        // building of the connection instance using the
method connect()
```

```
                Connection con = connect();
                Statement st;
                ResultSet result;
                st = con.createStatement();

                result = st.executeQuery(sql);
                while (result.next()) {
                      Statement st1;
                      st1 = con.createStatement();
                      ResultSet res;
                      int fileID = result.getInt("fileID");
                      String sql1 = "select fileName from absfiles
where fileID = "
                                  + fileID;

                      res = st1.executeQuery(sql1);
                      if (res.next()) {
                            if (choice != 1 && choice != 4) {
                                  resultSet.add("Article:      "      +
res.getString("fileName")
                                              +          "\n"          +
result.getString(column));
                            } else if (choice == 1) {
                                  resultSet.add("Defined term: "
                                              +
result.getString("pattern") + "  Article: "
                                              +
res.getString("fileName") + " "
                                              + "    Definition's name:
"
                                              +
result.getString("defName") + "\n"
                                              +
result.getString(column));
                            } else if (choice == 4) {
                                  resultSet.add("Article:      "      +
res.getString("fileName")
                                              + "    Theorem's name: "
                                              +
result.getString("theoName") + "\n"
                                              +
result.getString(column));
                            }
                      }
                      st1.close();
                }
                st.close();
          } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("SQL Exception has been thrown");

          }
          System.out.print("There is(are) " + resultSet.size()
                      + " match(es) for your entry!\n\n");
          int dis = 0;
          int todisplay = 0;
          boolean condition = false;
```

```
            int size = resultSet.size();
            int increment = 0;
            String yn = "";
            if (size > 0)
                  do {
                        System.out
                                    .println("Enter the number of
matches you want to get displayed.");
                        todisplay = IO1.einint();
                  } while (todisplay > size);
            dis = todisplay;
            do {
                  if (dis == size)
                        condition = true;
                  for (int i = increment; i < dis; i++) {
                        System.out.println(resultSet.get(i));
                  }
                  if (size > 0 && size - dis > 0)
                        do {
                              int check = 0;
                              if (size - dis >= todisplay)
                                    check = todisplay;
                              else {
                                    check = size - dis;
                              }
                              System.out.println("Would you like to get
the next "
                                          + check + " match(es) ?(y|n)");
                              yn = IO1.einstring();
                        }     while     (!yn.equalsIgnoreCase("y")     &&
!yn.equalsIgnoreCase("n"));
                  if (yn.equalsIgnoreCase("y")) {
                        increment = dis;
                        if (dis + todisplay >= size) {
                              dis = size;
                        } else
                              dis += todisplay;
                  } else {
                        condition = true;
                  }

            } while (condition == false);
      }

      public static String getSearchWord(String text) {
            // The searchWord has to be enclosed by the string sw
            String searchWord = "";
            String[] ret = null;
            try {
                  if (text.contains("ww")) {
                        ret = text.split("ww");
                        searchWord = ret[1];
                  } else {
                        searchWord = text;
                  }
            } catch (ArrayIndexOutOfBoundsException ae) {
                  System.out.println(ae);
```

```
            }
            return searchWord;
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            int menu = -1;
            do {
                  System.out.println("-----------------MIZAR-KMS------
------------");
                  System.out
                              .println("1.  findVoc  Command:  Please
enter a text whithout whitespace. ");
                  System.out.println("2.  select  program  to  display
Mizar elements.");
                  System.out.println("0.  Type  0  to  stop  the
application.");
                  menu = IO1.einint();
                  switch (menu) {
                  case 1: {
                        System.out
                                    .println("Enter a Mizar-Symbol which
you want information about");
                        String symbol = IO1.einstring();
                        System.out.println(startCommand(symbol));
                        break;
                  }
                  case 2: {
                        selectElement();
                        break;
                  }
                  }

