



Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik
Institut für Nachrichtentechnik
Bereich für Informatik

Master Thesis

im Studiengang
Master of Science Technische Informatik

Entwicklung des Naprobe-Proof-State-Datentyps

von
Sebastian Zittermann

Erstbetreuer: Prof. Dr. phil. Gregor Büchel
Zweitbetreuer: Prof. Dr. Peter Koepke

Eingereicht am: 30. September 2011

Eidesstattliche Erklärung

Hiermit erkläre ich, Sebastian Zittermann, an Eides statt, dass ich die Masterarbeit, "Entwicklung des Naproche-Proof-State-Datentyps", selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.
Bonn, den 28. September 2011

Unterschrift

Inhaltsverzeichnis

0	Abkürzungsverzeichnis	7
1	Einführung	9
1.1	Projektumfeld	9
1.2	Motivation	10
1.3	Aufgabenstellung der Master-Thesis	11
1.3.1	Allgemein	11
1.3.2	Analyse des Ist-Systems von Naproche (Version 0.47)	11
1.3.3	Anforderungen an ein Soll-System (Version 0.5)	11
1.4	Gliederung	13
2	Grundlagen	14
2.1	Logische Grundlagen	14
2.1.1	Prädikatenlogik erster Stufe	14
2.1.2	Syntax der Sprachen erster Stufe	15
2.1.3	Semantik der Sprachen erster Stufe	18
2.1.4	ATP (Automated Theorem Prover) und TPTP (Thousands of Problems for Theorem Provers)	22
2.2	Linguistische Grundlagen	26
2.2.1	Diskursrepräsentationsstruktur und Beweisrepräsentationsstruktur	26
2.2.2	Kontrollierte natürliche Sprache	32
2.2.3	Definite Clause Grammar	35
2.3	Informationstechnische Grundlagen	36
2.3.1	Prolog	36
2.3.2	Komplexität	36
2.3.3	Abstrakte Datentypen	37
2.3.4	Persistente Daten	38

2.3.5	Backus-Naur-Form	38
2.3.6	CSV-Dateien	39
2.3.7	Legende der Diagramme	40
2.4	Terminologische Besonderheiten	41
2.4.1	Datentypen und Typentheorie	41
2.4.2	Prolog und Java	44
3	Ist-Zustand des Naproche-Systems (Version 0.47)	46
3.1	Das Naproche-Projekt	46
3.2	Das Webinterface	49
3.3	Übersichtsdiagramm	51
3.4	Die wesentlichen Prologmodule	54
3.4.1	Build PRS	54
3.4.2	Create Obligations	56
3.4.3	Discharge Obligations	61
4	Anforderungsprofil an die Version 0.5 des Naproche-Systems	63
4.1	Modifikationsziele	63
4.2	Umstrukturierung der Prologmodule	64
4.3	Neuentwicklung des Proof-State-Datentyps	65
4.4	Neuentwicklung eines Grafical User Interface	66
5	Implementation des Prototyps	67
5.1	Allgemeine Architektur des Programmsystems	67
5.1.1	Die Hauptmodule	68
5.1.2	Der Preparser	70
5.1.3	Macroparser, Microparser und Formula	75
5.1.4	Datenstruktur, Dateiaufbau und die ProofState-Datei	77
5.1.5	Verbindung zwischen Java und Prolog	85
5.2	Die Javamodule	86
5.2.1	Das Grafical User Interface (GUI)	88
5.2.2	Wesentliche Attribute und Methoden zur Beweisüberprüfung . . .	95
5.2.3	Weitere Methoden	101
5.3	TPTP-AxSel	102
5.4	Aufruf des Prototypen	103

6	Test von Naproche 0.5	104
6.1	Versuchsaufbau	104
6.2	Testergebnisse des Lasttests	107
6.3	Vergleich zum alten Naproche-System Version 0.47	110
6.3.1	Version 0.47 vs Version 0.5 ohne Beweisdaten	110
6.3.2	Version 0.47 vs Version 0.5 mit Beweisdaten	111
6.4	Interpretation der Testergebnisse:	113
7	Zusammenfassung	114
7.1	Fazit	114
7.2	Ausblick	115
8	Quellen	116
	Abbildungsverzeichnis	119
	Tabellenverzeichnis	121
9	Anhang	122
9.1	Ordnerstruktur der Überprüften Beweise	122
9.1.1	Burali-Forti	122
9.1.2	Landau	128
9.1.3	Group-Theory	132
9.2	Java-Quelltexte	135

0 Abkürzungsverzeichnis

In diesem Abkürzungsverzeichnis sind alle Abkürzungen enthalten, die im Rahmen dieser Master Thesis verwendet werden.

- ACE: Attempto Controlled English
- API: Application Programm Interface
- ATP: Automated Theorem Prover
- BNF: Backus-Naur-Form
- cnf: clausal normal forms
- CNL: Controlled Natural Language
- CSV: Comma-Separated Values bzw. Character Separated Values
- DCG: Definite Clause Grammar
- DRS: Discourse Representation Structure
- DRT: Discourse Representation Theory
- FLI: Foreign Language Interface
- fof: first order formula
- GUI: Grafical User Interface
- GULP: Graph Unification Logic Programming

- JNI: Java Native Interface
- JPL: Java/Prolog-Interface
- JRE: Java Runtime Environment
- JVM: Java Virtual Machine
- Naproche: Natural Language Proof Checking
- PRS: Proof Representation Structur
- TPTP: Thousands of Problems for Theorem Provers
- TPTP-AxSel: TPTP- Axiom Selection

1 Einführung

1.1 Projektumfeld

Die Master-Thesis wurde im Rahmen des Forschungsprojektes *Naproche* geschrieben. Der Begriff Naproche steht für *Natural Language Proof Checking*. Es handelt sich hierbei um ein Gemeinschaftsprojekt der Universitäten Bonn und Duisburg-Essen, sowie der Fachhochschule Köln. Die Professoren, die das Naproche-Projekt leiten, sind Prof. Dr. Peter Koepke von der mathematischen Logikgruppe der Universität Bonn, Prof. Dr. Bernhard Schröder von dem linguistischen Bereich der Universität Duisburg-Essen und Prof. Dr. Gregor Büchel aus dem Bereich Informatik der Fachhochschule Köln.

Das Ziel des Projektes ist es, aus der semi-formalen Sprache der Mathematik mit linguistischen und logischen Methoden eine CNL („controlled natural language“) zu entwickeln. Beweistexte in dieser CNL sollen von einer Beweisüberprüfungssoftware auf ihre mathematische Korrektheit überprüft werden. Im Rahmen von Naproche wird untersucht, inwieweit Beweistexte automatisch in eine prädikatenlogische Formel (erster Stufe) umgewandelt werden können.

Von der Version 0.4 bis zur Version 0.47 des Naproche-Systems wurde von Marcos Cramer und Daniel Kühlwein das System so modifiziert, dass es über das Naproche-Webinterface aufgerufen werden kann (siehe [5] und [26]). Eine Untersuchung des Naproche-Systems 0.47 wird im Kapitel 4 „Anforderungsprofil an die Version 0.5 des Naproche-Systems“ beschrieben.

Zur Zeit entsteht neben dieser Master-Thesis eine Dissertation von Marcos Cramer, wobei die Prolog-Module zur linguistischen und logischen Überprüfung weiterentwickelt werden. Alle Publikationen des Projektes über die Homepage von Naproche [<http://naproche.net>] erreichbar.

1.2 Motivation

Im aktuellen Naproche-Programmsystem muss ein Beweistext immer komplett geparkt (Überprüfung auf semantische Richtigkeit) und überprüft (auf logische Richtigkeit) werden. Als Beispieltexre wurden bei der Entwicklung des Naproche-Systems hauptsächlich Beweise von Landau [17] und Euklid [10] überprüft. Der als LaTeX-Text gespeicherte Beweiskorpus wurde im Laufe der Zeit jeweils sehr umfangreich und komplex. Nach Änderungen im Text des Beweiskorpus muss der Verarbeitungsprozess neu gestartet werden. Selbst kleine Änderungen am Beweistext, wie eine Ergänzung des Textes, führen immer dazu, dass der gesamte Text komplett neu geparkt und überprüft werden muss. Es ist nicht möglich, die Überprüfung zu pausieren, wenn man z.B. einen Fehler gefunden hat. All diese Punkte erschweren das Neuerstellen und Bearbeiten von Beweistexten, insbesondere wenn diese einen größeren Umfang haben. Der Vorgang zum Überprüfen eines Beweises findet zum heutigen Zeitpunkt (Version 0.47, siehe [5]) wie folgt statt:

- Vorverarbeiten des Eingabetextes
- Umwandlung der Vorverarbeitung in eine linguistische Struktur
- Erstellen der zu überprüfenden Aussagen aus der linguistischen Struktur
- Überprüfen der Aussagen mit Hilfe eines automatischen Beweisüberprüfers
- Rückmeldung der Überprüfungsergebnisse an den Anwender

1.3 Aufgabenstellung der Master-Thesis

1.3.1 Allgemein

In diesem Kapitel werden alle Aufgaben beschrieben, die für die Entw Mit dem im Rahmen der Master-Thesis zu entwickelnden Proof-State-Datentyp soll das Erstellen und Überprüfen von Naproche-Texten vereinfacht und beschleunigt werden. Damit einher gehen allgemeine Veränderungen der Systemarchitektur in den verschiedenen vorhandenen Prolog-Modulen, die bei der Entwicklung berücksichtigt werden müssen.

1.3.2 Analyse des Ist-Systems von Naproche (Version 0.47)

Ein wesentlicher Teil der Master-Thesis ist die Analyse des Ist-Zustandes. Hier soll die aktuelle Version des Naproche-Systems 0.47 untersucht und die wesentlichen Module dokumentiert werden. Weiterhin geht es darum, die bisherige linguistische und logische Arbeitsweise zur Beweisschrittprüfung zu verstehen, um aus dieser den Persistenzmechanismus (nicht flüchtige Speicherung der Daten) des Proof-State-Datentyps zu spezifizieren.

1.3.3 Anforderungen an ein Soll-System (Version 0.5)

Um zu verhindern, dass ein Satz des Beweistextes mehrfach überprüft wird, der weder geändert wurde, noch von der Änderung einer oder mehrerer Sätze betroffen ist, muss eine Änderungsverwaltung implementiert werden. Bei einer Änderung muss neben der sprachlichen Formulierung des jeweiligen Satzes auch die logische Verknüpfung zwischen den einzelnen Sätzen überprüft werden. Aus jeder logischen Aussage werden Prämissen abgeleitet, die wiederum zusammen in einer PRS (siehe Kapitel 2.2.1 dargestellt werden. Die Zusammenhänge zwischen den Beweisschritten müssen in der neuen Version 0.5 berücksichtigt werden. Zu jedem Beweis werden verschiedene Auswertungsformen wie die Darstellung der linguistischen Struktur erzeugt, die ebenfalls

für das neue System angepasst werden müssen.

Weitere Anforderungen sind:

- Implementierung eines Java-Prototypen mit Schnittstelle zu den ATP- und PRS-Modulen (siehe Kapitel 2.1.4).
- Speichern und Laden von Beweisen ermöglichen: Es soll u"ber den zu entwickelnden Prototypen Beweisdaten, wie der Beweistext und vorhandene Überprüfungsergebnisse, gespeichert und geladen werden können.
- Vorhandene Dokumente sollen leicht fortgesetzt werden können.
- Die Spezifikation des Proof-State-Datentyps soll in Bezug auf die PRS- und ATP-Formate des Naproche-Systems erfolgen.
- **Laufzeituntersuchung:**
Der implementierte Prototyp mit der Version 0.5 des Naproche-Systems soll gegenüber dem IST-System von der Naproche-Version 0.47 unter den folgenden Aspekten untersucht werden:
 - Laufzeitverhalten
 - Erweiterbarkeit des Korpus
 - Änderungsfreundlichkeit

1.4 Gliederung

Diese Master-Thesis ist in folgende Kapitel gegliedert:

Das Kapitel „Einführung“ gibt einen Überblick über das Projekt Naproche, sowie die Motivation und den Anforderungen hinter der Master-Thesis. In dem folgenden Kapitel „Grundlagen“ werden die mathematischen und logischen Grundlagen zum Verständnis der Beweisstruktur erläutert. Die Grundlagen zur Entwicklung von abstrakten Datentypen werden hier ebenfalls erklärt. Der Ist-Zustand des Naproche-Systems zum Beginn der Thesis (Version 0.47) wird in dem Kapitel „Ist-Zustand des Naproche-Systems (Version 0.47)“ anhand von Ablaufplänen und der Beschreibung der wichtigsten Module dargestellt. Das nächste Kapitel „Anforderungen an die Version 0.5 des Naproche-Systems“ befasst sich zunächst mit den langfristigen Zielen, wie das Naproche-System modifiziert werden sollte. Die konkret geplanten Umsetzungen, die sowohl die Prolog-Module als auch den Proof-State Datentyp betreffen, werden anschließend in den Unterkapiteln 4.2 und 4.3 näher spezifiziert. Das Kapitel „Implementation des Prototyps“ beinhaltet die allgemeine Architektur des Systems mit den Java- und Prolog-Modulen. Die Funktionen der verschiedenen Java-Klassen werden, wie das Design des Programmsystems und die wesentlichen Java-Methoden hier ebenfalls beschrieben. Die Beschreibung der Tests des neuen Systems 0.5 sowie deren Vergleich mit dem alten System 0.47 finden in dem Kapitel „Test von Naproche 0.5“ statt. Im „Zusammenfassung“ wird ein Resüme über die erstellte Arbeit gezogen und zukünftige Entwicklungsvorschläge gemacht. Das Kapitel „Quellen“ gibt die Quellen der Master-Thesis an. Der Java-Quelltext des Prototypen befindet sich ebenso wie drei bereits erfasste Beweistexte mit ihrer jeweiligen Beweisdatenstruktur im Kapitel „Anhang“.

2 Grundlagen

2.1 Logische Grundlagen

In diesem Kapitel werden die Grundbegriffe der Logik, die dem Naproche- Projekt zu Grunde liegen, kurz beschrieben. Im Mittelpunkt stehen hierbei die Sprachen der Prädikatenlogik erster Stufe, die syntaktisch und semantisch definiert werden. Weiterhin werden Diskursrepräsentationsstrukturen erläutert und deren Aufbau an dem Beispiel 2.3 gezeigt. Die Quellen zu diesem Kapitel sind das Buch „Einführung in die mathematische Logik“ [9], ein Paper von Naproche [15] sowie die Diplomarbeit von Daniel Kühlwein [16] und die Master-Thesis von Marcos Cramer [8]. Zusätzliche Quelle für das Thema DRS ist die Webseite [12].

2.1.1 Prädikatenlogik erster Stufe

Die Prädikatenlogik erster Stufe ist eine formale Sprache der mathematischen Logik. In ihr werden einzelne Terme mit Relationssymbolen zueinander in Beziehung gesetzt, wobei jeder Term aus mehreren Bestandteilen zusammengesetzt werden kann. Dadurch entsteht eine Struktur, die logisch interpretiert werden kann.

2.1.2 Syntax der Sprachen erster Stufe

In diesem Abschnitt werden die wesentlichen Definitionen zur Sprache erster Stufe der Quelle [9] zitiert.

Definition 2.1:¹

Ist \mathfrak{A} ein Alphabet, also eine Menge von Zeichen, so bezeichnet \mathfrak{A}^* die Menge der Zeichenreihen bestehend aus Zeichen aus \mathfrak{A} .

Definition 2.2:²

„Das Alphabet einer Sprache erster Stufe umfasst folgende Zeichen:

1. v_0, v_1, v_2, \dots (Variablen)
2. $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ (nicht; und; oder; wenn ... so...; genau dann, wenn)
3. \forall, \exists (für alle, es gibt)
4. \equiv (Gleichheitszeichen)
5. $), ($ (Klammersymbole)
6. Eine Symbolmenge S besteht aus folgenden Zeichen:
 - a) für jedes $n \geq 1$ eine (eventuell leere) Menge aus n -stelligen Relationssymbolen;
 - b) für jedes $n \geq 1$ eine (eventuell leere) Menge aus n -stelligen Funktionssymbolen;
 - c) eine (eventuell leere) Menge von Konstanten.

S bestimmt eine Sprache erster Stufe; $\mathfrak{A}_S := \mathfrak{A} \cup S$ ist das Alphabet dieser Sprache und S ihrer Symbolmenge.“

¹siehe [9] S.13

²siehe [9] S.17

Definition 2.3:³

„S-Terme sind genau diejenigen Zeichenreihen in \mathfrak{A}_S^* , welche man durch endlichmalige Anwendung der folgenden Regeln erhalten kann:

1. Jede Variable ist ein S-Term
2. Jede Konstante aus S ist ein S-Term
3. Sind die Zeichenreihen t_1, \dots, t_n S-Terme und ist f ein n -stelliges Funktionssymbol aus S , so ist $f t_1, \dots, t_n$ ein S-Term.“

Definition 2.4:⁴

„S-Ausdrücke sind genau diejenigen Zeichenreihen in \mathfrak{A}_S^* , welche man durch endlichmalige Anwendung der folgenden Regeln erhalten kann:

1. Für S-Terme t_1, t_2 ist $t_1 \equiv t_2$ eine S-Formel
2. Sind t_1, \dots, t_n S-Terme und ist R ein n -stelliges Relationssymbol aus S , so ist $R t_1 \dots t_n$ eine S-Formel
3. Ist φ eine S-Formel, so ist $\neg \varphi$ eine S-Formel.
4. Sind φ und ψ S-Formeln, so sind $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi)$ und $(\varphi \leftrightarrow \psi)$ S-Formeln.
5. Ist φ eine S-Formel und x eine Variable, so sind $\forall x \varphi$ und $\exists x \varphi$ S-Formeln.“

³siehe [9] S.18

⁴siehe [9] S.19

Definition 2.5:⁵:

„(a) Die Funktion var (genauer: var_s), die jedem S -Term die Menge der in ihm vorkommenden Variablen zuordnet, lässt sich folgendermaßen definieren, wobei c eine Konstante und x eine Variable ist:

$$\begin{aligned} var(x) &:= \{x\} \\ var(c) &:= \emptyset \\ var(ft_1 \dots t_n) &:= var(t_1) \cup \dots \cup var(t_n). \end{aligned}$$

(b) Die Funktion TA , die jeder Formel die Menge seiner Teilausdrücke zuordnet, kann man induktiv über den Aufbau der Formeln definieren:

$$\begin{aligned} TA(t_1 \equiv t_2) &:= \{t_1 \equiv t_2\} \\ TA(Rt_1 \dots t_n) &:= \{Rt_1 \dots t_n\} \\ TA(\neg \varphi) &:= \{\neg \varphi\} \cup TA(\varphi) \\ TA((\varphi \circ \psi)) &:= \{(\varphi \circ \psi)\} \cup TA(\varphi) \cup TA(\psi) \\ &\text{für } \circ = \wedge, \vee, \rightarrow, \leftrightarrow \\ TA(\forall x \varphi) &:= \{\forall x \varphi\} \cup TA(\varphi) \\ TA(\exists x \varphi) &:= \{\exists x \varphi\} \cup TA(\varphi). \end{aligned}$$

⁵siehe [9] S.28

Definition 2.6:⁶:

„Die Menge der in einem Ausdruck φ frei vorkommenden Variablen, die wir mit $\text{frei}(\varphi)$ bezeichnen wollen, können wir induktiv über den Aufbau der Ausdrücke definieren. Wir beziehen uns dabei wieder auf eine vorgegebene Symbolmenge S .

$$\begin{aligned}\text{frei}(t_1 \equiv t_2) &:= \text{var}(t_1) \cup \text{var}(t_2) \\ \text{frei}(Pt_1 \dots t_n) &:= \text{var}(t_1) \cup \dots \cup \text{var}(t_n) \\ \text{frei}(\neg \varphi) &:= \text{frei}(\varphi) \\ \text{frei}((\varphi * \psi)) &:= \text{frei}(\varphi) \cup \text{frei}(\psi) \text{ für } * = \wedge, \vee, \rightarrow, \leftrightarrow \\ \text{frei}(\forall x \varphi) &:= \text{frei}(\varphi) \setminus \{x\} \\ \text{frei}(\exists x \varphi) &:= \text{frei}(\varphi) \setminus \{x\}.\end{aligned}$$

2.1.3 Semantik der Sprachen erster Stufe

Definition 2.7:⁷:

„Eine S -Struktur ist ein Paar $\mathfrak{A} = (A, \alpha)$ mit den folgenden Eigenschaften:

1. A ist eine nicht leere Menge, der sog. Grundbereich oder Träger von \mathfrak{A}
2. α ist eine auf S definierte Abbildung. Für sie gilt:
 - a) Für jedes n -stellige Relationssymbol R aus S ist $\alpha(R)$ eine n -stellige Relation über A .
 - b) Für jedes n -stellige Funktionssymbol f aus S ist $\alpha(f)$ eine n -stellige Funktion über A .
 - c) Für jede Konstante c aus S ist $\alpha(c)$ ein Element von A .

⁶siehe [9] S.30

⁷siehe [9] S.35

Definition 2.8:⁸:

„Eine *Belegung* in einer S -Struktur \mathfrak{A} ist eine Abbildung $\beta : \{v_n | n \in \mathbb{N}\} \rightarrow A$ der Menge der Variablen in den Träger A .“

Definition 2.9:⁹:

„ S -*Interpretation* \mathfrak{I} ist ein Paar (\mathfrak{A}, β) , bestehend aus einer S -Struktur \mathfrak{A} und einer Belegung β in \mathfrak{A} .

Ist β eine Belegung in \mathfrak{A} , $a \in A$ und x eine Variable, so sei β_x^a diejenige Belegung in \mathfrak{A} , die x auf a abbildet und für alle von x verschiedenen Variablen mit β übereinstimmt:

$$\beta_x^a(y) := \begin{cases} \beta(y) & \text{für } y \neq x \\ a & \text{für } y = x \end{cases} \quad (2.1)$$

Ist $\mathfrak{I} = (\mathfrak{A}, \beta)$, so sei $\mathfrak{I}_x^a := (\mathfrak{A}, \beta_x^a)$.“

Definition 2.10:¹⁰:

„Die Symbolmenge S ist vorgegeben, anstatt von S -Termen bzw. S -Ausdrücken und S -Interpretationen reden wir daher einfach von Termen, Ausdrücken und Interpretationen. Zur Vorbereitung wird jeder Interpretation $\mathfrak{I} = (\mathfrak{A}, \beta)$ und jedem Term t auf natürliche Weise ein Element $\mathfrak{I}(t)$ des Trägers A zugeordnet. Definiert wird $\mathfrak{I}(t)$ über den Aufbau der Terme:

1. Für eine Variable x sei $\mathfrak{I}(x) := \beta(x)$.
2. Für $c \in S$ sei $\mathfrak{I}(c) := c^{\mathfrak{A}}$.
3. Für n -stelliges $f \in S$ und Terme t_1, \dots, t_n sei $\mathfrak{I}(ft_1 \dots t_n) := f^{\mathfrak{A}}(\mathfrak{I}(ft_1), \dots, \mathfrak{I}(ft_n))$.“

⁸siehe [9] S.36

⁹siehe [9] S.36

¹⁰siehe [9] S.40

Definition 2.11:¹¹

„Definition der Modellbeziehung: Für alle $\mathcal{I} = (\mathfrak{A}, \beta)$ setzen wir:

$$\begin{aligned}
 \mathcal{I} &\models t_1 \equiv t_2 : \text{gdw } \mathcal{I}(t_1) = \mathcal{I}(t_2) \\
 \mathcal{I} &\models R t_1 \dots t_n : \text{gdw } R^{\mathfrak{A}} \mathcal{I}(t_1) \dots \mathcal{I}(t_n) \text{ d.h. } R^{\mathfrak{A}} \text{ trifft zu auf } \mathcal{I}(t_1), \dots, \mathcal{I}(t_n) \\
 \mathcal{I} &\models \neg \varphi : \text{gdw nicht } \mathcal{I} \models \varphi \\
 \mathcal{I} &\models (\varphi \wedge \psi) : \text{gdw } \mathcal{I} \models \varphi \text{ und } \mathcal{I} \models \psi \\
 \mathcal{I} &\models (\varphi \vee \psi) : \text{gdw } \mathcal{I} \models \varphi \text{ oder } \mathcal{I} \models \psi \\
 \mathcal{I} &\models (\varphi \rightarrow \psi) : \text{gdw } \mathcal{I} \models \varphi, \text{ so } \mathcal{I} \models \psi \\
 \mathcal{I} &\models (\varphi \leftrightarrow \psi) : \text{gdw } \mathcal{I} \models \varphi \text{ genau dann, wenn } \mathcal{I} \models \psi \\
 \mathcal{I} &\models \forall x \varphi : \text{gdw für alle } a \in A \text{ gilt } \mathcal{I}_{\frac{a}{x}} \models \varphi \\
 \mathcal{I} &\models \exists x \varphi : \text{gdw es gibt ein } a \in A \text{ mit } \mathcal{I}_{\frac{a}{x}} \models \varphi
 \end{aligned}$$

(Zur Definition von $\mathcal{I}_{\frac{a}{x}}$ vgl. Definition 2.9; gdw steht für „genau dann, wenn“)

Definition 2.12:¹²

„Definition der Folgerungsbeziehung: Φ sei eine Menge von Ausdrücken und φ ein Ausdruck:

Φ folgt aus φ (kurz: $\Phi \models \varphi$) :gdw jede Interpretation, die Modell von Φ ist, ist auch Modell von φ .¹³

Statt „ $\{\psi\} \models \varphi$ “ schreiben wir auch „ $\psi \models \varphi$ “.

Definition 2.13:¹⁴

„Ein Ausdruck φ heißt allgemeingültig (kurz: $\models \varphi$) :gdw $\emptyset \models \varphi$.“

¹¹siehe [9] S.40

¹²siehe [9] S.41

¹³Wir verwenden \models sowohl für die Modellbeziehung ($\mathcal{I} \models \varphi$) als auch für die Folgerungsbeziehung ($\Phi \models \varphi$). Mißverständnisse sind nicht zu befürchten, da der Bezug auf eine Interpretation \mathcal{I} bzw. eine Ausdrucksmenge Φ die Bedeutung festlegt.

¹⁴siehe [9] S.42

Definition 2.14:¹⁵:

„Ein Ausdruck φ heißt erfüllbar (kurz: Erf φ) genau dann, wenn es eine Interpretation gibt, die Modell von φ ist. Eine Menge Φ von Ausdrücken heißt erfüllbar (kurz: Erf Φ) genau dann, wenn es eine Interpretation gibt, die Modell aller Ausdrücke aus Φ ist.“

Definition 2.15:¹⁶:

„Zwei Ausdrücke φ und ψ heißen logisch äquivalent (kurz: $\varphi \models \psi$) :gdw $\varphi \models \psi$ und $\psi \models \varphi$.“

Neben diesen Definitionen existieren noch vollständige und korrekte Beweiskalküle, die hier nicht näher ausgeführt werden.

¹⁵siehe [9] S.43

¹⁶siehe [9] S.43

2.1.4 ATP (Automated Theorem Prover) und TPTP (Thousands of Problems for Theorem Provers)

Ein automatischer Beweiser (engl. Automated Theorem Prover, siehe [24]) ist ein Computerprogramm zum Beweisen von mathematischen Theoremen. Die Aussagen werden mittels eines Beweiskalküls auf Grundlage der Axiome belegt, oder durch die Erstellung eines Gegenmodells widerlegt. Sollte die Rechenzeit nicht ausreichen, um zu einem Ergebnis zu gelangen, gilt der jeweilige Beweisschritt als nicht gültig.

Es gibt verschiedene ATPs, im Naproche-System werden die ATPs „E“ und „Vampire“ benutzt.

Um die Ergebnisse und Effizienz der ATPs besser vergleichen zu können, wurde TPTP (Thousands of Problems for Theorem Provers, [22]) entwickelt. TPTP bietet eine eigene Syntax, um Probleme zu beschreiben. Im Rahmen von Naproche wird das Programm „SystemsOnTPTP“ genutzt, das die Probleme, die in TPTP-Syntax beschrieben sind, in das Eingabeformat des gewünschten ATPs umwandelt. Wenn also ein Problem in TPTP beschrieben wurde, kann man es an verschiedene ATPs übergeben und die Ergebnisse vergleichen.

Die TPTP-Syntax hat folgenden Aufbau:

`fof (<name>, <formula role>, <fof formula>, <annotations>)`

- fof steht für „first order formula“.
- <name> bezeichnet den Namen der mathematischen Aussage. Eine „Obligation“ entspricht einer Aussage (also keiner Annahme, Definition oder Axiom), die überprüft werden muss.
- <formula role> gibt dem Beweiser an, ob überprüft werden muss. Bei Naproche gibt es „axiom“, welche nicht überprüft werden und „conjecture“, die der Beweiser überprüfen muss. <name> und <formula role> müssen zusammen eindeutig sein.
- <fof formula> ist die Aussage in TPTP-Syntax geschrieben.

- <annotations> wird für zusätzliche Angaben optional bereit gestellt, aber in Naproche nicht genutzt.

In dem Naproche-System wird ein Beweistext dadurch überprüft, dass neue Aussagen in einem Beweis von einem ATP auf Grundlage vorheriger Aussagen und Annahmen bewiesen werden (siehe Kapitel 3.4.3). Um die Struktur zu verdeutlichen, wird hier ein einfaches Beispiel näher beschrieben. Der folgende natürlichsprachliche Eingabetext im LaTeX-Stil wird umgewandelt:

Beispiel 2.1

Axiom.

There is no y such that $y \in \emptyset$.

Define x to be transitive if and only if for all u , v ,
if $u \in v$ and $v \in x$ then $u \in x$.

Define x to be an ordinal if and only if x is transitive
and for all y , if $y \in x$ then y is transitive.

Then \emptyset is an ordinal.

Nach der Umwandlung in das TPTP-Format sieht die Eingabe des Beispiels 2.1, die zur Überprüfung, ob \emptyset eine Ordinalzahl ist, an den ATP gesendet wird, wie folgt aus:

```
fof('replace(pred(5, 1))', conjecture, rordinal(vemptyset)).
fof('def(cond(conseq(axiom(1)), 1), 1)', axiom,
    ![Vd10]:((rordinal(Vd10))<=>(![Vd12]:((rin(Vd12,Vd10))
    =>(rtransitive(Vd12))))&(rtransitive(Vd10))))).
fof('def(cond(conseq(axiom(1)), 0), 1)', axiom,
    ![Vd3]:((rtransitive(Vd3))<=>(![Vd5,Vd6]:
    (((rin(Vd6,Vd3))&(rin(Vd5,Vd6)))=>(rin(Vd5,Vd3)))))).
fof('neg(neg(axiom(1)))', axiom, ~(?[Vd1]:(rin(Vd1,vemptyset))))).
```

Die Reihenfolge des Inputs ist die umgekehrte des Fließtextes: Während im natürlichsprachlichen Text zunächst die Axiome in den ersten drei Sätzen beschreiben werden, wird am Ende des Textes die zu überprüfende Aussage formuliert (hier der letzte Satz:

Then \emptyset is an ordinal.). Aus allen vier Sätzen wird jeweils ein „fof()“-Eintrag mit drei Parametern erstellt.

Der erste Parameter ist der Name der Aussage, der sich auf dem Namen der jeweiligen PRS (siehe Kapitel 2.2.1) bezieht. Die zu überprüfenden Aussagen erkennt man an dem Schlüsselwort „conjecture“, welches das zweite Element der first-order-formula (fof) ist. Nicht zu überprüfende Aussagen enthalten an dieser Stelle ein „axiom“. Der Beweiser versucht nun diese Axiome so zu verbinden, dass die zu beweisende Aussage („conjecture“) entweder bestätigt oder widerlegt wird. Die Bearbeitung der Sätze findet in der natürlichen Reihenfolge statt, es wird alles gespeichert und anschließend ausgegeben.

Variablen beginnen in diesem Eingabeformat mit einem großen „V“, Konstanten mit einem kleinen „v“ und Relationen mit einem kleinen „r“. Das „!“ entspricht dem Allquantor (\forall), ein „?“ repräsentiert entsprechend dem Existenzquantor (\exists).

Die Variablen im ATP-Input entsprechen wie folgt den Variablen im Eingabetext:

- Vd1: y
- Vd3: x
- Vd5: v
- Vd6: u
- Vd10: x
- Vd12: y

In diesem Beispiel kommen die Variablen x und y jeweils in zwei verschiedenen Sätzen vor. Sie werden auch unabhängig voneinander betrachtet. Das bedeutet, dass man anstelle des zweiten x (oder y) die Variable auch z hätte nennen können.

Das Ergebnis der Überprüfung ist eine „.output - Datei“. Ein Auszug aus dieser Datei, die aus dem Beispiel 2.1 entstanden ist, wird im Folgenden angezeigt. Hierbei steht <pfad> für den Dateipfad, in dem sich die Dateien befinden:

```
fof(1, conjecture, vd2=vd4, file('
    <pfad>/3-holds(3, 6, 0).input', 'holds(3, 6, 0)')).
fof(2, axiom, vd1=vd4, file('<pfad>/3-holds(3, 6, 0).input', 'holds(2, 5, 0)')).
fof(3, axiom, vd1=vd2, file('<pfad>/3-holds(3, 6, 0).input', 'holds(1, 3, 0)')).
fof(4, negated_conjecture, ~(vd2=vd4),
    inference(assume_negation, [status(cth)], [1])).
fof(5, negated_conjecture, ~(vd2=vd4),
    inference(fof_simplification, [status(thm)], [4, theory(equality)]))).
cnf(6, negated_conjecture, (vd2!=vd4),
    inference(split_conjunct, [status(thm)], [5])).
cnf(7, plain, (vd1=vd4), inference(split_conjunct, [status(thm)], [2])).
cnf(8, plain, (vd1=vd2), inference(split_conjunct, [status(thm)], [3])).
cnf(9, negated_conjecture, (vd4!=vd1),
    inference(rw, [status(thm)], [6, 8, theory(equality)]))).
cnf(10, negated_conjecture, ($false),
    inference(rw, [status(thm)], [9, 7, theory(equality)]))).
cnf(11, negated_conjecture, ($false),
    inference(cn, [status(thm)], [10, theory(equality)]))).
cnf(12, negated_conjecture, ($false), 11, ['proof']).
```

In dem Dateiauszug wurde dargestellt, wie der ATP bei der Überprüfung der Eingabe vorgegangen ist. Die zu überprüfende Aussage ist der erste Eintrag, der diesmal alle vier Parameter benutzt.

Die ersten drei Parameter (<name>, <formula role>, <fof formula>) haben sich nicht verändert, sie bestehen aus dem Namen (hier ID), conjecture bzw. axiom und die entsprechende logische Aussage. Der vierte Parameter (<annotations>) beinhaltet den Pfad der Input-Datei, der zum Überprüfen mitgegeben wird. Im Bereich der cnf (clausal normal forms) findet die Überprüfung statt. Der erste Wert entspricht der fortlaufenden ID, danach wird der Typ der Aussage festgelegt. In dem Beispiel 2.3 gibt es entweder negierte oder einfache Aussagen. Der dritte Wert gibt die zu überprüfende Aussage an, die im vierten Wert näher spezifiziert wird. Lediglich die letzte Zeile enthält einen fünften Wert (hier proof), der das Ergebnis der Überprüfung angibt. „proof“ bedeutet, dass der Beweis gültig ist. Konnte die Aussage nicht bestätigt werden, steht an der Stelle ein „noproof“.

2.2 Linguistische Grundlagen

In diesem Kapitel werden die relevanten linguistischen Grundlagen näher beschrieben. Die Quellen zum Kapitel sind [14], [8], [15] und [12], in denen zum jeweiligen Themenbereich genauere Informationen gefunden werden können.

2.2.1 Diskursrepräsentationsstruktur und Beweisrepräsentationsstruktur

Diskursrepräsentationsstrukturen

Eine Diskursrepräsentationsstruktur (engl. „Discourse Representation Structure“, DRS) kommt in der semantischen Verarbeitung von Sprachen zum Einsatz. Sie ist eine Darstellungsform der Diskursrepräsentationstheorie („Discourse Representation Theory“, DRT). Quelle zu diesem Kapitel sind die Master-Thesis von Nickolay Kolev [14], die ebenfalls im Rahmen von Naproche entstanden ist, sowie eine Informatik-Internetseite der Universität Hamburg [12].

Ziel einer DRT ist es, die Bedeutung eines Satzes in ihrem Kontext richtig darzustellen. Die Interpretation eines Einzelsatzes setzt voraus, dass dieser in einem längeren Diskurs steht. Ein bekanntes Beispiel zur Darstellung eines Satzes in der Prädikatenlogik erster Stufe ist der „Donkey-Sentence“:

Beispiel 2.2

If a farmer owns a donkey, he beats it.

Die beiden Teilsätze sind inhaltlich voneinander abhängig. Hierbei bezieht sich das „he“ aus dem zweiten Teilsatz auf „a farmer“ aus dem ersten Teilsatz, also wird „he“ als Anapher von „a farmer“ bezeichnet.

Die systematische Übersetzung jedes existenziellen Ausdrucks in einen Existenzquantor ergibt eine falsche Darstellung des Satzes, da x und y in $beat(x, y)$ freie Variablen sind:

$$\exists x(farmer(x) \wedge \exists y(donkey(y) \wedge owns(x, y))) \rightarrow beat(x, y)$$

Eine bessere Übersetzung ist folgende:

$$\forall x \forall y (farmer(x) \wedge donkey(y) \wedge owns(x, y) \rightarrow beat(x, y))$$

Auf diese Übersetzung kommt man jedoch nicht durch das Zusammenführen der Übersetzung der Teilsätze. Die DRT bietet folgenden Lösungsansatz an: Es wird ein neuer Diskursreferent für gemeinsame semantische Funktion von nicht-anaphorischen Ausdrücken eingefügt, der die anaphorischen Ausdrücke bindet. Es werden keine Quantoren in die Darstellung eingeführt, womit das logische Übersetzungsproblem überwunden wurde.

Diese sprachlichen Zusammenhänge können in DRS-Boxen dargestellt werden.

Der Satz *If a farmer beats a donkey, it roars.* entspricht folgender Boxenstruktur:

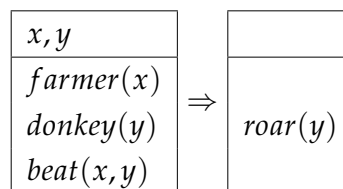


Abbildung 2.1: Beispiel einer DRS

Eine Box besteht aus zwei verschiedenen Bereichen, der obere Teil beinhaltet alle in der Box eingeführten Variablen (hier x und y), der untere Teil die Variablenzuordnung ($farmer(x)$ und $donkey(y)$) und die Relationen ($beat(x, y)$). Die Boxen können über logische Verknüpfungen zueinander in Beziehung gesetzt werden. Wie in der Abbildung 2.1 zu sehen ist, folgt aus der ersten die zweite Box.

Beweisrepräsentationsstrukturen

Beweisrepräsentationsstrukturen (engl. Proof Representation Structur, PRS) sind eine Erweiterung von DRSen, die aus mathematischen Texten relevante Informationen ergänzt. Eine PRS wird wie folgt dargestellt:

$$i$$

d_1, \dots, d_m	m_1, \dots, m_n
c_1	
\vdots	
c_l	
r_1, \dots, r_k	

Abbildung 2.2: Aufbau einer PRS

Eine PRS besteht aus einem Sieben-Tupel:

1. i : eine eindeutige Zahl oder ein zusammengesetzter Ausdruck (in der Art „n“, „f(n)“, „f(g(n))“ oder ähnliches) zur Identifizierung einer PRS
2. d_1, \dots, d_m (Drefs): eine Liste von zunächst uninstantiierten Prolog-Variablen, die im Bearbeitungsprozess mit natürlichen Zahlen instanziiert werden.
3. m_1, \dots, m_n (Mrefs): eine Liste von Variablen, Terme und/oder Formeln in einer Baumstruktur.
4. c_1, \dots, c_l (Conds): eine Liste von PRS-Bedingungen, die im Anschluß genauer erklärt wird.
5. r_1, \dots, r_k (Rrefs): eine Liste von PRS-Ids, mit denen auf andere PRSen verwiesen wird.
6. Accbefore und
7. Accafter: sind Listen von Termen der Form „math_id(n,tree)“, wobei „n“ ein dref und „tree“ eine Liste in Baumstruktur ist. Ziel ist es, die Zugänglichkeit in den PRSen zu bestimmen. Beide werden in der PRS-Darstellung weggelassen, da sie nur implementationstechnisch relevant sind.