            } while (menu != 0);
      }

}
```

Quelltext der Klasse `MizarParser.java`

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;

import org.xml.sax.helpers.*;

public class MizarParser {
      public static void main(String[] args) {
            parser();
      }

      public static void parser() {
```

```
            String filename = "";
            do {
                  System.out
                              .println("Mizar Parser: Enter the name of
the XML file to be parsed? (Please, the extension has to be .xml
!)");
                  filename = IO1.einstring();
            } while (filename.length() <= 4);
            MyContentHandler handler = new MyContentHandler();
            MyErrorHandler ehandler = new MyErrorHandler();
            System.out.println("Attempt: XML-File = " + filename + "
to open");
            parseXmlFile(filename, handler, ehandler, true);
      }

      public static class MyContentHandler implements ContentHandler
{
            String vocabularies = "vocabularies ";
            String  requirements  =  "requirements  BOOLE,  SUBSET,
NUMERALS, ARITHM, REAL;";
            String reservation = "";
            String theorem = "";
            String article = "";
            String proof = "";
            ArrayList<String> store1 = new ArrayList<String>();
            ArrayList<String> store2 = new ArrayList<String>();
            String vocab = "";
            /* proof is the generated skeleton steps for the written
theorem */
            int cm = 0;/* cm is equal to 1 if the XML-element has the
type #PCDATA */
            int key = 0;/* key is equal to 1 if the XML-element is a
keyword */
            int counter1 = 0, counter2 = 0;
            String[]  keywords  =  {  "contradiction",  "ex",  "for",
"holds", "iff",
                        "implies", "is", "not", "or", "st", "thesis" };
            // keywords contain some of the keywords that could occur
in a theorem
            int ra = 0; /* ra =1 <=> row active */
            int re = 0;/* re =1 <=> reservation is active */
            String actValue = ""; /* Value of the current XML element
*/
            RandomAccessFile raf, rac; /* Writing to random access
file */
            String dsn; /* Name of the output file */
            String  conList  =  "conList.txt";/*  Name  of  the  control
file */
            String errorMessages = "";/*
                                                * Error messages to
write in the control file
                                                */

            public void startDocument() {
                  System.out.println("Start of the Parsing: ");
            }
```

```
            public void endDocument() {
                  System.out.println("End of the Parsing: " + dsn + "
closed.");
                  System.out
                              .println("Open the file conList to check
whether there is an error in the XML-File.");
            }

            public void startElement(String uri, String localName,
String qName,
                        Attributes attributes) throws SAXException {

                  AttributesImpl a1 = new AttributesImpl(attributes);

                  if (qName.compareTo("reservation") == 0) {
                        reservation += "reserve ";
                        re = 1;
                  }
                  if (qName.compareTo("theorem") == 0) {
                        ra = 1;
                  }
                  /* Processing of the elements within a theorem */
                  if (ra == 1) /* ra==1 <=> row is active */
                  {  /*  figuring  out  whether  the  element  has  an
attribute */
                        /* Only elements with the type (#PCDATA) have
attributes */
                        if (a1.getLength() > 0) {// if
                              // (a1.getQName(0).compareTo("type")
                              // == 0)
                              cm = 1;
                              for (int i = 0; i < keywords.length; i++)
{
                                    if
(qName.equals(keywords[i].trim()))
                                          key = 1;
                              }
                        }
                  }
                  // the word x is not correctly spelled

                  //  ------------------------------------------------
---------//
                  if (qName.compareTo("reservation-segment") == 0) {
                        if (counter1 > 0) {
                              reservation += ", ";
                        }
                        counter1++;
                  }

            }

            public void characters(char[] ch, int start, int length)
                        throws SAXException {
                  String h = null;
                  h = new String(ch, start, length);
```

```
                actValue = h;
        }

        public void skippedEntity(String name) throws
SAXException {
        }

        public void processingInstruction(String target, String
data)
                throws SAXException {
        }

        public void ignorableWhitespace(char[] ch, int start, int
length)
                throws SAXException {
        }

        public void endElement(String uri, String localName,
String qName)
                throws SAXException {
              /* To set the article name */
              if (qName.compareTo("article") == 0) {
                article = actValue;
                dsn = article + ".miz";

              }
              //  ------------------------------------------------
---------//
              if (qName.compareTo("identifiers") == 0) {
                if (counter2 > 0) {
                      reservation += ", " + actValue;
                } else {
                      reservation += "" + actValue;
                }
                counter2++;
              }
              //  ------------------------------------------------
---------//
              if (qName.compareTo("reserved-identifiers") == 0) {
                reservation += " for ";
              }
              //  ------------------------------------------------
---------//
              if (qName.compareTo("type-expression") == 0) {
                // theorem = theorem + actValue;
                if (re == 1)
                      reservation += actValue;
              }
              //  ------------------------------------------------
---------//
              if (qName.compareTo("reservation-segment") == 0) {
                counter2 = 0;
              }

              //  ------------------------------------------------
---------//
              if (qName.compareTo("reservation") == 0) {
```

```
                    reservation = reservation + ";\r\n";
                    re = 0;
               }
               //  ------------------------------------------------
---------//
               if (qName.compareTo("theorem") == 0) {
                    ra = 0;

               }
               if (ra == 1) {
                    if (cm == 1) {
                         theorem = theorem + " " + actValue;
                         //--------------------------------------
------------------
                         // ------------------------------------
                         //  This  block  is  needed  to  provide  the
many different
                         // articles that have to be imported into
the vocabularies
                         // directive
                         String voc = "";
                         String[] out = actValue.split(" ");
                         for (int i = 0; i < out.length; i++) {
                              voc                                   =
mizHelpGen.genVocabularies(out[i]);
                              if  (!voc.trim().equals("HIDDEN")  &&
voc.trim() != ("")) {
                                   store1.add(voc.trim());
                              }
                         }
                         // this  for  loop  let  us  ensure  that  an
article will not
                         // occurred twice
                         for (int i = 0; i < store1.size(); i++) {
                              if (!vocab.contains(store1.get(i))
                                        &&  store1.get(i).length()
> 3) {
                                   vocab += store1.get(i) + "\n";
                              }
                         }
                         //--------------------------------------
------------------
                         // ------------------------------------

                         cm = 0;

                    }
                    if (key == 1) {
                         if (!qName.equals(actValue.trim())) {
                              errorMessages  +=  "The  keyword  "  +
actValue
                                        +   "   is   not   correctly
spelled\r\n";
                         }
                         key = 0;
                    }
```

```
                }
                //  ------------------------------------------------
---------//
                // theorem = theorem + actValue;
                if (qName.compareTo("section") == 0) {
                      String[] buffer = vocab.split("\n");
                      // This for loop is important in order to put
the coma where
                      // there have to be
                      for (int i = 0; i < buffer.length; i++) {
                            if (i < buffer.length - 1) {
                                  vocabularies += buffer[i] + " ,";
                            } else {
                                  vocabularies += buffer[i] + ";";
                            }
                      }
                      proof = mizHelpGen.genProof(theorem);
                      theorem += "\r\nproof\r\n" + proof + "end;";
                      try {
                            raf = new RandomAccessFile(dsn, "rw");
                            raf.setLength(0);
                            raf.writeBytes("environ\r\n");
                            // the vocabularies directive has to be
included
                            // when there is at least one file found.
                            if (vocabularies.length() > 15)