Eine PRS-Bedingung hat eine der folgenden Formen, mit der darunterstehenden Funktion bzw. Bedeutung:

- B : B ist eine PRS
- $B \Rightarrow C$: wobei B und C PRSen sind
Darstellung von einer Implikation oder einer universellen Quantifikation
- $B \Rightarrow C$: wobei B und C PRSen sind
Darstellung für eine Annahme (B) und aus der Annahme folgende Aussage (C)
- $B \Leftrightarrow C$: wobei B und C PRSen sind
Darstellung für eine logische Äquivalenz-Anweisung
- $B \Leftarrow C$: wobei B und C PRSen sind
Darstellung einer so genannten umgekehrten Implikation (B wenn C)
- $B = C$: wobei B und C PRSen sind
Darstellung einer Definition eines Prädikates
- $f::B \Rightarrow C$: wobei f ein Funktionssymbol ist und B und C PRSen sind
Darstellung einer Funktions-Definition
- $B \vee C$: wobei B und C PRSen sind
Darstellung einer Oder-Verknüpfung
- $\text{neg}(B)$: B ist eine PRS
Darstellung einer Negation
- $\text{static}(B)$: B ist eine PRS
Diese PRS-Bedingung wird beim Algorithmus zur Pluralinterpretation benötigt. Sie ähnelt der PRS-Bedingung „ B “, aber blockiert die Möglichkeit von Anaphern auf darin enthaltene Drefs.
- $\text{the}(n, B)$: n ist eine Zahl bzw. ein dref; B ist eine PRS
Darstellung einer bestimmten Kennzeichnung („the B “)

- $\text{plural}(n, B)$: n ist eine Zahl bzw. ein Dref; B ist eine PRS
Darstellung einer Nominalphrase im Plural; Diese PRS-Bedingung wird beim Algorithmus zur Pluralinterpretation benötigt.
- contradiction : ist eine PRS-Bedingung
diese Bedingung steht für einen Widerspruch.
- $\text{><}(PRSList)$: wobei $PRSList$ eine Prologliste von PRSen ist
Darstellung eines exklusiven Oders: genau ein Element aus der Liste ist gültig
- $\text{<>}(PRSList)$: wobei $PRSList$ eine Prologliste von PRSen ist
Darstellung eines NORs: höchstens ein Element aus der Liste ist gültig
- $\text{holds}(n)$: n ist ein Dref
die Behauptung, auf die mit n referenziert wird, ist wahr.
- $\text{math_id}(n, tree)$: n ist ein Dref; $tree$ ist eine Liste in Baumstruktur
Verknüpft einen Dref mit einem Mref.
- $\text{predicate}(n, atom)$: n ist ein Dref; $atom$ ist ein Prolog-Atom wobei $atom$ eine Relation ist, die auf n angewendet wird.
- $\text{predicate}(n, m, atom)$: n und m sind Drefs; $atom$ ist ein Prolog-Atom, wobei $atom$ eine Relation ist, die zwischen n und m angewendet wird.
- $\text{plural_dref}(n, list)$: n ist ein Dref; $list$ ist eine Liste von Drefs
Das Prädikat wird für den Pluralinterpretationsalgorithmus benötigt (siehe [6]):
Ausdrücken wie „there exist natural numbers x, y, z such that...“ wird jeder Variablen x, y, z ein Dref zugeordnet. Zusätzlich bekommt der Ausdruck „natural numbers x, y, z “ einen so genannten Plural-Dref. Über die Bedingung „ $\text{plural_dref}(n, list)$ “ wird der Zusammenhang zwischen den einzelnen Drefs und dem Pluraldref hergestellt.

Man betrachte nun einen Beispieltext:

Beispiel 2.3

„Let $x = y$ and $x = z$. Then $y = z$.“

Dieser Text wird von Naproche in folgende PRS umgewandelt:

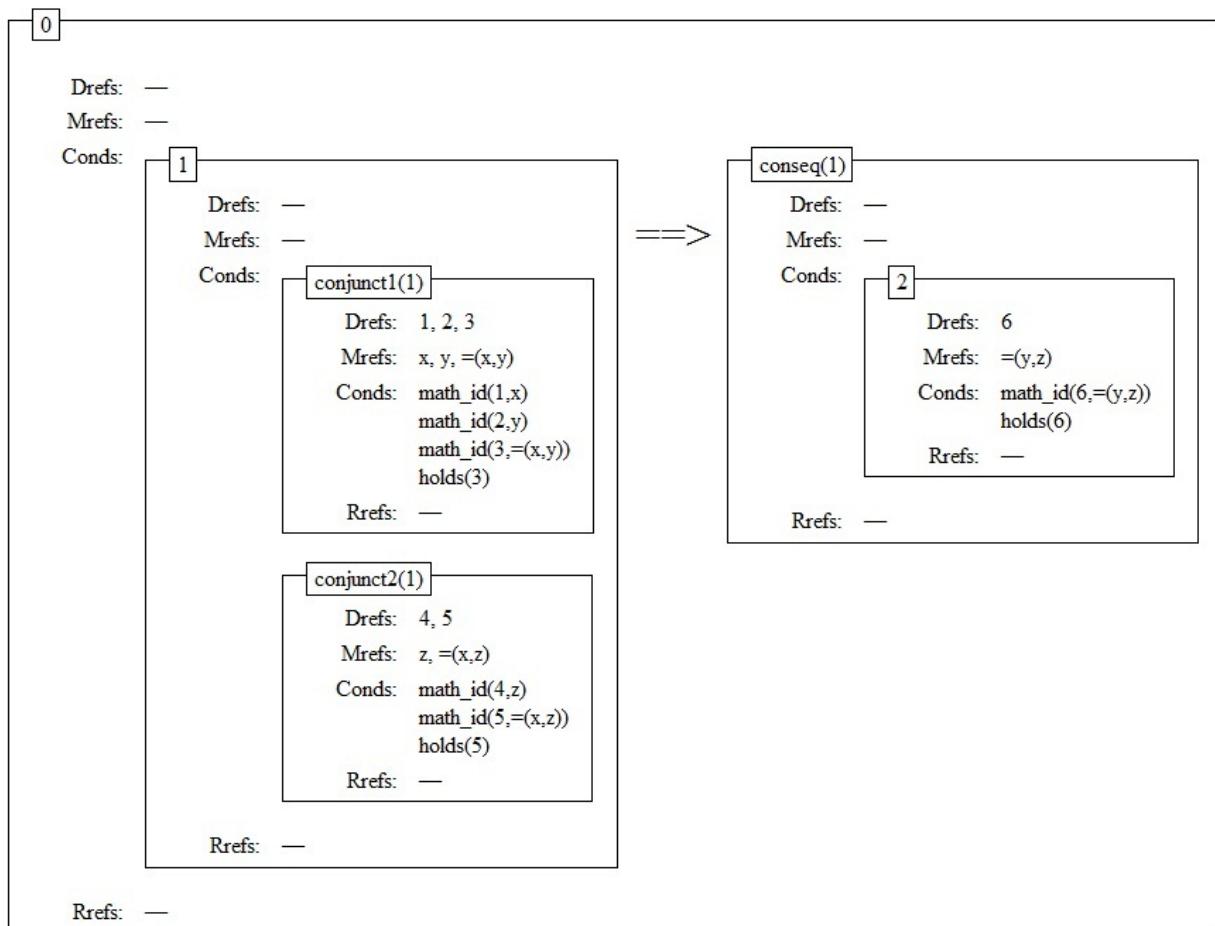


Abbildung 2.3: Beispiel einer PRS

2.2.2 Kontrollierte natürliche Sprache

Ein Ziel von Naproche ist es, eine kontrollierte natürliche Sprache (engl. „Controlled Natural Language“, CNL) zu entwickeln. Die Naproche-CNL basiert auf der englischen Fachsprache der Mathematik. Durch die Mehrdeutigkeiten, die in einer natürlichen Sprache auftreten, wird es schwer einen Text zu überprüfen. Beispielsweise kann das deutsche Wort *Bocksprung* falsch verstanden werden: entweder springt ein Bock über etwas, oder jemand springt über einen Bock. Mit einer klar definierten formalen Sprache ist es möglich, mathematische Texte strukturiert zu erfassen und diese dann maschinell auf ihre mathematische Richtigkeit überprüfen zu lassen. Die Naproche-CNL verwendet für die mathematischen Formeln LaTeX-Syntax, da die meisten mathematischen Paper in LaTeX geschrieben werden. Der Textinhalt wird anhand von Schlüsselwörtern strukturiert: Axiom, Theorem, Lemma, Proof und Qed. So wird nach jedem „Theorem“ im Text ein „Proof“ erwartet. Nach „Proof“ findet der eigentliche Beweis statt, der mit einem „Qed.“ (quod erat demonstrandum) abgeschlossen wird. Weiterhin können Statements, Definitionen, Implikationen und Annahmen als solche verarbeitet werden. So wird beispielsweise bei Annahmen ein Trigger geöffnet, wenn Schlüsselwörter (let, consider, assume that, ...) im Text vorkommen. Geschlossen wird der Trigger durch weitere Schlüsselwörter (thus, Qed.). Zusätzlich können Negationen, Konjunktionen, Disjunktionen, Quantifizierungen, Äquivalenzen und Implikationen in natürlicher Sprache im Beweistext geschrieben und richtig interpretiert werden. Im folgenden wird der Beispieltext 2.4 gezeigt, der in der Naproche-CNL geschrieben ist und vom Naproche-System 0.47 auf seine Richtigkeit überprüft werden kann:

Beispiel 2.4

Axiom.

There is no y such that $y \in \emptyset$.

Axiom.

For every x it is not the case that $x \in x$.

Define x to be transitive if and only if for all u, v , if $u \in v$ and $v \in x$ then $u \in x$.

Define x to be an ordinal if and only if x is transitive

and for all y , if $y \in x$ then y is transitive.

Theorem.

\emptyset is an ordinal.

Proof.

Assume $u \in v$ and $v \in \emptyset$. Then there is an x such that $x \in \emptyset$. Contradiction.

Thus \emptyset is transitive.

Assume $y \in \emptyset$. Then there is an x such that $x \in \emptyset$. Contradiction.

Thus for all y , if $y \in \emptyset$ then y is transitive. Hence \emptyset is an ordinal.

Qed.

Die Syntax und Semantik der englischen Sprache findet sich auch in der mathematischen Fachsprache wieder, wie das Beispiel zeigt. Es gibt jedoch Besonderheiten zu allgemeinsprachlichen Texten:

- Mathematische Symbole und Formeln im Beweistext entsprechen Nominalphrasen und Teilsätzen und werden mit natürlichsprachlichen Ausdrücken verbunden.
- Neue Ausdrücke und Symbole können über Definitionen, die deren Bedeutung eindeutig klären, zum Beweistext hinzugefügt werden.
- Mehrdeutige Aussagen werden vermieden.
- Um einen Text möglichst eindeutig zu schreiben, werden bevorzugt mathematische Symbole verwendet.
- Eine Annahme kann im Beweisverlauf zunächst eingeführt und später zurückgezogen werden. Ein Beispiel hierfür ist der Beweis für die Irrationalität von $\sqrt{2}$. Bei diesem Widerspruchsbeweis ist die Annahme eine Negation der Theoremaussage: $\sqrt{2}$ ist rational. Alle folgenden Aussagen werden aus dieser Annahme abgeleitet. Nachdem die Rationalitätsannahme zu einem Widerspruch geführt hat, wird die Annahme zurückgezogen. Daraus ergibt sich, dass die ursprüngliche Annahme falsch ist: $\sqrt{2}$ ist irrational. Die einzelnen Unterannahmen haben in Naproche einen „skopus“ (Geltungsbereich). Dieser bezeichnet einen Bereich, der mit der Einführung einer Annahme beginnt und bis zu ihrer Aufhebung reicht.
- Die Struktur der mathematischen Texte ist leicht zu erkennen: Bereiche wie Theoreme, Lemmata und Definitionen und der eigentliche Beweis sind sofort zu erkennen. Die Schachtelungen innerhalb eines Beweises liefern durch die Geltungsbereiche von Annahmen eine hierarchische Struktur.
- Durch Referenzen auf bereits vorhandene Ergebnisse können einzelne Beweisschritte begründet werden.

Die Grammatik der Naproche-Sprache wurde in Prolog geschrieben und findet sich in den Modulen „dcg“ und „fo_grammar“ wieder (siehe Kapitel 3.4.1).

2.2.3 Definite Clause Grammar

Definite Clause Grammar (DCG, [28]) ist ein Grammatikformalismus zum Beschreiben von formalen Sprachen, und damit auch von kontrollierten natürlichen Sprachen. Eine Sprache besteht aus verschiedenen Worttypen (u.a. Substantiv und Adjektiv), die zu verschiedenen Klassen (u.a. Nominalphrasen) zusammengefasst werden können. In der natürlichen Sprache entstehen durch weitere Kombinationsmöglichkeiten Teilsätze (u.a. Relativ- und Fragesätze, die wiederum zu vollständigen Sätzen zusammengesetzt werden können). Es entstehen auf diese Art und Weise Gruppen von Wörtern, Gruppen von diesen Gruppen und wiederum Gruppen von Gruppen von Gruppen (usw.). Eine DCG beschreibt formal die Kombinationsmöglichkeiten in einer Sprache unter Verwendung von bestimmten Regeln, die im englischen als „definite clauses“ bezeichnet werden. Es gibt verschiedene Arten von Parsern für das Parsen von Texten nach den Regeln einer DCG. In Prolog ist bereits ein sehr effizienter Top-Down-Parser implementiert. Mit dem DCG-Parser kann man neben einer reinen Analyse auch eine semantische Repräsentation des Eingabetextes erstellen lassen. Im Kapitel 3.4.1 wird der Einsatz der DCG im Naproche-System zur Erstellung einer PRS beschrieben.

2.3 Informationstechnische Grundlagen

In diesem Kapitel werden die informationstechnischen Grundlagen erläutert, die in der Dokumentation der Master-Thesis verwendet werden.

2.3.1 Prolog

Prolog ist eine deklarative Programmiersprache und gilt als die wichtigste logische Programmiersprache (siehe [18] und [1]). Die Bezeichnung „Prolog“ ist eine Kurzform von „Programmation en Logique“. Ein Prolog-Programm besteht aus einer Wissensbasis, auf die beim Ausführen des Programms durch Abfragen zugegriffen werden kann. Eingesetzt wird Prolog unter anderem zum Parsen (Analysieren bzw. Zergliedern) von Texten, um zu überprüfen, ob sie die vorgegebenen grammatikalischen Regeln befolgen.

2.3.2 Komplexität

Die Anzahl der Rechenschritte zur Lösung eines Problems durch einen Algorithmus gibt seine Zeitkomplexität an (siehe [25]). Es wird hierbei zwischen drei möglichen Fällen unterschieden, nach denen der Algorithmus eine Eingabe bearbeitet und die zur Laufzeitabschätzung benutzt werden:

Den besten (best-case), den durchschnittlichen (average-case) und den schlechtesten (worst-case) Fall. Sowohl das best- als auch das worst-case Szenario werden in der Praxis sehr selten erreicht. Sie dienen als obere und untere Schranke für die Berechnung der Laufzeit. Der average-case ist stark von der Verteilung der Eingabe des Algorithmus, die jedoch unbekannt sein kann, abhängig. So kann dieser Berechnungswert nur eingeschränkt verwendet werden.

Wie aufwändig ein einzelnes Problem sein kann, wird in der O -Notation, die dem oben genannten worst-case und einer Klassifizierung der Laufzeit entspricht, angegeben. Hierbei werden verschiedene Komplexitätsklassen unterschieden. Es gilt, dass f eine Funktion und n die Anzahl der einzelnen Berechnungsschritte ist. Die gängigen Klassen sind folgende:

- konstantes Wachstum: $O(1)$
- logarithmisches Wachstum: $O(\log n)$
- lineares Wachstum: $O(n)$
- n-log-n Wachstum: $O(n \log n)$
- polynomiales Wachstum: $O(n^k)$ für $k \geq 1$
- exponentielles Wachstum: $O(d^n)$ für $d > 1$

2.3.3 Abstrakte Datentypen

Abstrakte Datentypen entstehen, wenn mehrere Datentypen zusammenfasst, sie also gekapselt werden. So entsteht eine Gruppierung von logisch zusammenhängenden Daten mit ihren jeweiligen Methoden. Generell gilt: Abstrakter Datentyp = Datentypen + Methoden. Mit einer Kapselung können unzulässige Zugriffe und versehentliche Fehler vermieden werden. So kann der sichergestellt werden, dass Datenstrukturen nicht nach außen sichtbar sind. Jeder abstrakte Datentyp hat alle seine Variablen und Methoden, die einen kontrollierten vordefinierten Zugriff ermöglichen. Für jede Operation muss eine Methode implementiert worden sein, sonst ist es eine ungültige Operation. Eine direkte Änderung der Daten in einem abstrakten Datentyp ist nicht möglich (siehe [2]).

2.3.4 Persistente Daten

Eine Aufgabe dieser Master-Thesis ist die Entwicklung eines persistenten (nicht flüchtigen) Datentyps. Im Gegensatz zu transienten (flüchtigen) Daten sind persistente Daten solche, die in Dateien oder Datenbanken gespeichert werden (siehe [11]). Diese Daten sind somit auch nach dem Beenden (oder Absturz) eines Programms noch vorhanden und können beim nächsten Programmaufruf wieder verwendet werden. Außerdem kann man von verschiedenen Instanzen auf die Daten zugreifen. Bei der Berechnung von diversen Werten können die Daten auch aus dem Speicher ausgelagert und bei Bedarf wieder eingelesen werden.

2.3.5 Backus-Naur-Form

Um kontextfreie Grammatiken korrekt zu definieren kann man die Backus-Naur-Form (BNF) benutzen (siehe [3]). Mit dieser Metasprache kann beispielsweise die Syntax einer Programmiersprache formal exakt beschrieben werden. Der generelle Aufbau einer BNF besteht aus folgenden Elementen:

- Terminalsymbole: können nicht weiter abgeleitet werden.
- Nicht-Terminalsymbole: werden durch die Anwendung von Ableitungsregeln in Terminalsymbole umgewandelt.
- Ableitungsregeln: Geben die Grammatik der Sprache vor.

Die verschiedenen Symbole werden in Ableitungsregeln miteinander kombiniert, so dass der gewünschte Kontext entsteht. In den Ableitungsregeln wird das Zeichen „|“ als Oder-Verknüpfung benutzt; das Zeichen „::=“ steht für eine Definition und zwischen „<“ und „>“ werden die Nicht-Terminalsymbole benannt. Die evtl. vorhandenen Symbole hinter den Nicht-Terminalsymbolen in den Ableitungsregeln sagen etwas über die Häufigkeit aus, in der das Symbol vorkommt:

- (nichts): genau einmal
- + : mindestens einmal
- * : beliebig oft
- ? : einmal oder keinmal

Zur Veranschaulichung der BNF werden hier anhand des Beispiels für die deutschen KFZ-Kennzeichen die Ableitungsregeln aufgelistet:

Beispiel 2.4

$\langle \text{Alphabet} \rangle ::= A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I \mid J \mid K \mid L \mid M \mid N \mid O \mid P \mid Q \mid R \mid S \mid T \mid U \mid V \mid W \mid X \mid Y \mid Z$

$\langle \text{ErsteZiffer} \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{Ziffern} \rangle ::= 0 \mid \langle \text{ErsteZiffer} \rangle$

$\langle \text{Trennung} \rangle ::= -$

$\langle \text{Leerzeichen} \rangle ::=$

$\langle \text{Kreis} \rangle ::= \langle \text{Alphabet} \rangle \langle \text{Alphabet} \rangle? \langle \text{Alphabet} \rangle?$

$\langle \text{Erkennungsnummer} \rangle ::= \langle \text{Alphabet} \rangle \langle \text{Alphabet} \rangle? \langle \text{Leerzeichen} \rangle \langle \text{ErsteZiffer} \rangle \langle \text{Ziffern} \rangle? \langle \text{Ziffern} \rangle? \langle \text{Ziffern} \rangle?$

$\langle \text{KFZ-Kennzeichen} \rangle ::= \langle \text{Kreis} \rangle \langle \text{Trennung} \rangle \langle \text{Erkennungsnummer} \rangle$

2.3.6 CSV-Dateien

Die CSV (Comma-Separated Values)-Dateien beinhalten verschiedene Daten, die jeweils durch ein Komma getrennt werden. Etwas freier kann man CSV auch als Abkürzung für „Character Separated Values“ interpretieren, bei der die Trennung der Datensätze durch ein beliebig auswählbares Zeichen erfolgt. Durch zusätzliche Trennsymbole können in CSV-Dateien auch komplexe Tabellen und Listen gespeichert werden.

2.3.7 Legende der Diagramme

In diesem Kapitel werden die graphischen Elemente der verschiedenen Diagramme dieser Master-Thesis beschrieben.

Es wurden die im Anschluss dargestellten Symbole verwendet:





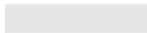
Symbol	Beschreibung
	Der rechteckige Kasten gibt das Modul (zum Beispiel Java-Methoden und Prolog-Prädikate) an, welches im dargestellten Verlauf bearbeitet wird. Hierbei wird unterschieden, ob ein konkreter Modulaufruf oder ein Programmblock beschrieben wird: der Modulaufruf endet mit zwei runden Klammern (), während der Programmblock ausschließlich aus textlichen Inhalten besteht.
	Die Wolke steht für ein komplexes Modul, welches in der Diagrammbeschreibung selber nicht explizit erläutert wird.
	Dateien in den Diagrammen werden durch den Zylinder dargestellt, der den jeweiligen Datei- oder Ordnernamen beinhaltet.
	Die Raute steht für eine Aktion des Anwenders, auf den das im Diagramm beschriebene System reagieren muss.
	Grau unterlegt sind die Zusatzinformationen zu einem dargestellten Symbol im selben Diagramm, die nur eine erklärende aber keine funktionale Aufgabe haben.

Tabelle 2.1: Legender der Diagramme

2.4 Terminologische Besonderheiten

In der Mathematik und der Informatik gibt es Begriffe, die für ihren Bereich jeweils eindeutig definiert sind, aber in anderen Themengebieten nicht exakt die selbe Bedeutung haben. Um solchen Missverständnissen vorzubeugen werden in diesem Kapitel die vieldeutigen Begriffe, die in der Thesis benutzt werden, beschrieben.

2.4.1 Datentypen und Typentheorie

Der Begriff des „Typs“ ist missverständlich, da in der Informatik oft der Begriff „Datentyp“ assoziiert wird, während in der Mathematik die „Typentheorie“ gemeint sein könnte. Aus diesem Grund werden diese beiden Begriffe erläutert.

Datentypen

In der Informatik steht der Begriff Typ oft für einen Datentyp. Ein Datentyp ist formal definiert als Zusammenfassung von Objektmengen, auf denen bestimmte Operationen angewendet werden können. Ein konkreter Datentyp hat einen klar definierten Wertebereich und kann Operationen ausführen. Die grundlegenden Datentypen sind folgende (in Klammern steht als Beispiel die Syntax aus der Programmiersprache Java):

- Ganze Zahlen (int, smallint, bigint, long, short)
- Natürliche Zahlen (unsigned int, unsigned long, unsigned short)
- Festkommazahlen (decimal)
- Gleitkommazahlen (double, float, long double, long float)
- Boolean (boolean)
- Zeichen (char)

Auf alle Zahlen können die Operationen $+$, $-$, $*$, $<$, $>$, $=$ und Division mit Rest und Modulo ausgeführt werden. Die boolschen Operationen sind *NOT*, *AND*, *XOR*, *NOR*, *NAND*, *OR*, $=$ und \neq . Für die Zeichen (char) gelten die Operationen $<$, $>$ und $=$. Weiterhin können sie beispielsweise in das Integerformat übertragen werden. Zusätzlich zu den grundlegenden Datentypen gibt es zusammengesetzte Datentypen. In Java ist beispielsweise ein „String“ eine Aneinanderreihung von beliebig vielen Zeichen. Auf einem String können verschiedene Operationen durchgeführt werden; u.a. Ermittlung der Anzahl der Zeichen, Suchen nach einem Teilstring, Herausschneiden eines Bereiches und viele mehr.

In Prolog gibt es ebenfalls verschiedene Datentypen, die sich teilweise von den Java-Datentypen unterscheiden. Die Prolog-Terme werden in folgender Baumstruktur aufgelistet und anschließend erläutert:

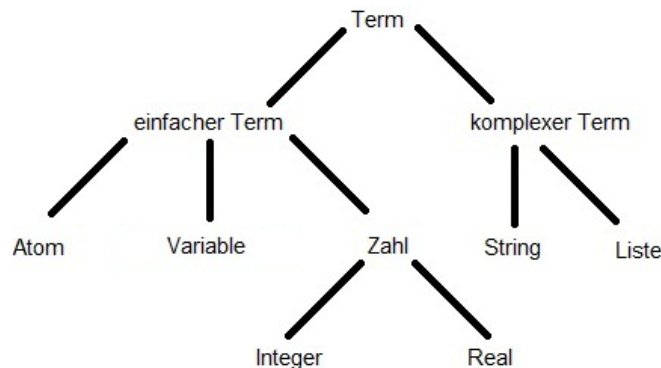


Abbildung 2.4: Datentypen in Prolog

Atome bestehen aus alphanumerischen Zeichen, die mit einem Kleinbuchstaben beginnen. Sie können auch Variablen enthalten, die in einfachen Anführungszeichen geschrieben wurden. Variablen fangen mit einem Großbuchstaben oder Unterstrich an und können alphanumerisch sein. Der Zahlentyp Integer entspricht den natürlichen Zahlen, der Zahlentyp Real dem der Gleitkommazahlen. Ein Spezialfall von komplexen Termen sind Listen. Eine Liste beinhaltet mehrere Einträge, auf die rekursiv zugegriffen wird. Strings zählen ebenfalls zu den komplexen Datentypen in Prolog und werden üblicherweise in doppelten Hochkommata gesetzt, wobei es da auch andere Konventionen gibt.

Wenn man, wie es in Naproche der Fall ist, Prolog durch das Hinzuladen des Moduls „gulp4swi.pl“ erweitert, gibt es mit den GULP-Listen einen weiteren Spezialfall von komplexen Termen. GULP steht für Graph Unification and Logic Programming und ermöglicht die einfache Erstellung von Graphen. Mit GULP ist es möglich, gezielt auf die Werte in einer Struktur zugreifen zu können, ohne die genaue Kenntnis über diese Struktur zu besitzen; Übertragen auf eine Datenbank heißt es, dass man lediglich die Bezeichnung einer vorhandenen Spalte und die Nummer der Spalte benötigt, um auf den Inhalt zugreifen zu können.

Typentheorie

Die Typentheorie ist eine von Bertrand Russel gefundene Lösung zu der nach ihm benannten Russellschen Antinomie. In einer Antinomie hat man zwei Aussagen, die einander widersprechen. Sehr bekannt ist das Beispiel von einem Barbier, der alle Männer in seinem Wohnort rasiert, die sich nicht selbst rasieren. Die Frage ob sich der Barbier selbst rasiert oder nicht, führt zu einem Widerspruch. Bertrand Russel fand einen solchen Widerspruch in der naiven Mengenlehre, wonach eine Menge M als die „Menge aller Mengen, die sich nicht selbst enthalten“ definiert werden kann. Diese Definition führt zu einem Widerspruch, da sich demzufolge M sowohl selbst als auch nicht selbst enthält. Russell löst diese Antinomie durch seine Typentheorie. Nach dieser Theorie gibt es einfache Mengen, die nur einfache Elemente, jedoch keine Mengen als Elemente enthalten können. Mengen, die einfache Mengen und Elemente enthalten, gehören zum zweiten Typ. Mengen, die Mengen des zweiten Typs enthalten können zum dritten Typ usw. Formal ausgedrückt: Die Aussagen: $x \in x$ und $x \notin x$ sind nicht erlaubt. In diesem System ist die Darstellung der Menge aller sich nicht selbst enthaltenden Mengen schon aus syntaktischen Gründen nicht möglich (siehe [23]).

Für die Alltagssprache bedeutet das, dass man zwischen verschiedenen Aussageebenen unterscheiden muss. Ein Ausdruck, der sich auf alle Gegenstände eines Typs bezieht, befindet sich auf einer höheren Ebene als die Gegenstände, auf die er sich bezieht. Der Allsatz: „Alle Männer werden rasiert.“ ist von einem anderen (höherem) Typ als der Satz: „Der Barbier wird rasiert.“. Daher ist das oben genannte Beispiel nach der Typenlehre nicht erlaubt. Die verschiedenen Sprachebenen können durch die Begriffe Objektsprache und Metasprache beschrieben werden. In der Objektsprache sind die direkten Ausdrücke einer Sprache enthalten, in der Metasprache wird über Aussagen dieser Objektsprache gesprochen.

Ein Beispiel: Eine „kurzer Satz“ wird als einen Satz mit nicht mehr als fünf Worten definiert. Die Aussage „Dieser Satz ist kurz.“ scheint also wahr zu sein. Es werden jedoch zwei Welten beschrieben: die Welt aller kurzen Sätze und der Satz selber. Probleme entstehen durch den Bezug der Aussage auf sich selbst, wodurch sich die erste Ebene mit der zweiten Metaebene vermengt. Dadurch ist der Satz im Sinne der Typentheorie keine Aussage, da zwei Ebenen vermischt werden (siehe [21]).

2.4.2 Prolog und Java

In diesem Unterkapitel werden die Unterschiede zwischen Prolog und Java näher erläutert. Weiterhin werden die Begriffe, die in dem Zusammenhang der beiden Programmiersprachen in dieser Thesis häufig verwendet werden, erklärt.

2.4.2.1 Instanziiieren

In Prolog muss einer Variable im Quellcode weder einen Wert noch einen Wertebereich (Datentyp) zugewiesen werden. Erhält eine Variable einen Wert, wird diese von da an als Konstante angesehen. Ist sie uninstanziiert, also ohne Wert, wird sie weiter als Variable verwendet. Bei Java wird der Wertebereich der Variablen durch den Typ bei der Instanziierung festgelegt. Bei einer Variablendefinition ist der Wert typenunabhängig immer „null“. Im Laufe des Programms kann sich der Wert im Rahmen des Datentyps durch eine einfache Zuweisung ändern.

2.4.2.2 Prolog-Prädikate und Java-Methoden

Bei Prolog-Prädikaten handelt es sich um eine Menge von Klauseln, deren Kopfzeile die selben Argumente haben. Klauseln sind eine Generalisierung aus Fakten und Regeln. Die Rückgabewerte werden in den Argumenten aufgeführt.

Java-Methoden werden benutzt, um das Verhalten von Objekten anzugeben. Jede Methode hat einen Namen, eine bestimmte Anzahl von Argumenten und einen definierten Rückgabewert.

Beispiel 2.5

In Java kann eine Methode zum addieren wie folgt programmiert werden:

```
int add(int a, int b){ return a+b; }
```

Das Äquivalent dazu ist dieses Prolog-Prädikat:

```
add(A,B,C) :- C is A + B.
```

3 Ist-Zustand des Naproche-Systems (Version 0.47)

Eine wesentliche Aufgabe der Master-Thesis ist es, eine Ist-Analyse der Version 0.47 des Naproche-Systems durchzuführen. In diesem Kapitel befindet sich die Dokumentation des Ist-Zustandes. Ergänzende Quellen hierfür sind [26] und [20]. Zunächst wird ein Überblick über die Entstehung und Ziele des Naproche-Projekts gegeben. Die Funktionen des Webinterfaces, welches die Schnittstelle des Naproche-Systems zum Anwender bildet, werden im Anschluß beschrieben. Die einzelnen Module und sowie die Kommunikation zwischen ihnen werden dargestellt und erläutert. Dies beinhaltet insbesondere die wesentlichen Module, die in Prolog die logische und linguistische Verarbeitung übernehmen.

3.1 Das Naproche-Projekt

Der Ansatz des Naproche-Projektes liegt in der Analyse von natürlich-sprachlichen mathematischen Beweisen. Konkret geht es zunächst darum, die Werke von Landau [17] und Euklid [10] in einer geeigneten formalen Sprache zu übertragen. So entstand die kontrollierte natürliche Sprache („controlled natural language“, CNL), die sich unter anderem an schon bekannte Sprachen wie ACE (Attempto Controlled English) von Attempto (siehe [27]) orientiert. Die CNL umfasst einen Bereich der mathematischen Fachsprache auf Englisch. Die entsprechende Grammatik wurde speziell in Hinsicht auf die Verarbeitung der Prädikatenlogik erster Stufe entwickelt. Um die mathematischen Symbole korrekt und einfach darzustellen, werden für diese die LaTeX-Syntax verwendet. So können in dem mathematischen Bereich zwischen zwei „\$“-Zeichen alle

Symbole und Zeichen in den Beweistext eingegeben werden. Aus dem Beweistext kann jederzeit durch eine einfache LaTeX-Kompilierung ein PDF-Dokument erstellt werden.

In den verschiedenen Versionen von Naproche wurde die Sprache und die Grammatik weiterentwickelt, so dass sie immer komplexer und mächtiger wurde. Beispielsweise ist in der Version 0.47 auch erstmals die Bedeutung des regelmäßig benutzten Wortes „the“ implementiert (siehe [6]).

Tritt ein „the“ in einem Satz auf („... the thing...“) , präsupponiert man die Existenz eines Objekts mit den Eigenschaften „thing“. Unter einer Präsupposition versteht man die Voraussetzung, dass etwas existiert, obwohl dies nicht explizit ausgesagt wurde. Durch die Erweiterung der CNL um die Präsuppositionen können nun mathematische Ausdrücke, die ein „the“ beinhalten, logisch korrekt verarbeitet werden.

Mit Hilfe von verschiedenen Prologmodulen ging es nun darum, Beweistexte, die in der CNL geschrieben wurden, maschinell überprüfen zu lassen. Dazu wurde die CNL in Prolog implementiert, u.a. steht die Grammatik der natürlichen Sprache in der Datei „dcg.pl“, während die mathematischen Ausdrücke mit den Regeln aus der Datei „fo_grammar.pl“ überprüft werden (siehe Kapitel 2.2.3). Als linguistisches Format zur Darstellung der Zusammenhänge in einem Satz und zwischen mehreren Sätzen werden PRSs verwendet (siehe Kapitel 2.2.1). Mit diesen kann man erkennen, welche Informationen aus einfachen Sätzen für den Beweis benutzt werden.

Weitere Prolog-Module übersetzen die PRS in die Prädikatenlogik erster Stufe im TPTP-Format. Eine Liste von Prämissen wird aufgebaut und jede neue Aussage muss auf Grundlage der erstellten Prämissenliste bewiesen werden. Die Überprüfung der einzelnen Schritte findet im ATP (siehe Kapitel 2.1.4) statt, in dem die gegebenen Voraussetzungen und Annahmen (Prämissen) so kombiniert werden, dass der Beweis belegt werden kann. Wird in der vorgegebenen einstellbaren Überprüfungszeit des ATPs kein Beweis für die Aussage auf Grundlage der Prämissen gefunden, gilt der Beweis als fehlgeschlagen. Dieses wiederum kann auf Grund der wachsenden Anzahl der zu übergebenden Prämissen sehr lange dauern; der Überprüfungsprozess hat dadurch eine polynomielle oder im worst-case exponentielle Laufzeit (siehe Kapitel 2.3.2).

Das Naproche-System 0.47 hat durch die aufgeführten Module (CNL, ATP, PRS) verschiedene Grenzen: Bei längeren Beweistexten wie Landau[17] muss man den Text nach den Regeln der CNL erfassen und gegebenenfalls an einigen Stellen umformulieren. Die groben Strukturen der Sprache wurden in grammatische Regeln der CNL

spezifiziert, so dass eine automatische semantische Verarbeitung möglich wurde. Es ist in der Version 0.47 nicht möglich, jeden natürlichsprachlichen mathematischen Text zu überprüfen. Die aus einer CNL entstandene PRS ist zwar gut strukturiert, aber im Gesamten nicht mehr übersichtlich. Weiterhin benötigen die ATPs unter Umständen viel Zeit, um einen gültigen Beweis zu finden.

Die CNL wird sich immer weiter entwickeln, wodurch der Originaltext besser verarbeitet werden kann. Um die Laufzeit zu reduzieren wurde eine Prämissenauswahl implementiert, die eine Vorauswahl aller an den ATP zu übergebenen Prämissen eines Beweisschrittes trifft. Die Prämissenauswahl wird mit Hilfe eines Beweisgraphen getroffen, der die logische und textuelle Nähe von Prämissen sowie ihre explizierte Referenzen betrachtet.

3.2 Das Webinterface

Das Webinterface (<http://naproche.net/inc/webinterface.php>) ist die Schnittstelle zwischen dem Programm und dem Anwender. Es folgt die Abbildung 3.1 des Webinterfaces, mit dessen Hilfe der Aufbau und die Funktionen kurz erläutert werden:

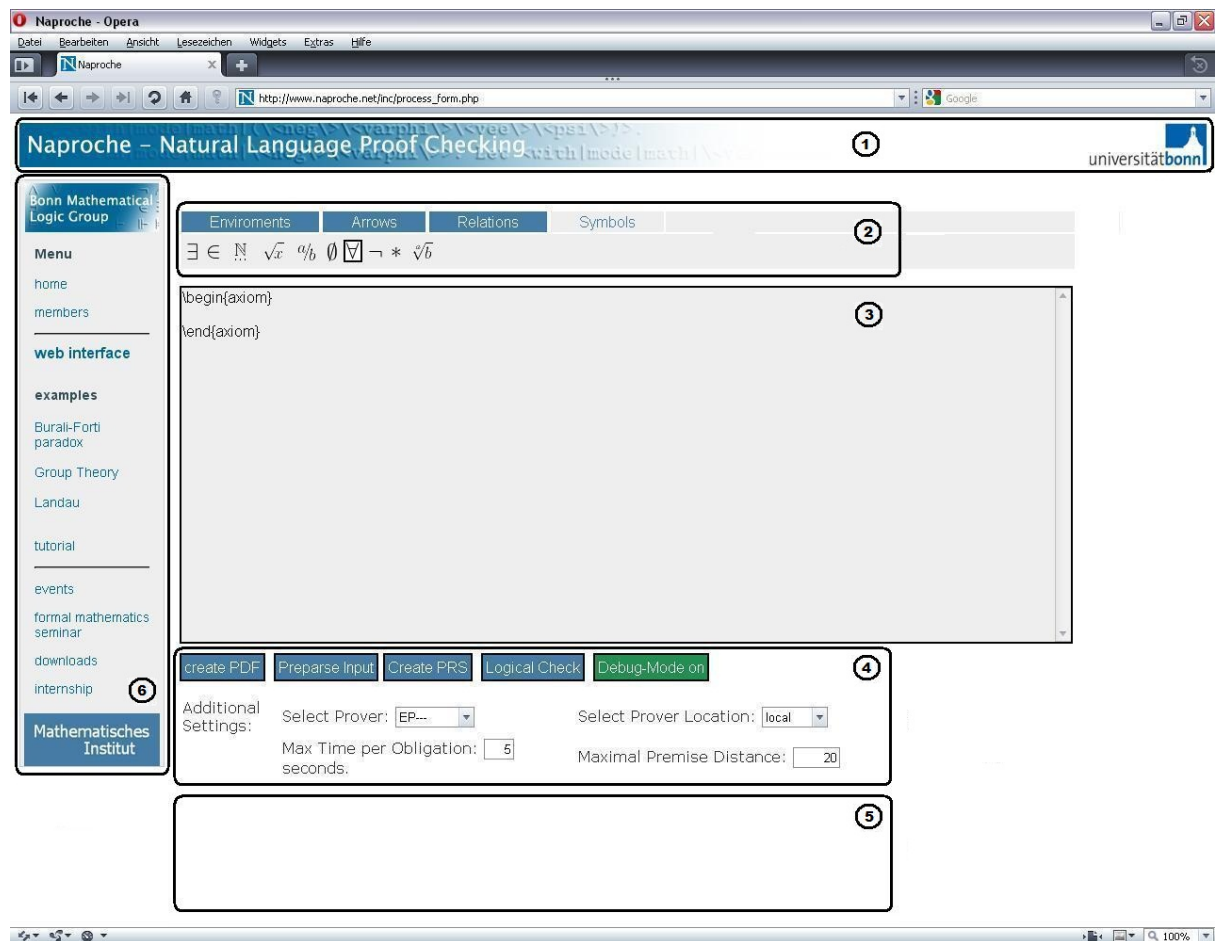


Abbildung 3.1: Bereiche des Webinterfaces

Der Bereich ① ist die Kopfzeile, die mit den Webseiten zur Naproche-Homepage und der Universität Bonn verlinkt ist. Im Bereich ② befinden sich Eingabehilfen zum Erfassen von Beweisen. Klickt man auf eines der Symbole (hier ist \forall markiert), erscheint das ausgewählte Symbol im Textfeld an der Stelle, an der sich vorher der Textcursor befunden hat. Das Textfeld ist der Bereich ③, in dem der Anwender im LaTeX-Stil einen Beweistext eingeben kann. Die Funktionen des Webinterfaces befinden sich im Bereich ④. In der Darstellung ist der Debug-Modus eingeschaltet. Neben den Funktionen „Create PDF“ (erstellt eine PDF-Datei mit dem Inhalt des Eingabefeldes) und „Logical Check“ (startet die Beweisüberprüfung) hat man weitere Möglichkeiten, Hilfsausgaben zu erzeugen („Preparse Input“ und „Create PRS“) und Einstellungen zu ändern (Bereich „Additional Settings“). Die Ausgaben und Ergebnisse der Beweisüberprüfung werden im Bereich ⑤ angezeigt. Die Navigation befindet sich am linken Rand ⑥. Hier kann man über einen Klick die Seite wechseln bzw. sich vorgefertigte Beispieltex-te (Burali-Forti Paradox, Group Theory und Landau) in das Eingabefeld laden.

3.3 Übersichtsdiagramm

Der Arbeitsablauf der Version 0.47 des Naproche-Systems wird im Diagramm 3.2 dargestellt:

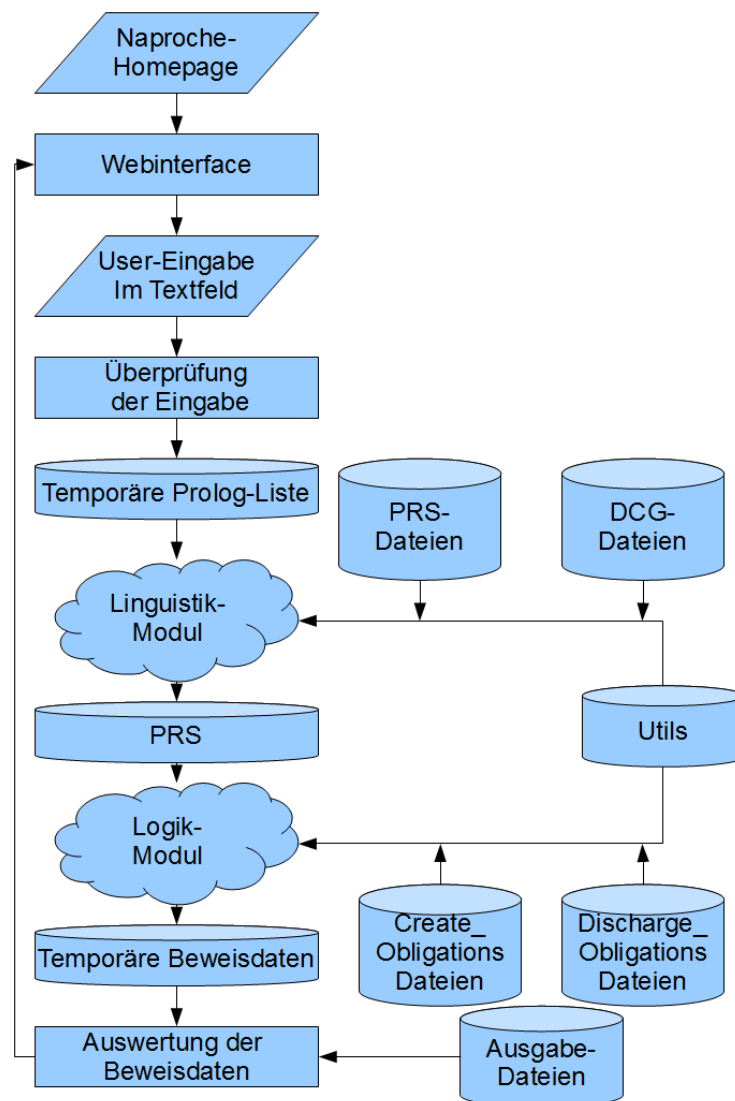


Abbildung 3.2: Übersichtsdiagramm Naproche-System 0.47

Über die Homepage des Naproche-Projekts [<http://naproche.net/>] gelangt man auf das Webinterface (siehe Abbildung 3.2). Im dazugehörigen Textfeld kann ein mathematischer Beweis eingegeben werden. Als Eingabebeispieltext für dieses Kapitel dient hier:

Beispiel 3.1

Axiom.

There is no y such that $y \in \emptyset$.

Define x to be transitive if and only if for all u , v ,
if $u \in v$ and $v \in x$ then $u \in x$.

Define x to be an ordinal if and only if x is transitive
and for all y , if $y \in x$ then y is transitive.

Then \emptyset is an ordinal.

Startet man die logische Beweisüberprüfung wird die Eingabe zunächst über PHP und Javascript-Methoden vorverarbeitet und dabei eine temporäre Satzliste in Prolog-Struktur erzeugt. Diese Liste enthält zu jedem Satz ein Sentence-Objekt.

Die Struktur eines Sentence-Objektes kann wie folgt beschrieben werden:

- Sentence: sentence(<ID>,[<Content>]).
- <ID> : Fortlaufende bei 1 beginnende Nummer
- <Content>: <Atom> oder <MathList> durch Kommata getrennt in beliebiger Reihenfolge und Kombination
- <Atom>: ein beliebiges kleingeschriebenes englischsprachiges Wort oder Ziffern und Satzpunkte außerhalb des LaTeX-Mathebereichs oder Benennung von Axiomen, Definitionen, Lemmata und Theoremen
- <MathList>: Liste von mathematischen Symbolen in LaTeX-Syntax

Das Beispiel 3.1 kann entsprechend in folgender Prolog-Liste dargestellt werden:

```
sentence(1,['axiom']),
sentence(2,['there','is','no',math([y]),
'such','that',math([y,'\in','\emptyset'])]),
sentence(3,['define',math([x]),'to','be','transitive',
'if','and','only','if','for','all',math([u]),',',',math([v]),',',',
'if',math([u,'\in',v]),'and',math([v,'\in',x]),'then',math([u,'\in',x])]),
sentence(4,['define',math([x]),'to','be','an','ordinal',
'if','and','only','if',math([x]),'is','transitive',
'and','for','all',math([y]),',',',if',math([y,'\in',x]),
'then',math([y]),'is','transitive']),
sentence(5,['then',math(['\emptyset']),'is','an','ordinal'])
```

Wie im Übersichtsdiagramm 3.2 zu erkennen ist, wird die Satzliste an das Linguistik-Modul übergeben. Dieses Modul erstellt durch das Prolog-Prädikat *Build PRS* eine PRS (siehe Kapitel 2.2.1). Dabei werden weitere Prologmodule benutzt, die sich in der Datei „prs.pl“ befinden. In der Datei „prs_export.pl“ sind die Prädikate vorhanden, die für die Darstellung einer PRS benötigt werden. Ebenfalls benutzt werden die Prädikate für den DCG-Parser (Definit-Klausel-Grammatiken, siehe Kapitel 2.2.3), die sich in den Dateien „dcg.pl“, „dcg_error.pl“, „dcg_error_utils.pl“, „dcg_simple.pl“, „dcg_utils.pl“ und „fo_grammar.pl“ befinden. Die genauere Funktionsweise des Moduls *Build PRS* wird im Kapitel 3.4.1 näher beschrieben.

Die Ausgabe (output) des Linguistik-Moduls ist die PRS des Beweises, die als Eingabewert (input) an das Logik-Modul übergeben wird. Hier werden mit dem Prädikat *create_obligations* die Obligationen erstellt (siehe Kapitel 3.4.2) und mit dem Prädikat *discharge_obligations* (siehe Kapitel 3.4.3) die Obligationen unter Verwendung eines ATP-Programms bewiesen. Die Dateien, die die Prologprädikate zur Obligationserstellung zur Verfügung stellen, sind „create_obligations.pl“, „premises.pl“ und „graph.pl“.

Ferner sind „discharge_obligations.pl“ und „translation_tptp.pl“, die Dateien, in denen sich die Prädikate für die Umwandlung in die für ATPs taugliche TPTP-Syntax befinden. Der ATP erzeugt bei der Überprüfung des Beweises temporäre Beweisdatei, die ausgewertet und auf dem Webinterface benutzerfreundlich ausgegeben wird. In-

formationen zu der Überprüfung der einzelnen Beweisschritte (ATP In- und Output) können über die entsprechenden Links des farblich gekennzeichneten Beweistextes auf der Oberfläche des Webinterfaces abgerufen werden. Die Links zu den Statistiken zum vollständigen Beweis sowie dem Beweisgraph befinden sich unterhalb des farblichen Beweistextes. Die Dateien „output.pl“ und „stats.pl“ beinhalten die Prädikate für die Ausgaben im Webinterface.

3.4 Die wesentlichen Prologmodule

In diesem Kapitel werden die wesentlichen Prologmodule beschrieben, die zunächst zum überprüfenden Beweistext die PRS und die Obligationen erstellen, bevor die Anfragen an den ATP geschickt werden. Der Aufbau der PRS wurde im Kapitel 2.2.1 näher erläutert; das Kapitel 2.1.4 gibt einen Überblick über ATPs. Mit der Datei „load.pl“ werden alle Dateien geladen, die für alle Module Prädikate zur Verfügung stellen sollen. Die allgemeingültigen Prädikate stehen in der Datei „utils.pl“. Die Dateien mit den Prädikaten zur spezifischen Weiterverarbeitung werden an den entsprechenden Stellen in diesem Kapitel beschrieben.

Eine Besonderheit, die hier in einigen Fällen (z.B. „check_conditions()“) auftritt, ist der rekursive Aufruf eines Prädikates. Hierbei werden Baumstrukturen durchgearbeitet, wobei für jedes Element bestimmt wird, in welchen der verschiedenen Knoten es bearbeitet werden muss. Ist die Baumstruktur durchgearbeitet, also die Liste leer, wird das Prädikat anhand einer Haltemarke beendet.

3.4.1 Build PRS

Das Prädikat „Build_PRS()“ befindet sich in der Datei „prs.pl“. Es wird genutzt, um aus einer Prolog-Liste eine PRS zu erstellen. Die Liste muss eine Satzliste sein, wie sie im vorangegangenen Kapitel 3.3 anhand des Beispiels 3.1 beschrieben wurde. Hilfsprädikate zum Exportieren einer PRS sind in der Datei „prs_export.pl“ vorhanden. Zur Überprüfung der Sprache werden die Prädikate in den Dateien „dcg_error.pl“, „dcg_error_utils.pl“, „dcg_utils.pl“ und „dcg_simple.pl“ benutzt. Die Datei „dcg.pl“ beinhaltet die Grammatik der Sprache, nach der der Eingabetext überprüft wird. Die

mathematische Formelstruktur ist in der Datei „fo_grammar.pl“ definiert.
Die Verarbeitung von „Build_PRS()“ läuft wie in Abbildung 3.3 gezeigt ab:

build_prs():

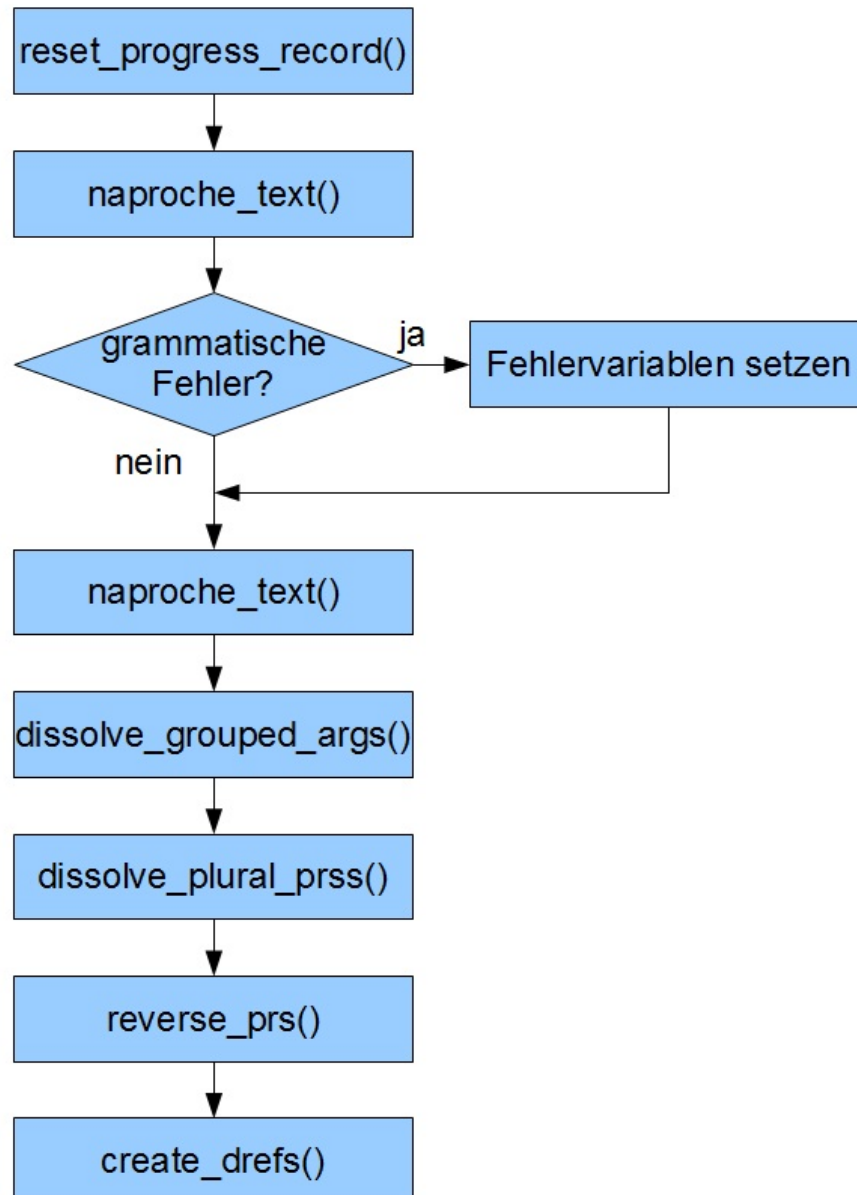


Abbildung 3.3: „Build_PRS()“

Zunächst werden die Werte von globalen Variablen, die den Fortschritt des Parsing-Prozesses messen, mit der Methode „reset_progress_record()“ zurückgesetzt (siehe Abbildung 3.3). Der übergebene Text wird eingelesen und als erstes mit dem Prädikat „naproche_text()“ aus der Datei „dgc_error.pl“ überprüft, ob es sich bei dem Eingabetext um einen gültigen Text im Sinne der Naproche-Sprache (siehe Kapitel 2.2.2) handelt. Sollten grammatikalische Fehler vorhanden sein, werden die entsprechenden Werte in die dafür vorgesehenen Variablen übertragen.

Anschließend wird fehlerunabhängig mit dem gleichnamigen Prädikat „naproche_text()“ aus der Datei „dgc.pl“ eine vorläufige PRS erstellt: „naproche_text()“ ist über eine DCG-Grammatik definiert: In dieser Grammatik ist „statement()“ ein wichtiges Prädikat, welches auf Satzebene überprüft, ob es sich um einen Aussagesatz handelt. Das Prädikat „propositon_cord()“ parst den Hauptteil eines Aussagesatzes. Handelt es sich nicht um eine Aussage, werden andere Prädikate wie „definition_text()“ (für Definitionen) oder „assumption_text()“ (für Voraussetzungen) aufgerufen. Die Unterscheidung hierbei ist wichtig, da die Sätze nach anderen Kriterien verarbeitet werden müssen.

Die provisorische PRS wird über die Prädikate „dissolve_grouped_args()“ und „dissolve_plural_prss()“ so umstrukturiert, dass Pluralkonstruktionen im Eingabetext richtig interpretiert werden (siehe [6]). Das rekursive Prädikat „reverse_prs()“ dreht die so entstandene PRS um, wodurch die Sub-PRSsen berücksichtigt werden, die weiterhin ihrer jeweiligen übergeordneten PRS zugeordnet bleiben. An das anschließende Prädikat „create_drefs()“ werden die PRSsen mit nicht instanziierten Drefs übergeben. „create_drefs()“ instanziiert alle Dref-Variablen mit aufsteigenden natürlichen Zahlen beginnend bei 1.

3.4.2 Create Obligations

In den Dateien „create_obligations.pl“, „premises.pl“ und „graph.pl“ befinden sich die Prädikate für die Erstellung der Obligations. Hauptsächlich beinhaltet die Datei „create_obligations.pl“ verschiedene Fallunterscheidungen des Prädikates „check_conditions()“, welches wiederum von dem Hauptprädikat „create_obligations()“ aufgerufen wird. Zusätzlich wird über das Prädikat „create_obligations()“ in Abhängigkeit der PRS der Beweisgraph erstellt.

Mit „check_conditions()“ werden die verschiedenen PRS-Bedingungen nach der Baumstruktur der PRS verarbeitet. Dabei wird die Prämissenliste erweitert und auf Grundlage dieser die Obligationen erstellt (siehe [6]). Die beiden am häufigsten auftretenden Fälle sind PRS und Assumption, die in jeweils einem Diagramm (siehe Abbildung 3.4 und Abbildung 3.5) dargestellt und näher beschrieben werden:

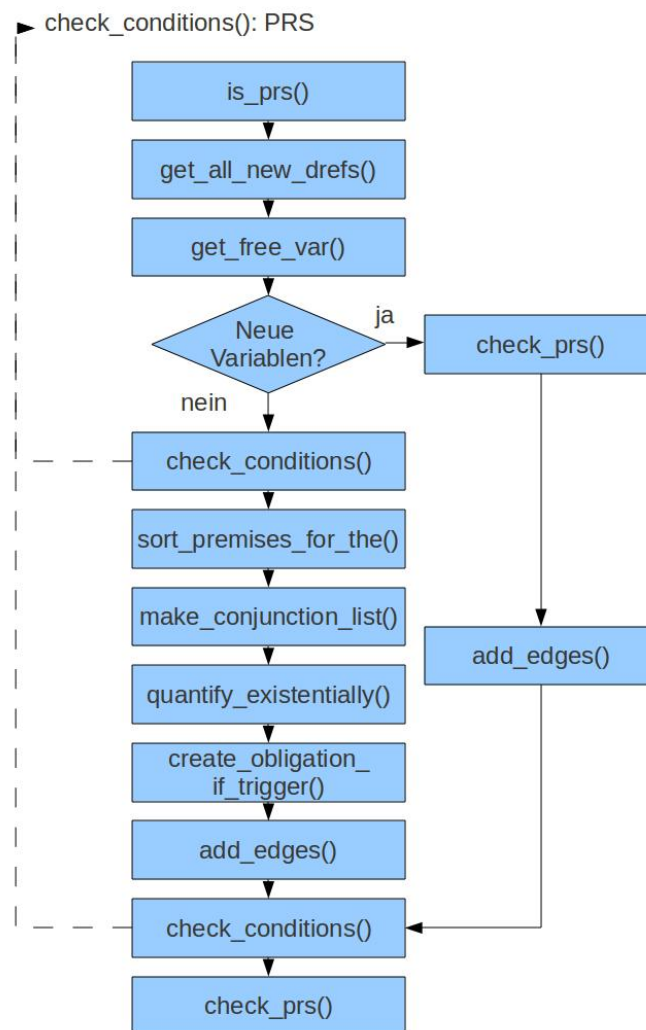


Abbildung 3.4: „check_conditions()“: PRS

Durch den Aufruf des Prädikates „is_prs()“ wird erkannt, dass eine PRS überprüft werden muss (siehe Abbildung 3.4). Hier werden die Werte der PRS den entsprechenden Variablen zugewiesen. Mit „get_all_new_drefs()“ und „get_free_var()“ werden die Drefs und freien Variablen ebenfalls nacheinander eingelesen und gespeichert. Falls noch unbekannte Variablen in der PRS vorhanden sind, die noch nicht zugeordnet werden konnten, wird über „check_prs()“ und „add_edges()“ die PRS erneut überprüft, bevor die neu gefundenen PRS-Elemente dem Beweisgraphen hinzugefügt werden. Das rekursive Prädikat „check_conditions()“ wird mit dem Wert „nocheck“ aufgerufen. Der Wert „nocheck“ ist ein interner Wert zum Aufbau von Prämissen und Conjecture für den ATP (siehe Kapitel 2.1.4): bei „nocheck“ handelt es sich um ein Axiom, bei „check“ um eine Conjecture. Generell gilt, dass sich ein „nocheck“ in die PRS-Boxen hineinvererbt. Die Ausnahme ist das englische Wort „the“. Steht dieses in einer PRS-Box, wird das „nocheck“ zum „check“, und damit vom ATP überprüft. Um diesen „the“-Fall korrekt verarbeiten zu können, werden die Voraussetzungen für das „the“ explizit mit dem Prädikat „sort_premises_for_the()“ sortiert, mittels „make_conjunction_list()“ und „quantify_existentially()“ konjugiert und existentiell quantifiziert, und aus der daraus resultierenden Formel mit „create_obligation_if_trigger()“ eine Obligation erstellt. Nachdem der Beweisgraph mit „add_edges()“ aktualisiert wurde, wird rekursiv erneut „check_conditions()“ aufgerufen. Eine weitere Überprüfung der PRS mit „check_prs()“ beendet diesen Fall.

In dem Fall „Assumption“ im Prädikat „check_conditions()“ sollen alle Annahmen aus einer PRS gefunden werden, wie in Abbildung 3.5 dargestellt und erläutert wird.

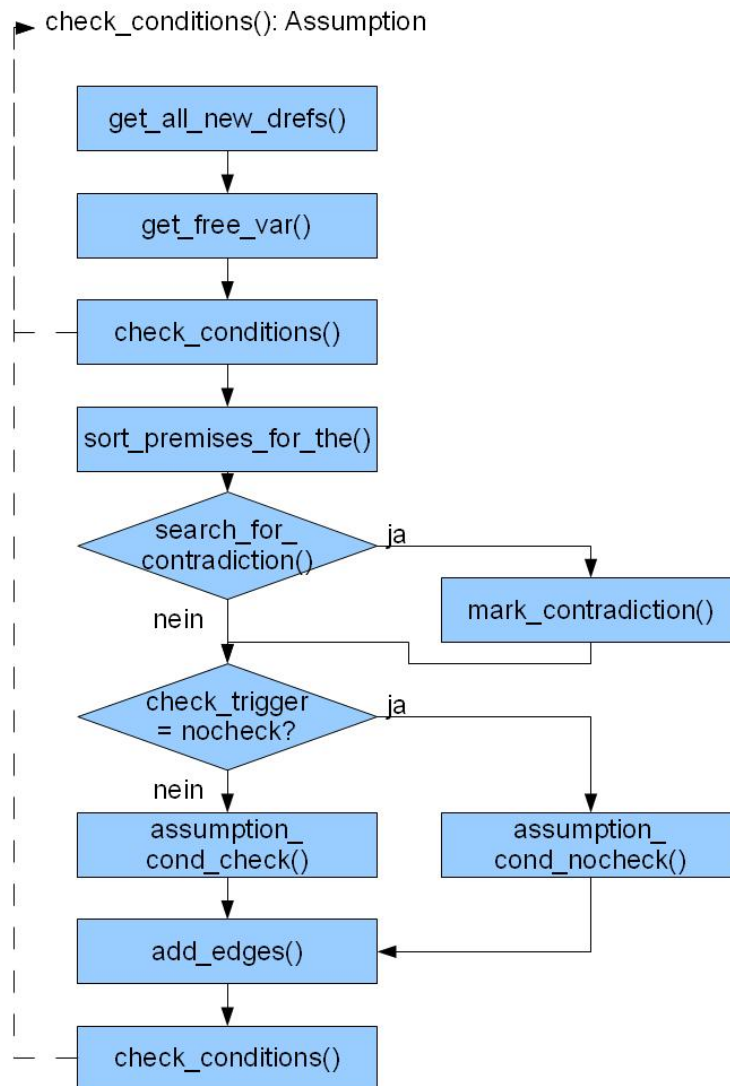


Abbildung 3.5: „check_conditions()“: Assumption

Mit den Prädikaten „get_all_new_drefs()“ und „get_free_var()“ werden die Drefs und freien Variablen nacheinander eingelesen und gespeichert (siehe Abbildung 3.5). Das rekursive Prädikat „check_conditions()“ wird aufgerufen, um die Weiterverarbeitung zu gewährleisten. Mit „sort_premises_for_the()“ wird die Liste der Voraussetzungen sortiert und anschließend mit „search_for_contradiction()“ nach einem Widerspruch durchsucht. Wurde ein Widerspruch gefunden, wird dieser mit dem Prädikat „mark_contradiction()“ markiert. Sollten die Annahmen mit einem Widerspruch enden, wird dieser mit „check_trigger()“ gekennzeichnet. Ist „nocheck“ der Wert von „check_trigger()“, wird das Prädikat „assumption_cond_nocheck()“ aufgerufen, um Annahmen zu erstellen, die nicht überprüft werden müssen. Alternativ werden mit „assumption_cond_check()“ die Annahmen analysiert in dem festgestellt wird, ob das, was aus den Annahmen folgt, überprüft werden muss. Unabhängig vom „check_trigger“-Wert wird der Beweisgraph aktualisiert und abschließend das Prädikat „check_conditions()“ aufgerufen.

3.4.3 Discharge Obligations

In den Dateien „discharge_obligations.pl“, „output.pl“, „stats.pl“ und „translation_tptp.pl“ befinden sich die Prädikate für die eigentliche Beweisüberprüfung. Der mathematische Beweis wird hier schrittweise in ATP-lesbare Abfragen (TPTP-Format, siehe Kapitel 2.1.4) umgewandelt und aufgerufen. Der Ablauf des Hauptprädikates „discharge_obligations()“ ist in der Abbildung 3.6 dargestellt:

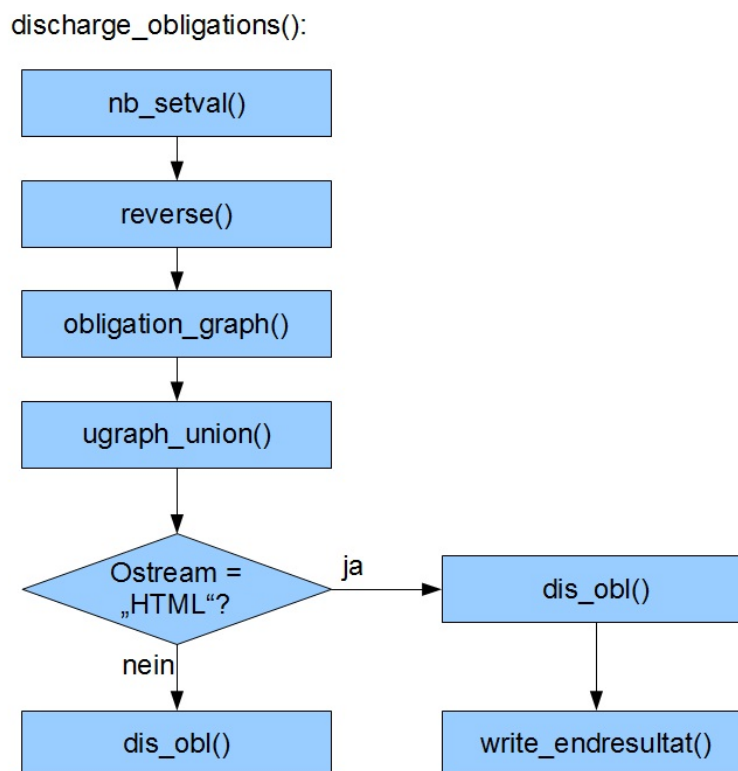


Abbildung 3.6: „discharge_obligations()“

Mit dem Prädikat „nb_setval()“ werden die globalen Variablen, die für die Generierung der ATP-Anfragen benötigt werden, gesetzt. Die Liste der Obligationen wird anschließend mit „reverse()“ umgedreht. Das Prädikat „obligation_graph()“ erstellt den Obligationsgraphen, wobei mit „ugraph_union()“ der Obligationsgraph an den globalen Graphen angefügt wird. Es folgt eine abschließende Überprüfung des Wertes „ostream“. Beinhaltet dieser den Wert „HTML“ wird zunächst „dis_obl()“ aufgerufen, welches im Anschluss näher beschrieben wird. Dies ist der Fall, wenn Naproche über das Webinterface und nicht lokal über die Prolog-Module aufgerufen wird. Die Ergebnisse der Berechnungen werden mit dem Prädikat „write_endresult()“ in HTML-Format an die Datei „process_form.php“ während der Bearbeitung des Beweises hinzugefügt. Ist „ostream“ nicht der Wert „HTML“, wird lediglich „dis_obl()“ aufgerufen.

In „dis_obl()“ findet ein rekursiver Aufruf von „check_conj()“ statt. „check_conj()“ wandelt die jeweilige Anfrage in TPTP-Syntax um, berechnet die Distanzen zu den Prämissen und ruft den ATP auf.

4 Anforderungsprofil an die Version 0.5 des Naproche-Systems

In diesem Kapitel werden die Ziele aufgelistet und erläutert, die ab der Version 0.5 des Naproche-Systems realisiert werden sollen.

4.1 Modifikationsziele

Das Naproche-System soll leistungsfähiger werden. Werden Beweistexte eingegeben und überprüft, dauert es mitunter sehr lange bis es zu einem Ergebnis der Beweisüberprüfung kommt. Durch ein Axiomauswahlmodul soll die Geschwindigkeit der Überprüfung erhöht werden. Die den Überprüfungsmodulen zu Grunde liegende Formelgrammatik soll ebenfalls überarbeitet werden. Die Version vor 0.5 hat mit der Prädikatenlogik der ersten Stufe gearbeitet, das neue System soll auch mit höheren Stufen umgehen können. Weiterhin ist die Formelgrammatik der Version 0.47 wenig flexibel. So standen beispielsweise lediglich „f“, „g“ und „h“ und keine anderen Buchstaben für mathematische Funktionen im Beweistext zur Verfügung.

Erfasste Beweistexte sollen mit ihren Überprüfungsergebnissen gespeichert werden können. Wenn nun ein Anwender einen bereits vorhandenen Beweistext fortsetzen möchte, sollen die Ergebnisse der letzten Überprüfung sowohl für die Beweisüberprüfungsmodule als auch für den Anwender sichtbar zur Verfügung stehen. Durch die Speicherverwaltung wird auch inkrementelles Parsen des Beweistextes möglich.

Ein weiteres Ziel ist es, die Benutzerfreundlichkeit zu erhöhen. Zu diesem Zweck soll ein Grafical User Interface (GUI) entwickelt werden. Über das GUI soll ein Anwen-

der die Beweise speichern, laden und auswerten können. Mathematische Formeln im Beweistext sollen weiterhin in LaTeX-Syntax eingegeben werden.

4.2 Umstrukturierung der Prologmodule

Um die Modifikationsziele erreichen zu können, müssen einige der Prologmodule überarbeitet werden. Die wesentlichen Prologmodule wurden bereits im Kapitel 3.4 beschrieben.

Zur Verbesserung des Ablaufs der Prologmodule soll in der Version 0.5 der Aufbau der PRS zum Beweis nicht mehr vor der logischen Überprüfung der Aussagen sondern parallel dazu stattfinden. So können die Ergebnisse der logischen Überprüfung schon mit in die PRS einfließen.

Neu wird auch das Speichern und Laden von Zwischenergebnissen der Beweisüberprüfung in die entsprechenden Prologmodule. Das Speichern muss in einer externen Datei in einem geeigneten Format stattfinden, so dass die Daten eindeutig zu jedem Beweisschritt zugeordnet werden können. Geladen werden muss in eine Prologstruktur, die die jeweilige Beweisstruktur widerspiegelt.

Weiterhin soll die linguistische Verarbeitung aufgeteilt werden. Der Macroparser soll die strukturelle Gliederung des Textes in verschiedenartige Sätze und Markierungen wie „Theorem“, „Proof“ und „Qed“ erkennen, während der Mikroparser einzelne Sätze linguistisch parst. Um die Formelgrammatik flexibler zu gestalten und mehr als die Prädikatenlogik erster Stufe verarbeiten zu können, muss die Formelgrammatik neu implementiert werden.

Bei allen Änderungen und Anpassungen müssen immer alle von der Modifikation betroffenen Module überprüft und angepasst werden.

4.3 Neuentwicklung des Proof-State-Datentyps

Ziel dieser Master-Thesis ist die Entwicklung eines persistenten Datentyps. Es soll ein Datentyp entwickelt werden, mit dessen Hilfe Beweistexte nicht jedesmal komplett neu auf ihre Richtigkeit überprüft werden müssen. Die Daten zu einem Beweis sollen mit dem Datentyp gespeichert werden. Dies muss persistent, also nicht flüchtig geschehen. Eine transiente (übergangsweise) Datenspeicherung reicht nicht aus, da die Daten im Speicher bei einem Neustart nicht mehr zur Verfügung stehen, so dass der Beweis jedes Mal neu überprüft werden müsste. Dazu müssen alle Daten, die während der Überprüfung entstehen, strukturiert verwaltet werden. Das Ziel dieser Verwaltung ist der Proof-State-Datentyp. Durch diesen Datentyp soll inkrementelles Parsen auf Satzebene ermöglicht werden. Beim inkrementellen Parsen wird die Syntax-Analyse des Satzes durchgeführt, ohne dass der komplette Beweis bei einer Änderung neu überprüft wird. Die Rückmeldung zwischen Anwender und Programm wird durch den Datentyp ebenfalls vereinfacht.

4.4 Neuentwicklung eines Grafical User Interface

Die Analyse des Webinterfaces ergab, dass es zwar im Rahmen der definierten Sprache und Grammatik fehlerfrei funktioniert, aber eine Erweiterung um einen persistenten Datentyp zusätzliche Ressourcen benötigt:

Es würde zusätzliche Hardware, wie ein leistungsfähiger Server, benötigt, um auf verschiedene Benutzerkonten mit unterschiedlichen Rechten Daten zu verwalten. Jedem Anwender muss genug Speicherkapazität zur Verfügung stehen, um die temporären Daten der Beweistextüberprüfung speichern zu können. Da diese Ressourcen auf absehbarer Zeit nicht zur Verfügung stehen, wird folgende Alternative bevorzugt:

Den Rahmen für die gewünschten Modifikationsziele soll ein Grafical User Interface (GUI) liefern. Diese neu zu entwickelnde GUI soll benutzerfreundlich und intuitiv zu bedienen sein, und vom Anwender als Offline-Software lokal installiert werden. Die Rückmeldung zwischen Anwender und Programm sollen vereinfacht und besser veranschaulicht werden. Sie wird als Prototyp für den Proof-State-Datentyp entworfen und benötigt eine Anbindung an die Prolog-Module. Neben der damit verbundenen Verwaltung von Beweistexten und deren Überprüfungsschritten soll es über die GUI noch möglich sein, alle Informationen und statistischen Auswertungen, die im Webinterface zur Verfügung stehen, zu gelangen. Sinnvolle weitere Funktionen wie beispielsweise das Drucken des Beweistextes sollen ebenfalls realisiert werden bzw. die GUI um diese Funktionen leicht erweiterbar sein.

5 Implementation des Prototyps

In diesem Kapitel wird die Entwicklung des Prototypen des Naproche-Systems der Version 0.5 beschrieben. Zunächst wird die allgemeine Architektur des Programmsystems erläutert, die aus dem Grafical User Interface und dem Prototypen besteht. Insbesondere wird auf die Hauptmodule des Programms sowie die Kommunikation zwischen Java und Prolog eingegangen. Die Kommunikation der einzelnen Javamodule untereinander wird anhand des Klassendiagramms näher erläutert, ebenso die einzelnen Klassen und deren Funktion. Das Design des Programmsystems wird genau wie die wesentlichen Methoden in weiteren Kapiteln beschrieben. Zu den wesentlichen Aufgaben der Master-Thesis gehören die Entwicklung der Java-Module, die hier dokumentiert wurden. Die Prolog-Module wurden von Marcos Cramer, Julian Schlöder und Johannes Seeler als angestellte Naproche-Entwickler weiterentwickelt.

5.1 Allgemeine Architektur des Programmsystems

Die Architektur des Programmsystems wurde anhand der Anforderungen (siehe Kapitel 1.3) spezifiziert. In dem Unterkapitel 5.1.1 werden die Hauptmodule des neuen Naproche-Systems erläutert. Um die Kommunikation mit einem Anwender möglichst intuitiv zu gestalten, wurde ein Graphical User Interface (GUI) parallel zu dem Prototypen dieser Master-Thesis entwickelt. Grundsätzlich soll der Prototyp leicht erweiterbar sein, da die Entwicklung der CNL noch nicht abgeschlossen ist, wodurch neue Anforderungen an das System gestellt werden.

Das System wurde nach dem Top-Down Prinzip entworfen. Wird eine Software nach diesem Prinzip entwickelt, steht das Gesamtsystem im Vordergrund. Die erwarteten funktionalen Anforderungen an das System werden erst im Laufe der Entwicklung

näher spezifiziert. Die Implementation beginnt erst, wenn die Anwendungsziele klar geplant sind. Eine wesentliche Aufgabe des Prototypen besteht aus der Kommunikation zwischen den Java- und Prologmodulen, die im Unterkapitel 5.1.5 beschrieben wird.

5.1.1 Die Hauptmodule

Die Hauptmodule des Prototyps sind im folgenden Datenflussplan dargestellt:

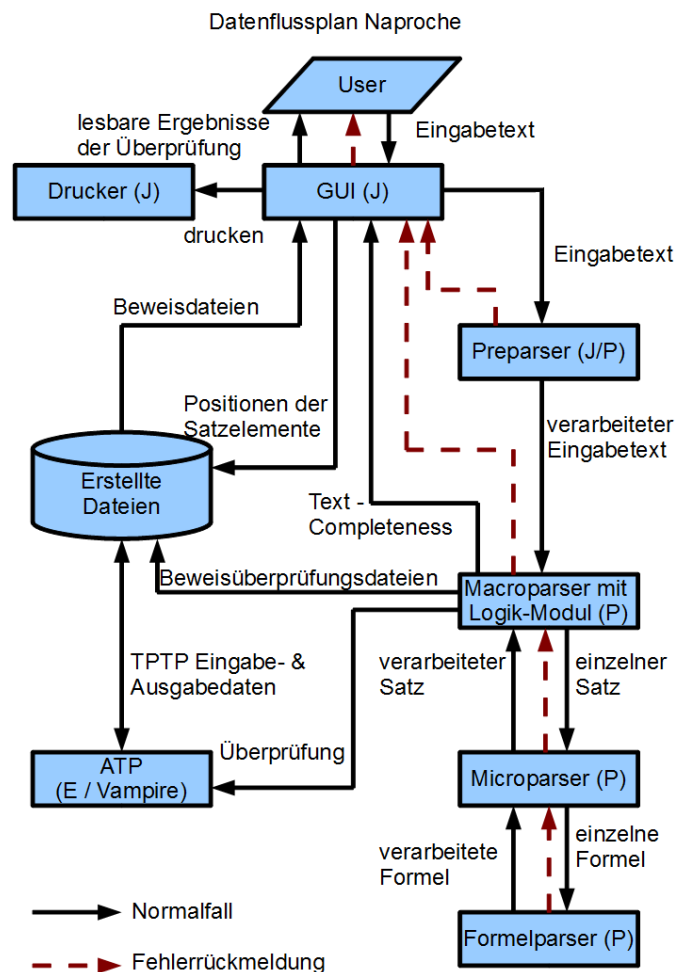


Abbildung 5.1: Übersichtsdiagramm Naproche-System 0.5

Die mit (J) gekennzeichneten Diagrammelemente der Abbildung 5.1 sind Java-Module. Elemente mit einem (P) sind Prolog-Module. Der Preparser (siehe Kapitel 5.1.2) hat als einziges Modul sowohl Java- als auch Prolog-Module.

Der Anwender startet das Grafical User Interface (GUI) und gibt in dem Textfeld den Eingabetext ein. Dieser Text kann über einen dafür vorgesehenen Button ausgedruckt werden. Soll der Eingabetext überprüft werden, wird zunächst der Preparser aufgerufen. Dieses Prologmodul wandelt die vorhandene Satzstruktur des Eingabetextes in eine Prolog-lesbare Liste um. Konnte diese Liste nicht erstellt werden, erfolgt eine Fehlermeldung über das GUI an den Anwender. Ansonsten wird die Liste über ein weiteres Prologmodul, den Macroparser, weiterverarbeitet.