                                  raf.writeBytes(vocabularies        +
"\r\n");
                            raf.writeBytes(requirements + "\r\n");
                            raf.writeBytes("begin\r\n");
                            raf.writeBytes(reservation);
                            raf.writeBytes(theorem);
                            raf.close();
                      } catch (IOException ex1) {
                            System.out
                                        .println("An error has occured
when opening/writing "
                                                    + dsn + " : " +
ex1.toString());
                      }
                      try {
                            rac    =    new    RandomAccessFile(conList,
"rw");
                            rac.setLength(0);
                            rac.writeBytes(errorMessages);
                            rac.close();
                      } catch (IOException ex1) {
                            System.out
                                        .println("An error has occured
when opening/writing "
                                                    + conList + " : " +
ex1.toString());
                      }

                }

          }
```

```
            public   void   endPrefixMapping(String   prefix)   throws
SAXException { // System

            }

            public void startPrefixMapping(String prefix, String uri)
                        throws SAXException {
            }

            public void setDocumentLocator(Locator locator) {
            }
      }

      public static class MyErrorHandler implements ErrorHandler {
            public   void   warning(SAXParseException   ep)   throws
SAXException {
                  System.out.println("Parser   meldet   WARNUNG:   "   +
ep.toString());
                  System.out.println("an  der  Entity            :  "  +
ep.getPublicId());
                  System.out.println("Zeile,Spalte               :  "  +
ep.getLineNumber()
                              + "," + ep.getColumnNumber());
            }

            public   void   error(SAXParseException   ep)   throws
SAXException {
                  System.out.println("Parser   meldet   FEHLER   :   "   +
ep.toString());
                  System.out.println("an  der  Entity            :  "  +
ep.getPublicId());
                  System.out.println("Zeile,Spalte               :  "  +
ep.getLineNumber()
                              + "," + ep.getColumnNumber());
            }

            public   void   fatalError(SAXParseException   ep)   throws
SAXException {
                  System.out.println("Fataler  FEHLER  !!!      :  "  +
ep.toString());
                  System.out.println("an  der  Entity            :  "  +
ep.getPublicId());
                  System.out.println("Zeile,Spalte               :  "  +
ep.getLineNumber()
                              + "," + ep.getColumnNumber());
                  System.exit(3);
            }
      }

      public     static     void     parseXmlFile(String     filename,
MyContentHandler handler,
                  MyErrorHandler ehandler, boolean val) {
            try {
                  SAXParserFactory              factory              =
SAXParserFactory.newInstance();
                  factory.setValidating(val);
```

```
                SAXParser saxpars1 = factory.newSAXParser();
                XMLReader read1 = saxpars1.getXMLReader();
                read1.setContentHandler(handler);
                read1.setErrorHandler(ehandler);
                boolean w1 = saxpars1.isValidating();
                if (w1)
                     System.out.println("--->       The       Parser
validates.");
                String             h1             =             new
File(filename).toURI().toURL().toString();
                System.out.println("URI = " + h1);
                read1.parse(h1);
           } catch (SAXParseException ep) {// A parsing error
occurred; the xml
                // input is not valid
                System.out.println("SAx-Parser-Exception in " +
filename + " :\n");
                System.out.println("Parser throws ERROR : " +
ep.toString());
                System.out.println("at the Entity : " +
ep.getPublicId());
                System.out.println("Line,Column : " +
ep.getLineNumber()
                          + "," + ep.getColumnNumber());
                System.out.println(ep.getMessage());
           } catch (SAXException e) {// A parsing error occurred;
the xml input
                // is
                // not valid
                System.out.println("The " + filename
                          + "is not conform to the defined DTD" + "
:\n" + e);
                System.out.println(e.getMessage());
           } catch (ParserConfigurationException e) {
                System.out.println("A     Parser     configurations
problem.");
                System.out.println(e.getMessage());
           } catch (IOException e) {
                System.out.println("XML-File = " + filename + "
could not be open");
                System.out.println(e.getMessage());
           }
      }
}
```

Quelltext der Klasse `mizHelpGen.java`

```
public class mizHelpGen {

      /**
       * This method returns the characters around the keyword
implies
       *
       * @param theorem
```

```
      * @return an array which contains the both parts around the
keyword implies
      */

     public static String[] implies(String theorem) {
          if (theorem.endsWith(";"))
               theorem = theorem.substring(0, theorem.length() -
1);
          // to remove the semicolon at the end of the theorem
          int no = 0;// the number of occurrence of the word
implies
          int k = 0, i = 0;
          int index = 0; // index of the middle implies
          String[] output = new String[2];
          String buffer = "", part1 = "", part2 = "";// help
variables
          /*
           * The parts above are the ones needed to provide the
different skeleton
           * steps
           */
          if (theorem.contains(" implies ")) {
               String[] parts = theorem.split(" implies ");
               no = parts.length - 1;

               do {
                    buffer += theorem.charAt(k);
                    if (buffer.length() > 7 || theorem.charAt(k) ==
32) {
                         buffer = "";
                    } else if (buffer.equals("implies")) {
                         i++;
                         if (i == (no / 2) + 1)
                              index = k - 7;
                         // 7 is the length of the literal string
implies
                    }
                    k++;
               } while (k < theorem.length());
               part1 = theorem.substring(0, index);
               part2 = theorem.substring(index + 8,
theorem.length());
          }
          output[0] = part1;
          output[1] = part2;
          return output;
     }

     /**
      * This method returns the characters around the keyword holds
      *
      * @param theorem
      * @return an array which contains the different parts around
the keyword
      *         holds
      */
     public static Object[] holds(String theorem) {
```

```
            if (theorem.endsWith(";"))
                  theorem = theorem.substring(0, theorem.length() -
1);
            // to remove the semicolon at the end of the theorem
            int no = 0;// the number of occurrence of the word
implies
            int k = 0, i = 0;
            int index = 0; // index of the middle implies

            Object[] output = new Object[3];
            String buffer = "", part3 = "", part4 = "";
            /*
             * The parts above are the ones needed to provide the
different skeleton
             * steps
             */
            String temporary = "";
            // help variables
            if (theorem.contains(" holds ")) {
                  String[] parts = theorem.split(" holds ");
                  no = parts.length - 1;

                  do {
                        buffer += theorem.charAt(k);
                        if (buffer.length() > 5 || theorem.charAt(k) ==
32) {
                              buffer = "";
                        } else if (buffer.equals("holds")) {
                              i++;
                              if (i == (no / 2) + 1)
                                    index = k - 5;
                              // 5 is the length of the literal string
holds
                        }
                        k++;
                  } while (k < theorem.length());

                  temporary = theorem.substring(0, index);
                  if (theorem.substring(index + 6, theorem.length())
!= "")
                        part4 = theorem.substring(index + 6,
theorem.length());
            }

            //-----------------------------------st-----------------
----------------

            no = 0;
            k = 0;
            i = 0;
            index = 0;
            buffer = "";
            if (temporary.contains(" st ")) {
                  String[] parts = temporary.split(" st ");
                  no = parts.length - 1;

                  do {
```

```
                    buffer += temporary.charAt(k);
                    if (buffer.length() > 2 || temporary.charAt(k)
== 32) {
                         buffer = "";
                    } else if (buffer.equals("st")) {
                         i++;
                         if (i == (no / 2) + 1)
                              index = k - 2;
                         // 2 is the length of the literal string
st
                    }
                    k++;
               } while (k < temporary.length());
               part3     =     temporary.substring(index     +     3,
temporary.length());
               temporary = temporary.substring(0, index);
          } else {
               part3 = "";
          }

          //  ----------------------------------for----------------
--------------
          String[] fors = temporary.split("for");
          Object[]  loci  =  new  Object[2];//  loci  is  the  loci-
declaration
          String[] variables = new String[fors.length - 1];
          String[] type = new String[fors.length - 1];
          for (int n = 1; n < fors.length; n++) {

               if (fors[n].contains(" being ")) {
                    String[] parts = fors[n].split(" being ");
                    variables[n - 1] = parts[0];
                    type[n - 1] = parts[1];

               } else {
                    variables[n - 1] = fors[n];
                    type[n - 1] = "";
               }
               loci[0] = variables;
               loci[1] = type;
          }
          // part3 is the part preceded by the keyword assume
          // part4 is the part preceded by the keyword thus
          output[0] = loci;
          output[1] = part3;
          output[2] = part4;

          return output;
     }

     /**
      * This method generates the skeleton steps for a proof that
contains the
      * keyword holds
      *
      * @param theorem
      *            is the given formula containing the word holds
```

```
	 * @return String consists of the skeleton steps of the theorem
	 */

	public static String proofHolds(String theorem) {
		// call to the method holds()
		Object[] b = holds(theorem);
		String output = "";

		for (int i = 0; i < ((String[]) ((Object[])
b[0])[0]).length; i++) {
			output += "    let " + ((String[]) ((Object[])
b[0])[0])[i];
			if (((String[]) ((Object[]) b[0])[1])[i] != "") {
				output += " be " + ((String[]) ((Object[])
b[0])[1])[i];
			}
			output += ";\r\n  ";
		}

		// split b[1] based on the sign &
		if ((String) b[1] != "") {
			if (((String) b[1]).contains(" implies ")) {
				output += proofImplies((String) b[1]);
			} else if (((String) b[1]).contains(" & ")) {
				String[] pts = ((String) b[1]).split("&");