Im Marcoparser ist ein Logikmodul implementiert; der Macroparser parst den Satz, während das Logik-Modul die logische Überprüfung durchführt. Hierbei werden durch die logische Überprüfung die Beweisüberprüfungsdateien erzeugt. Zum Parsen der einzelnen Formeln wird der Formelparser, welcher durch den Microparser aufgerufen wird, verwendet. Sollten hierbei Fehler auftreten, findet eine Fehlerrückmeldung an die nächsthöhere Ebene statt (Formelparser nach Microparser, Microparser nach Macroparser, Macroparser zum GUI und Preparser zum GUI). Durch den Aufruf des Macroparsers wird der Eingabetext in entsprechend viele ATP-Input-Dateien im TPTP-Format (siehe Kapitel 2.1.4) umgewandelt und anschließend der ATP ausgeführt. Mit Hilfe des ATPs werden alle Aussagesätze überprüft und die jeweiligen Ausgabedateien durch den ATP erzeugt. Die Rückmeldung des Macroparsers an das GUI erfolgt über den Wert „Text Completeness“, der angibt, ob der Beweistext als in sich abgeschlossener Text oder als noch zu erweiternder Text erkannt wurde. Die entstandene Datenstruktur wird im Unterkapitel 5.1.4 genau wie der Aufbau der im Prozess erzeugten Dateien näher beschrieben.

5.1.2 Der Preparser

Prolog-Prädikat zum Preparser

Der Preparser wurde im Rahmen eines Praktikums bei Naproche von Mona Rahn entworfen [19] und von Julian Schlöder weiterentwickelt [20]. Der Preparser ist zum Zeitpunkt der Entstehung dieser Master-Thesis bereits in seiner Entwicklung abgeschlossen. Die eigentliche Überprüfung der Satzstruktur durch den Preparser findet in dem Prolog-Modul „input_parser.pl“ statt. Das folgende Diagramm stellt den Verlauf des Hauptprädikats „create_naproche_input()“ dar:

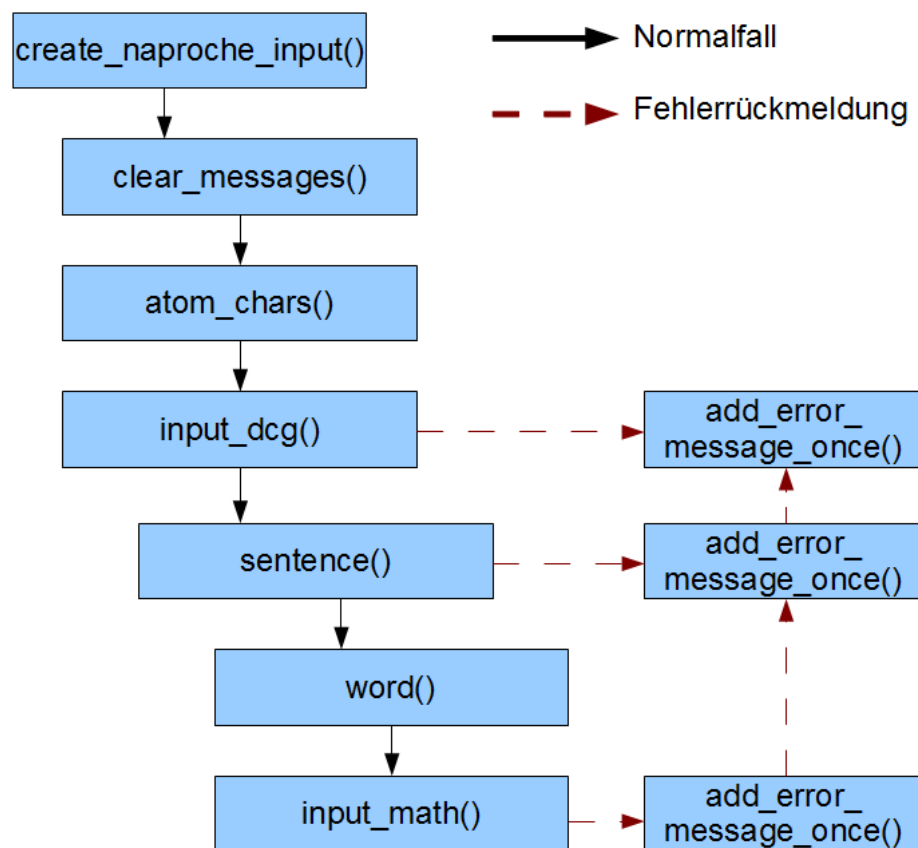


Abbildung 5.2: "Preparser"

Der unverarbeitete Eingabetext des Anwenders wird dem Modul „input_parser“ übergeben. Von diesem ausgehend verarbeitet das Prädikat „create_naproche_input()“ (siehe Abbildung 5.2) den Eingabetext, in dem zuerst mit „clear_messages()“ die aktuelle Fehlerliste gelöscht wird. Mit „atom_chars()“ werden die Atome des Beweistextes in eine Prolog-Liste aufgeteilt. Dabei handelt es sich um ein atomares Prädikat von Prolog (siehe Kapitel 2.3.1, welches wie folgt funktioniert: `atom_chars('inhalt',X)` weist dem Parameter `X` die einzelnen Buchstaben in einer Liste zu (`X=['i','n','h','a','l','t']`).

Diese Liste wird mit dem Prädikat „input_dcg()“ auf Satzebene geparkt: Der Text wird über das Prädikat „sentence()“ in all seine Sätze aufgeteilt. Jedem Satz wird eine eindeutige ID zugeordnet, der SatzID. Zu jedem Satz werden die Anfangs- und Endpositionen im Bezug auf den Gesamttext ermittelt. Weiterhin wird eine Wortliste mit dem Prädikat „word()“ erstellt, wobei zu jedem Wort ebenfalls die Anfangs und Endpositionen ermittelt werden. Als Wörter werden auch die mathematischen Bereiche gesehen, die im LaTeX-Code durch die `$`-Zeichen begrenzt sind und mit dem Prädikat „input_math()“ bestimmt werden. Anhand dieser Kriterien entsteht eine Prolog-Liste mit den Positionen des Satzes, der einzelnen Wort-Elemente sowie deren Inhalt.

Beispiel 5.1

Der Eingabetext „There is no $\$y\$$ such that $\$y \in \emptyset\$$.“ wird nach dem Parser zu „[sentence(1, 0, 44, [word(0, 5), word(6, 8), word(9, 11), math(13, 14), word(16, 20), word(21, 25), math(27,42)], [there, is, no, math([y)], such, that, math([y, 'in', '\emptyset'])])]“ umgewandelt.

Sollte ein Fehler auftreten, wird dieser auf der untersten Ebene protokolliert und anschließend an die nächst höhere Ebene weitergegeben. Die Fehlerliste wird ebenunabhängig mit dem Prädikat „add_error_message_once()“ erweitert. Der gefundene Fehler wird mit dem Index des Fehlers in die Fehlerliste hinzugefügt. Die Position der einzelnen Zeichen (Buchstaben und weitere Symbole) im Eingabetext werden hier und für den folgenden Text als Index bezeichnet. „sentence“ erkennt den Fehler und ergänzt die Fehlerliste um die Information, welcher Satz mit welchem Index nicht korrekt formuliert wurde. Der DCG-Parser erweitert die Fehlerliste ein weiteres Mal um die Information, dass überhaupt ein Fehler aufgetreten ist. Mit diesen Informationen kann der Fehlerfall klassifiziert sowie die Position des Satzes und des Wortes angegeben werden.

Als Beispiel für einen Fehlerfall wird das Hauptprädikat „create_naproche_input()“ direkt von Prolog aus aufgerufen. Der Fehler wird durch einen Backslash im mathematischen LaTeX-Bereich ausgelöst, der von der Vorverarbeitung verdoppelt wurde:

Beispiel 5.2

```
?- create_naproche_input('Then $\\$.',T).
false.
?- get_error_messages(X).
X = [message(error, inputError, 'create_naproche_input, 0, 0', 'Then
$\\$.', 'Could not parse input.'), message(error, inputError,
'sentence, 5, 8', '$\\$', 'Could not parse math mode.'), message(error,
inputError, 'input_math, 6, 7', '\\', 'Missing Latex command.')].
```

Wird das Prädikat mit dem Übergabewert „Then \$ \\\$.“ aufgerufen, wird das Ergebnis der Überprüfung in die Variable „T“ geschrieben. Der Rückgabewert „false“ des Aufrufs bedeutet, dass ein Fehler auftrat. Mit dem Aufruf des Prädikates „get_error_messages(X)“ wird die Fehlerliste der Variablen „X“ zugeordnet. Der Inhalt der Variablen „X“ zeigt deutlich, wie die Fehlermeldung aufgebaut ist:

Es handelt sich um einen Eingabefehler (inputError) beim Aufruf des Prädikats „create_naproche_input()“, der an Position 0,0 auftrat. Der Inhalt „Then \$ \\\$.“ konnte nicht geparkt werden. Die folgende Fehlermeldung wurde von „sentence“ ausgelöst. Zwischen den Positionen 5 und 8 konnte der Mathematikmodus „\$ \\\$“ nicht geparkt werden. Abschließend gibt „input_math“ den eigentlichen Fehler an: Das Zeichen „\\“ von Position 6 bis 7 konnte nicht als Latex-Befehl interpretiert werden.

Javaklassen zum Parser

Um die Informationen in Java entsprechend verarbeiten zu können, wurde ein eigenes Java-Package erstellt. Dieses beinhaltet die Dateien „sentence.java“ , „word.java“ und „error.java“.

Die Klasse „sentence“ hat die Attribute „id“, „start“ und „end“ als Integerwerte, sowie eine Liste „content“ vom Typ „word“:

- *id*: die eindeutige Satz-ID des jeweiligen Satzes
- *start*: der Index des Satzanfanges
- *end*: der Index des Satzendes
- *content*: eine Liste, die alle Wörter des Satzes beinhaltet

Die Attribute der Klasse „word“ sind ebenfalls „start“ und „end“ als Integerwerte, „type“ und „wordContent“ als Strings, sowie „mathContent“ als die aus einzelnen Strings bestehende Liste:

- *start*: der Index des Wortanfanges
- *end*: der Index des Wortendes
- *type*: die eindeutige Satz-ID des jeweiligen Satzes
- *wordContent*: eine Liste, die alle Wörter des Satzes beinhaltet
- *mathContent*: eine Liste, die alle mathematischen Zeichen des Satzes beinhaltet

Durch die Klasse „error“ werden eventuell auftretende Fehler genau bestimmt. Sie enthält folgende Attribute:

- *type*: Bezeichnet den aufgetretenen Fehlertyp beim Parsen.
- *position*: Gibt an, in welchem Modul bzw. auf welchem Parsermodul der Fehler aufgetreten ist: im Prepaser, im Microparser, im Logikmodul, im Macroparser oder im Formelparser.
- *start*: Index des Fehleranfangs
- *end*: Index des Fehlerendes
- *content*: Die Fehlerumgebung *content* gibt den Inhalt des Beweistextes an, ab dem dieser nicht mehr vollständig geparkt werden konnte. Tritt beim Parsen ein Fehler auf, wird der Fehler und der Rest des noch ungeparkten Textes in den Wert „content“ geschrieben.
- *description*: Eine genauere Beschreibung des aufgetretenen Fehlers

In allen Klassen sind neben dem Konstruktoren auch Konvertierungsmethoden vorhanden, um mit den Rückgabewerten des Preparsers die Werte der Attribute richtig zu setzen.

5.1.3 Macroparser, Microparser und Formula

Diese verschiedenen Prologmodule werden zur Zeit im Rahmen einer Doktorarbeit und mit studentischen Hilfskräften weiterentwickelt. Die Spezifikation der genauen Aufgaben der Prologmodule ist noch nicht vollständig abgeschlossen, weshalb diese Module hier nur abstrakt beschrieben werden.

In der folgenden Abbildung wird die Modul-Struktur-Veränderung von der Version 0.47 zu 0.5 dargestellt:

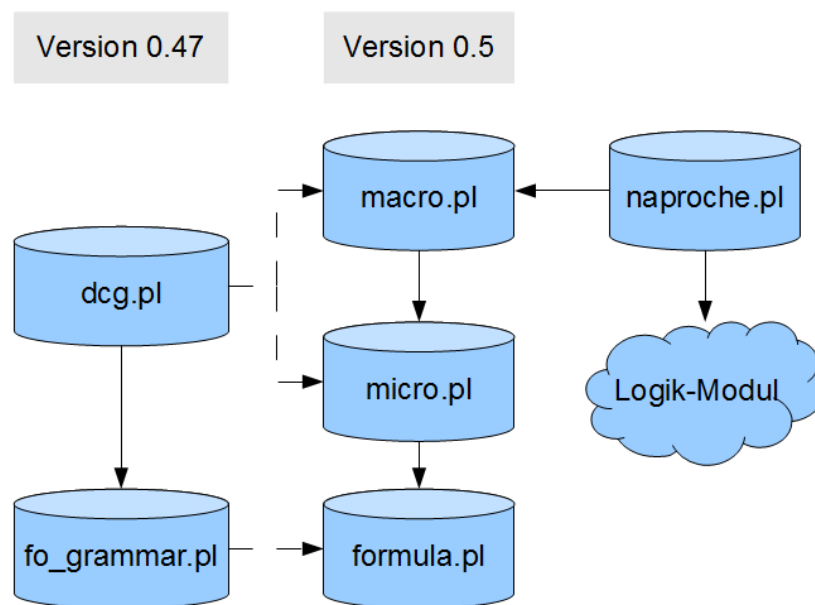


Abbildung 5.3: Änderungen der Versionen

Die Bearbeitungsmodule der Datei „dcg.pl“ aus der Version 0.47, die die CNL-Grammatik beinhaltet, wurde in der neuen Version 0.5 aufgeteilt: In der Datei „macro.pl“ ist die Macroparsergrammatik beschrieben. Dieses definiert die Regeln, wie sich die Sätze eines natürlichsprachlichen Textes zusammensetzen dürfen. Das Modul Microparser befindet sich in der Datei „micro.pl“, in dem definiert ist, wie ein Satz aus Wörtern und Formeln zusammengesetzt werden darf. Der Microparser ist in der DCG-Syntax (siehe Kapitel 2.2.3) geschrieben.

Die Regeln für die Zusammensetzung der Formeln und mathematischen Zeichen von Naproche 0.5 sind in der Datei „formula.pl“ aufgeführt, die der Datei „fo_grammar.pl“ aus Naproche 0.47 entspricht. Dieses Modul wurde von Grund auf neu implementiert, um die Formelgrammatik flexibler zu gestalten und Formeln der Prädikatenlogik höherer Ordnung parsen zu können. Aufgerufen wird der Macroparser aus dem Naproche-Modul in der Datei „naproche.pl“. Dort ist ein inkrementeller Parser für die Überprüfung der Textstruktur implementiert, über den auch die eigentliche Beweistextüberprüfung gestartet wird.

Die Makrogrammatik wird nicht von dem in Prolog eingebautem DCG-Parser geparkt, da dieser kein inkrementelles parsen ermöglicht.

Weiterhin hat sich der Ablauf der logischen Überprüfung einzelner Beweisschritte geändert: Zuerst wird der jeweilige Satz des Beweistextes linguistisch überprüft. Anschließend wird das unveränderte Prädikat „check_sentence()“ aufgerufen, wodurch das Prädikat „create_obligations()“ den ATP-Input erzeugt und diesen an den gewählten ATP weiterleitet. Damit erfolgt die logische Überprüfung nicht erst nach der kompletten linguistischen Verarbeitung, sondern direkt nach der linguistischen Verarbeitung jedes einzelnen Satzes.

5.1.4 Datenstruktur, Dateiaufbau und die ProofState-Datei

Anhand der ersten 13 natürlichsprachlichen Sätze des Burali-Forti-Beweises (siehe Beispiel 6.1) wird in diesem Kapitel die Struktur der Dateien erklärt, die während der Beweisüberprüfung erstellt werden.

Die Struktur der Dateien und Ordner ist in der Abbildung 5.4 dargestellt:

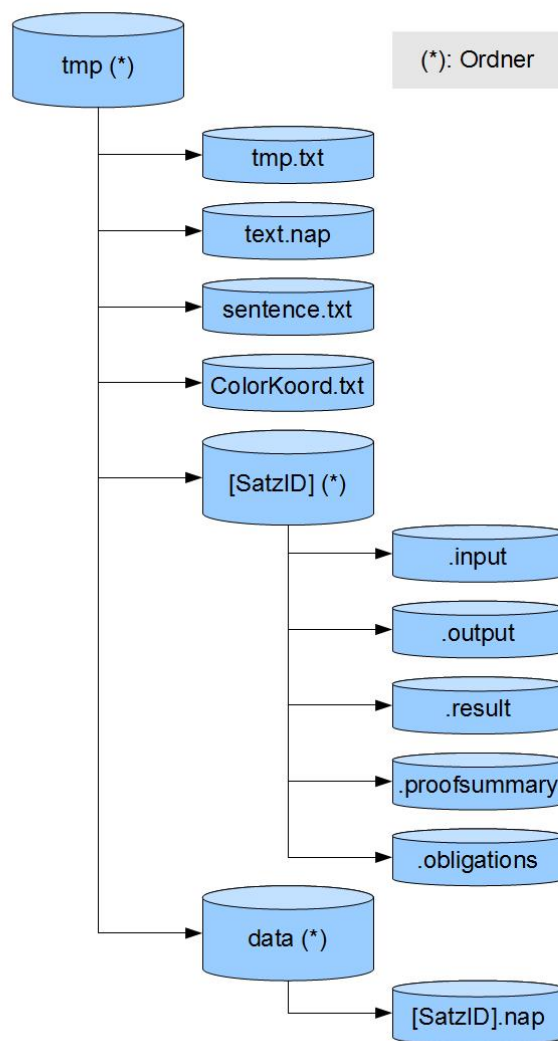


Abbildung 5.4: Aufbau der Datenstruktur

Zunächst wird, falls noch nicht vorhanden, ein lokaler Unterordner „tmp“ angelegt, in dem die im Laufe des Prozesses erzeugten Daten gespeichert werden. Der Beweistext aus dem Textfeld des GUI wird unverändert in die Datei „tmp.txt“ geschrieben:

Beispiel 5.3

Assume that there is a relation \in and
an object \emptyset satisfying the following axioms:

Axiom.

There is no y such that $y \in \emptyset$.

Axiom.

For every x it is not the case that $x \in x$.

Define x to be transitive if and only if for all u, v ,
if $u \in v$ and $v \in x$ then $u \in x$.

Define x to be an ordinal if and only if x is transitive
and for all y , if $y \in x$ then y is transitive.

Theorem.

\emptyset is an ordinal.

Das Prologmodul „preparser“ liefert die entstandene Satzliste des Textes von Beispiel 5.3, die in die Datei „text.nap“ hinterlegt wird:

```
sentence(1,[assume,that,there,is,a,relation,math(['\in']),and,an,object,
math(['\emptyset']),satisfying,the,following,axioms]).
sentence(2,[##]).
sentence(3,[axiom]).
sentence(4,[there,is,no,math([y]),such,that,
math([y,'\in','\emptyset'])]).
sentence(5,[##]).
sentence(6,[axiom]).
```

```

sentence(7, [for, every, math([x]), it, is, not, the, case, that,
math([x, '\in', x]))).
sentence(8, [##]).
sentence(9, [define, math([x]), to, be, transitive, if, and, only, if,
for, all, math([u]), ', ', math([v]), ', ', if, math([u, '\in', v]), and,
math([v, '\in', x]), then, math([u, '\in', x]))).
sentence(10, [define, math([x]), to, be, an, ordinal, if, and, only, if,
math([x]), is, transitive, and, for, all, math([y]), ', ', if,
math([y, '\in', x]), then, math([y]), is, transitive])).
sentence(11, [##]).
sentence(12, [theorem]).
sentence(13, [math(['\emptyset']), is, an, ordinal]).

```

Aus dem Beispiel 5.3 wurden verschiedene „sentence“-Objekte erzeugt. Diese Objekte sind in einer Syntax geschrieben, die ein effektiveres Einlesen der Datei durch Prolog erlaubt. Sätze, deren Inhalt aus den Zeichen „##“ bestehen, sind lediglich Leerzeilen und haben keine inhaltliche Bedeutung. Der Aufbau dieser Liste wurde bereits im Kapitel 3.3 erklärt.

Die Datei „sentence.txt“ ist ähnlich wie eine CSV-Datei aufgebaut(siehe Kapitel 2.3.6):

```

1#0#96#0#6&word&assume&[]#7&11&word&that&[]#12&17&word&there&[]
#18&20&word&is&[]#21&22&word&a&[]#23&31&word&relation&[]
#33&36&math&&['\in']#38&41&word&and&[]#42&44&word&an&[]
#45&51&word&object&[]#53&62&math&&['\emptyset']
#64&74&word&satisfying&[]#75&78&word&the&[]
#79&88&word&following&[]#89&95&word&axioms&[]#
2#96#98#96&98&word&##&[]#
3#98#104#98&103&word&axiom&[]#
4#105#149#105&110&word&there&[]#111&113&word&is&[]
#114&116&word&no&[]#118&119&math&&[y]#121&125&word&such&[]
#126&130&word&that&[]#132&147&math&&[y, '\in', '\emptyset']#
5#149#151#149&151&word&##&[]#
6#151#157#151&156&word&axiom&[]#

```

```
7#158#206#158&161&word&for&[]#162&167&word&every&[]
#169&170&math&&[x]#172&174&word&it&[]#175&177&word&is&[]
#178&181&word&not&[]#182&185&word&the&[]#186&190&word&case&[]
#191&195&word&that&[]#197&204&math&&[x, '\\in', x]#
8#206#208#206&208&word&##&[]#
9#208#310#208&214&word&define&[]#216&217&math&&[x]
#219&221&word&to&[]#222&224&word&be&[]#225&235&word&transitive&[]
#236&238&word&if&[]#239&242&word&and&[]#243&247&word&only&[]
#248&250&word&if&[]#251&254&word&for&[]#255&258&word&all&[]
#260&261&math&&[u]#262&263&word&', '&[]#265&266&math&&[v]
#267&268&word&', '&[]#269&271&word&if&[]#273&280&math&&[u, '\\in', v]
#282&285&word&and&[]#287&294&math&&[v, '\\in', x]
#296&300&word&then&[]#302&308&math&&[u, '\\in', x]#
10#311#425#311&317&word&define&[]#319&320&math&&[x]
#322&324&word&to&[]#325&327&word&be&[]#328&330&word&an&[]
#331&338&word&ordinal&[]#339&341&word&if&[]#342&345&word&and&[]
#346&350&word&only&[]#351&353&word&if&[]#355&356&math&&[x]
#358&360&word&is&[]#361&371&word&transitive&[]#372&375&word&and&[]
#376&379&word&for&[]#380&383&word&all&[]#385&386&math&&[y]
#387&388&word&', '&[]#389&391&word&if&[]#393&400&math&&[y, '\\in', x]
#402&406&word&then&[]#408&409&math&&[y]#411&413&word&is&[]
#414&424&word&transitive&[]#
11#425#427#425&427&word&##&[]#
12#427#435#427&434&word&theorem&[]#
13#436#462#437&446&math&&['\\emptyset']#448&450&word&is&[]
#451&453&word&an&[]#454&461&word&ordinal&[]#
```

Das erste Element eines Eintrags bezieht sich jeweils auf die Satz-ID. Nach dieser erscheint das Trennsymbol: „#“. Die nächste Ziffer enthält die Anfangsposition des Satzes im Bezug auf den Gesamttext. Die nachfolgende Ziffer gibt die Endposition des Satzes an. Hinter einem erneuten Trennzeichen werden die einzelnen Wörter und mathematischen Zeichen und deren Position im Eingabetext angegeben. Getrennt werden die einzelnen Wörter bzw. mathematischen Ausdrücke durch das Trennzeichen „#“. Zu jedem Wort bzw. mathematischen Ausdruck wird eine weitere CSV-ähnliche Struktur

aufgebaut, in der das Trennzeichen ein „&“ ist: „Wortanfangsposition & Wortendposition & word oder math & natürlichsprachliches Wort & [durch Kommata getrennte Liste der Inhalte des Latex-Mathemodus]“. Wird ein Wort im Text aufgelistet, ist der dritte Wert „word“ und der vierte Wert wird mit dem entsprechenden natürlichsprachlichen Wort gefüllt, während die Liste für die mathematischen Zeichen und Symbole des Latex-Mathemodus leer bleibt. Alternativ wird der dritte Wert auf „math“ gesetzt, sollte es sich um einen mathematischen Ausdruck im Beweistext handeln. Der Wert für das natürlichsprachliche Wort bleibt leer, die Liste mit den mathematischen Zeichen wird mit dem mathematischen Elementen aufgefüllt. Die einzelnen mathematischen Zeichen und Symbole werden in dieser Liste durch „“ (Kommata) getrennt.

Als Rückmeldung an den Anwender werden die einzelnen Sätze des Eingabetextes eingefärbt, wobei die Farbwerte die folgende Bedeutung haben:

- 1: „Rot“ - Es konnte kein Beweis gefunden werden.
- 2: „Orange“ - Es konnte ein Beweis gefunden werden, der darauf beruht, dass die Prämissen widersprüchlich sind.
- 3: „Grün“ - Es konnte ein Beweis gefunden werden, der nicht darauf beruht, dass die Prämissen widersprüchlich sind.
- 4: „Grau“ - der Satz enthält keine zu überprüfende Aussage

Der Java-Thread „Parser()“ erzeugt während der Überprüfung die Datei „ColorKo-ord.txt“. In der Datei steht in einer CSV-Struktur neben der Positionen für den Anfang und das Ende im Beweistext der Farbwert des jeweiligen Satzes:

```
0!96!4
96!98!4
98!104!4
105!149!4
149!151!4
151!157!4
158!206!4
```


206!208!4
208!310!4
311!425!4
425!427!4
427!435!4
436!462!3

Der einzige zu überprüfende Satz in dem Beispiel 5.3 ist der (letzte) Satz 13, der grün gefärbt wird.

Die Datei „prs.html“ wird durch ein weiteres Prologmodul („make_super_prs()“) erzeugt. Bei der Beweisüberprüfung wird die Datei nicht automatisch erstellt, es muss der entsprechende Button auf dem GUI betätigt werden. In dieser Datei befindet sich die Darstellung der PRS im HTML-Format. Mit Hilfe eines Internet-Browsers kann diese Datei angezeigt werden.

Neben den bereits beschriebenen Dateien werden bei der Beweisüberprüfung Unterordner für jeden Satz erstellt. Die Bezeichnungen dieser Ordner sind die jeweiligen Satz-IDs. In dem Ordner befinden sich folgende Dateien:

- <name>.input
- <name>.output
- <name>.result
- <name>.proofsummary
- obligations.nap

Die „.input“-Datei beinhaltet den Eingabetext, der an den ATP gesendet wird. Der Ausgabertext des ATPs wird in die „.output“-Datei geschrieben. Die Strukturen der beiden Dateien wurden bereits in dem Kapitel 2.1.4 beschrieben.

Die „result“-Datei des Satzes mit der ID 13 hat folgenden Inhalt, wobei <Pfad> hier für den Pfad des Ordners steht, in dem sich die GUI befindet:

```
<Pfad>/tmp/13/13-13-0-0.input;true;false;  
E;9;UsedAxioms;ProofTime;TotalTime;1;E
```

In der „result“-Datei befindet sich folgende Datenstruktur:

```
<Pfad>/<SatzID>/<name>.input;[Beweisüberprüfungsergebnis];[Inkon-  
sistenzwarnung];[ATP];[maximale Länge einer Prämisse]; [benutzte Axio-  
me];[Beweisüberprüfungszeit];[Gesamtüberprüfungszeit]; [Anzahl der Ver-  
suche des ATPs];[mögliche ATPs]
```

Der Pfad mit der Satz-ID gibt an, in welchem Verzeichnis sich die „input“-Datei befindet. Der <name> setzt sich aus folgenden Bestandteilen zusammen:

```
[SatzID] - [PRSID] - [Nummer der Bedingung in PRS] - [Nummer der  
Obligationen in den Bedingungen]
```

Das Ergebnis der Beweisüberprüfung ist ein boolescher Wert: „true“ bedeutet, dass der Beweis gefunden wurde; „false“ das Gegenteil. Ähnlich verhält es sich bei der Inkonsistenzwarnung: „true“ bedeutet, dass eine Warnung aufgetreten ist, also die Prämissen widersprüchlich sind, bei „false“ ist nichts passiert. Die Wertekombination „false;true“ ist nicht denkbar, da die Erkennung eines Widerspruchs in den Prämissen immer als Beweis gewertet wird („ex falso sequitur quodlibet“). Alle anderen Kombinationsmöglichkeiten können auftreten. In dem Beispiel 5.3 konnte ein Beweis ohne Inkonsistenzwarnung gefunden werden.

Die anderen Werte ([ATP]; [maximale Länge einer Prämisse]; [benutzte Axiome]; [Beweisüberprüfungszeit]; [Gesamtüberprüfungszeit]; [Anzahl der Versuche des ATPs]; [mögliche ATPs]) entsprechen den Beweisüberprüfungseinstellungen des Benutzers. Sie geben Aufschluss darüber, unter welchen Bedingungen die Beweisüberprüfung durchgeführt wurde. Diese Werte werden für statistische Auswertungen erfasst und können in einem späteren Schritt interpretiert und betrachtet werden.

In der Datei mit der Endung „proofsummary“ befinden sich zum Satz 13 folgende Daten:

```
% SZS status Success for <pfad>/tmp/13/13-13-0-0.output
% SZS output start Summary
% The conjecture was proved by contradiction
proved('replace(pred(13,0))', ['pred(24,1)', 'pred(24,0)']) .
% SZS output end Summary
```

Aus der „proofsummary“-Datei wird das Beweisverfahren und das Ergebnis der Überprüfung des jeweiligen Satzes ersichtlich. Hier wurde der Beweis über einen Widerspruch erbracht. Die nächste Zeile gibt die konkreten Beweisschritte an, mit denen der Beweis erbracht wurde.

Die Datei „obligations.nap“ enthält eine Liste von allen Obligationen, die zu dem jeweiligen Satz überprüft werden müssen:

```
<Pfad>/tmp/13/13-13-0-0.input
```

In Satz 13 gibt es genau eine Obligation, die in der Datei „13-13-0-0.input“ steht. Zu jeder Obligation in dem jeweiligen Satz wird genau eine „input“, „output“, „result“ und „proofsummary“-Datei erzeugt.

Weiterhin wird die Datei „theorem-obligations.nap“ erzeugt, wenn es sich bei dem zu überprüfenden Satz um ein Theorem handelt. Der Inhalt der Datei besteht aus einer GULP-Liste (siehe Kapitel 2.4.2), die die einzelnen Obligationen dieses Theorems beinhaltet.

Neben den Ordnern mit den SatzIDs wird noch ein Ordner „data“ durch die Beweisüberprüfung erstellt. In diesem Ordner wird bei der Beweisüberprüfung zu jedem einzelnen Satz, eine „nap“-Datei unabhängig davon erzeugt, ob der jeweilige Satz einen Beweisschritt beinhaltet oder nicht. In jeder „nap“-Datei befindet sich eine GULP-Liste (siehe Kapitel 2.4.2), die alle Informationen zum betreffenden Satz beinhaltet. Diese Dateien werden gebraucht, um das inkrementelle Parsen und Überprüfen des Textes mit

den Prädikaten in den Dateien „naproche.pl“ und „macro.pl“ zu ermöglichen. Eine erste Überlegung war es, die GULP-Daten in eine einzige Datei zu speichern und diese beim Öffnen eines Beweises zu laden. Diese Idee wurde wieder verworfen, da mit vielen kleinen GULP-Listen in entsprechend vielen kleinen Dateien bei einer Änderung oder Ergänzung des Beweistextes die Daten schneller eingelesen werden können. Nachdem ein Beweistext überprüft und die geschilderten Ordner und Dateien erstellt wurden, kann diese Ordnerstruktur gespeichert werden. Dazu wird über das GUI eine Java-Methode „zipProof()“ aufgerufen, welche die Ordnerstruktur des aktuellen Beweises in das ZIP-Format komprimiert.

Die entstandene ZIP-Datei beinhaltet somit die für die Master-Thesis zu entwickelnden ProofState-Daten. Sie enthält den kompletten Beweiskorpus, entspricht den Kriterien der persistenten Datenspeicherung und ist somit die zu erzeugende „ProofState“-Datei. Dies hat den Vorteil, dass die Datenmenge bei gleichzeitiger Beibehaltung der Ordnerstruktur komprimiert wird. Der Beweis kann von des GUI aus jederzeit geladen, beliebig modifiziert und erneut geprüft werden. Auch außerhalb des GUI kann mit jedem ZIP-tauglichen Entpackungsprogramm ein einzelner Beweis geöffnet und die erstellten Dateien in einem beliebigen Textverarbeitungsprogramm analysiert werden.

5.1.5 Verbindung zwischen Java und Prolog

Um von Java aus Prologprädikate aufrufen zu können, wurde „JPL“ importiert. JPL ist ein Java/Prolog-Interface, um Verbindungen zwischen der Java Virtual Machine (JVM) und SWI-Prolog herstellen zu können. Dabei wird das Java Native Interface (JNI) benutzt, welches mit dem Prolog Foreign Language Interface (FLI) verbunden wird. Es bietet zwei Schnittstellen an: Ein low-level Interface zum FLI und ein high-level Interface zum JNI. Das Interface zum JNI wurde im Rahmen des Prototypes benutzt. JPL stellt unter anderem den Datentyp „Query“ zur Verfügung, mit dem man verschiedene Prolog-Befehle ausführen kann. Die Rückgabewerte werden in einer „Hashtable“ (bildet Schlüssel auf Werte ab, sodass über einen Schlüsselbegriff auf einen konkreten Wert zugegriffen werden kann) an Java zurückgegeben und können anschließend für Berechnungen im Java-Quellcode benutzt werden. Eine genauere Beschreibung zur JPL ist auf der Homepage [http://www.swi-prolog.org/packages/jpl/java_api/index.html]

zu finden.

Im Prototypen gibt es drei verschiedene Stellen, an denen Prolog-Aufrufe erfolgen. Beim Initialisieren des GUI wird ein Query aufgerufen, welches alle relevanten Prologprädikate für Naproche lädt. Der Preparser wird als Query aufgerufen, wenn der Beweistext vorverarbeitet werden soll (siehe Kapitel 5.2.3). Das Prologprädikat „naproche()“ startet den Macroparser und damit die linguistische und logische Überprüfung des eingegebenen Beweistextes.

5.2 Die Javamodule

Im Prototyp wurden diverse Java-Klassen und Module entwickelt. Alle in dem Klassendiagramm (Abbildung 5.5) gekennzeichneten Methoden (die wesentlichen Methoden sind dick umrahmt, die weiteren Methoden dünn umkreist) werden in den folgenden Unterkapiteln genauer beschrieben.

Die Hauptklasse ist die GUI-Klasse, in der diverse Java-Swing-Elemente zur Gestaltung der GUI-Oberfläche definiert sind. Zusätzlich gibt es noch einige Variablen in verschiedenen Datentypen, die für die Berechnungen von Zwischenwerten benutzt werden.

Mit der GUI-Klasse verbunden sind die Klassen zum Preparsen: „Sentence“, „Word“ und „Error“ (siehe 5.1.2). Die weiteren Klassen („FileOperations“, „AttributedTextPane“, „PrintTool“, „TPTPAxSel“ und „NaturalOrderComparator“) wurden hinzugefügt, um die Java-Module zu ordnen. Diese Klassen sind kein Bestandteil des Prototyps und werden in dieser Master-Thesis nicht detailliert erläutert. Die Klasse „FileOperations“ bietet alle Methoden, Dateien zu erstellen, zu schreiben, als String einzulesen und zu löschen. Weiterhin können vorhandene Ordner im Zip-Format gepackt und entpackt werden. Die Klasse „AttributedTextPane“ ermöglicht es, ein Textfeld zu erzeugen, dass neben den üblichen Methoden eines Textfeldes den Textinhalt verschiedenfarbig gestalten kann. Um den Textfeldinhalt ausdrucken zu können, wurde die Klasse „PrintTool“ implementiert. Für eine Auswahlliste, die langfristig bei der Anbindung der GUI-Klasse an die Klasse „TPTPAxSel“ (siehe Kapitel 5.3) benötigt wird, wurde bereits eine eigene Klasse vorbereitet. Der „NaturalOrderComparator“ wird benutzt, um eine beliebige String-Liste zu sortieren. Die Liste wird entweder in der alphabetischen Reihenfolge oder in ihrem Gegenteil sortiert.

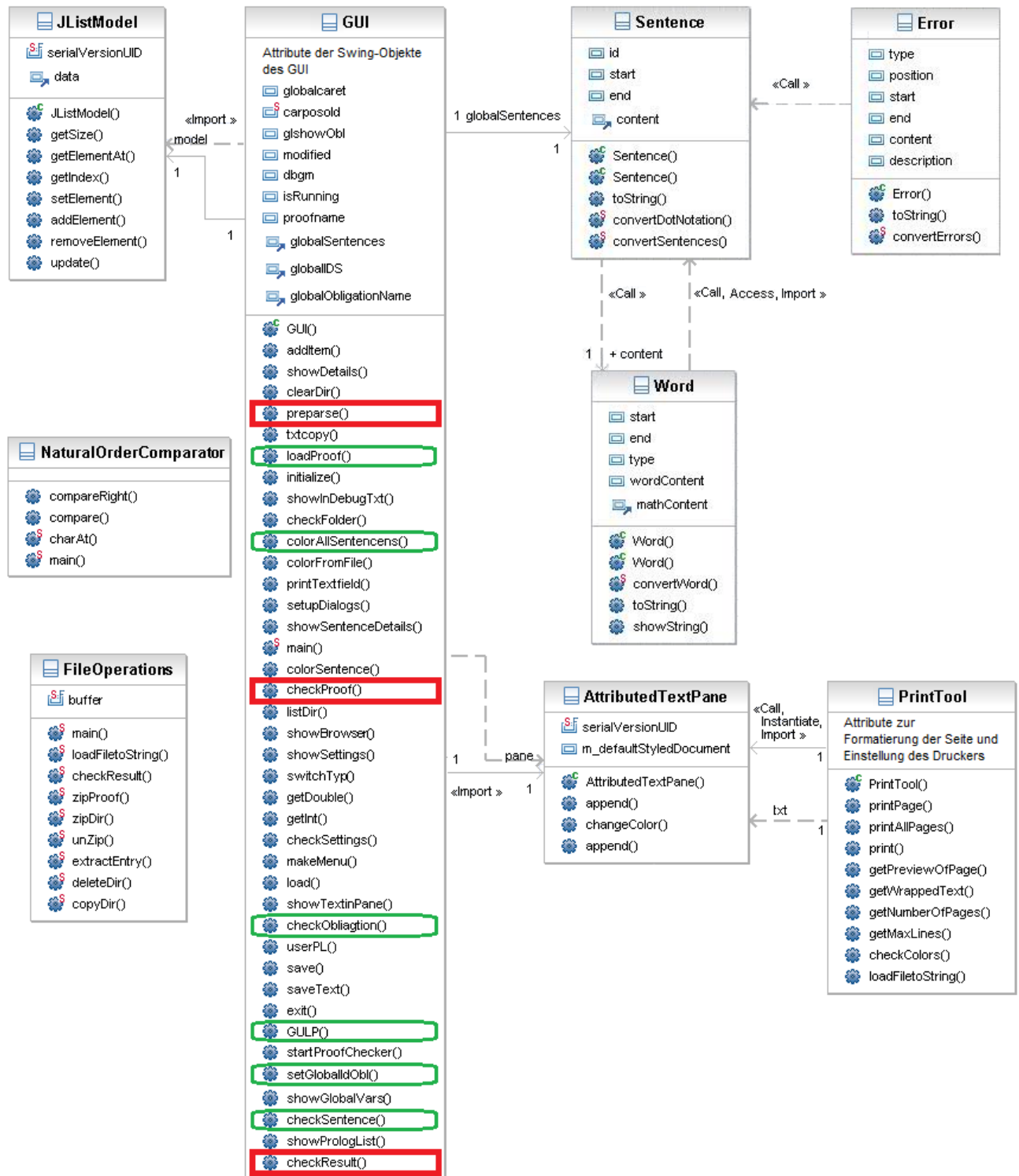


Abbildung 5.5: Klassendiagramm

5.2.1 Das Grafical User Interface (GUI)

Das hier beschriebene Grafical User Interface wurde als Benutzeroberfläche des Prototypen für die Naproche-Version 0.5 entwickelt. Die Abbildung 5.6 zeigt den generellen Aufbau des GUI:

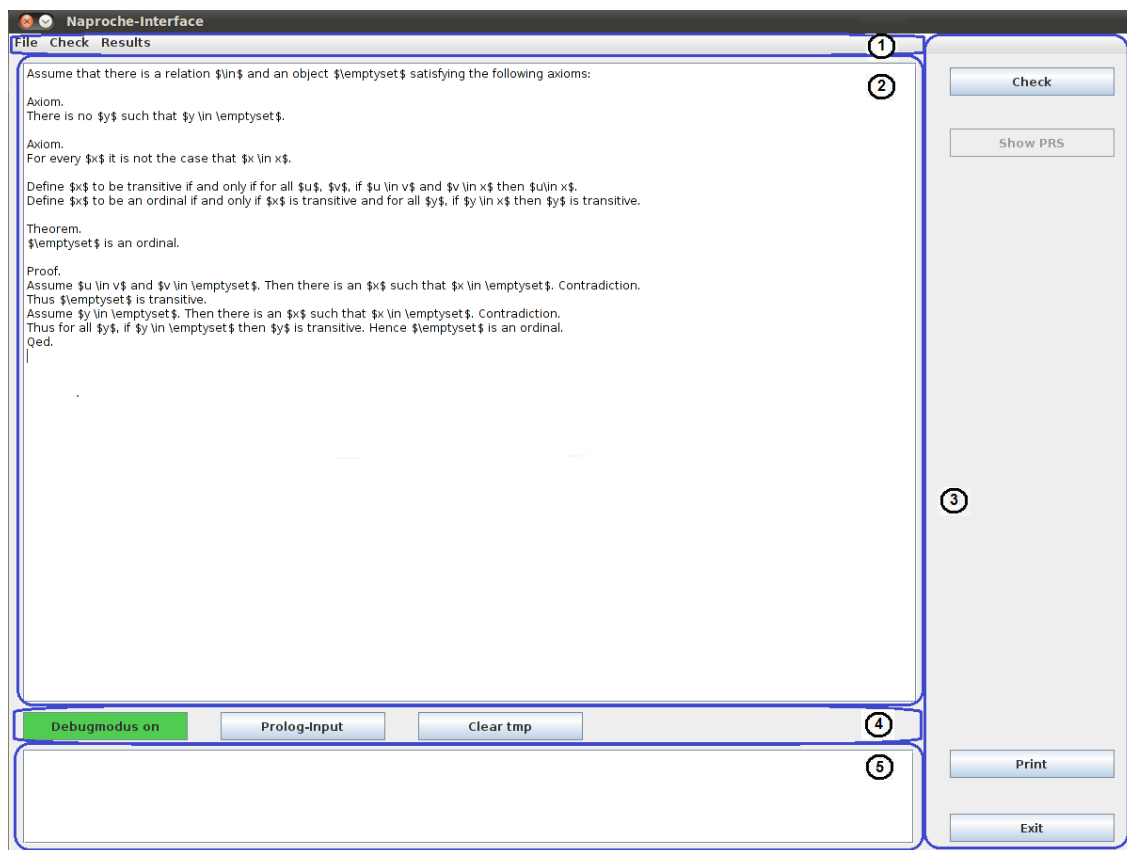


Abbildung 5.6: Screenshot des GUI

Die hier dargestellten Bereiche des GUI werden im Folgenden beschrieben:

- ①: Menüleiste
- ②: Eingabetextfeld
- ③: Funktionsbuttons
- ④: Debugmodus-Buttons
- ⑤: Debugmodus-Textfeld

Eine in der Abbildung 5.6 nicht zu sehende Funktion besteht in zwei verschiedenen Popup-Menüs. Ein Popup-Menü erscheint, wenn man mit der rechten Maustaste auf eines der beiden Textfelder klickt. Die GUI bietet einem Anwender die anschließend dargestellten funktionalen Möglichkeiten:

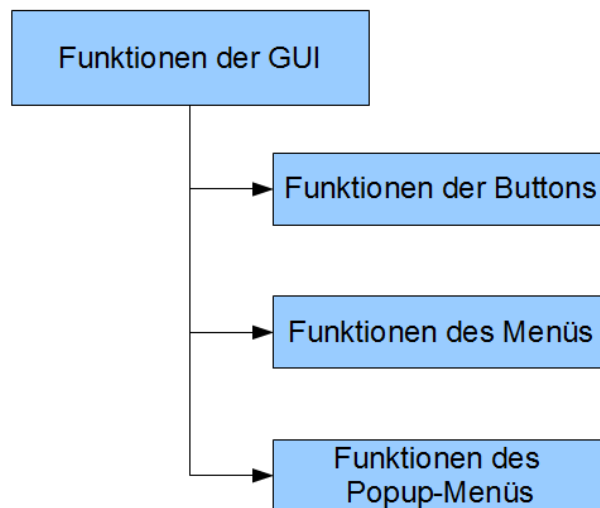


Abbildung 5.7: Funktionen des GUI

Die Funktionen der GUI können über drei verschiedene Ereignisarten ausgelöst werden: Es können die Buttons auf der Oberfläche (3,4) betätigt, ein Menü-Unterpunkt (1) aufgerufen oder ein Popup-Menüpunkt (2,5) eines Textfeldes ausgewählt werden.

Funktionen der Buttons

In der folgenden Abbildung werden die Buttons der GUI aufgelistet:

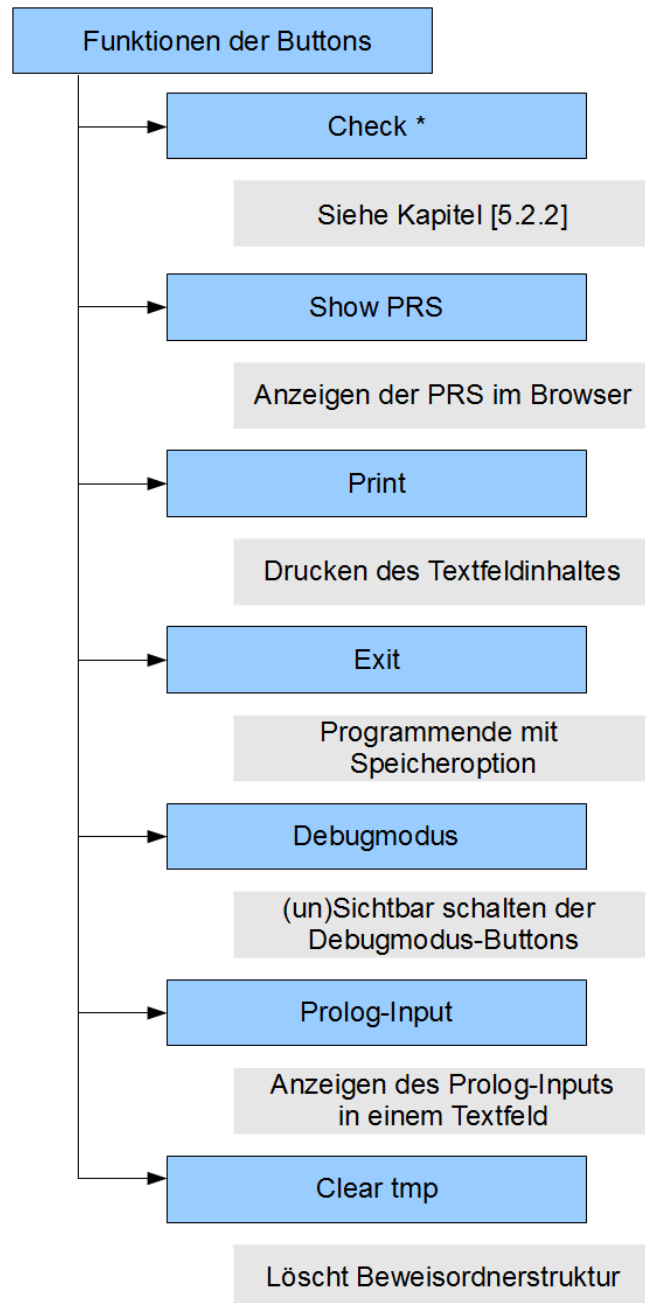


Abbildung 5.8: Funktionen der GUI-Buttons

Die Überprüfung des Beweises wird mit der Betätigung des „Check“-Buttons ausgelöst, die im folgenden Kapitel 5.2.2 beschrieben wird.

Über den Button „Show PRS“ wird mit dem Prolog-Prädikat „make_super_prs“ eine HTML-Datei erzeugt, die die aktuelle PRS beinhaltet. Nach der Erstellung wird ein von der GUI unabhängiges Browser-Fenster geöffnet, in der die PRS dargestellt wird. Der Button „Print“ öffnet zunächst das systemeigene Druck-Fenster, in dem man die Eigenschaften des Ausdrucks festlegen kann. Bei Bestätigung der Druckeinstellungen wird der aktuelle Inhalt des Eingabetextfeldes ausgedruckt.

Beendet wird das GUI mit dem Button „Exit“. Bevor das GUI endgültig geschlossen wird, findet eine Sicherheitsabfrage zur Speicherung des aktuellen Beweises statt. Diese kann bestätigt werden (der Beweis wird unter einem, vom Anwender anzugebenden, Dateinamen gespeichert, bevor die GUI beendet wird), abgelehnt (sofortige Beendigung des GUI) oder abgebrochen (GUI schließt sich nicht) werden.

Der Debugmodus bietet dem Anwender weitere Funktionsmöglichkeiten: Die Buttons „Prolog-Input“ und „Clear tmp“ werden sichtbar und damit verfügbar gemacht. Um den Debugmodus auszuschalten, muss der Debugmodus-Button erneut gedrückt werden.

Nach der Betätigung des Buttons „Prolog-Input“ wird die aktuelle Satzstruktur als Prolog-Liste in dem Ausgabefeld des Debugmodus angezeigt. So kann schnell und einfach überprüft werden, ob bei der Erstellung dieser Liste Fehler aufgetreten sind.

Über den Button „Clear tmp“ wird die komplette Beweisüberprüfungsstruktur des „tmp“-Ordners gelöscht. Die einzige Datei, die nicht gelöscht wird, ist die Datei „tmp.txt“, die den Beweistext enthält.

Funktionen der Menü-Unterpunkte

In dem GUI gibt es folgende Menü-Unterpunkte:

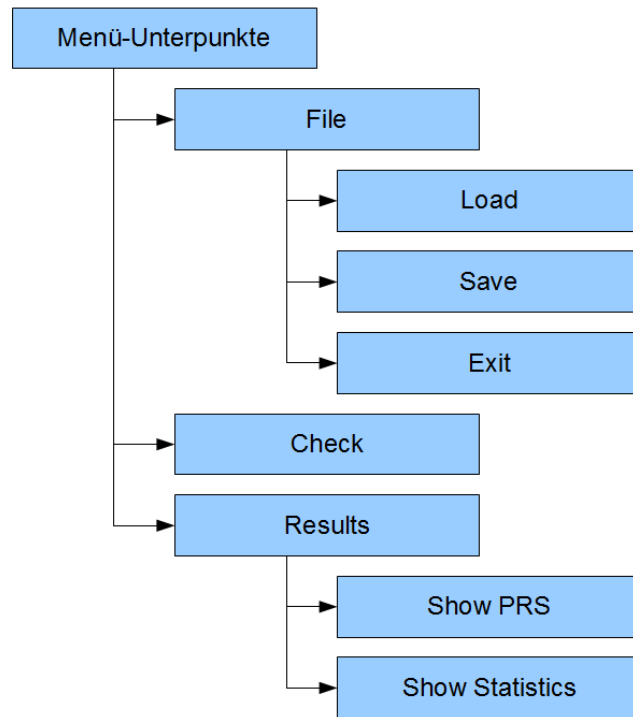


Abbildung 5.9: Funktionen des GUI-Menüs

Mit dem Menü-Unterpunkt „File“ können die Optionen „Load“, „Save“ und „Exit“ ausgewählt werden. Bei „Load“ öffnet sich ein Dateiauswahlfenster, in dem der Anwender den gewünschten Beweis auswählt. Das Ladeverfahren wird im Kapitel 5.1.4 näher beschrieben. Das Speichern des Beweises erfolgt über die Auswahl des Unterpunktes „Save“. Die Datenstruktur wird wie im Kapitel 5.1.4 beschrieben in eine Zip-Datei gepackt. Alternativ zu dem Exit-Button kann das GUI auch über den Menüpunkt „Exit“ beendet werden.

Über den Menü-Unterpunkt „Check“ kann die Überprüfung des Beweistextes genau wie mit dem gleichnamigen Button „Check“ (siehe Kapitel 5.2.2) gestartet werden. Die verschiedenen Resultate zum jeweiligen Beweis sind über den Unterpunkt „Results“ zugänglich.

Mit „Show PRS“ wird die PRS (siehe Kapitel 2.2.1) in einem Browserfenster angezeigt. Vorbereitet für die Anbindung von TPTP-AxSel (siehe Kapitel 5.3) an das GUI ist die Anzeige von Statistiken über den Unterpunkt „Show Statistics“.

Funktionen des Popup-Menüs

Bei einem Klick mit der rechten Maustaste auf eines der beiden Textfelder erscheint das jeweilige Popup-Menü:

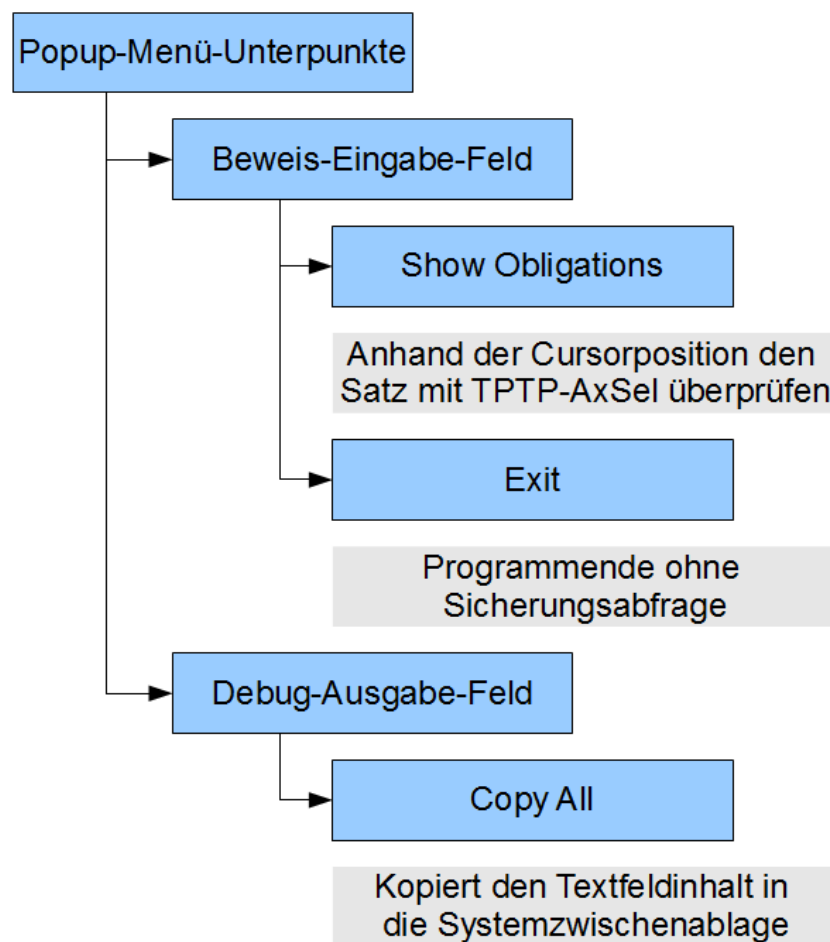


Abbildung 5.10: Funktionen des Popup-Menüs

Die verschiedenen Popup-Menüs sind in der Abbildung 5.10 dargestellt. Klickt man im Eingabetextfeld mit der rechten Maustaste auf einen Satz im Beweistext, wird anhand der Cursorposition die SatzID ermittelt. Wenn dieser Satz Obligationen enthält, können diese nun angezeigt werden. Langfristig sollen die einzelnen Obligationen mit Hilfe von „TPTP-AxSel“ mit anderen ATP-Parametern erneut überprüft werden können (siehe Kapitel 5.3).

Weiterhin kann das GUI mit einem Aufruf des Popup-Menü-Unterpunkts „Exit“ ohne Sicherheitsabfrage beendet werden. Dieser Unterpunkt wurde eingefügt, um bei Tests der Prolog-Module das GUI schnell zu beenden.

Das Ausgabefeld des Debug-Modus bietet die Möglichkeit, den kompletten Textfeldinhalt in die System-Zwischenablage zu Kopieren. So kann der Inhalt beispielsweise in Prolog oder einen Texteditor eingefügt werden, so dass Fehler besser reproduziert werden können.

Initialisierung des GUI

Über den Konstruktor des GUI wird die Methode „initialize()“ aufgerufen, deren Arbeitsweise in der folgenden Abbildung 5.11 verdeutlicht wird:

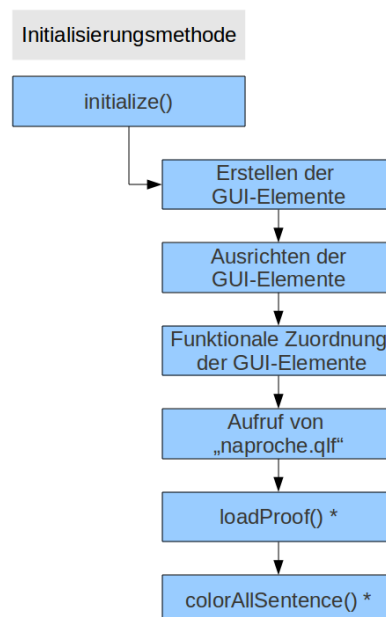


Abbildung 5.11: Initialisierung des GUI

Die Methoden, die in der Abbildung 5.11 mit einem (*) gekennzeichnet sind, werden in dem Unterkapitel 5.2.3 genauer beschrieben.

Die verschiedenen Elemente wie die Buttons, die Textfelder, das Menü und die Popup-Menüs werden zunächst erstellt und anschließend auf der Oberfläche ausgerichtet. Die Funktionen der jeweiligen GUI-Elemente, so wie sie im Kapitel 5.2.1 beschrieben wurden, werden mit den entsprechenden Methoden verbunden.

Nachdem die Oberfläche gestaltet wurde, wird über ein Prolog-Query „consult“ die Datei „naproche.qlf“ geladen, die die Prolog-Pradikate des naproche-systems zur Verfügung stellt. In diesem werden alle Prolog-Module geladen, die für die Beweisüberprüfung benötigt werden.

Mit der Methode „loadproof()“ werden die Dateien eingelesen und ausgewertet, die sich in dem Unterordner „tmp“ befinden. Diese Beweisinformationen sind Grundlage für die Einfärbung des Eingabetextes durch den abschließenden Aufruf der Methode „colorAllSentence()“ (siehe Kapitel 5.2.3).

5.2.2 Wesentliche Attribute und Methoden zur Beweisüberprüfung

In diesem Unterkapitel werden die Attribute und Methoden beschrieben, die direkt mit der Überprüfung des Beweises in Verbindung stehen. Alle Methoden zur Beweisüberprüfung haben keinen Rückgabewert (*void*), da evtl. auftretende Fehler über die Prolog-Rückmeldungen verarbeitet werden.

Attribute:

Attribute des GUI, die bei der Überprüfung des Beweistextes benutzt werden:

- globalSentence: Eine Liste aus Sentence-Objekten, die aus dem Beweistext erzeugt werden konnte.
- globalIDs: Liste mit allen IDs der Beweissätze
- globalObligations: Liste mit allen Obligationsnamen, die aus dem kompletten Beweistext erstellt werden

Diese drei Attribute beinhalten die Daten, die durch die Prolog-Module erzeugt werden. Anhand der Informationen, die in den Variablen gespeichert werden, findet die Auswertung der Beweisüberprüfungsergebnisse durch die nachfolgend beschriebenen Methoden statt.

checkProof():

Die Java-Methode „checkProof()“ wird bei der Betätigung des Buttons „Check“ aufgerufen. Über diese Methode wird die Beweisüberprüfung gestartet und die Ergebnisse mit Hilfe von zwei Threads direkt in Form einer Einfärbung des Beweistextes ausgegeben, wie in der Abbildung 5.12 zu sehen ist:

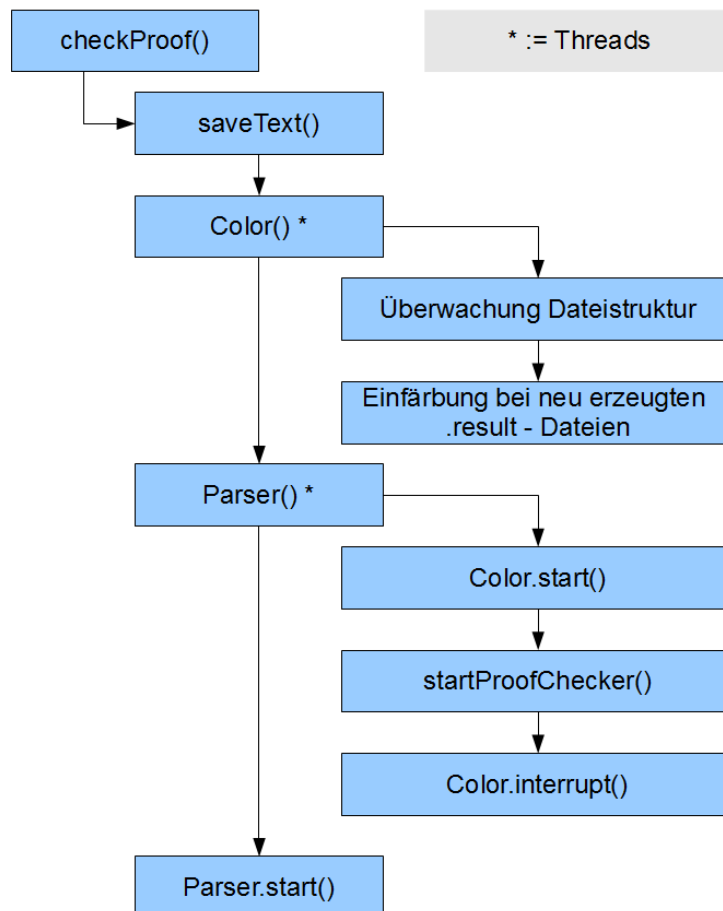


Abbildung 5.12: checkproof()

Damit die Dateien, die bei der Beweisüberprüfung erstellt werden, mit dem Inhalt des Textfeldes übereinstimmen, wird in der Datei „tmp.txt“ der aktuelle Textfeldinhalt gespeichert (siehe Abbildung 5.12).

Der Thread „color()“ wird initialisiert: Alle 1,5 Sekunden wird überprüft, welche „result“ Dateien in ihrem jeweiligen Ordner mit dem Namen der SatzID erstellt wurden. Entsprechend werden die Attribute „globalIDs“ und „globalObligations“ dem GUI mit der Methode „setGlobalIdObl()“ aktualisiert. Der Textfeldinhalt wird mit der Methode „colorAllSentence()“ neu eingefärbt.

Ein weiterer Thread „Parser()“ wird ebenfalls erstellt, in dem der „Color-Thread“ zunächst gestartet wird. Der Aufruf der im Anschluss erklärten Methode „startProofChecker()“ startet die eigentliche Überprüfung des Beweistextes. Nach dem Abschluss der Überprüfung wird der „Color-Thread“ mit „Color.interrupt()“ beendet.

Abschließend wird der Parser-Thread gestartet.

startProofChecker():

Die Aufgabe der Methode „startProofChecker()“ ist es, die Überprüfung des Beweises durchzuführen (Abbildung 5.13):

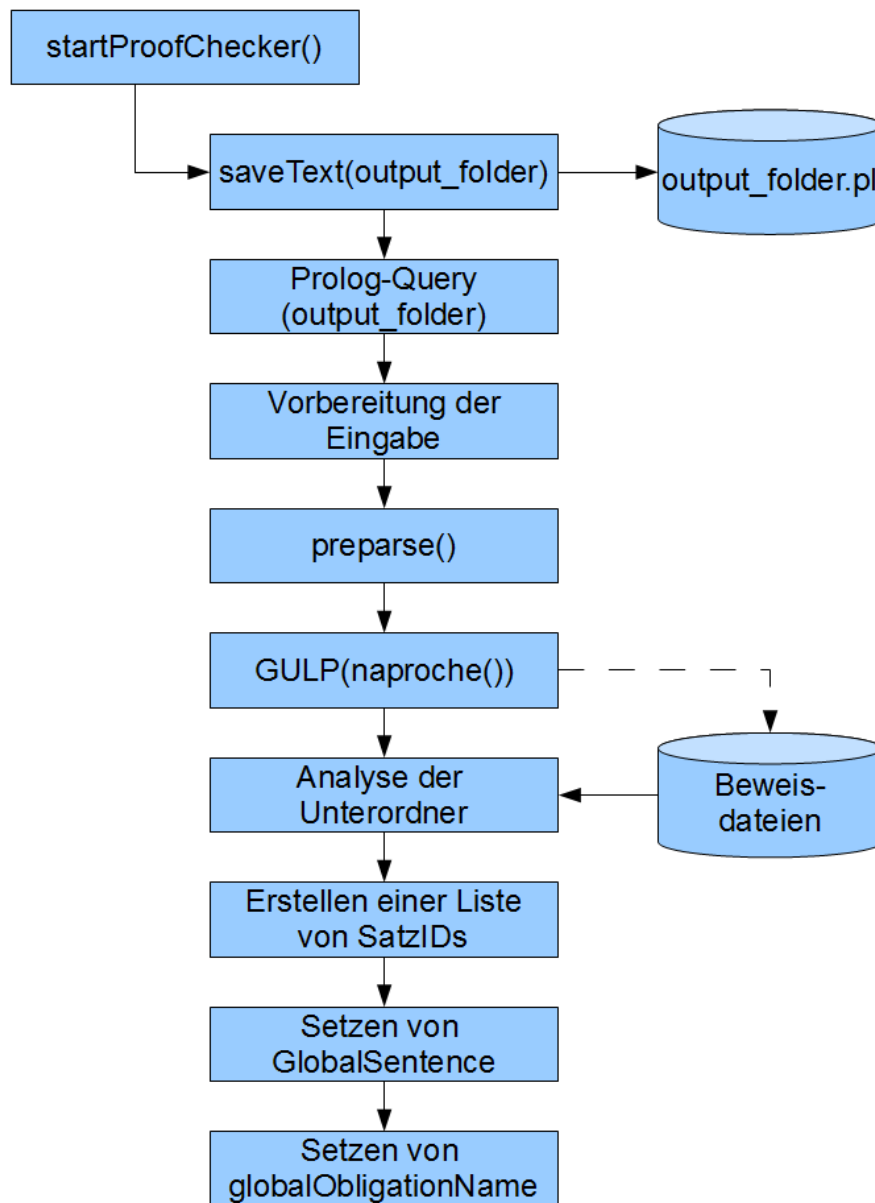


Abbildung 5.13: startProofChecker()

In der Datei „output_folder.pl“ wird die Bezeichnung des Ordners, in dem die Beweisdaten geschrieben werden, festgelegt. Die Methode „saveText()“ legt „tmp“ als Bezeichnung für diesen Ordner fest (siehe Abbildung 5.13). Danach wird die Datei „output_folder.pl“ von Prolog kompiliert. Der Ordner mit der Bezeichnung „tmp“, in dem die Beweisdateien angelegt werden, wird in die Prolog-Datei „output_folder.pl“ geschrieben. Diese Datei wird von Prolog eingelesen und benutzt die Information der Ordnerbezeichnung für die weitere Verarbeitung des Beweistextes. Der Textfeldinhalt wird vor dem preparsen vorbereitet, alle Zeilenumbrüche (\n) werden durch das Zeichen „#“ ersetzt, damit die Zeichen auch in Prolog erfasst werden. Der ersetzte Text wird durch Aufruf der weiter unten erläuterten Methode „preparse()“ von dem Prolog-Prädikat „create_naproche_input()“ vorverarbeitet. Mit der Methode „GULP()“ wird das Prädikat „naproche()“ aufgerufen, in dem die eigentliche Beweisüberprüfung durchgeführt wird und aus der die Beweisüberprüfungsdateien erzeugt werden. Die so vom Prologmodul erzeugten Unterordner werden analysiert und den entsprechenden Java-Variablen („globalIDs“ und „globalObligationName“) die jeweiligen Werte zugewiesen. Abschließend wird der Eingabetext anhand der ermittelten Werte mit der Methode „colorAllSentence()“ neu eingefärbt.

preparse():

Die Hauptaufgabe des Preparsers besteht in der Umwandlung des Eingabetextes in eine Prolog-Liste:

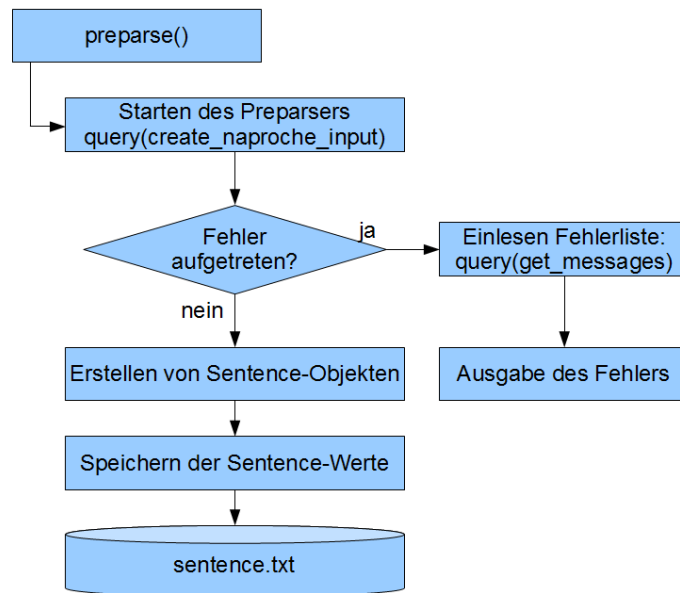


Abbildung 5.14: preparse()

Hierzu wird das Prolog-Prädikat „create_naproche_input()“ aufgerufen. Konnte der Eingabetext vollständig verarbeitet werden, findet eine Konvertierung der Rückgabewerte aus Prolog in „sentence“-Objekte (siehe Kapitel 3.3) statt (siehe Abbildung 5.14). Anschließend werden die Werte nach dem in Kapitel 5.1.4 beschriebenen Aufbau in die Datei „sentence.txt“ geschrieben.

Sollte bei der Verarbeitung des Eingabetextes ein Fehler auftreten, wird über das Prolog-Prädikat „get_messages()“ die Fehlerliste eingelesen (siehe Kapitel 5.1.4). Die Auswertung dieser Liste findet in Form von Ausgaben statt. Der aufgetretene Fehler wird klassifiziert, die Position angegeben und die Fehlerbeschreibung ausgegeben.

5.2.3 Weitere Methoden

In diesem Abschnitt werden kurz die Java-Methoden beschrieben, die bereits in vorherigen Kapiteln genannt wurden, aber keinen direkten Einfluss auf die Beweisüberprüfung haben.

boolean loadProof():

In dieser Methode wird die Datei „sentence.txt“ eingelesen. Die Inhalte werden verarbeitet und „word“- bzw. „sentence“-Objekten gespeichert. Abschließend wird die Methode „setGlobalIDObl()“ aufgerufen. Der Rückgabewert gibt an, ob die Werte aus der Datei verarbeitet werden konnten (true) oder nicht (false).

void colorAllSentence():

Die Einträge der Liste „globalSentences“ werden nacheinander untersucht, wobei für jeden Satz die Methode „checkSentence()“ aufgerufen wird, die den Farbwert des Satzes zurück gibt. Anschließend wird ein String mit dem Start- und End-Index des Satzes sowie der ermittelten Satzfarbe zusammengesetzt. Dieser erzeugte String wird in die Datei „ColorKoord.txt“ geschrieben.

Color checkSentence():

Zu jedem Satz, der überprüfbare Informationen enthält, wird die Methode „checkSentence()“ aufgerufen. Die Ergebnisse aller Obligationen zu dem Satz werden untersucht. Konnten alle Obligationen bewiesen werden, ist der Rückgabewert „grün“, gab es eine Inkonsistenz in den Prämissen wird „orange“ zurückgegeben, und konnte mindestens eine Obligation des Satzes nicht bewiesen werden, so ist der Rückgabewert „rot“. Gab es mindestens eine Obligation, die den Farbwert „rot“ erhalten hat, wird der komplette Satz rot gefärbt. Ist keine „rote“ Obligation dabei, aber mindestens eine orange, wird der Satz „orange“. Wenn es nur „grüne“ Resultate gibt, wird der Satz „grün“ gefärbt. Die ermittelte Farbe wird von der Methode zurückgegeben.

Diese Methode analysiert die von dem Prologmodul erzeugten Unterordner: Anhand der Namen der Ordner werden die Werte der Liste „globalIDs“ bestimmt und mit den Obligationenamen werden der Liste „globalObligationName“ ihre Werte zugewiesen.

Color checkObligation():

Liest die „result“-Datei der jeweiligen Obligation ein und bestimmt anhand der Werte die Farbe, in der die Obligation eingefärbt werden muss.

Hashtable GULP():

Über diese Methode kann ein beliebiges Prädikat aufgerufen werden. Die Rückgabewerte des Prädikates werden in einer Hashtable zwischengespeichert und zurückgegeben.

5.3 TPTP-AxSel

Daniel Kühlwein hat eine Java-API (Application Programm Interface) entwickelt: TPTP-AxSel (Axiom Selection). Hierbei handelt es sich um einen Auswahl-Algorithmus, von dem nur bestimmte Voraussetzungen in die Beweisüberprüfung eingehen. Zur Überprüfung eines Beweises wird ein Unterverzeichnis mit beliebig vielen „input“-Dateien im TPTP-Format an die API übergeben. Beim Start der Überprüfung werden mit den eingestellten Parametern alle Beweisdateien untersucht und die Ergebnisse zur Laufzeit über einen Ausgabestream zur Verfügung gestellt.

In der Weiterentwicklung der GUI soll TPTP-AxSel eingebunden werden, um einzelne Obligationen eines Beweises erneut mit geänderten Parametern überprüfen zu können. Sollte beispielsweise eine Obligation nicht bewiesen werden können, weil dem ATP zuwenig Zeit zur Verfügung stand, kann diese angepasst und so die Obligation in einem neuen Versuch bewiesen werden.

Eine ausführliche Dokumentation findet man auf der Internetseite:

[http://korpora-exp.zim.uni-duisburg-essen.de/naproche/naproche/wiki/doku.php?id=dokumentation:atpapi_interface_documentation].

5.4 Aufruf des Prototypen

Im Rahmen seiner studentischen Hilfskraftstelle wurde von Julian Schlöder ein „Naproche-Installationspaket“ erstellt. Mit diesem Paket kann auf einem beliebigen Linux-System, auf dem das Java Runtime Environment (JRE) installiert wurde, der Prototyp installiert und gestartet werden. Das Installationspaket sowie der Installationsanleitung wird mit der Veröffentlichung der Naproche Version 0.5 über die Homepage [<http://naproche.net>] zum Download bereit stehen.

Installation:

Hierfür muss lediglich das Paket mit dem Dateinamen „naproche-i686.tgz“ oder „naproche-x86_64.tgz“ heruntergeladen werden. Nach dem Entpacken der jeweiligen Datei wird der Prototyp mit einem Aufruf „naproche.sh“ gestartet.

Weiterentwicklung der Prolog-Prädikate:

Nach der Installation des Prototypen können die Prolog-Prädikate modifiziert werden. Um mit Prolog zu arbeiten, muss lokal „SWI-Prolog“ mit dem Befehl „install_swipl.sh“ kompiliert werden. Um die Änderungen in das Naproche-System einzubringen, muss der Prolog-Code mit der Script-Datei „compile.sh“ neu kompiliert werden. Der Aufruf des Prototypen erfolgt unverändert mit der Datei „naproche.sh“.

Erstellung einer Version:

Mit dem Aufruf „make_release.sh“ wird ein Paket erstellt, welches sich aus SWI-Prolog, Naproche, Javakomponenten und dem ATP „E“ zusammensetzt. Dieses Paket kann anschließend wie oben beschrieben installiert werden.

6 Test von Naproche 0.5

Im Folgenden wird die Überprüfung der Naproche-Version 0.5 beschrieben. Ziel der Testlauf ist es, die Funktionen des in Rahmen der Master-Thesis entwickelten Prototypen zu überprüfen und mit der Version 0.47 zu vergleichen. Hierbei wurde ermittelt, wie sich die Änderungen und Weiterentwicklungen des neuen Systems auf die Beweisüberprüfungszeit auswirken.

6.1 Versuchsaufbau

In dem Prototyp wurde folgender Beweistext getestet:

Beispiel 6.1:

Assume that there is a relation \in
and an object \emptyset satisfying the following axioms:

Axiom.

There is no y such that $y \in \emptyset$.

Axiom.

For every x it is not the case that $x \in x$.

Define x to be transitive if and only if for all u , v ,
if $u \in v$ and $v \in x$ then $u \in x$.

Define x to be an ordinal if and only if x is transitive

and for all y , if $y \in x$ then y is transitive.

Theorem.

\emptyset is an ordinal.

Proof.

Assume $u \in v$ and $v \in \emptyset$.

Then there is an x such that $x \in \emptyset$.

Contradiction.

Thus \emptyset is transitive.

Assume $y \in \emptyset$. Then there is an x such that $x \in \emptyset$.

Contradiction.

Thus for all y , if $y \in \emptyset$ then y is transitive.

Hence \emptyset is an ordinal.

Qed.

Theorem.

For all x , y , if $x \in y$ and y is an ordinal then x is an ordinal.

Proof.

Suppose $x \in y$ and y is an ordinal.

Then for all v , if $v \in y$ then v is transitive.

Hence x is transitive.

Assume that $u \in x$. Then $u \in y$, i.e. u is transitive.

Thus x is an ordinal.

Qed.

Theorem: There is no x such that for all u ,

$u \in x$ iff u is an ordinal.

Proof.

Assume for a contradiction that there is an x such that for all u ,

$u \in x$ iff u is an ordinal.

Lemma: x is an ordinal.

Proof:

Let $u \in v$ and $v \in x$. Then v is an ordinal,
i.e. u is an ordinal, i.e. $u \in x$. Thus x is transitive.
Let $v \in x$. Then v is an ordinal, i.e. v is transitive.
Thus x is an ordinal. Qed.

Then $x \in x$. Contradiction.
Qed.

Auf zwei verschiedenen Rechnern wurde dieser Beispieltext 6.1 ins Eingabefeld des Prototypen eingefügt und anschließend die Beweisüberprüfung gestartet.
Die Daten der beiden Rechner sind genau wie die Daten des Webserver, der zum Vergleich der beiden Naproche-Systeme benötigt wurde, hier aufgeführt:

Spezifikation Rechner 1:

- Betriebssystem: Ubuntu 10.04
- Prozessor: Pentium 4, 2.66 GHz
- Arbeitsspeicher: 2 GB RAM

Spezifikation Rechner 2: Im Gegensatz zu dem anderen Rechner fand auf diesem Rechner die Berechnung nicht auf einer Festplatte, sondern im Flash-Speicher statt.

- Betriebssystem: Ubuntu 11.04
- Prozessor: Dual Core mit 2.5 GHz
- Arbeitsspeicher: 8 GB RAM

Spezifikation Webserver:

- System: Virtueller Server
- Prozessor: Xeon (Core 2) 3 GHz
(Prozessor in etwa doppelt so schnell wie Rechner 1)
- Arbeitsspeicher: 1 GB RAM

Um die Belastung des Systems bei relativ großen Eingabetexten zu testen, wurde die Textlänge der Tests durch Kopieren von Textteilen künstlich erhöht. Dabei wurden fünf Varianten des zu überprüfenden Beweistextes erstellt: Die Axiome und Definitionen (von Satz 1 bis Satz 12) sind in jeder Variante gleich geblieben. Der eigentliche Beweis (von Satz 13 bis Satz 58) wird je nach Variante mehrfach wiederholt: In der Ausgangsvariante kommt der Burali-Forti-Beweis (BF, siehe [4]) genau einmal vor. Weitere Varianten sind ein zweimaliges, fünfmaliges, zehnmaliges und elfmaliges Wiederholen der Sätze 13 bis 58. Auf diese Art und Weise wurden bis zu 538 Sätze erzeugt und überprüft.