				for (int i = 0; i < pts.length; i++) {
					output += "assume " + pts[i].trim() +
";\r\n";
				}
			} else
				output += "assume " + ((String) b[1]).trim() +
";\r\n";
		}

		// split parts[1] based on the sign &
		if (((String) b[2]).contains(" ex ")) {
			output += proofEx(((String) b[2]));
		} else if (((String) b[2]).contains(" implies ")) {
			output += proofImplies((String) b[2]);
		} else
			output += "thus " + b[2] + ";\r\n";

		return output;
	}

	/**
	 * This method generates the skeleton steps for a proof that
contains the
	 * keyword implies
	 *
	 * @param theorem
	 *            is the given formula containing the word implies
	 * @return String consists of the skeleton steps of the theorem
	 */

	public static String proofImplies(String theorem) {
		String output = "";
```

```
            String[] parts = implies(theorem);
            if (parts[0].contains(" & ")) {
                  String[] pts = parts[0].split(" & ");
                  for (int i = 0; i < pts.length; i++) {
                        output += "assume " + pts[i].trim() + ";\r\n";
                  }
            } else
                  output += "assume " + parts[0].trim() + ";\r\n";

            if (parts[1].contains(" holds ")) {
                  output += proofHolds(parts[1]);

            } else if (parts[1].contains(" ex ")) {
                  output += proofEx(parts[1]);
            }

            else
                  output += "thus " + parts[1].trim() + ";\r\n";

            return output;
      }

      /**
       * This method generates the skeleton steps for a proof that
contains the
       * keyword implies
       *
       * @param theorem
       *            is the given formula containing the word iff
       * @return String consists of the skeleton steps of the theorem
       */
      public static String proofIff(String theorem) {
            // the part before iff has to be assumed
            // the part after iff has to be considered as proved
            if (theorem.endsWith(";"))
                  theorem = theorem.substring(0, theorem.length() -
1);
            String output = "";
            String[] parts = null;
            if (theorem.contains(" iff ")) {// if
                  parts = theorem.split(" iff ");
                  output += "hereby \r\n";
                  output += "assume " + parts[0] + ";\r\n";
                  if (parts[1].contains(" implies ")) {
                        output += proofImplies(parts[1]);
                  } else if (parts[1].contains(" holds ")) {
                        output += proofHolds(parts[1]);
                  } else if (parts[1].contains(" ex ")) {
                        output += proofEx(parts[1]);
                  } else
                        output += "thus " + parts[1] + ";\r\n";
                  output += "end;\r\n";
                  output += "assume " + parts[1] + ";\r\n";
                  if (parts[0].contains(" implies ")) {
                        output += proofImplies(parts[0]);
                  } else if (parts[0].contains(" holds ")) {
                        output += proofHolds(parts[0]);
```

```
                } else if (parts[0].contains(" ex ")) {
                      output += proofEx(parts[0]);
                } else
                      output += "thus " + parts[0] + ";\r\n";

          }// end if
          return output;
     }

     /**
      * This method generates the skeleton steps for a proof that
contains the
      * keyword ex
      *
      * @param theorem
      *            is the given formula containing the word iff
      * @return String consists of the skeleton steps of the theorem
      */
     public static String proofEx(String theorem) {
          String output = "";
          if (theorem.contains(" ex ")) {
                String[] part = theorem.split("st");
                // to remove the word ex
                output += "consider " + part[0].substring(3).trim()
+ ";\r\n";
                if (part[0].contains(" being ")) {
                      String[] take = part[0].split("being");
                      output += "take " + take[0].substring(3).trim()
+ ";\r\n";
                } else
                      output += "take " + part[0].substring(3).trim()
+ ";\r\n";
                if (part[1].contains(" & ")) {
                      String[] pt = part[1].split("&");
                      for (int i = 0; i < pt.length; i++) {
                            output += "thus " + pt[i].trim() +
";\r\n";
                      }
                } else
                      output += "thus " + part[1].trim() + ";\r\n";
          }
          return output;
     }

     /**
      * This method generates the skeleton steps for any proof that
contains the
      *
      * @param theorem
      *            is the given formula that one wants to prove
      * @return String consists of the skeleton steps of the theorem
      */
     public static String genProof(String theorem) {
          String output = "";
          if (theorem.contains(" iff ")) {
                output = proofIff(theorem);
```

```
            }   else   if   (theorem.contains("   implies   ")   &&
theorem.contains(" holds ")) {
                  int posIm = 0;// index of implies
                  int posHol = 0;// index of holds
                  posIm = theorem.indexOf(" implies ");
                  posHol = theorem.indexOf(" holds ");
                  if (posIm > posHol) {
                        output = proofHolds(theorem);
                  } else {
                        output = proofImplies(theorem);
                  }

            } else if (theorem.contains(" implies ")) {
                  output = proofImplies(theorem);
            } else if (theorem.contains(" holds ")) {
                  output = proofHolds(theorem);
            }
            return output;
      }

      /**
       * @param symbol
       *              is the one we want to figure out the vocabulary
article it
       *            comes from
       *@return  article  is  the  article  to  be  imported  into  the
vocabularies
       *         directive
       */
      public static String genVocabularies(String symbol) {
            String voc = SelectMethods.startCommand("-w " + symbol);
            String[] output = voc.split("\n");
            String article = "";
            String[] out;
            String line = "no";
            for (int i = 0; i < output.length; i++) {
                  if (output[i].trim().startsWith("vocabulary"))
                        line = output[i];
            }
            if (line != "no") {
                  out = line.split(" ");
                  article = out[1];
            }
            return article;
      }

}
```