6.2 Testergebnisse des Lasttests

Die Tabellen mit den Testergebnissen zeigen, wie viele Sätze in welcher Zeit überprüft werden. Zuerst wird der Anzahl der vorkommenden Burali-Forti-Beweise angegeben („Anz. BF“), die Anzahl der zu überprüfenden Sätze („Anz. Sätze“) ist daneben positioniert. Die Spalte „neu“ steht dafür, dass der Beweis jeweils komplett neu, ohne auf vorhandene Daten zuzugreifen, überprüft wurde. Die Nachbarspalte „mit“ bedeutet, dass die Überprüfung mit vorhandenen Daten ablief. Die vorhandenen Daten beziehen sich hierbei immer auf die vorangegangene Versuchsvariante. Also sind beispielsweise bei der „5 x“- Burali-Forti-Variante die Daten der Überprüfung der „2 x“- Burali-Forti-Variante noch vorhanden.

Rechner 1:

Anz. BF	Anz. Sätze	neu (s)	mit (s)
1 x	58	55.73	-
2 x	106	126.35	63.83
5 x	250	448.57	279.03
10 x	490	1330.96	765.02
11 x	538	1520.43	228.25

Tabelle 6.1: Rechner 1

Rechner 2:

Anz. BF	Anz. Sätze	neu (s)	mit (s)
1 x	58	24.12	-
2 x	106	58.15	22.51
5 x	250	151.77	96.25
10 x	490	332.40	197.14
11 x	538	363.95	41.62

Tabelle 6.2: Rechner 2

Die Abbildung 6.1 zeigt die Ergebnisse des Laufzeittests der beiden Rechner ohne alte Beweisergebnisse:

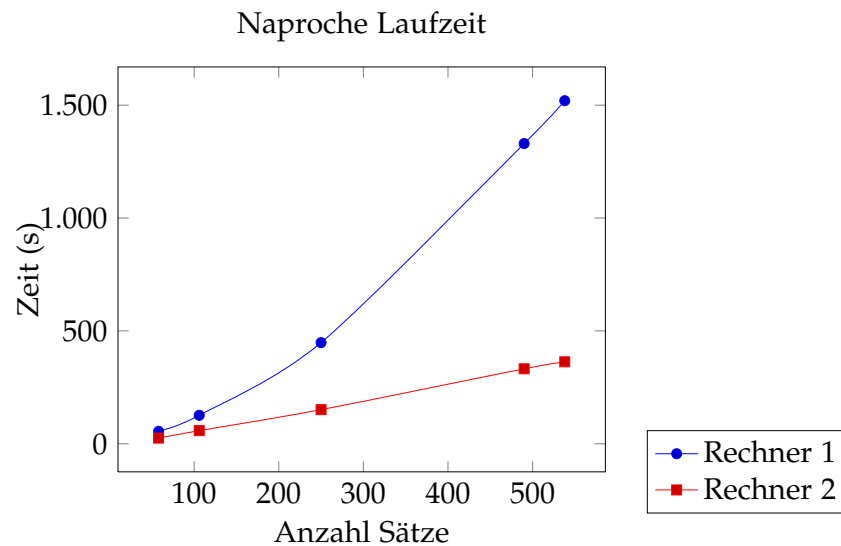


Abbildung 6.1: Laufzeit beider Rechner

Aus einem Vergleich der beiden Rechner kann man deutlich erkennen, dass es einen großen Unterschied macht, wie leistungsfähig ein Rechner ist. Hinzu kommt, dass auf „Rechner 2“ die Berechnungen statt auf der Festplatte auf dem Flash-Speicher durchgeführt wurden. Während die Zeiten bei Rechner 1 schnell in einen kritischen Bereich der Bearbeitungszeit für einen Anwender gehen, sind die Zeiten des 2. Rechners in einem akzeptablen Bereich. Für beide Rechner gilt, dass je länger der Eingabetext ist, desto mehr Zeit für die Überprüfung benötigt wird.

In beiden Fällen handelt es sich ab 250 Sätzen ungefähr um ein lineares Wachstum der Laufzeit. Steigt die Anzahl der zu überprüfenden Sätze, steigt die Überprüfungszeit proportional an.

Für die Anwendung des Prototypen ist es zwar nicht erforderlich, aber deutlich angenehmer einen leistungsstarken Rechner zu benutzen. Gerade bei längeren Texten macht sich der Leistungsunterschied sehr schnell bemerkbar.

6.3 Vergleich zum alten Naproche-System Version 0.47

6.3.1 Version 0.47 vs Version 0.5 ohne Beweisdaten

Um einen möglichst guten Vergleich zwischen den Versionen 0.47 und 0.5 herstellen zu können, musste der Beispieltext 6.1 in zwei Punkten geändert werden, um diesen an die modifizierte Formelgrammatik der Version 0.5 anzupassen:

- Der für die Version 0.5 benötigte neue Satz am Anfang des Beispieltextes musste wieder gelöscht werden:

Assume that there is a relation $\text{\textit{\$in\$}}$
and an object $\text{\textit{\$emptyset\$}}$ satisfying the following axioms:

- Am Anfang des ersten Beweises (Satz 16) konnte „Consider“ durch „Assume“ ersetzt werden, wobei „Assume“ in beiden Versionen funktioniert.

Der Test der Naproche-Version 0.47 ist genauso aufgebaut wie der des Prototypen: Die Axiome und Definitionen stehen einmalig im Beweistext, die eigentlichen Beweisschritte wurden bis zu 11 mal ergänzt:

Anz. BF	Anz. Sätze	Dauer (s)
1 x	58	11
2 x	106 x	18
5 x	250 x	44
10 x	490 x	154
11 x	538 x	185

Tabelle 6.3: Naproche 0.47

Es folgt der Graph 6.2 mit dem Vergleich der Werte ohne Zwischenspeicherung:

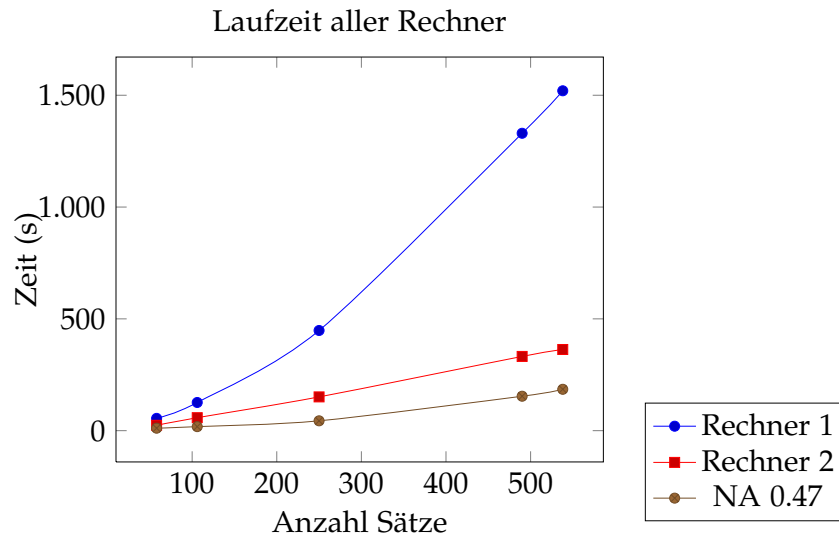


Abbildung 6.2: Laufzeit aller Rechner

Die Naproche-Version 0.47 ist eindeutig am schnellsten, wenn es darum geht, einen Beweistext zu überprüfen, ohne auf Überprüfungsergebnisse zuzugreifen.

6.3.2 Version 0.47 vs Version 0.5 mit Beweisdaten

Um die Datenspeicherung der Beweisüberprüfung zu testen, wurden erneut Tests mit beiden Rechnern durchgeführt. Hier wurde der Beweis überprüft, nachdem die Daten der vorherigen Überprüfung vorhanden waren: Bei einmal Burali-Forti wurden keine Beweisdaten benutzt, bei zweimal Burali-Forti die von einem mal, bei 5 mal die von 4 mal, bei 10 mal die von 9 mal und bei 11 mal die von 10 mal.

Anz. BF	Anz. Sätze	Rechner 1	Rechner 2
1 x	58	57.35	24.12
2 x	106	65.25	22.51
5 x	250	107.82	32.45
10 x	490	204.44	38.34
11 x	538	228.62	41.62

Tabelle 6.4: Rechner 1 & 2

Vergleicht man die so entstandenen Überprüfungszeiten der beiden Rechner mit dem Webinterface entsteht dieses Diagramm:

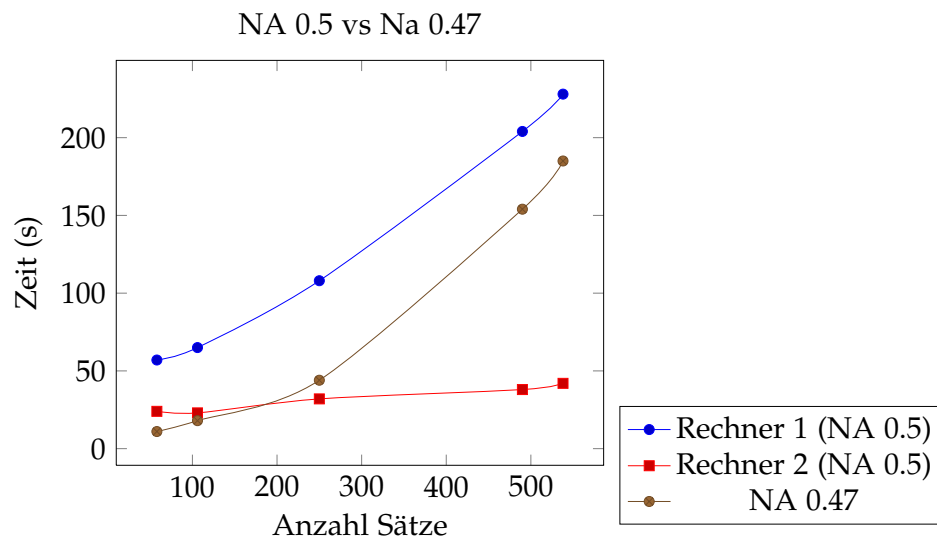


Abbildung 6.3: NA 0.5 vs Na 0.47

Mit vorhandenen Beweisdaten ist „Rechner 2“ klar schneller als das Webinterface (42 Sekunden zu 185 Sekunden), wenn der 10-malige Burali-Forti-Beweis auf den 11-maligen erhöht wird. Hingegen ist „Rechner 1“ (228 Sekunden) zu langsam. Die Grenze, ab der sich die Speicherung der Beweisdaten lohnt, liegt beim Vergleich von Rechner 2 zum Webinterface bei ca 180 Sätzen.

6.4 Interpretation der Testergebnisse:

Der Test hat gezeigt, dass mit der Speicherung der Daten einer Beweisüberprüfung für längere Texte der Gesamtbeweistext wesentlich schneller überprüft werden konnte. Das Webinterface hat eine polynomielle Laufzeit, während die Laufzeit des Prototypen einen linearen Verlauf hat.

Weiterhin darf nicht vernachlässigt werden, dass sich die Leistungsfähigkeit und damit die Komplexität der Beweisüberprüfungsmodule erhöht hat. Durch diese Steigerung wurde das Gesamtsystem langsamer, was die schlechteren Zeiten bei einem Test ohne die vorher vorhandenen Daten erklärt.

Der Einsatz der Beweisspeicherung macht auf jeden Fall dann Sinn, wenn längere Beweistexte erfasst und ergänzt werden sollen. Sollen viele neue Beweisschritte überprüft werden, lohnt es sich, an einem leistungsstarken Rechner zu arbeiten, um so die Überprüfungszeit zu verkürzen.

7 Zusammenfassung

In diesem Kapitel wird die Arbeit rund um die entstandene Master-Thesis zusammengefasst und ein Ausblick gegeben.

7.1 Fazit

Die im Rahmen dieser Master-Thesis gestellten Aufgaben wurden alle erfolgreich abgeschlossen.

Die Analyse der vorherigen Version 0.47 des Naproche-Systems wurde im Kapitel 3 dokumentiert und als Grundlage genutzt, um Anforderungen an das Naproche-System 0.5 (siehe Kapitel 4) zu spezifizieren. Diese Spezifikation beinhaltet nicht nur die Bereiche des Prototypen, der in dieser Master-Thesis entwickelt wurde, sondern stellt ebenfalls Anforderungen an die Prolog-Module für die Version 0.5 des Naproche-Systems. Der Java-Prototyp wurde entwickelt und hat seine volle Funktionsfähigkeit unter anderem im Lasttest unter Beweis gestellt. Im Prototyp wurden Schnittstellen zum Aufruf von Prolog-Prädikaten, die wiederum die ATP-Module für die logische und die PRS-Module für die linguistische Überprüfung aufrufen, implementiert. Das Naproche-System der Version 0.5 kann im Gegensatz zu der Version 0.47 bereits prädikatenlogische Ausdrücke höherer Ordnungen überprüfen. Weiterhin ist hier auch das inkrementelle Parsen auf Satzebene möglich gemacht worden.

Das Grafikal User Interface bietet als Programmsystem (inklusive Prototyp) die Möglichkeit, Beweise zu speichern und mit ihren Beweisdaten zu laden. Realisiert wurde dies mit einer Zip-Datei, welche die Beweisüberprüfungsdaten beinhaltet. Die Zip-Datei ist die in dieser Master-Thesis zu erstellende ProofState-Datei die alle Kriterien eines persistenten Datentyps erfüllt und die Daten dabei gleichzeitig komprimiert. Die

Daten, die im temporären Ordner „tmp“ vorhanden sind und bei jedem Neustart des GUI geladen werden, sind ebenfalls persistent.

Jeder Beweistext kann beliebig modifiziert und dieser anschließend ab der ersten Änderung neu überprüft werden. Der Test des Prototypen hat ergeben, dass mit dem Zugriff auf vorhandene Beweisdateien die Beweisüberprüfung schneller abläuft als mit der Version 0.47 des Naproche-Systems. Die Laufzeit steigt mit zunehmender Beweislänge mit dem neuen Naproche-System 0.5 nicht mehr polynomial, sondern annähernd nur noch linear an.

7.2 Ausblick

Im Folgenden werden einige Punkte für eine Weiterentwicklung der Version 0.5 aufgezählt.

Der durch die Arbeit entwickelte Prototyp ist voll funktionsfähig und einsatzbereit. Das GUI, welches die Umgebung des Prototypen darstellt, kann noch benutzerfreundlicher gemacht werden:

Während des Lasttests ist aufgefallen, dass sich die GUI-Benutzerelemente im Laufe der eigentlichen Beweisüberprüfung nicht anwenderfreundlich verhalten. Der Inhalt des Textfeldes sollte beispielsweise solange nicht verändert werden dürfen, bis die Überprüfung abgeschlossen ist.

Weiterhin kann das Java-Projekt „TPTP-AxSel“ vollständig in das GUI eingearbeitet werden, so dass einzelne Obligationen eines Beweissatzes mit geänderten Einstellungen neu überprüft werden können.

Da sich auch die Prolog-Module in Zukunft weiterentwickeln werden, müssen auch entsprechende Anpassungen in dem GUI vorgenommen werden.

Ein weiteres Ziel ist es, die vorhandenen linguistischen und logischen Prolog-Module der Version 0.5 des Naproche-Systems zu optimieren und weiterentwickeln. Langfristig soll es beispielsweise möglich werden, eine Referenzierung auf Theoreme und Lemmata aus anderen Beweistexten anzubieten, so dass ein Beweistext am Anfang keine Axiome und Definitionen mehr benötigt.

8 Quellen

- [1] P. Blackburn, Learn PROLOG Now!, 1. Auflage 2006
- [2] U. Breymann, Der Programmierer, Carl Hanser Verlag München, 2009
- [3] G. Büchel, Datenbanken Vorlesungsskript, Fachhochschule Köln, 2008
- [4] Cesare Burali-Forti. Una questione sui numeri transfiniti. Rendiconti del Circolo Matematico di Palermo, 11:154,Äì164, 1897.
- [5] M. Cramer, B. Fisseni, P. Koepke, D. Kühlwein, B. Schröder, and J. Veldman, The Naproche Project, Controlled Natural Language Proof Checking of Mathematical Texts, Universität Bonn, 2009
- [6] M. Cramer, P. Koepke, and B. Schröder, Parsing and Disambiguation of Symbolic Mathematics in the Naproche System, Universität Bonn, 2011
- [7] M. Cramer, D. Kühlwein und B. Schröder: Presupposition Projection and Accommodation in Mathematical Texts. Semantic Approaches in Natural Language Processing: Proceedings of the Conference on Natural Language Processing, Universität Bonn, 2010
- [8] M. Cramer, Master Thesis, Mathematisch-logische Aspekte von Beweisrepräsentationsstrukturen, Universität Bonn, 2009
- [9] H.-D. Ebbinghaus, J. Flum, and W. Thomas. Einführung in die mathematische Logik. Spektrum, 1992.
- [10] Euklid: Die Elemente, 3. Auflage 1997

- [11] M.Fischer, Persistente Datenspeicherung, http://mf-home.net/index.php?option=com_content&id=33:persistente-datenspeicherung&catid=14:frameworks&Itemid=59
Stand: 26. September 2011
- [12] M. Guhe, F. Schilder, Semantische Sprachverarbeitung, <http://www.informatik.uni-hamburg.de/WSV/teaching/vorlesungen/SemSprachFolien/SemSprach4-DRT03.pdf>
Stand: 26. September 2011
- [13] P. Koepke, Vorlesungsskript Logik, <http://www.math.uni-bonn.de/people/logic/teaching>
Stand: 26. September 2011
- [14] N. Kolev, Magister thesis, Generating Proof Representation Structures in the Project NAPROCHE, 2008
- [15] D. Kühlwein, M. Cramer, P. Koepke, and B. Schröder, The Naproche System, Universität Bonn, 2009
- [16] D. Kühlwein, Diplomarbeit, A calculus for Proof Representation Structures, Universität Bonn, 2009
- [17] E. Landau: Grundlagen der Analysis (Third Edition, 1960).
- [18] A. W. Lockermann, Script Prolog-Tutorium der Fachhochschule Köln, April 2009
- [19] M. Rahn, Praktikumsbericht, Universität Bonn, 2009
- [20] J. Schlöder, Praktikumsbericht, Universität Bonn, 2010
- [21] R. Sponsel, Typentheorie, <http://www.sgipt.org/wisms/gb/typenth.htm>, Erlangen, Stand: 26. September 2011
- [22] TPTP-Homepage, <http://www.cs.miami.edu/~tptp/>, Stand: 26. September 2011
- [23] Wikipedia, Russellsche Antinomie, http://de.wikipedia.org/wiki/Russellsche_Antinomie
Stand: 26. September 2011

- [24] A. Wenzel, T. Kraußer, Automatische Theorem-Beweiser, Universität Bremen, 20.01.2004
- [25] K.-U. Witt, Grundkurs Theoretische Informatik, 3. Auflage 2004
- [26] S. Zittermann, Praxissemesterbericht, Fachhochschule Köln 2010
- [27] University of Zurich, Projekt-Webseite, <http://attempto.ifi.uzh.ch/site/>, Stand: 26. September 2011
- [28] Projekt-Webseite, <http://www.cotilliongroup.com/arts/DCG.html>, Stand: 26. September 2011

Abbildungsverzeichnis

2.1	Beispiel einer DRS	27
2.2	Aufbau einer PRS	28
2.3	Beispiel einer PRS	31
2.4	Datentypen in Prolog	42
3.1	Bereiche des Webinterfaces	49
3.2	Übersichtsdiagramm Naproche-System 0.47	51
3.3	„Build_PRS()“	55
3.4	„check_conditions()“: PRS	57
3.5	„check_conditions()“: Assumption	59
3.6	„discharge_obligations()“	61
5.1	Übersichtsdiagramm Naproche-System 0.5	68
5.2	"Preparser	70
5.3	Änderungen der Versionen	75
5.4	Aufbau der Datenstruktur	77
5.5	Klassendiagramm	87
5.6	Screenshot des GUI	88
5.7	Funktionen des GUI	89
5.8	Funktionen der GUI-Buttons	90
5.9	Funktionen des GUI-Menüs	92
5.10	Funktionen des Popup-Menüs	93
5.11	Initialisierung des GUI	94
5.12	checkproof()	96
5.13	startProofChecker()	98
5.14	preparse()	100

6.1	Laufzeit beider Rechner	109
6.2	Laufzeit aller Rechner	111
6.3	NA 0.5 vs Na 0.47	112

Tabellenverzeichnis

2.1	Legender der Diagramme	40
6.1	Rechner 1	108
6.2	Rechner 2	108
6.3	Naproche 0.47	110
6.4	Rechner 1 & 2	112

9 Anhang

9.1 Ordnerstruktur der Überprüften Beweise

9.1.1 Burali-Forti

- /tmp:
 - /13:
 - * 13-13-0-0.input
 - * 13-13-0-0.output
 - * 13-13-0-0.proofsummary
 - * 13-13-0-0.result
 - * obligations.nap
 - * theorem-obligations.nap
 - /17:
 - * 17-17-0-0.input
 - * 17-17-0-0.output
 - * 17-17-0-0.proofsummary
 - * 17-17-0-0.result
 - * obligations.nap
 - /18:
 - * 18-18-0-0.input
 - * 18-18-0-0.output
 - * 18-18-0-0.proofsummary
 - * 18-18-0-0.result
 - * obligations.nap
 - /19:
 - * 19-19-0-0.input
 - * 19-19-0-0.output
 - * 19-19-0-0.proofsummary
 - * 19-19-0-0.result
 - * obligations.nap

- /21:
 - * 21-21-0-0.input
 - * 21-21-0-0.output
 - * 21-21-0-0.proofsummary
 - * 21-21-0-0.result
 - * obligations.nap
- /22:
 - * 22-22-0-0.input
 - * 22-22-0-0.output
 - * 22-22-0-0.proofsummary
 - * 22-22-0-0.result
 - * obligations.nap
- /23:
 - * 23-apod(scope(23))-0-0.input
 - * 23-apod(scope(23))-0-0.output
 - * 23-apod(scope(23))-0-0.proofsummary
 - * 23-apod(scope(23))-0-0.result
 - * obligations.nap
- /24:
 - * 24-24-0-0.input
 - * 24-24-0-0.output
 - * 24-24-0-0.proofsummary
 - * 24-24-0-0.result
 - * obligations.nap
- /28:
 - * 28-apod(scope(28))-0-0.input
 - * 28-apod(scope(28))-0-0.proofsummary
 - * obligations.nap
 - * 28-apod(scope(28))-0-0.output
 - * 28-apod(scope(28))-0-0.result
 - * theorem-obligations.nap
- /32:
 - * 32-apod(scope(32))-0-0.input

- * 32-apod(scope(32))-0-0.output
- * 32-apod(scope(32))-0-0.proofsummary
- * 32-apod(scope(32))-0-0.result
- * obligations.nap
- /33:
 - * 33-33-0-0.input
 - * 33-33-0-0.output
 - * 33-33-0-0.proofsummary
 - * 33-33-0-0.result
 - * obligations.nap
- /35:
 - * 35-conseq_conjunct1(35)-holds-0.input
 - * 35-conseq_conjunct1(35)-holds-0.result
 - * 35-conseq_conjunct2(35)-0-0.proofsummary
 - * 35-conseq_conjunct1(35)-holds-0.output
 - * 35-conseq_conjunct2(35)-0-0.input
 - * 35-conseq_conjunct2(35)-0-0.result
 - * 35-conseq_conjunct1(35)-holds-0.proofsummary
 - * 35-conseq_conjunct2(35)-0-0.output
 - * obligations.nap
- /36:
 - * 36-36-0-0.input
 - * 36-36-0-0.output
 - * 36-36-0-0.proofsummary
 - * 36-36-0-0.result
 - * obligations.nap
- /40:
 - * 40-neg(conjunct1(40))-0-0.input
 - * 40-neg(conjunct1(40))-0-0.proofsummary
 - * obligations.nap
 - * 40-neg(conjunct1(40))-0-0.output
 - * 40-neg(conjunct1(40))-0-0.result
 - * theorem-obligations.nap

- /45:
 - * 45-45-0-0.input
 - * 45-45-0-0.output
 - * 45-45-0-0.proofsummary
 - * 45-45-0-0.result
 - * obligations.nap
 - * theorem-obligations.nap
- /49:
 - * 49-conseq_conjunct1(49)-0-0.input
 - * 49-conseq_conjunct1(conseq_conjunct2(49))-0-0.result
 - * 49-conseq_conjunct1(49)-0-0.output
 - * 49-conseq_conjunct2(conseq_conjunct2(49))-holds-0.input
 - * 49-conseq_conjunct1(49)-0-0.proofsummary
 - * 49-conseq_conjunct2(conseq_conjunct2(49))-holds-0.output
 - * 49-conseq_conjunct1(49)-0-0.result
 - * 49-conseq_conjunct2(conseq_conjunct2(49))-holds-0.proofsummary
 - * 49-conseq_conjunct1(conseq_conjunct2(49))-0-0.input
 - * 49-conseq_conjunct2(conseq_conjunct2(49))-holds-0.result
 - * 49-conseq_conjunct1(conseq_conjunct2(49))-0-0.output
 - * obligations.nap
 - * 49-conseq_conjunct1(conseq_conjunct2(49))-0-0.proofsummary
- /50:
 - * 50-50-0-0.input
 - * 50-50-0-0.output
 - * 50-50-0-0.proofsummary
 - * 50-50-0-0.result
 - * obligations.nap
- /52:
 - * 52-conseq_conjunct1(52)-0-0.input
 - * 52-conseq_conjunct1(52)-0-0.result
 - * 52-conseq_conjunct2(52)-0-0.proofsummary
 - * 52-conseq_conjunct1(52)-0-0.output
 - * 52-conseq_conjunct2(52)-0-0.input

- * 52-conseq_conjunct2(52)-0-0.result
- * 52-conseq_conjunct1(52)-0-0.proofsummary
- * 52-conseq_conjunct2(52)-0-0.output
- * obligations.nap
- /53:
 - * 53-53-0-0.input
 - * 53-53-0-0.output
 - * 53-53-0-0.proofsummary
 - * 53-53-0-0.result
 - * obligations.nap
- /56:
 - * 56-56-holds-0.input
 - * 56-56-holds-0.output
 - * 56-56-holds-0.proofsummary
 - * 56-56-holds-0.result
 - * obligations.nap
- /57:
 - * 57-57-0-0.input
 - * 57-57-0-0.output
 - * 57-57-0-0.proofsummary
 - * 57-57-0-0.result
 - * obligations.nap
- /data:
 - * 1.nap
 - * 2.nap
 - * 3.nap
 - * 4.nap
 - * 5.nap
 - * 6.nap
 - * 7.nap
 - * 8.nap
 - * 9.nap
 - * 10.nap

- * 11.nap
- * 12.nap
- * 13.nap
- * 14.nap
- * 15.nap
- * 16.nap
- * 17.nap
- * 18.nap
- * 19.nap
- * 20.nap
- * 21.nap
- * 22.nap
- * 23.nap
- * 24.nap
- * 25.nap
- * 26.nap
- * 27.nap
- * 28.nap
- * 29.nap
- * 30.nap
- * 31.nap
- * 32.nap
- * 33.nap
- * 34.nap
- * 35.nap
- * 36.nap
- * 37.nap
- * 38.nap
- * 39.nap
- * 40.nap
- * 41.nap
- * 42.nap
- * 43.nap

- * 44.nap
- * 45.nap
- * 46.nap
- * 47.nap
- * 48.nap
- * 49.nap
- * 50.nap
- * 51.nap
- * 52.nap
- * 53.nap
- * 54.nap
- * 55.nap
- * 56.nap
- * 57.nap
- text.nap
- tmp.txt
- ColorKoord.txt
- sentence.txt
- theorem-obl-list.nap

9.1.2 Landau

- /tmp:
 - /33:
 - * 33-apod(33)-holds-0.input
 - * 33-apod(33)-holds-0.output
 - * 33-apod(33)-holds-0.proofsummary
 - * 33-apod(33)-holds-0.result
 - * obligations.nap
 - * theorem-obligations.nap
 - /38:
 - * 38-38-holds-0.input
 - * 38-38-holds-0.output
 - * 38-38-holds-0.proofsummary

- * 38-38-holds-0.result
- * obligations.nap
- /43:
 - * 43-scope(43)-holds-0.input
 - * 43-scope(43)-holds-0.output
 - * 43-scope(43)-holds-0.proofsummary
 - * 43-scope(43)-holds-0.result
 - * obligations.nap
 - * theorem-obligations.nap
- /47:
 - * 47-the(47)-1-0.input
 - * 47-the(47)-1-0.output
 - * 47-the(47)-1-0.result
 - * obligations.nap
- /49:
 - * 49-49-holds-0.input
 - * 49-49-holds-0.output
 - * 49-49-holds-0.proofsummary
 - * 49-49-holds-0.result
 - * obligations.nap
- /51:
 - * 51-conseq_conjunct1(apod(51))-holds-0.input
 - * 51-conseq_conjunct1(apod(51))-holds-0.output
 - * 51-conseq_conjunct1(apod(51))-holds-0.proofsummary
 - * 51-conseq_conjunct1(apod(51))-holds-0.result
 - * 51-conseq_conjunct2(apod(51))-holds-0.input
 - * 51-conseq_conjunct2(apod(51))-holds-0.output
 - * 51-conseq_conjunct2(apod(51))-holds-0.proofsummary
 - * 51-conseq_conjunct2(apod(51))-holds-0.result
 - * obligations.nap
- /52
- /57
- /61

- /63
- /66
- /67
- /69
- /70
- /75
- /83
- /85
- /87
- /88
- /89
- /91
- /94
- /97
- /98
- /99
- /102
- /103
- /105
- /107
- /109
- /114
- /119
- /122
- /126
- /128
- /133
- /137
- /139
- /142
- /143
- /144
- /145
- /147
- /148

- /153
- /158
- /159
- /162
- /163
- /164
- /166
- /171
- /176
- /179
- /182
- /184
- /186
- /191
- /196
- /199
- /202
- /203
- /207
- /212
- /215
- /218
- /220
- /data:
 - * 1.nap
 - * 2.nap
 - * 3.nap
 - * 4.nap
 - * ...
 - * 222.nap
- ColorKoord.txt
- sentence.txt
- text.nap
- theorem-obl-list.nap
- tmp.txt

9.1.3 Group-Theory

- /tmp:
 - /13:
 - * 13-apod(13)-holds-0.input
 - * 13-apod(13)-holds-0.output
 - * 13-apod(13)-holds-0.proofsummary
 - * 13-apod(13)-holds-0.result
 - * obligations.nap
 - * theorem-obligations.nap
 - /17:
 - * 17-17-holds-0.input
 - * 17-17-holds-0.output
 - * 17-17-holds-0.proofsummary
 - * 17-17-holds-0.result
 - * obligations.nap
 - /18:
 - * 18-18-holds-0.input
 - * 18-18-holds-0.output
 - * 18-18-holds-0.proofsummary
 - * 18-18-holds-0.result
 - * obligations.nap
 - /19:
 - * 19-19-holds-0.input
 - * 19-19-holds-0.output
 - * 19-19-holds-0.proofsummary
 - * 19-19-holds-0.result
 - * obligations.nap
 - /20:
 - * 20-20-holds-0.input
 - * 20-20-holds-0.output
 - * 20-20-holds-0.proofsummary
 - * 20-20-holds-0.result
 - * obligations.nap

- /24:
 - * 24-apod(24)-holds-0.input
 - * 24-apod(24)-holds-0.output
 - * 24-apod(24)-holds-0.proofsummary
 - * 24-apod(24)-holds-0.result
 - * obligations.nap
 - * theorem-obligations.nap
- /28:
 - * 28-conseq_conjunct1(28)-holds-0.input
 - * 28-conseq_conjunct1(28)-holds-0.output
 - * 28-conseq_conjunct1(28)-holds-0.proofsummary
 - * 28-conseq_conjunct1(28)-holds-0.result
 - * 28-conseq_conjunct2(28)-holds-0.input
 - * 28-conseq_conjunct2(28)-holds-0.output
 - * 28-conseq_conjunct2(28)-holds-0.proofsummary
 - * 28-conseq_conjunct2(28)-holds-0.result
 - * obligations.nap
- /29:
 - * 29-conseq_conjunct1(29)-holds-0.input
 - * 29-conseq_conjunct1(29)-holds-0.output
 - * 29-conseq_conjunct1(29)-holds-0.proofsummary
 - * 29-conseq_conjunct1(29)-holds-0.result
 - * 29-conseq_conjunct2(29)-holds-0.input
 - * 29-conseq_conjunct2(29)-holds-0.output
 - * 29-conseq_conjunct2(29)-holds-0.proofsummary
 - * 29-conseq_conjunct2(29)-holds-0.result
 - * obligations.nap
- /data:
 - * 1.nap
 - * 2.nap
 - * 3.nap
 - * 4.nap
 - * 5.nap

- * 6.nap
- * 7.nap
- * 8.nap
- * 9.nap
- * 10.nap
- * 11.nap
- * 12.nap
- * 13.nap
- * 14.nap
- * 15.nap
- * 16.nap
- * 17.nap
- * 18.nap
- * 19.nap
- * 20.nap
- * 21.nap
- * 22.nap
- * 23.nap
- * 24.nap
- * 25.nap
- * 26.nap
- * 27.nap
- * 28.nap
- * 29.nap
- * 30.nap
- ColorKoord.txt
- sentence.txt
- text.nap
- theorem-obl-list.nap
- tmp.txt

9.2 Java-Quelltexte

Listing 9.1: FileOperation.java

```

1 package net.naproche.GUI;
2 import java.awt.Color;
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileInputStream;
6 import java.io.FileNotFoundException;
7 import java.io.FileOutputStream;
8 import java.io.FileReader;
9 import java.io.IOException;
10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.util.Enumeration;
13 import java.util.zip.ZipEntry;
14 import java.util.zip.ZipFile;
15 import java.util.zip.ZipOutputStream;
16
17 /**
18  * Diese Klasse bietet den Zugriff auf Dateien,
19  * sowie das packen und entpacken von Dateien an.
20  */
21 public class FileOperations {
22     private static final byte[] buffer = new byte[0xFFFF];
23
24     // Main zum Testen der Funktionen
25     public static void main(String[] args){
26         //System.out.println("Jetzt wird gezippt");
27         //zipProof("sinn.zip", "tmp");
28         //System.out.println("Jetzt wird entzippt");
29         //unzipDir("sinn.zip");
30
31         File x = new File(".");
32         String pfad = x.getAbsolutePath();
33         //pfad = "/home/sebastian/workspace/test/";
34         pfad = "/home/sebastian/workspace/javanaproche/";
35         //String fname = "2cases.zip";
36         String fname = "sinn.zip";
37
38         unzip(fname, pfad);
39
40         System.out.println("Entpackt in " + pfad);
41     }
42
43     /** Liest eine Datei als einen String ein
44      * @param File: einzulesende Datei
45      * @return String: Inhalt der Datei als String
46      */
47     public static String loadFileToString(File file){
48         StringBuffer buf = new StringBuffer();
49         if(file.exists()){
50             try {
51                 BufferedReader reader = new BufferedReader(new FileReader(file));
52                 String line = "";
53                 while((line = reader.readLine()) != null){
54                     buf.append(line+"\n");
55                 }

```

```

56         reader.close( );
57     }
58     catch (FileNotFoundException e) {
59         e.printStackTrace( );
60     }
61     catch (IOException e) {
62         e.printStackTrace( );
63     }
64 }
65 else {
66     System.out.println("NICHT DA!");
67 }
68 return (buf.toString());
69 }
70
71
72 /** Zippt einen Beweis
73  * @param zipfilename: Dateiname
74  * @param dirtozip: zu packendes Verzeichnis
75  */
76 public static void zipProof(String zipfilename, String dirtozip){
77     try {
78         File x = new File(".");
79         String dirname = dirtozip;
80         String zipname = zipfilename;
81         String pfad = x.getAbsolutePath();
82
83         System.out.println("Pfad: " + pfad + "\t fname: " + dirname);
84         ZipOutputStream zos = new ZipOutputStream(new FileOutputStream(zipname));
85
86         if (new File (dirname).exists()){
87             zipDir(dirname, zos);
88             zos.close();
89             System.out.println("Gezippt");
90         }
91         else {
92             System.out.println(dirname + " konnte nicht gefunden werden");
93         }
94     }
95 }
96 catch(Exception e){
97     System.out.println("Exception: " + e);
98 }
99 }
100
101 /** Zippt ein Verzeichnis mit Hilfe eines Outputstreams
102  * @param zos: ZipOutputStream
103  * @param dir2zip: zu packendes Verzeichnis
104  */
105 public static void zipDir(String dir2zip, ZipOutputStream zos) {
106     try {
107         //create a new File object based on the directory we have to zip File
108         File zipDir = new File(dir2zip);
109         //get a listing of the directory content
110         String[] dirList = zipDir.list();
111         byte[] readBuffer = new byte[2156];
112         int bytesIn = 0;
113         //loop through dirList, and zip the files
114         for(int i=0; i<dirList.length; i++) {
115             File f = new File(zipDir, dirList[i]);

```

```

116         if(f.isDirectory()) {
117             //if the File object is a directory, call this
118             //function again to add its content recursively
119             String filePath = f.getPath();
120             zipDir(filePath, zos);
121             //loop again
122             continue;
123         }
124         //if we reached here, the File object f was not a directory
125         //create a FileInputStream on top of f
126         FileInputStream fis = new FileInputStream(f);
127         //create a new zip entry
128         ZipEntry anEntry = new ZipEntry(f.getPath());
129         //place the zip entry in the ZipOutputStream object
130         zos.putNextEntry(anEntry);
131         //now write the content of the file to the ZipOutputStream
132         while((bytesIn = fis.read(readBuffer)) != -1) {
133             zos.write(readBuffer, 0, bytesIn);
134         }
135         //close the Stream
136         fis.close();
137     }
138 }
139 catch(Exception e) {
140     System.out.println("Es ist ein Fehler aufgetreten:\n" + e);
141 }
142 }
143
144
145 /** Entpackt ein gezipptes Verzeichnis ins Zielverzeichnis
146  * @param zip: einzulesende Datei
147  * @param zip: einzulesender Pfad
148  * @return boolean: konnte entpackt werden? true = ja, false = nein
149  */
150 public static boolean unzip(String zip, String path){
151     if(path.length() == 1){
152         System.out.println("Benutzung: unzip <zipfile> <destination>");
153     }
154     else{
155         try {
156             ZipFile zipFile = new ZipFile(zip);
157             Enumeration<? extends ZipEntry> zipEntryEnum = zipFile.entries();
158
159             while(zipEntryEnum.hasMoreElements()){
160                 ZipEntry zipEntry = zipEntryEnum.nextElement();
161                 System.out.print(zipEntry.getName() + ".");
162                 extractEntry(zipFile, zipEntry, path);
163                 System.out.println("... entpackt");
164             }
165             return true;
166         }
167         catch(FileNotFoundException e){
168             System.err.println("Fehler: ZipFile nicht gefunden!");
169         }
170         catch(IOException e){
171             System.err.println("Fehler: Allgemeiner Ein-/Ausgabefehler!");
172         }
173     }
174     return false;
175 }

```



```
176
177  /** Entpackt den Eintrag ins Zielverzeichnis
178   * @param zf: zu entpackende Datei
179   * @param entry: zu entpackender Eintrag
180   * @param destDir: Zielverzeichnis
181   */
182  private static void extractEntry(ZipFile zf, ZipEntry entry, String destDir) throws IOException{
183      File file = new File(destDir, entry.getName());
184
185      if(entry.isDirectory()){
186          file.mkdirs();
187      }
188
189      else {
190          new File(file.getParent()).mkdirs();
191
192          InputStream is = null;
193          OutputStream os = null;
194
195          try {
196              is = zf.getInputStream(entry);
197              os = new FileOutputStream(file);
198
199              for (int len; (len = is.read(buffer)) != (-1); ){
200                  os.write(buffer, 0, len);
201              }
202          }
203          finally{
204              if (os != null) os.close();
205              if (is != null) is.close();
206          }
207      }
208  }
209
210  /** Loescht ein Verzeichniss mit seinen Unterverzeichnissen
211   * @param dir: zu loeschendes Verzeichnis
212   */
213  public static void deleteDir(File dir) {
214      File[] files = dir.listFiles();
215      if (files != null) {
216          for (int i = 0; i < files.length; i++) {
217              if (files[i].isDirectory()) {
218                  deleteDir(files[i]); // Verzeichnis leeren und anschliessend loeschen
219              }
220              else {
221                  files[i].delete(); // Datei loeschen
222              }
223          }
224          dir.delete(); // Ordner loeschen
225      }
226  }
227
228  /** Kopiert ein Verzeichnis in ein anders
229   * @param sourceLocation: Quellverzeichnis
230   * @param targetLocation: Zielverzeichnis
231   */
232  public static void copyDir(File sourceLocation, File targetLocation) throws IOException {
233
234      if (sourceLocation.isDirectory()) {
235          if (!targetLocation.exists()) {
```

```

236         targetLocation.mkdir();
237     }
238
239     String[] children = sourceLocation.list();
240     for (int i = 0; i < children.length; i++) {
241         copyDir(new File(sourceLocation, children[i]), new File(targetLocation, children[i]));
242     }
243 }
244 else {
245
246     InputStream in = new FileInputStream(sourceLocation);
247     OutputStream out = new FileOutputStream(targetLocation);
248     byte[] buf = new byte[1024];
249     int len;
250     while ((len = in.read(buf)) > 0) {
251         out.write(buf, 0, len);
252     }
253
254     in.close();
255     out.close();
256 }
257 }
258 }

```

Listing 9.2: Error.java

```

1 package net.naproche.preparser;
2 import java.util.LinkedList;
3
4 /**
5  * Containerclass for one Error, may be thought of like a "struct" in C-like languages.
6  * @author Julian Schloeder
7  * modified: Sebastian Zittermann
8  */
9
10 public class Error{
11     // Each error has a type (e.g. "inputError"), a syntactical position (e.g. "sentence"), start and end (position of
12     // leading and trailing character), content (Atom which triggered the error) and description (e.g. "Could not parse
13     // input.")
14
15     // As I said, it's essentially a struct, so every field is public. Live with it.
16     public String type;
17     public String position;
18     public int start;
19     public int end;
20     public String content;
21     public String description;
22
23     /**
24     * Constructor. Takes a String – starting with "message(error, ".
25     * @param inString
26     */
27     public Error(String inString){
28         String[] arr = new String[5], split = new String[3]; //there are five elements to each string: type, [pos,
29         //start,end], content, description and the static atom "error" in the beginning.
30         //When in a future version stuff like "warning" comes up (i.e. "message(warning,...") it will be arr[0] below
31         .
32         int arr_index=0, str_index=0, pairs=0;
33
34         // We iterate over each character. pairs==1 means we are inside of an atom.