Quelltext der Klasse `KMS_MIZAR.java`

```
public class KMS_MIZAR {

      public static void main(String[] a) {
            int choice = -1;
            do {
```

```
                System.out.println("-----------------MIZAR-KMS------
-----------");
                System.out
                           .println("To  ensure  that  this  system
works properly you should have the abstr directory located ");
                System.out
                           .println("at  C:\\mizar.  In  additional,
make sure that an appropriate database\nhas already been designed as
well.");
                System.out
                           .println("------------------------------
----------------------");
                System.out.println("1.  Type  1  to  enter  the  Mizar
insert program.");
                System.out.println("2.  Type  2  to  enter  the  Mizar
select program.");
                System.out.println("3. Type 3 to run the Mizar Proof
Generator.");
                System.out.println("0.   Type   0   to   leave   the
application.");
                choice = IO1.einint();
                switch (choice) {
                case 1: {
                     int menu = -1;

                     do {
                          System.out
                                     .println("-----------------
MIZAR-KMS------------------");
                          System.out.println("1. Inserts all of the
Mizar articles.");
                          System.out
                                     .println(" This  task  needs  to
be performed when the system is being runned  for the first time.");
                          System.out.println("2.   Inserts   a   new
File.");
                          System.out
                                     .println(" This  task  needs  to
be performed when a new file has been added to the MML.");
                          System.out.println("3. Updates the Mizar-
KMS");
                          System.out.println("0. Type 0 to stop the
insert program.");

                          menu = IO1.einint();
                          switch (menu) {
                          case 1: {
                               String yn = "";
                               System.out
                                          .println("Do  you  really
want to complete this command? (Y/N)");
                               yn = IO1.einstring();
                               if (yn.equalsIgnoreCase("y")) {
                                    System.out
                                               .println("Please
wait.The System is running.\n");
                                    InsertUpdate.insertAllFiles();
```

```
                                    } else {
                                    }
                                    break;
                              }
                              case 2: {
                                    System.out
                                                .println("Please,    enter
an article's name from the MML.\n");

      InsertUpdate.insertFile(IO1.einstring());
                                    break;
                              }
                              case 3: {
                                    String yn = "";
                                    System.out
                                                .println("Do   you   really
want to complete this command? (Y/N)");
                                    yn = IO1.einstring();
                                    if (yn.equalsIgnoreCase("y")) {
                                          System.out
                                                      .println("Please
wait. The database is being updated.\n");
                                          InsertUpdate.update();
                                    } else {
                                    }
                                    break;
                              }
                              }

                        } while (menu != 0);

                        break;
                  }
                  case 2: {
                        int menu = -1;
                        do {
                              System.out
                                          .println("-----------------
MIZAR-KMS------------------");
                              System.out
                                          .println("1.  findVoc  Command:
Please enter a text whithout whitespace. ");
                              System.out
                                          .println("2. select program to
display Mizar elements.");
                              System.out.println("0. Type 0 to stop the
select program.");
                              menu = IO1.einint();
                              switch (menu) {
                              case 1: {
                                    System.out
                                                .println("Enter  a  Mizar-
Symbol which you want information about");
                                    String symbol = IO1.einstring();

      System.out.println(SelectMethods.startCommand(symbol));
                                    break;
```

```
                              }
                              case 2: {
                                    SelectMethods.selectElement();
                                    break;
                              }
                              }

                        } while (menu != 0);
                        break;
                  }
                  case 3: {
                        System.out
                                    .println("-----------------MIZAR-
KMS-------------------");
                        MizarParser.parser();
                        break;
                  }
                  }// end switch

            } while (choice != 0);
      }
}
```

Anhang C: SQL-Skripte

C.1: Skript zur Erstellung der DB-Tabellen (createTables.sql)

```
create  table absFiles(
fileID integer not null constraint fileID_a primary key,
fileName varchar2(1000) not null,
description varchar2(4000) not null,
defNo integer not null,
notNo integer not null,
regNo integer not null,
theoNo integer not null,
locked integer not null check(locked = 0 or locked = 1));


create table definitions(
defID integer not null constraint defID primary key,
fileID integer not null constraint fileID_b references
absFiles(fileID),
text varchar2(4000) not null,
redefine integer not null check(redefine = 0 or redefine = 1)
pattern varchar2(4000) not null,
defName varchar2(100) not null);


create table notations(
notID integer not null constraint notID primary key,
fileID integer not null constraint fileID_c references
absFiles(fileID),
text varchar2(4000) not null
);


create table Registrations(
regID integer not null constraint regID primary key,
fileID integer not null constraint fileID_d references
absFiles(fileID),
text varchar2(4000) not null,
existential integer not null check(existential = 0 or existential =
1)
);


create table theorems(
theoID integer not null constraint theoID primary key,
fileID integer not null constraint fileID_e references
absFiles(fileID),
text varchar2(4000) not null,
theoName varchar2(4000)
)
```

C.2: Skript zur Erstellung der Sequenzen (sequence.sql)

```
create sequence file_seq
start with 1
increment by 1
nomaxvalue;
create sequence def_seq
start with 1
increment by 1
nomaxvalue;
create sequence not_seq
start with 1
increment by 1
nomaxvalue;
create sequence reg_seq
start with 1
increment by 1
nomaxvalue;
create sequence theo_seq
start with 1
increment by 1
nomaxvalue
```

C.3: Skript zur Erstellung der Trigger (trigger.sql)

```
CREATE  TRIGGER file_trig
BEFORE INSERT ON ABSFILES
FOR EACH ROW
BEGIN
SELECT file_seq.NEXTVAL INTO :NEW.fileid FROM dual;
END;
CREATE  TRIGGER def_trig
BEFORE INSERT ON DEFINITIONS
FOR EACH ROW
BEGIN
SELECT def_seq.NEXTVAL INTO :NEW.defid FROM dual;
END;
CREATE  TRIGGER not_trig
BEFORE INSERT ON NOTATIONS
FOR EACH ROW
BEGIN
SELECT not_seq.NEXTVAL INTO :NEW.notid FROM dual;
END;
CREATE  TRIGGER reg_trig
BEFORE INSERT ON REGISTRATIONS
FOR EACH ROW
BEGIN
SELECT reg_seq.NEXTVAL INTO :NEW.regid FROM dual;
END;
CREATE  TRIGGER theo_trig
BEFORE INSERT ON THEOREMS
FOR EACH ROW
BEGIN
SELECT theo_seq.NEXTVAL INTO :NEW.theoid FROM dual;
END;
```

Abkürzungsverzeichnis

MML	Mizar Mathematical Library
SQL	Structured Query Language
RDBMS	Relational Database Management System
SAX	Simple API for XML
DOM	Document Object Model
JDBC	Java Database Connectivity
URL	Unified Resource Locator
ERD	Entity Relationship Diagram
PRIK	Primary Key
FKEY	Foreign Key

Literaturverzeichnis

[1] Mizar System
http://mizar.org/system/

[2] http://mizar.org/language/vocabularies.html

[3] Mizar Syntax(Bauckus-Naur Form)
http://mizar.org/language/syntax.xml

[4] Freek Wiedijk's Page about Mizar
Writing a Mizar article in nine easy steps
http://www.cs.ru.nl/~freek/mizar/

[5] O. Forster
Analysis 1
vieweg Verlag

[6] Rolf Walter
Einführung in die Analysis 1
de Gruyter

[7] Skripte von Pr. Dr. phil. G. Büchel
www.nt.fh-koeln.de/fachgebiete/inf/buechel/GesamtskriptDB0506.pdf

[8] SQL Tutorial
http://www.w3schools.com/sql/

[9] SAX-Parser
http://www.saxproject.org/