```

```

31         // So when we find a comma and are not inside an atom that means we found a new argument
32         for (int i=0; i<inString.length(); i++){
33             if (inString.substring(i,i+1).equals(",") && pairs==0)
34                 pairs++;
35             else if (inString.substring(i,i+1).equals("'") && pairs==1)
36                 pairs--;
37             else if (inString.substring(i,i+1).equals(",") && pairs==0){
38                 // theres the comma(1) and theres a space (2), so we take everything in between the last comma
39                 // +2 and the current comma
40                 arr[arr_index]=inString.substring(str_index+2,i);
41                 str_index=i;
42                 arr_index++;
43             }
44             //do it once more in the end for the rest:
45             arr[arr_index] = inString.substring(str_index+2,inString.length());
46
47             split = arr[2].split(", ");
48
49             // trim everything (programmers and mathematicians are lazy...) and cut off various irrelevant characters (
50             // like ' or ,) with substring.
51             this.type = arr[1].trim();
52             this.position = split[0].trim().substring(1);
53             this.start = Integer.valueOf(split[1].trim());
54             this.end = Integer.valueOf(split[2].substring(0,split[2].length()-1).trim());
55             this.content = arr[3].trim();
56             this.description = arr[4].trim().substring(1,arr[4].trim().length()-2);
57         }
58     /**
59     * Constructs a String that should be exactly like the raw output of the error when called in swi-prolog
60     * @return String
61     */
62     public String toString(){
63         String retVal="message(error, ";
64         retVal = retVal + type+", '"+position+", "+start+", "+end+"', "+content+", '"+description+"')";
65         return retVal;
66     }
67
68     /** Takes the String which is returned by get_messages (the prolog-predicate) and returns a List containing the errors.
69     * DEBUG: This is a static utility-procedure and may be moved.
70     * @param inString
71     * @return LinkedList<Error>
72     */
73     public static LinkedList<Error> convertErrors(String inString){
74         LinkedList<String> temp = Sentence.convertDotNotation(inString);
75         LinkedList<Error> retVal = new LinkedList<Error>();
76         for (String error : temp)
77             retVal.add(new Error(error));
78         return retVal;
79     }
80 }

```

Listing 9.3: GUI.java

```

1 package net.naproche.GUI;
2
3 import java.awt.*;
4
5 import javax.swing.*;

```

```

6 import javax.swing.text.*;
7 import javax.swing.AbstractListModel;
8 import javax.swing.JFileChooser;
9 import javax.swing.JList;
10 import javax.swing.JMenuBar;
11 import javax.swing.JButton;
12 import javax.swing.JMenuItem;
13 import javax.swing.JScrollPane;
14 import javax.swing.SwingUtilities;
15 import javax.swing.JPopupMenu;
16
17 import java.awt.datatransfer.StringSelection;
18 import java.awt.event.ActionEvent;
19 import java.awt.event.ActionListener;
20 import java.awt.event.KeyEvent;
21 import java.awt.event.KeyListener;
22 import java.awt.event.MouseAdapter;
23 import java.awt.event.MouseEvent;
24 import java.awt.print.PageFormat;
25 import java.awt.print.PrinterJob;
26
27
28 import java.io.File;
29 import java.io.FileWriter;
30 import java.io.IOException;
31
32 import java.util.ArrayList;
33 import java.util.Arrays;
34 import java.util.Collections;
35 import java.util.LinkedList;
36 import java.util.List;
37
38 import tptpaxsel.*; // jar zur ueberpruefung von Ordnern
39
40 import jpl.Query;
41 import net.naproche.preparser.Error;
42 import net.naproche.preparser.*;
43
44 public class GUI
45 {
46     // GUI-Elemente
47     static JPopupMenu menu = new JPopupMenu("Popup");
48     static JPopupMenu plListmenu = new JPopupMenu("Popup");
49     JFrame f=new JFrame();
50     JFrame prs=new JFrame();
51     JFrame oblDetails=new JFrame();
52     JPanel oblpnl = new JPanel();
53     AttributedTextPane pane = new AttributedTextPane();
54     JTextPane txtPrologList = new JTextPane();
55     JButton btnClose= new JButton();
56     JButton btnExit= new JButton();
57     JButton btnPrologInput= new JButton();
58     JButton btnPrint= new JButton();
59     JButton btnDbg = new JButton();
60     JButton btnClearData= new JButton();
61     JButton btnlist= new JButton();
62     JButton btnCheckProof= new JButton();
63     JButton btnShowPrs= new JButton();
64     JButton oblbtnExit= new JButton();
65     JButton oblbtnSave= new JButton();

```

```

166 JButton oblbtnApply= new JButton();
167 JButton oblbtnCheck= new JButton();
168 JLabel obllabelTyp = new JLabel();
169 JLabel obllabelProver = new JLabel();
170 JLabel obllabelStatus = new JLabel();
171 JLabel obllabelBezStatus = new JLabel();
172 JLabel obllabelTime = new JLabel();
173 JLabel obllabelThreads = new JLabel();
174 JLabel obllabelWObl = new JLabel();
175 JLabel obllabelWAprils = new JLabel();
176 JLabel obllabelWNaproche = new JLabel();
177 JLabel obllabelgrOperator = new JLabel();
178 JLabel obllabelgrStartValue = new JLabel();
179 JLabel obllabelgrIncValue = new JLabel();
180 JLabel obllabelgrProver = new JLabel();
181 JLabel obllabelgrTimeStart = new JLabel();
182 JLabel obllabelgrTimeInc = new JLabel();
183 JLabel llabel = new JLabel();
184 JComboBox oblcbProver = new JComboBox();
185 JComboBox oblcbTyp = new JComboBox();
186 JComboBox oblcbgrOperation = new JComboBox();
187 JComboBox oblcbgrProver = new JComboBox();
188 JTextField obltfWObl = new JTextField();
189 JTextField obltfWAprils = new JTextField();
190 JTextField obltfWNaproche = new JTextField();
191 JTextField obltfProver = new JTextField();
192 JTextField obltfTime = new JTextField();
193 JTextField obltfThreads = new JTextField();
194 JTextField obltfgrStartValue = new JTextField();
195 JTextField obltfgrIncValue = new JTextField();
196 JTextField obltfgrTimeStartValue = new JTextField();
197 JTextField obltfgrTimeIncValue = new JTextField();
198 JScrollPane sp;
199     JScrollPane scroll = new JScrollPane(pane);
200     JScrollPane txtproll = new JScrollPane(txtPrologList);
201 JList liste;
202 JListModel model = new JListModel();
203
204 // Globale Variablen
205 int globalcaret = -1; // Position des Textcoursers
206 static int carposold = 99999999; // Alte Position des Textcoursers
207 boolean glshowObl = false; // wurde der Beweis bereits einmal ueberprueft? – fuer
    TPTPAxsel vorbereitet
208 boolean dbg = false; // Debugmodus
209 boolean isRunning = false; // fuer Threads
210
211 String proofname = "tmp"; // globale Variable fuer aktuellen Beweis
212 LinkedList<Sentence> globalSentences; // globale Variable mit allen Saetzen
213 LinkedList<Integer> globalIDS; // globale Variable mit allen zu ueberpruefenden SatzIDs
214 LinkedList<String> globalObligationName; // globale Variable mit allen zu ueberpruefenden Obligationsnamen
215 String OblName = new String(); // Fuer TPTP-AXsel
216 String glPreparseList = new String(); // Der vorverarbeitete Eingabetext in Form einer Prolog-Liste
217
218 // Farbwerte
219 Color glGreen = new Color(80,204,80);
220 Color glRed = new Color(205,0,0);
221 Color glOrange = new Color(254,158,0);
222 Color glGray = new Color(111,111,111);
223 Color glBlue = new Color(0,0,255);
224

```

```

125 // Standardbuttongroesse
126 int glbtnx = 1000;
127 int glbtny = 700;
128
129 /**
130  * Konstruktor der Klasse GUI
131  */
132 public GUI() {
133     super();
134     initialize();
135     pane.requestFocusInWindow();
136     pane.setCaretPosition(pane.getText().length());
137     // Abhaengig von PRS-Datei, die dargestellt werden soll.
138     if (new File (proofname + "/prs.html").exists()){
139         btnShowPrs.setEnabled(true);
140     }
141     else {
142         btnShowPrs.setEnabled(false);
143     }
144     btnShowPrs.setVisible(true);
145     btnCheckProof.setVisible(true);
146     btnCheckProof.setEnabled(true);
147     btnDbg.setVisible(true);
148     btnDbg.setEnabled(true);
149     btnExit.setVisible(true);
150     btnExit.setEnabled(true);
151     btnClearData.setVisible(false);
152     btnPrologInput.setVisible(false);
153     f.repaint();
154 }
155
156 /**
157  * Fuegt die Menueunterpunkte ueber die rechte Maustaste ein
158  * und startet die entsprechenden Methoden bei der Auswahl
159  * eines Unterpunktes
160  * @param s: Bezeichnung des Menueunterpunktes
161  * @param i: Position, an der der Punkt eingefuegt wird
162  */
163 void addItem(final String s, int i) {
164     JMenuItem item = new JMenuItem(s);
165     item.addActionListener(new ActionListener() {
166         public void actionPerformed(ActionEvent e) {
167             String tmp = e.toString();
168             if (tmp.indexOf("Exit") > 0){
169                 System.out.println("EXIT");
170                 System.exit(0);
171             }
172             else if (tmp.indexOf("show Obligations") > 0){
173                 showSentenceDetails();
174             }
175             else if (tmp.indexOf("Copy All") > 0){
176                 System.out.println("Copy All...");
177                 txtcopy();
178             }
179
180             else {
181                 System.out.println("Nichts bekanntes gewaehlt");
182             }
183         }
184     });

```

```

185         if (i == 0){
186             menu.add(item);
187         }
188         else {
189             plListmenu.add(item);
190         }
191     }
192
193     /**
194     * Bereitet den Inhalt des Textfeldes auf das preparieren vor
195     * und ruft dieses anschliessend auf und gibt es aus.
196     */
197     void showDetails() {
198         String inhalt = this.pane.getText();
199         inhalt = inhalt.replaceAll("\\n", "#");
200         inhalt = inhalt.replaceAll("\\\\", "!");
201         inhalt = inhalt.replaceAll("!", "\\\\\\\\\\\\\\\\\\");
202         try {
203             preparse(inhalt);
204         }
205         catch (Exception ex) {
206             String tmp = "Es ist etwas schiefgegangen:\n" + ex.toString();
207             System.out.println(tmp);
208         }
209     }
210
211     /**
212     * Anzeigen der Prologliste im Debug-Ausgabefeld
213     */
214     void showPrologList() {
215         //if (glPreparseList.length() > 1){
216         if (globalSentences.size() > 0){
217             scroll.setVisible(true);
218             scroll.setAutoscrolls(true);
219
220             //1. kommaposition bis erste eckigeKlammer zu + 1 loeschen, und fertig
221             String x = new String();
222             for (int i = 0; i < globalSentences.size(); i++){
223                 String tmp = globalSentences.get(i).toString();
224                 int poskom = tmp.indexOf(",");
225                 int posklm = tmp.indexOf("]") + 1;
226                 String erg = tmp.substring(0, poskom);
227                 erg = erg + tmp.substring(posklm, tmp.length());
228                 x = x + erg + ", ";
229             }
230             x = x.substring(0, x.length() - 2);
231
232             System.out.println("Show Prolog List: \n" + x.toString());
233             txtPrologList.setText(x.toString());
234             txtPrologList.setVisible(true);
235             txtproll.setVisible(true);
236             addItem("Copy All", 1);
237         }
238         else {
239             System.out.println("No Prolog-List...");
240             scroll.setVisible(true);
241             scroll.setAutoscrolls(true);
242         }
243     }
244 }

```

```

245
246 /**
247  * Kopiert den Inhalt des unteren Fensters in die Zwischenablage
248  */
249 void txtcopy(){
250     String cp = txtPrologList.getText();
251     StringSelection ss = new StringSelection(cp);
252     Toolkit.getDefaultToolkit().getSystemClipboard().setContents(ss, null);
253
254     System.out.println("Selection: " + cp.toString());
255 }
256
257 /**
258  * Ruft das Prologmodul zum Preparieren auf
259  * @param input: String, der geprepariert werden soll
260  */
261 public void prepare(String input){
262     //new Query("'src/prolog/load_jpl '.").oneSolution();
263     //new Query("asserta(output_folder(\"+proofname+\")).").oneSolution();// spaeter
264
265     Query create_naproche_input =
266         // new Query("create_naproche_input('Let $n$ be in $\mathbb{N}$. Then $n > 0$.',L).");
267         new Query("create_naproche_input(' " + input + "',L).");
268
269     String output;
270     try{
271         output = create_naproche_input.oneSolution().get("L").toString();
272         LinkedList<Sentence> ll = Sentence.convertSentences(output);
273         globalSentences = ll;
274
275         //Simulation der Ergebnis-Anzeige beim Start der GUI
276         String sentdata = new String();
277         for (int i = 0; i < ll.size(); i++){
278             sentdata = sentdata + ll.get(i).id + "#";
279             sentdata = sentdata + ll.get(i).start + "#";
280             sentdata = sentdata + ll.get(i).end + "#";
281
282             LinkedList<Word> ww = ll.get(i).content;
283             for (int k = 0; k < ww.size(); k++){
284                 sentdata = sentdata + ww.get(k).start + "&";
285                 sentdata = sentdata + ww.get(k).end + "&";
286                 sentdata = sentdata + ww.get(k).type + "&";
287                 sentdata = sentdata + ww.get(k).wordContent.toString() + "&";
288                 sentdata = sentdata + ww.get(k).mathContent.toString() + "#";
289             }
290             //sentdata = sentdata + "blue#"; // TBC: Test
291             sentdata = sentdata + "\n";
292
293         }
294         // schreibe Daten in Datei
295         saveText(new File(proofname + "/sentence.txt"), sentdata);
296     }
297     catch (Exception ex){
298         Query get_messages = new Query("get_messages(Messages)");
299         output = get_messages.oneSolution().get("Messages").toString();
300         System.out.println("Output: \n" + output);
301
302         // Wenn die Liste aus Prolog nicht leer ist
303         if (output.length() > 2){
304             LinkedList<Error> ll = Error.convertErrors(output);

```



```

305         System.out.println("Nicht OK: " + ll.get(0).toString());
306         int kkk = 0;
307         int start = ll.get(kkk).start;
308         int end = ll.get(kkk).end;
309         if (start == 0 && end == 0){
310             start = 0;
311             end = pane.getText().length();
312         }
313         // Keine Farbaenderung, sondern Markierung im Fehlerfall setzen!
314         //changeColor(ll.get(kkk).start, ll.get(kkk).end,new Color(255,0,0));
315         pane.setSelectionStart(start);
316         pane.setSelectionEnd(end);
317         System.out.println("Pos: " + ll.get(0).position);
318         System.out.println("Content: " + ll.get(0).content);
319         System.out.println("Discription: " + ll.get(0).description);
320     }
321     else {
322         System.out.println("grober Fehler bei der Java-Prolog-Umwandlung!");
323     }
324 }
325 }
326
327 /**
328  * Main: Erstellt die GUI
329  */
330 public static void main(String[] args){
331     GUI x = new GUI();
332 }
333
334 /**
335  * Interpretiert die Datei "sentence.txt" und setzt entsprechend die globalen Variablen
336  * @return boolean: true: beweis geladen, false: beweis nicht geladen
337  */
338 public boolean loadProof(){
339     String hui = new String();
340     //String pop = loadFileToString(new File(proofname + "/sentence.txt"));
341     String pop = FileOperations.loadFileToString(new File("tmp/sentence.txt"));
342     //System.out.println("Inhalt:\n" + pop + "");
343     if (pop.length() < 1) return false;
344     else {
345         String[] sentencesdata = pop.split("\n");
346
347         LinkedList<Sentence> lls = new LinkedList<Sentence>();
348
349         for(int h = 0; h < sentencesdata.length; h++){
350             String[] da = sentencesdata[h].split("#");
351             hui = hui + sentencesdata[h] + "\n";
352             // Werte fuer Konstruktor: i, s, e, content[sc, ec, typ, mcontent | wcontent]
353
354             int i = Integer.parseInt(da[0]);
355             int s = Integer.parseInt(da[1]);
356             int e = Integer.parseInt(da[2]);
357             String mhui = i + "\t" + s + "\t" + e + "\t";
358             LinkedList<Word> llw = new LinkedList<Word>();
359             // Bestimmung des Contents
360             for (int p = 3; p < da.length; p++){
361                 String rest = da[p];
362                 //System.out.println("Rest: " + rest + "\tzaehler: " + p + "\tLaenge: " + da.length);
363                 String []words = rest.split("&");
364                 int sc = Integer.parseInt(words[0]);

```

```

365         int ec = Integer.parseInt(words[1]);
366         String typ = words[2];
367         String mcontent = new String();
368         LinkedList<String> wcontent = new LinkedList<String>();
369         if (words.length > 3){
370             mcontent = words[3];
371             if (words.length > 4){
372                 String []wc = words[4].split(",");
373
374                 for(int b = 0; b < wc.length; b++){
375                     wcontent.add(wc[b]);
376                 }
377             }
378         }
379         // Wenn kein Wort vorhanden ist, erstelle auch keine Wortliste
380         else {
381             sc = -1;
382             break;
383         }
384
385         mhui = mhui + "{" + sc + " " + ec + " " + typ + " " + mcontent + " " +
            wcontent + "}";
386         if (sc == -1){
387             Word w = new Word(sc,ec,typ,mcontent, wcontent);
388             llw.add(w);
389         }
390     }
391
392     Sentence ss = new Sentence(i, s, e, llw);
393     lls.add(ss);
394     hui = hui + mhui + "\n";
395 }
396
397 txtPrologList.setText("Testausgabe: \n" + hui + "\n\n" + lls.toString() + "\nENDE Testausgabe");
398 // ENDE umwandlung der Datei in die Linkedlist
399 globalSentences = lls;
400 setGlobalIdObl();
401 return true;
402 }
403 }
404
405 /**
406  * Initialisierungsmethode:
407  * Baut alle Elemente des Hauptfensters, des Menues und der rechten Maustaste auf
408  */
409 private void initialize() {
410     globalSentences = new LinkedList<Sentence>();
411     globalIDS = new LinkedList<Integer>();
412     globalObligationName = new LinkedList<String>();
413
414     makeMenu();
415
416     // rechte Maustasten-Menu
417     addItem("Exit", 0);
418
419     f.setBounds(new Rectangle(0, 0, 1200, 800)); // Aufloesung lanfristig 1280 x 720
420     f.setMaximumSize(new Dimension(1200,800));
421     f.setMinimumSize(new Dimension(1200,800));
422     f.setResizable(false);
423 }

```

```

424     f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
425     f.setVisible(true);
426     f.setLayout(null);
427     f.setTitle("Naproche-Interface");
428
429     llabel.setText("Obligationen zu Satz x");
430     llabel.setVisible(true);
431     llabel.setBounds(10, 5, 200, 15);
432     oblpnl.add(llabel);
433
434     liste = new JList(model);
435     sp = new JScrollPane(liste);
436     sp.setVisible(true);
437     sp.setBounds(10, 30, 175, 110);
438     oblpnl.add(sp);
439
440     btnlist.setText("Details");
441     btnlist.addActionListener(new java.awt.event.ActionListener() {
442         public void actionPerformed(java.awt.event.ActionEvent e) {
443             int pl = liste.getSelectedIndex();
444             if (pl > -1){
445                 System.out.println("Ausgewählt: " + pl+ " Name: " + model.getElementAt(pl));
446             }
447             OblName = model.getElementAt(pl).toString();
448             showSettings(pl,model.getElementAt(pl).toString());
449         }
450     });
451     btnlist.setVisible(true);
452     btnlist.setBounds(10, 150, 175, 30);
453     oblpnl.add(btnlist);
454
455     oblpnl.setBounds(glbtnx-10, 180, 200, 200);
456     oblpnl.setVisible(false);
457     oblpnl.setBackground(glGray);
458     oblpnl.setLayout(null);
459     f.add(oblpnl);
460
461     btnDbg.setText("Debugmodus off");
462     btnDbg.setBackground(glRed);
463     btnDbg.setBounds(15, glbtny, 175, 30);
464     btnDbg.addActionListener(new java.awt.event.ActionListener() {
465         public void actionPerformed(java.awt.event.ActionEvent e) {
466             if (dbgm == true){
467                 btnDbg.setText("Debugmodus off");
468                 btnDbg.setBackground(glRed);
469                 txtproll.setVisible(false);
470                 txtproll.setBounds(15, 800, pane.getWidth(), 100);
471                 btnPrologInput.setVisible(false);
472                 btnClearData.setVisible(false);
473                 f.setBounds(f.getX(), f.getY(), f.getWidth(), 900);
474                 dbgm = false;
475             }
476             else {
477                 btnDbg.setText("Debugmodus on");
478                 btnDbg.setBackground(glGreen);
479                 txtproll.setBounds(15, 740, pane.getWidth(), 100);
480                 txtproll.setVisible(true);
481                 btnPrologInput.setVisible(true);
482                 btnClearData.setVisible(true);
483                 f.setBounds(f.getX(), f.getY(), f.getWidth(), 900);

```

```

484         dbgmn = true;
485     }
486
487     }
488     });
489     btnDbg.setVisible(true);
490     f.add(btnDbg);
491
492     btnExit.setText("Exit");
493     btnExit.setBounds(glbtnx, glbtny, 175, 30);
494     btnExit.addActionListener(new java.awt.event.ActionListener() {
495         public void actionPerformed(java.awt.event.ActionEvent e) {
496             showInDebugTxt("Programm wird beendet");
497             exit();
498         }
499     });
500     f.add(btnExit);
501
502     btnPrint.setText("Print");
503     btnPrint.setVisible(true);
504     btnPrint.setBounds(glbtnx, glbtny-60, 175, 30);
505     btnPrint.addActionListener(new java.awt.event.ActionListener() {
506         public void actionPerformed(java.awt.event.ActionEvent e) {
507             printTextfield();
508             showInDebugTxt("Inhalt des Feldes wurde gedruckt");
509         }
510     });
511     f.add(btnPrint);
512
513     btnPrologInput.setText("Prolog-Input");
514     btnPrologInput.setVisible(false);
515     btnPrologInput.setBounds(225, glbtny, 175, 30);
516     btnPrologInput.addActionListener(new java.awt.event.ActionListener() {
517         public void actionPerformed(java.awt.event.ActionEvent e) {
518             for (int i = 1; i < 58; i++){
519                 System.out.println("\\item " + i + ".nap");
520             }
521         }
522     });
523     f.add(btnPrologInput);
524
525     txtPrologList.addMouseListener(new MouseAdapter() {
526     public void mousePressed(MouseEvent e) {
527         if(e.isPopupTrigger()){
528             // Setzen des Cursors
529             int offset = txtPrologList.viewToModel( e.getPoint() );
530             globalcaret = offset;
531             txtPrologList.setCaretPosition(offset);
532
533             // Anzeigen des Popup-Menus
534             plListmenu.show(e.getComponent(),e.getX(),e.getY());
535         }
536     }
537     public void mouseReleased(MouseEvent e) {
538         if(e.isPopupTrigger()){
539             // Setzen des Cursors
540             int offset = pane.viewToModel( e.getPoint() );
541             globalcaret = offset;
542             txtPrologList.setCaretPosition(offset);
543

```

```

544         // Anzeigen des Popup-Menus
545         plListmenu.show(e.getComponent(), e.getX(), e.getY());
546     }
547 }
548 });
549 f.add(txtproll);
550
551 btnClearData.setText("Clear " + proofname);
552 btnClearData.setBounds(435, glbtiny, 175, 30);
553 btnClearData.setVisible(false);
554 btnClearData.addActionListener(new java.awt.event.ActionListener() {
555     public void actionPerformed(java.awt.event.ActionEvent e) {
556         System.out.println("Testergebnisse zu " + proofname + " werden gelöscht!");
557         clearDir(proofname);
558         File f = new File(proofname);
559         f.mkdir();
560         saveText(new File(proofname + "/" + proofname + ".txt"), pane.getText());
561         showInDebugTxt("Ordner " + proofname + " wurde gelöscht!");
562         pane.setColor(0, pane.getText().length(), new Color(0,0,0));
563     }
564 });
565 f.add(btnClearData);
566
567 btnCheckProof.setText("Check");
568 btnCheckProof.setBounds(glbtinx, 15, 175, 30);
569 btnCheckProof.setVisible(true);
570 btnCheckProof.addActionListener(new java.awt.event.ActionListener() {
571     public void actionPerformed(java.awt.event.ActionEvent e) {
572         System.out.println("Check!");
573         showInDebugTxt("Check!");
574         checkProof();
575     }
576 });
577 f.add(btnCheckProof);
578
579 btnShowPrs.setText("Show PRS");
580 btnShowPrs.setBounds(glbtinx, 80, 175, 30);
581 btnShowPrs.setEnabled(true);
582 btnShowPrs.setVisible(true);
583 btnShowPrs.addActionListener(new java.awt.event.ActionListener() {
584     public void actionPerformed(java.awt.event.ActionEvent e) {
585         showPrs();
586         showInDebugTxt("PRS in Browser angezeigt!");
587     }
588 });
589 f.add(btnShowPrs);
590
591 // Keylistener fuers Textfeld
592 KeyListener kl = new KeyListener() {
593     public void keyPressed(KeyEvent keyEvent) {
594     }
595
596     public void keyReleased(KeyEvent keyEvent) {
597     }
598
599     public void keyTyped(KeyEvent keyEvent) {
600     }
601     if (keyEvent.getKeyCode() > 0 && keyEvent.getKeyCode() < 48){
602     }
603     else {

```

```

604         if (glshowObl == true){
605             int cposType = pane.getCaretPosition();
606             int cpos = pane.getCaretPosition()+1;
607             // cpos muss den Startwert des Satzes haben.
608             int z = 0;
609             while (cpos > globalSentences.get(z).end && z < globalSentences.size()-1){
610                 z++;
611                 if (z > globalSentences.size()){
612                     z--;
613                     break;
614                 }
615             }
616             cpos = globalSentences.get(z).start;
617             int limit = globalSentences.getLast().end;
618             int max = pane.getText().length();
619
620             if (cposType >= limit){
621                 pane.changeColor(cposType, (max-cposType), new Color(0,0,255));
622                 pane.setCaretPosition(cposType);
623             }
624             else if (cposType < cpos){
625                 carposold = cpos;
626                 pane.changeColor(cposType-1, (max-cposType+1), new Color(0,0,255));
627                 pane.setCaretPosition(cposType);
628             }
629             else if (cpos < carposold){
630                 carposold = cpos;
631                 pane.changeColor(cpos, (max - cpos), new Color(0,0,255));
632                 pane.setCaretPosition(cposType);
633             }
634         }
635     }
636 }
637 };
638 pane.addKeyListener(kl);
639
640 pane.addMouseListener(new MouseAdapter() {
641     public void mousePressed(MouseEvent e) {
642         if(e.isPopupTrigger()){
643             // Setzen des Cursors
644             int offset = pane.viewToModel( e.getPoint() );
645             globalcaret = offset;
646             pane.setCaretPosition(offset);
647
648             // Anzeigen des Popup-Menus
649             menu.show(e.getComponent(), e.getX(), e.getY());
650         }
651     }
652 }
653
654 public void mouseReleased(MouseEvent e) {
655     if(e.isPopupTrigger()){
656         // Setzen des Cursors
657         int offset = pane.viewToModel( e.getPoint() );
658         globalcaret = offset;
659         pane.setCaretPosition(offset);
660
661         // Anzeigen des Popup-Menus
662         menu.show(e.getComponent(), e.getX(), e.getY());
663     }
664 }

```

```

664    });
665    pane.setBounds(15,10,950,680);
666    scroll.setBounds(pane.getX(), pane.getY(), pane.getWidth(), pane.getHeight());
667    scroll.setVisible(true);
668    scroll.setAutoscrolls(true);
669    f.add(scroll);
670
671    txtPrologList.setText(proofname);
672    File ftmp = new File(proofname + "/" + proofname + ".txt");
673    if (ftmp.exists()){
674        showTextinPane(ftmp);
675        globalObligationName.add("");
676    }
677
678    // fuer die Comboboxen der Settings:
679    oblcbProver.addItem("Vampire");
680    oblcbProver.addItem("E");
681    oblcbgrProver.addItem("Vampire");
682    oblcbgrProver.addItem("E");
683    oblcbgrProver.addItem("Vampire & E");
684    oblcbgrProver.addItem("E & Vampire");
685    oblcbgrOperation.addItem("+");
686    oblcbgrOperation.addItem("*");
687    oblcbTyp.addItem("Simple");
688    oblcbTyp.addItem("Growth");
689
690    // Rest der Settings-Objekte:
691    oblDetails.add(oblLabelTime);
692    oblDetails.add(oblTfTime);
693    oblDetails.add(oblLabelThreads);
694    oblDetails.add(oblTfThreads);
695    oblDetails.add(oblLabelProver);
696    oblDetails.add(oblcbProver);
697    oblDetails.add(oblLabelWAprils);
698    oblDetails.add(oblTfWAprils);
699    oblDetails.add(oblLabelWObl);
700    oblDetails.add(oblTfWObl);
701    oblDetails.add(oblLabelWNaproche);
702    oblDetails.add(oblTfWNaproche);
703    oblDetails.add(oblLabelgrStartValue);
704    oblDetails.add(oblTfgrStartValue);
705    oblDetails.add(oblLabelgrIncValue);
706    oblDetails.add(oblTfgrIncValue);
707    oblDetails.add(oblLabelgrProver);
708    oblDetails.add(oblcbgrProver);
709    oblDetails.add(oblLabelgrTimeStart);
710    oblDetails.add(oblTfgrTimeStartValue);
711    oblDetails.add(oblLabelgrTimeInc);
712    oblDetails.add(oblTfgrTimeIncValue);
713    oblDetails.add(oblLabelgrOperator);
714    oblDetails.add(oblcbgrOperation);
715
716    oblDetails.add(oblLabelBezStatus);
717    oblDetails.add(oblLabelStatus);
718    oblDetails.add(oblbtnSave);
719    oblDetails.add(oblLabelTyp);
720    oblDetails.add(oblcbTyp);
721    oblDetails.add(oblbtnExit);
722    oblDetails.add(oblbtnCheck);
723    oblbtnCheck.setText("Check again");

```

```

724
725     oblbtnCheck.setBounds(100,220,100,30);
726     oblbtnCheck.addActionListener(new java.awt.event.ActionListener() {
727         public void actionPerformed(java.awt.event.ActionEvent e) {
728             System.out.println("Check Obligations!");
729             String finput = OblName + ".input";
730             String fresult = OblName + ".result";
731             LinkedList<String> eee = checkSettings();
732             int sentID = Integer.parseInt(OblName.substring(0,OblName.indexOf("-")));
733
734             // Zuweisung des korrekten Pfades
735             final String ordner = proofname + "/" + sentID + "/";
736
737             final int start = globalSentences.get(sentID-1).start;
738             final int end = globalSentences.get(sentID-1).end;
739
740             if (eee.size() > 0){
741                 System.out.println("Folgende Werte muessen ueberprueft werden: \n" + eee.toString());
742             }
743             else {
744                 final File obl = new File(ordner + "obligation.txt");
745                 if (obl.exists()){
746                     System.out.println("Obligationsdatei muss zuerst geloescht werden: " + obl);
747                     obl.delete();
748                     saveText(obl,finput);
749                     System.out.println("Inhalt der Obligations-Datei: " + FileOperations.loadFileToString(
750                         obl));
751                 }
752                 else {
753                     System.out.println("Obligationsdatei muss erstellt werden: " + obl);
754                     saveText(obl,finput);
755                 }
756
757                 System.out.println("Alle Werte in den Textfeldern wurden akzeptiert!");
758                 int pr = oblcbProver.getSelectedIndex();
759                 String g = new String();
760                 if (pr == 0){
761                     g = "V";
762                 }
763                 else {
764                     g = "E";
765                 }
766                 final String fpr = "" + g + "";
767
768                 String ttt = ordner;
769                 AtpApi aaa = new AtpApi(ttt, ttt + "obligation.txt");
770                 aaa.setSimple(getDouble(obltfWObl.getText()), getDouble(obltfWAprils.getText()),
771                     getDouble(obltfWNaproeche.getText()), fpr, getInt(obltfTime.getText()), getInt(
772                     obltfThreads.getText()));
773
774                 aaa.runSingleByFilename(ttt + finput);
775                 oblStatusLabel.setForeground(checkResult(ordner + fresult));
776                 if (oblStatusLabel.setForeground() == glRed){
777                     oblStatusLabel.setText("#NOPROOF");
778                     pane.changeColor(start, end-start, glRed);
779                     f.setVisible(true);
780                     System.out.println("Check-Ergebnis: Satz " + sentID + " in der Farbe " + glRed + "
781                         eingeferbt");
782                 }
783                 else if (oblStatusLabel.setForeground() == glOrange){
784                     oblStatusLabel.setText("#Warning");
785                 }
786             }
787         }
788     }
789 }

```



```

781         pane.changeColor(start, end-start, glOrange);
782         System.out.println("Check-Ergebnis: Satz " + sentID + " in der Farbe " + glOrange + "
           eingefärbt");
783     }
784     else if (obllabelStatus.getForeground() == glGreen){
785         obllabelStatus.setText("PROOF");
786         pane.changeColor(start, end-start, glGreen);
787         f.setVisible(true);
788         System.out.println("Check-Ergebnis: Satz " + sentID + " in der Farbe " + glGreen + "
           eingefärbt");
789     }
790     else {
791         obllabelStatus.setText("???");
792     }
793     oblplnl.setVisible(false);
794 }
795 }
796 });
797 oblDetails.add(oblbtnApply);
798
799
800     new Query("[ 'naproche' ] ").oneSolution();
801     loadProof();
802     // Einfärben
803     colorAllSentencens(); // 190 & 958
804
805 }
806
807 /**
808  * Ausgabe eines Strings im Debug-Unterfenster
809  * @param h: Auszugebender String
810  */
811 void showInDebugTxt(String h){
812     txtproll.setBounds(15, 740, pane.getWidth(), 100);
813     f.setBounds(f.getX(), f.getY(), f.getWidth(), 900);
814     btnExit.setBounds(btnExit.getX(), f.getHeight()-92, btnExit.getWidth(), btnExit.getHeight());
815     btnPrint.setBounds(btnExit.getX(), f.getHeight()-160, btnExit.getWidth(), btnExit.getHeight());
816     txtproll.setVisible(true);
817     txtPrologList.setText(h.toString());
818 }
819
820 /**
821  * Methode zu PRS-Darstellung
822  */
823 void showPrs(){
824     File fprs = new File(proofname + "/prs.html");
825
826     if (fprs.exists()){
827         showBrowser(fprs);
828         System.out.println("PRS angezeigt: " + fprs.toString());
829         showInDebugTxt("PRS angezeigt: " + fprs.toString());
830     }
831     else {
832         showInDebugTxt("PRS-Datei wird erzeugt...");
833         new Query("make_super_prs.").oneSolution();
834         fprs = new File(proofname + "/prs.html");
835         showBrowser(fprs);
836         showInDebugTxt("PRS wird im Browser angezeigt.");
837     }
838 }

```

```

839     }
840
841     /**
842     * Hauptmethode zur Beweisueberpruefung
843     */
844     void checkProof() {
845         final long zstVorher;
846         zstVorher = System.currentTimeMillis();
847
848         showInDebugTxt("ueberpruefung des Beweises laeuft...");
849         // Speichern des aktuellen Beweisinhaltes
850         saveText(new File (proofname + "/" + proofname + ".txt"), pane.getText());
851
852         // Erstellen eines Threads zum Einfärben während das Prologmodul rechnet
853         final Thread Color = new Thread() {
854             LinkedList <String> ordner;
855
856             public void run() {
857
858                 System.out.println("Start des Color-Threads");
859                 ordner = checkFolder(proofname);
860                 System.out.println("Anzahl der Ordner: " + ordner.size() + "\nInhalt:");
861                 System.out.println("#####");
862
863                 for (int i = 0; i < ordner.size(); i++){
864                     System.out.println(ordner.get(i));
865                 }
866
867                 System.out.println("#####");
868                 try {
869                     while(isRunning == true){
870                         Thread.sleep(1500); // 1.5 Sekunden
871
872                         // geht noch effizienter, aber so gehts halt:
873                         setGlobalIdObl();
874                         colorAllSentences();
875                     }
876                 } catch (InterruptedException ex) {
877                     System.out.println("Es ist ein Fehler aufgetreten: \n" + ex);
878                 }
879
880                 //Abbruchkriterien
881                 SwingUtilities.invokeLater(new Runnable() {
882                     public void run() {
883                         System.out.println("Ende Color-Thread");
884
885                         //colorFromFile();
886                     }
887                 });
888             }
889         }; // Ende Color
890
891         Thread Parser = new Thread() {
892
893             public void run() {
894                 System.out.println("Start des Parser-Threads");
895                 btnShowPrs.setEnabled(false);
896                 isRunning = true;
897                 Color.start(); // test
898                 startProofChecker(); // startet Parser und initialisiert globale Variablen

```

```

899         Color.interrupt();
900         isRunning = false;
901         System.out.println("----- Microparser ist fertig -----");
902         btnShowPrs.setEnabled(true);
903
904         SwingUtilities.invokeLater(new Runnable() {
905             public void run() {
906                 System.out.println("Ende Parser-Thread");
907                 long zstNachher;
908                 zstNachher = System.currentTimeMillis();
909                 System.out.println("Zeit benoetigt: " + ((zstNachher - zstVorher)/1000) + " sec");
910                 showInDebugTxt("... ueberpruefung des Beweises abgeschlossen. \n Berechnungszeit: " +
911                     ((zstNachher - zstVorher)/1000) + "." + (((zstNachher - zstVorher)/10) - 100 *
912                     ((zstNachher - zstVorher)/1000)) + " Sekunden");
913                 String fff = ("... ueberpruefung des Beweises abgeschlossen. \n Berechnungszeit: " +
914                     ((zstNachher - zstVorher)/1000) + "." + (((zstNachher - zstVorher)/10) - 100 *
915                     ((zstNachher - zstVorher)/1000)) + " Sekunden");
916                 fff = fff + "\nLetzte ID: \t" + globalIDS.getLast();
917                 saveText(new File (proofname + "/" + globalIDS.getLast() + ".txt"), fff);
918                 isRunning = false;
919             }
920         });
921     }
922
923     }; // Ende Parser
924
925     Parser.start();
926 }
927
928 /**
929  * Erstellt eine String-Liste, die die Verzeichnisstruktur eines
930  * uebergeben Ordernamen beinhaltet
931  * @param Fname: zu untersuchende Ordnerstruktur
932  * @return LinkedList <String>: Ordnerstruktur
933  */
934 LinkedList <String> checkFolder(String Fname){
935     String[] tmp;
936     File dir = new File(Fname);
937     LinkedList <String> erg = new LinkedList <String>();
938
939     if (dir.isDirectory()){
940         tmp = dir.list();
941         int z = 0;
942         for (int i = 0; i < tmp.length; i++){
943             String x = Fname + "/" + tmp[i];
944             if (new File (x).isDirectory()){
945                 z++;
946                 erg.add(x);
947             }
948         }
949         return erg;
950     }
951     else return null;
952 }
953
954 /**
955  * Faerbt alle Saetze ein
956  */
957 void colorAllSentencens(){
958     String tmp = new String();
959     for (int i = 0; i < globalSentences.size(); i++){

```

```

956         tmp = tmp + globalSentences.get(i).start + "!" + globalSentences.get(i).end + "!";
957         Color c = checkSentence(i+1);
958         if (c == glBlue){
959             tmp = tmp + "5";
960         }
961         else if (c == glRed){
962             tmp = tmp + "1";
963         }
964         else if (c == glOrange){
965             tmp = tmp + "2";
966         }
967         else if (c == glGreen){
968             tmp = tmp + "3";
969         }
970         else if (c == glGray){
971             tmp = tmp + "4";
972         }
973         tmp = tmp + "\n";
974         pane.requestFocusInWindow();
975         pane.setCaretPosition(pane.getText().length());
976     }
977     saveText(new File(proofname + "/ColorKoord.txt"), tmp);
978 }
979
980 /**
981  * Faerbt den Textinhalt anhand der Daten in der Datei ColorKoord.txt ein
982  */
983 void colorFromFile(){
984     String data = FileOperations.loadFileToString(new File(proofname + "/ColorKoord.txt"));
985     String[] sent = data.split("\n");
986     for (int i = 0; i < sent.length; i++){
987         String[] x = sent[i].split("!");
988         int start = Integer.parseInt(x[0]);
989         int end = Integer.parseInt(x[1]);
990         int color = Integer.parseInt(x[2]);
991         Color c = new Color(0,0,0);
992         if (color == 1){
993             c = glRed;
994         }
995         else if (color == 2){
996             c = glOrange;
997         }
998         else if (color == 3){
999             c = glGreen;
1000         }
1001         else if (color == 4){
1002             c = glGray;
1003         }
1004         else if (color == 5){
1005             c = glBlue;
1006         }
1007         System.out.println("Satz: " + i + "\tFarbwert: " + c);
1008         pane.changeColor(start, end-start, c);
1009     }
1010 }
1011
1012 /**
1013  * Methode zum Drucken des Inhaltes des Textfelds
1014  */
1015 public void printTextfield(){

```

```

1016     System.out.println("Druckauftrag wird vorbereitet...");
1017
1018     PrintTool pt = new PrintTool(pane.getText(), null, pane.getFont(), proofname, false);
1019     System.out.println("PrintTool initialisiert...");
1020
1021     pt.printAllPages();
1022     System.out.println("Alle Seiten wurden gedruckt!");
1023 }
1024
1025 /**
1026  * Methode zum erstellen des Druckdialog-Fensters
1027  */
1028 public boolean setupDialogs(PageFormat pfUse, PrinterJob prjob){
1029     PageFormat pfDflt = pfUse;
1030     pfUse = prjob.pageDialog( pfDflt );
1031     return ( pfUse == pfDflt ) ? false : prjob.printDialog();
1032 }
1033
1034
1035 /**
1036  * TBC: Methode fuer die Anbindung an TPTP-AXsel
1037  */
1038 public void showSentenceDetails(){
1039     int pos = globalcaret;
1040     int i = -1;
1041     do {
1042         i++;
1043     }while (globalSentences.get(i).end < pos && i < globalSentences.size()-1);
1044     String satz = pane.getText().substring(globalSentences.get(i).start, globalSentences.get(i).end);
1045     int satzID = i + 1; // zum Abgleich mit der von Prologerstellten IDs der Saetze (faengt da bei 1 an, nicht bei 0)
1046     System.out.println(satzID + ". Satz: " + satz);
1047     pane.select(globalSentences.get(i).start, globalSentences.get(i).end);
1048     if (globalIDS.indexOf(satzID) > -1){
1049         System.out.println(satzID + ". Satz enthaelt mindestens eine Abfrage!");
1050         // Setzen der benoetigten Felder auf "sichtbar"
1051         llabel.setText("Obligationen zu Satz " + satzID);
1052         oblpnl.setVisible(true);
1053
1054         int lsize = model.getSize();
1055
1056         // loeschen der alten Liste
1057         if (lsize > 0){
1058             for (int j = 0; j < lsize; j++){
1059                 model.removeElement(0);
1060             }
1061         }
1062
1063         String iok = satzID + "-";
1064         for (int k = 0; k < globalObligationName.size(); k++){
1065             String gon = globalObligationName.get(k).toString();
1066             if (gon.indexOf(iok)>-1){
1067                 model.addElement(gon);
1068             }
1069         }
1070         liste.setSelectedIndex(0);
1071     }
1072     else {
1073         llabel.setText("Keine Obl. \nin Satz " + satzID);
1074         llabel.setVisible(true);
1075         sp.setVisible(false);
1076         btnlist.setVisible(
1077             false);

```

```

1075     }
1076
1077 }
1078
1079
1080 /**
1081  * Methode zum Einfärben eines Satzes
1082  * @param proofErg: Liste mit allen Beweisschritten des Satzes
1083  * @param id: id des Satzes
1084  */
1085 public void colorSentence(LinkedList<String> proofErg, int id){
1086     Color c = new Color(0,0,0);
1087     if (proofErg != null){
1088
1089         for (int i = 0; i < proofErg.size(); i++){
1090             String tmp = proofErg.get(i);
1091
1092             if (tmp.indexOf(";") > 0){
1093                 String[] x = tmp.split(";");
1094                 if (x[1].equals("false")){
1095                     c = glRed;
1096                 }
1097                 else {
1098                     if (x[1].equals("true")){
1099                         if (x[2].equals("true")){
1100                             c = glOrange;
1101                         }
1102                         else {
1103                             c = glGreen;
1104                         }
1105                     }
1106                 }
1107             }
1108             else {
1109                 c = Color.pink; // soll nie kommen ; )
1110             }
1111         }
1112     }
1113     else {
1114         c = new Color(110,110,110);
1115     }
1116     // id faengt bei 1 an, Sentences bei 0.
1117     int start = globalSentences.get(id-1).start;
1118     int end = globalSentences.get(id-1).end;
1119     pane.changeColor(start, end-start, c);
1120 }
1121
1122
1123 /**
1124  * Erstellt Stringliste mit allen Dateinamen
1125  * @param dir: zu untersuchendes Verzeichnis
1126  * @return String[]: Stringliste mit Verzeichnisnamen
1127  */
1128 public String[] listDir(File dir) {
1129
1130     File[] files = dir.listFiles();
1131     String[] tmp = new String[files.length];
1132     int j = 0;
1133     if (files != null) { // Erforderliche Berechtigungen etc. sind vorhanden
1134         for (int i = 1; i < files.length; i++) {

```

```

1135         if ( files[i].isFile() ) {
1136             if ( files[i].getName() != null && files[i].getName() != "null"){
1137                 tmp[j] = files[i].getName();
1138                 j++;
1139             }
1140         }
1141     }
1142 }
1143 String[] erg = new String[j];
1144 for (int i = 0; i < j; i++) {
1145     erg[i] = tmp[i];
1146 }
1147 j = 0;
1148 return erg;
1149 }
1150
1151 /**
1152  * Anzeige einer Datei im Browser (fuer PRS)
1153  * @param file: anzuzeigende Datei
1154  */
1155 void showBrowser(File file){
1156     try {
1157         Desktop.getDesktop().browse( file.toURI() );
1158     }
1159     catch ( Exception e ){
1160         e.printStackTrace();
1161     }
1162 }
1163
1164 /**
1165  * TBC: Settings pro Abfrage einstellen – TPTP–AxSel
1166  * @param oblID: ID der Obligation
1167  * @param fname: Dateiname der Obligation
1168  */
1169 void showSettings(int oblID, String fname){
1170     int sentID = Integer.parseInt(fname.substring(0,fname.indexOf("-")));
1171     String tmp = "Obligation im "+ (sentID) +". Satz :"+ fname + " ";
1172
1173     final String ordner = proofname + "/" + sentID + "/";
1174
1175     final int start = globalSentences.get(sentID-1).start;
1176     final int end = globalSentences.get(sentID-1).end;
1177
1178     String typ = oblcbTyp.getSelectedItem().toString();
1179     String inhalt = "simple;1.0;0.0;1.0;\r\n";
1180     String[] data = inhalt.split(";");
1181
1182     // Einlesen einer evtl vorhandenen spezifischen Settingsdatei
1183     final File fsetx = new File(ordner + fname + ".set");
1184
1185     if (fsetx.exists()){
1186         inhalt = FileOperations.loadFileToString(fsetx);
1187         data = inhalt.split(";");
1188     }
1189     else {
1190         System.out.println("keine .set-Datei da, deswegen Standartwerte");
1191     }
1192
1193     final String fresult = fname + ".result";
1194     final File obl = new File(ordner + "obligation.txt");

```

```

1195     if (obl.exists()) {
1196         obl.delete();
1197         saveText(obl, fname + ".input");
1198     }
1199     else {
1200         //System.out.println("???");
1201         saveText(obl, fname + ".input");
1202     }
1203
1204     oblDetails.setBounds(new Rectangle(400, 300, 600, 300));
1205     oblDetails.setLayout(null);
1206     oblDetails.setVisible(true);
1207     oblDetails.setTitle(tmp);
1208
1209     // Simple
1210     obllabelTime.setText("Max Time per Obligation: ");
1211     obllabelTime.setVisible(true);
1212     obllabelTime.setBounds(20, 60, 250, 30);
1213
1214     obltime.setText(data[5].trim());
1215     obltime.setVisible(true);
1216     obltime.setBounds(220, 60, 50, 30);
1217
1218     obllabelThreads.setText("Threads: ");
1219     obllabelThreads.setVisible(true);
1220     obllabelThreads.setBounds(20, 110, 250, 30);
1221
1222     oblthreads.setText(data[6].trim());
1223     oblthreads.setVisible(true);
1224     oblthreads.setBounds(220, 110, 50, 30);
1225
1226     obllabelProver.setText("Select Prover: ");
1227     obllabelProver.setVisible(true);
1228     obllabelProver.setBounds(20, 160, 250, 30);
1229
1230     oblcbProver.setSelectedItem(data[4].substring(1, data[4].length() - 1));
1231     oblcbProver.setVisible(true);
1232     oblcbProver.setBounds(180, 160, 90, 30);
1233
1234     obllabelWAprils.setText("Weight Aprils: ");
1235     obllabelWAprils.setVisible(true);
1236     obllabelWAprils.setBounds(320, 60, 250, 30);
1237
1238     obltimeWAprils.setText(data[2].trim());
1239     obltimeWAprils.setVisible(true);
1240     obltimeWAprils.setBounds(520, 60, 50, 30);
1241
1242     obllabelWObl.setText("Weight Obligations: ");
1243     obllabelWObl.setVisible(true);
1244     obllabelWObl.setBounds(320, 110, 250, 30);
1245
1246     obltimeWObl.setText(data[1].trim());
1247     obltimeWObl.setVisible(true);
1248     obltimeWObl.setBounds(520, 110, 50, 30);
1249
1250     obllabelWNa proche.setText("Weight Na proche: ");
1251     obllabelWNa proche.setVisible(true);
1252     obllabelWNa proche.setBounds(320, 160, 250, 30);
1253
1254     obltimeWNa proche.setText(data[3].trim());

```



```

1255 obltfWNAproche.setVisible(true);
1256 obltfWNAproche.setBounds(520,160,50,30);
1257
1258 // Growth
1259 obllabelgrStartValue.setText("Start Value: ");
1260 obllabelgrStartValue.setVisible(false);
1261 obllabelgrStartValue.setBounds(20,60,250,30);
1262
1263 obltfgrStartValue.setText(data[8].trim());
1264 obltfgrStartValue.setVisible(false);
1265 obltfgrStartValue.setBounds(220,60,50,30);
1266
1267 obllabelgrIncValue.setText("Inc. Value: ");
1268 obllabelgrIncValue.setVisible(false);
1269 obllabelgrIncValue.setBounds(20,110,250,30);
1270
1271 obltfgrIncValue.setText(data[9].trim());
1272 obltfgrIncValue.setVisible(false);
1273 obltfgrIncValue.setBounds(220,110,50,30);
1274
1275 obllabelgrProver.setText("Select Prover: ");
1276 obllabelgrProver.setVisible(false);
1277 obllabelgrProver.setBounds(20,160,250,30);
1278
1279 oblcbrgrProver.setSelectedItem(data[10].substring(1, data[10].length()-1));
1280 oblcbrgrProver.setVisible(false);
1281 oblcbrgrProver.setBounds(180,160,90,30);
1282
1283 obllabelgrTimeStart.setText("Time start: ");
1284 obllabelgrTimeStart.setVisible(false);
1285 obllabelgrTimeStart.setBounds(320,60,250,30);
1286
1287 obltfgrTimeStartValue.setText(data[11].trim());
1288 obltfgrTimeStartValue.setVisible(false);
1289 obltfgrTimeStartValue.setBounds(520,60,50,30);
1290
1291 obllabelgrTimeInc.setText("Inc. Time: ");
1292 obllabelgrTimeInc.setVisible(false);
1293 obllabelgrTimeInc.setBounds(320,110,250,30);
1294
1295 obltfgrTimeIncValue.setText(data[12].trim());
1296 obltfgrTimeIncValue.setVisible(false);
1297 obltfgrTimeIncValue.setBounds(520,110,50,30);
1298
1299 obllabelgrOperator.setText("Operator: ");
1300 obllabelgrOperator.setVisible(false);
1301 obllabelgrOperator.setBounds(320,160,250,30);
1302
1303 oblcbrgrOperation.setSelectedItem(data[13].substring(1, data[13].length()-1));
1304 oblcbrgrOperation.setVisible(false);
1305 oblcbrgrOperation.setBounds(520,160,50,30);
1306
1307 // Rest
1308 obllabelBezStatus.setText("Status: ");
1309 obllabelBezStatus.setVisible(true);
1310 obllabelBezStatus.setBounds(320,20,90,30);
1311
1312 obllabelStatus.setForeground(checkResult(ordner + fresult));
1313 if (obllabelStatus.getForeground() == glRed){
1314     obllabelStatus.setText("NOPROOF");

```

```

1315     }
1316     else if (obllabelStatus.getForeground() == glOrange){
1317         obllabelStatus.setText("#warning");
1318     }
1319     else if (obllabelStatus.getForeground() == glGreen){
1320         obllabelStatus.setText("PROOF");
1321     }
1322     else {
1323         obllabelStatus.setText("???");
1324     }
1325
1326     obllabelStatus.setVisible(true);
1327     obllabelStatus.setBounds(520,20,90,30);
1328
1329     obllabelTyp.setText("Proof type:");
1330     obllabelTyp.setVisible(true);
1331     obllabelTyp.setBounds(20,20,150,30);
1332
1333     oblcbTyp.setSelectedItem("Simple");
1334     oblcbTyp.setVisible(true);
1335     oblcbTyp.setBounds(180,20,90,30);
1336     oblcbTyp.addActionListener(new java.awt.event.ActionListener() {
1337         public void actionPerformed(java.awt.event.ActionEvent e) {
1338             switchTyp(oblcbTyp.getSelectedItem().toString());
1339         }
1340     });
1341
1342     oblbtnExit.setText("Close");
1343     oblbtnExit.setBounds(480,220,100,30);
1344     oblbtnExit.addActionListener(new java.awt.event.ActionListener() {
1345         public void actionPerformed(java.awt.event.ActionEvent e) {
1346             oblDetails.setVisible(false);
1347         }
1348     });
1349 }
1350
1351 /**
1352  * TBC: TPTP-AxSel: die beiden Typen ("simple" und "growth") bekommen
1353  * unterschiedliche GUI-Elemente
1354  * @param typ
1355  */
1356 void switchTyp(String typ){
1357     if (typ.equals("Simple")){
1358         obllabelTime.setVisible(true);
1359         obltfTime.setVisible(true);
1360         obllabelThreads.setVisible(true);
1361         obltfThreads.setVisible(true);
1362         obllabelProver.setVisible(true);
1363         oblcbProver.setVisible(true);
1364         obllabelWAprils.setVisible(true);
1365         obltfWAprils.setVisible(true);
1366         obllabelWObl.setVisible(true);
1367         obltfWObl.setVisible(true);
1368         obllabelWNaProche.setVisible(true);
1369         obltfWNaProche.setVisible(true);
1370
1371         obllabelgrStartValue.setVisible(false);
1372         obltfgrStartValue.setVisible(false);
1373         obllabelgrIncValue.setVisible(false);
1374         obltfgrIncValue.setVisible(false);

```

```

1375         obllabelgrProver.setVisible(false);
1376         oblcgrProver.setVisible(false);
1377         obllabelgrTimeStart.setVisible(false);
1378         obltfgrTimeStartValue.setVisible(false);
1379         obllabelgrTimeInc.setVisible(false);
1380         obltfgrTimeIncValue.setVisible(false);
1381         obllabelgrOperator.setVisible(false);
1382         oblcgrOperation.setVisible(false);
1383     }
1384     else if (typ.equals("Growth")){
1385         obllabelTime.setVisible(false);
1386         obltfTime.setVisible(false);
1387         obllabelThreads.setVisible(false);
1388         obltfThreads.setVisible(false);
1389         obllabelProver.setVisible(false);
1390         oblcgrProver.setVisible(false);
1391         obllabelWAprils.setVisible(false);
1392         obltfWAprils.setVisible(false);
1393         obllabelWObl.setVisible(false);
1394         obltfWObl.setVisible(false);
1395         obllabelWNaproche.setVisible(false);
1396         obltfWNaproche.setVisible(false);
1397
1398         obllabelgrStartValue.setVisible(true);
1399         obltfgrStartValue.setVisible(true);
1400         obllabelgrIncValue.setVisible(true);
1401         obltfgrIncValue.setVisible(true);
1402         obllabelgrProver.setVisible(true);
1403         oblcgrProver.setVisible(true);
1404         obllabelgrTimeStart.setVisible(true);
1405         obltfgrTimeStartValue.setVisible(true);
1406         obllabelgrTimeInc.setVisible(true);
1407         obltfgrTimeIncValue.setVisible(true);
1408         obllabelgrOperator.setVisible(true);
1409         oblcgrOperation.setVisible(true);
1410     }
1411 }
1412
1413 /**
1414  * Wandelt einen String in einen double-Wert um
1415  * @param d: umzuwandelner String
1416  * @return double: doublewert des Strungs
1417  */
1418 double getDouble(String d){
1419     return Double.parseDouble(d);
1420 }
1421
1422 /**
1423  * Wandelt einen String in einen Integer-Wert um
1424  * @param i: umzuwandelner String
1425  * @return int: Integer-Wert des Strungs
1426  */
1427 int getInt(String i){
1428     return Integer.parseInt(i);
1429 }
1430
1431 /**
1432  * TBC: TPTP-AxSel: Speichern der ueberpruefungseinstellungen des Obligationsfensters
1433  * in einer String-Liste
1434  * @return int: Integer-Wert des Strings

```

```

1435  */
1436  LinkedList<String> checkSettings() {
1437      LinkedList<String> erg = new LinkedList<String>();
1438      try {
1439          Double.parseDouble(obltfWObl.getText());
1440      }
1441      catch (Exception ex) {
1442          System.out.println("Fehler bei WOBL: \n" + ex);
1443          erg.add("Fehler bei WOBL");
1444      }
1445      try {
1446          Double.parseDouble(obltfWAprils.getText());
1447      }
1448      catch (Exception ex) {
1449          System.out.println("Fehler bei Aprils: \n" + ex);
1450          erg.add("Fehler bei Aprils");
1451      }
1452      try {
1453          Double.parseDouble(obltfWNa proche.getText());
1454      }
1455      catch (Exception ex) {
1456          System.out.println("Fehler: bei Na pr \n" + ex);
1457          erg.add("Fehler bei Na pr");
1458      }
1459      try {
1460          Integer.parseInt(obltfTime.getText());
1461      }
1462      catch (Exception ex) {
1463          System.out.println("Fehler: \n" + ex);
1464          erg.add("Fehler bei Time");
1465      }
1466      try {
1467          Integer.parseInt(obltfThreads.getText());
1468      }
1469      catch (Exception ex) {
1470          System.out.println("Fehler: \n" + ex);
1471          erg.add("Fehler bei Threads");
1472      }
1473      return erg;
1474  }
1475
1476  /**
1477   *   Erstellen des Menues
1478   */
1479  void makeMenu() {
1480      //Erstellen einer Menueleiste
1481      JMenuBar menuBar = new JMenuBar();
1482
1483      //Hinzufuegen von Menues
1484      JMenu menuFile = new JMenu("File");
1485      JMenu menuRun = new JMenu("Check");
1486      JMenu menuResults = new JMenu("Results");
1487
1488      menuBar.add(menuFile);
1489      menuBar.add(menuRun);
1490      menuBar.add(menuResults);
1491
1492      JMenuItem menuItemFileOpen = new JMenuItem("Open");
1493      JMenuItem menuItemFileSave = new JMenuItem("Save");
1494      JMenuItem menuItemFileExit = new JMenuItem("Exit");

```

```

1495
1496 JMenuItem menuItemRun = new JMenuItem("Check");
1497
1498 menuItemFileExit.addActionListener(new java.awt.event.ActionListener() {
1499     public void actionPerformed(java.awt.event.ActionEvent e) {
1500         System.out.println("EXIT!");
1501         proofname = "tmp";
1502         userPL();
1503         System.exit(0);
1504     }
1505 });
1506
1507 menuItemFileOpen.addActionListener(new java.awt.event.ActionListener() {
1508     public void actionPerformed(java.awt.event.ActionEvent e) {
1509         if (load()){
1510             System.out.println("Open!");
1511             showInDebugTxt("Datei geoeffnet!");
1512         }
1513         else {
1514             System.out.println("Keine gueltigen Beweisdaten im Zip-Archiv vorhanden!");
1515             showInDebugTxt("Keine gueltigen Beweisdaten im Zip-Archiv vorhanden!");
1516         }
1517     }
1518 });
1519
1520 menuItemFileSave.addActionListener(new java.awt.event.ActionListener() {
1521     public void actionPerformed(java.awt.event.ActionEvent e) {
1522         System.out.println("Save!");
1523         // Speichern des aktuellen Beweisinhaltes
1524         saveText(new File (proofname + "/" + proofname + ".txt"), pane.getText());
1525         save();
1526     }
1527 });
1528
1529 menuItemRun.addActionListener(new java.awt.event.ActionListener() {
1530     public void actionPerformed(java.awt.event.ActionEvent e) {
1531         checkProof();
1532     }
1533 });
1534
1535 JMenuItem menuItemResPRS = new JMenuItem("Show PRS");
1536 menuItemResPRS.addActionListener(new java.awt.event.ActionListener() {
1537     public void actionPerformed(java.awt.event.ActionEvent e) {
1538         showPrs();
1539     }
1540 });
1541
1542 JMenuItem menuItemResstats = new JMenuItem("Show global stats");
1543
1544 menuFile.add(menuItemFileOpen);
1545 menuFile.add(menuItemFileSave);
1546 menuFile.addSeparator();
1547 menuFile.add(menuItemFileExit);
1548 menuRun.add(menuItemRun);
1549 menuResults.add(menuItemResPRS);
1550 menuResults.add(menuItemResstats);
1551
1552 f.setJMenuBar(menuBar);
1553 }
1554

```

```

1555  /**
1556   *   Filechooser wird aufgerufen und ausgewaehlte Datei im GUI-Textfeld angezeigt
1557   *   @return boolean: true: konnte geladen werden, false: konnte nicht geladen werden
1558   */
1559  protected boolean load() {
1560      boolean erg = true;
1561
1562      final JFileChooser fc = new JFileChooser();
1563
1564      File x = new File (".");
1565      System.out.println("Pfad: " + x.getAbsolutePath());
1566
1567      fc.setCurrentDirectory(x);
1568
1569      int returnVal = fc.showOpenDialog(pane);
1570      if (returnVal == JFileChooser.APPROVE_OPTION) {
1571          File file = fc.getSelectedFile();
1572          String fname = file.toString();
1573
1574          if (fname.endsWith(".zip")){
1575              System.out.println("Das ist eine Zip-Datei \nLoeschen des alten Verzeichnisses");
1576              // loesche alten Ordner nach Bestaetigung der gueltigen Datei
1577              FileOperations.deleteDir(new File("tmp"));
1578
1579              boolean unzip = FileOperations.unZip(fname,x.getAbsolutePath());
1580
1581              if (unzip == true){
1582                  // ueberpruefung der vorhandenen Datenstruktur
1583                  if (!(new File("tmp/tmp.txt").exists())){
1584                      return false;
1585                  }
1586
1587                  showTextinPane(new File("tmp/tmp.txt"));
1588
1589                  // ueberpruefen, ob es eine zip-datei ist, die eine sentence.txt-Datei enthaelt.
1590                  // Darstellung der txt-Datei im Textfeld
1591
1592                  System.out.println("-----");
1593                  System.out.println("Beweis entpackt!");
1594                  System.out.println("-----");
1595
1596                  // Was soll passieren, wenn keine Beweisstruktur gefunden wurde?
1597                  boolean lp = loadProof();
1598                  if (lp == true){
1599                      // Einfarben
1600
1601                      // Testausgaben:
1602                      System.out.println("-----");
1603                      System.out.println("Einfarben der Saetze des Beweises");
1604                      System.out.println("-----");
1605                      for (int i = 0; i < globalSentences.size(); i++){
1606                          checkSentence(i+1);
1607                      }
1608                      colorAllSentencens();// 190 & 958
1609                      System.out.println("Beweis vollstaendig geladen!");
1610                      System.out.println("-----");
1611                  }
1612                  else {
1613                      System.out.println("keine Beweisstruktur zur Datei " + file.getName() + "
                      vorhanden!");

```

```

1614         }
1615     }
1616     else {
1617         System.out.println("keine gueltige Beweis-Datei geladen!");
1618     }
1619 }
1620 }
1621 else {
1622     System.out.println("keine gueltige Beweis-Datei geladen!");
1623 }
1624 }
1625 }
1626 return erg;
1627 }
1628
1629 /**
1630  * Zeigt Dateiinhalte im GUI-Textfeld an
1631  * @param file: Anzuzeigende Datei
1632  */
1633 private void showTextinPane(File file){
1634     this.pane.setText(FileOperations.loadFileToString(file));
1635 }
1636
1637 /**
1638  * Speichern der Einstellungen auch in der user.pl-datei
1639  */
1640 void userPL(){
1641     File user = new File("src/prolog/user.pl");
1642     File dummy = new File("dummy");
1643     String plContent = dummy.getAbsolutePath();
1644     plContent = plContent.substring(0, plContent.lastIndexOf("/"));
1645     plContent = "check_src(' " + plContent + "')";
1646     plContent = plContent + "\nsession_id(' " + proofname + "')";
1647     saveText(user, plContent);
1648 }
1649
1650
1651 /**
1652  * Speichert den Inhalt des Textfeldes in eine ueber den Filechooser
1653  * auswahlbare/neu zu erstellende Datei
1654  */
1655 protected void save() {
1656     final JFileChooser fc = new JFileChooser();
1657     File x = new File(".");
1658     fc.setCurrentDirectory(x);
1659     int returnVal = fc.showSaveDialog(pane);
1660
1661     if (returnVal == JFileChooser.APPROVE_OPTION) {
1662         File file = fc.getSelectedFile();
1663         String pname = file.toString();
1664
1665         if (!pname.endsWith(".zip")){
1666             System.out.println("Endung drangehangen!!!");
1667             pname = pname + ".zip";
1668         }
1669
1670         FileOperations.zipProof(pname, "tmp");
1671     }
1672 }
1673

```

```

1674  /**
1675   * Speichert den Inhalt des Strings in die uebergebene Datei
1676   * @param file: uebergebene Datei
1677   * @param text: zu speichernder Inhalt
1678   */
1679  private void saveText(File file , String text) {
1680      try {
1681          FileWriter writer = new FileWriter(file);
1682          writer.write(text);
1683          writer.flush();
1684          writer.close();
1685      }
1686      catch (IOException e) {
1687          e.printStackTrace();
1688      }
1689  }
1690
1691  /**
1692   * Beendet das Programm mit einer Speicherabfrage
1693   * Soll gespeichert werden, wird die Methode saveText aufgerufen
1694   */
1695  private void exit(){
1696      System.out.println("EXIT!");
1697      //Custom button text
1698      Object[] options = {"Yes, please",
1699                          "No, thanks",
1700                          "Cancel"};
1701      int n = JOptionPane.showOptionDialog(f,
1702          "Save proof?",
1703          "One last question",
1704          JOptionPane.YES_NO_CANCEL_OPTION,
1705          JOptionPane.QUESTION_MESSAGE,
1706          null,
1707          options,
1708          options[2]);
1709
1710      if (n == 1){ // 1 => nein
1711          System.exit(0);
1712      }
1713      else if (n == 0){
1714          saveText(new File(proofname + "/" + proofname + ".txt"), pane.getText());
1715          // Anpassen der User.pl - Datei
1716          proofname = "tmp";
1717          userPL();
1718          System.exit(0);
1719      }
1720  }
1721
1722  /**
1723   * Fuehrt ueber einen Prolog-Call ein Prolog-Query aus.
1724   * @param query: auszufuehrender Query
1725   * @return Hashtable: Ergebnis des Querys
1726   */
1727  java.util.Hashtable GULP(String query){
1728
1729      java.util.Hashtable erg = new java.util.Hashtable();
1730
1731      Query gulp = new Query("call((set_prolog_flag(occurs_check,true),term_expansion(" + query + ",Expanded),call(
1732          Expanded)))");
1733      try{

```



```

1733         erg = gulp.oneSolution();
1734
1735         return erg;
1736     }
1737     catch (Exception ex){
1738         System.out.println("Gulp - Error: \n" + ex);
1739         return erg;
1740     }
1741 }
1742
1743 /**
1744  * Beweisueberpruefungsmethode:
1745  * – Vorverarbeitung des Eingabestrings
1746  * – Aufruf des Naproche-Praedikates
1747  * – Abschliessende Einfaerbung der Ergebnisse
1748  */
1749 void startProofChecker(){
1750
1751     String ordner = proofname;
1752
1753     // Ordner fuer den Beweis erstellen
1754     File f = new File(ordner);
1755     if (f.isDirectory()) {
1756         //System.out.println("Das Verzeichnis \"\" + ordner + "\" existiert bereits.\n");
1757     } else {
1758         f.mkdir();
1759         //System.out.println("Das Verzeichnis \"\" + ordner + "\" wurde erzeugt.\n");
1760     }
1761     System.out.println("...Beweisordner zugeordnet...");
1762     try{
1763
1764         // Speichere den Ausgabeordner in die dafuer vorgesehene von Prolog lesbare Datei
1765         saveText(new File("output_folder.pl"), "output_folder("+ordner+")." );
1766         new Query("[ 'output_folder.pl' ].").oneSolution();
1767         System.out.println("Folder gesetzt");
1768
1769         System.out.println("vorm Ersetzen der Zeichen");
1770
1771         // Ersetze die "\" prologtauglich
1772         String inhalt = this.pane.getText();
1773         inhalt = inhalt.replaceAll("\\n", "#");
1774         inhalt = inhalt.replaceAll("\\\\", "!");
1775         inhalt = inhalt.replaceAll("!", "\\\\\\\\\\");
1776         System.out.println("nach dem Ersetzen");
1777         // Preparsen des Beweistextes mit Prolog
1778         prepare(inhalt);
1779         System.out.println("Nach dem Preparsen");
1780
1781         int anzSentence = globalSentences.size();
1782
1783         System.out.println("Preparse abgeschlossen; "+ anzSentence + " Saetze gefunden: \n" + pane.getText());
1784         java.util.Hashtable ht = GULP("naproche(' " + inhalt + "',L,T0)");
1785         gIPreparseList = ht.get("L").toString();
1786
1787         // Prolog hat Dateien im vorgegebenen Ordner erzeugt:
1788         System.out.println("nach Aufruf von Naproche-Praedikat\nAuswertung der Unterordner [" +ordner+ "]:\n");
1789         String[] filenames;
1790         LinkedList<Integer> ids = new LinkedList<Integer>();
1791         LinkedList<String> ObligationName = new LinkedList<String>();
1792         anzSentence = globalSentences.size();

```

```

1793
1794         for (int h = 0; h < anzSentence; h++){
1795             int id = h + 1;
1796             File ftmp = new File(ordner + "/" + id);
1797
1798             if (ftmp.exists()){
1799
1800                 // wenn ID noch nicht erfasst wurde
1801                 if (!ids.contains(id)){
1802                     ids.add(id);
1803                 }
1804
1805                 carposold = pane.getText().length() + 1;
1806                 filenames = listDir(ftmp);
1807
1808                 List<String> sort = Arrays.asList(filenames);
1809                 extracted(sort);
1810
1811                 // filenames ausgeben:
1812                 for (int k = 0; k < sort.size(); k++){
1813
1814                     // Einfarben des Satzes:
1815                     if (filenames[k].indexOf(".") > -1){
1816                         // Füge den Obligationsnamen in eine Liste ein (falls noch nicht vorhanden)
1817                         String obname = sort.get(k).toString().substring(0, sort.get(k).toString().indexOf("."));
1818
1819                         //System.out.println("Obligationsname: " + obname);
1820                         if (!ObligationName.contains(obname)){
1821                             if (obname.indexOf("-") > 0){
1822                                 ObligationName.add(obname);
1823                             }
1824                         }
1825                     }
1826                 }
1827                 else{
1828                     System.out.println("Keinen Satz " + ordner + "/" + id + " gefunden...");
1829                 }
1830             }
1831
1832             if (ids.size() > 0){
1833                 globalIDS = ids;
1834                 globalObligationName = ObligationName;
1835             }
1836
1837             // Positionen der Sätze markieren
1838             if (glshowObl == false){
1839                 addItem("show Obligations", 0);
1840                 glshowObl = true;
1841             }
1842             colorAllSentences(); // Alles einfärben
1843         }
1844     }
1845     catch (Exception ex){
1846         System.out.println("List - Error: \n" + ex);
1847     }
1848 }
1849
1850 /**
1851  * Sortiert die uebergebene Liste korrekt alphanumerisch

```

```

1852     * @param sort
1853     */
1854     private void extracted(List<String> sort) {
1855         Collections.sort(sort, new NaturalOrderComparator());
1856     }
1857
1858     /**
1859     * Setzt anhand der Ordnerstruktur die globalen Variablen
1860     */
1861     void setGlobalIdObl() {
1862         // Prolog hat Dateien im vorgegebenen Ordner erzeugt:
1863         String ordner = proofname;
1864         //System.out.println("\nAuswertung der Unterordner [" + ordner + "]\n");
1865         String[] filenames;
1866         LinkedList<Integer> ids = new LinkedList<Integer>();
1867         LinkedList<String> ObligationName = new LinkedList<String>();
1868
1869         for (int h = 0; h < globalSentences.size(); h++){
1870             int id = h + 1;
1871             File ftmp = new File(ordner + "/" + id);
1872
1873             if (ftmp.exists()){
1874
1875                 // wenn ID noch nicht erfasst wurde
1876                 if (!ids.contains(id)){
1877                     ids.add(id);
1878                 }
1879
1880                 carposold = pane.getText().length() + 1;
1881                 filenames = listDir(ftmp);
1882
1883                 List<String> sort = Arrays.asList(filenames);
1884                 extracted(sort);
1885
1886                 // filenames ausgeben:
1887                 for (int k = 0; k < sort.size(); k++){
1888
1889                     // Einfärben des Satzes:
1890                     if (filenames[k].indexOf(".") > -1){
1891                         // Füge den Obligationsnamen in eine Liste ein (falls noch nicht vorhanden)
1892                         String obname = sort.get(k).toString().substring(0, sort.get(k).toString().indexOf("."));
1893                         if (!ObligationName.contains(obname)){
1894                             if (obname.indexOf("-") > 0){
1895                                 ObligationName.add(obname);
1896                             }
1897                         }
1898                     }
1899                 }
1900             }
1901             else{
1902                 //System.out.println("Keinen Satz " + ordner + "/" + id + " gefunden...");
1903             }
1904         }
1905
1906         if (ids.size() > 0){
1907             globalIDS = ids;
1908             globalObligationName = ObligationName;
1909         }
1910     }
1911

```

```

1912         // Positionen der Sätze markieren
1913         if (glshowObl == false){
1914             addItem("show Obligations", 0);
1915             glshowObl = true;
1916         }
1917     }
1918
1919     /**
1920     * Ausgabemethode der globalen Variablen
1921     */
1922     void showGlobalVars() {
1923         System.out.println("Globale Variablen: \n");
1924
1925         System.out.println("\nSentences");
1926         for (int i = 0; i < globalSentences.size(); i++){
1927             System.out.println(i + " : " + globalSentences.get(i));
1928         }
1929
1930         System.out.println("\nIDS");
1931         for (int i = 0; i < globalIDS.size(); i++){
1932             System.out.println(i + " : " + globalIDS.get(i));
1933         }
1934
1935         System.out.println("\nObligationennamen");
1936         for (int i = 0; i < globalObligationName.size(); i++){
1937             System.out.println(i + " : " + globalObligationName.get(i));
1938             String tmp = globalObligationName.get(i);
1939             String dateipfad = proofname + "/" + tmp.substring(0,tmp.indexOf("-")) + "/" + tmp + ".result";
1940             System.out.println("Pfad der Datei: " + dateipfad);
1941             File dat = new File(dateipfad);
1942             if (dat.exists()){
1943                 System.out.println("Datei " + dat.toString() + " existiert!");
1944             }
1945         }
1946     }
1947
1948     /**
1949     * ueberprueft einen Satz anhand seiner uebergeben ID durch Ermittlung der Farbe
1950     * @param sentID: Id des zu ueberpruefenden Satzes
1951     * @return Color: Farbe des Satzes
1952     */
1953     Color checkSentence(int sentID){
1954         Color c = glGray;
1955         String ID = "" + sentID;
1956         LinkedList<Color> cl = new LinkedList<Color>();
1957         for (int i = 0; i < globalObligationName.size(); i++){
1958             String name = globalObligationName.get(i);
1959             // wenn im Ordner noch keine result-Datei vorhanden ist, faerbe grau
1960             if (name.indexOf("-") < 0) return c;
1961             String id = name.substring(0,name.indexOf("-"));
1962             if (id.equals(ID)){
1963                 cl.add(checkObligation(i));
1964             }
1965         }
1966     }
1967     for (int j = 0; j < cl.size(); j++){
1968         if (cl.get(j) == glGreen && c != glOrange && c != glRed){
1969             c = glGreen;
1970         }
1971         else if (cl.get(j) == glOrange && c != glRed){

```

```

1972         c = glOrange;
1973     }
1974     else if (cl.get(j) == glRed){
1975         c = glRed;
1976     }
1977 }
1978 pane.changeColor(globalSentences.get(sentID-1).start,globalSentences.get(sentID-1).end-globalSentences.get(
    sentID-1).start,c);
1979
1980 return c;
1981
1982 }
1983
1984 /**
1985  * ueberprueft eine einzelne Obligation anhand der uebergebenen ID
1986  * @param id: id der zu ueberpruefenden obligation
1987  * @return color: Farbe des Satzes
1988  */
1989 Color checkObligation(int id){
1990     String tmp = globalObligationName.get(id);
1991     String dateipfad = proofname + "/" + tmp.substring(0,tmp.indexOf("-")) + "/" + tmp + ".result";
1992     File dat = new File(dateipfad);
1993     if (dat.exists()){
1994         Color c = checkResult(dat.toString());
1995         return c;
1996     }
1997     else {
1998         return glGray;
1999     }
2000 }
2001
2002 /**
2003  * ueberprueft des Ergebnis einer Result-Datei
2004  * @param fname: name der Result-Datei
2005  * @return color: resultierende Farbe
2006  */
2007 Color checkResult(String fname){
2008     String tmp = FileOperations.loadFileToString(new File (fname));
2009     Color c = new Color (180,104,80);
2010     if (tmp.indexOf(";") > 0){
2011         String[] x = tmp.split(";");
2012         if (x[1].equals("false")){
2013             c = glRed;
2014         }
2015         else {
2016             if (x[1].equals("true")){
2017                 if (x[2].equals("true")){
2018                     c = glOrange;
2019                 }
2020                 else {
2021                     c = glGreen;
2022                 }
2023             }
2024         }
2025     }
2026     return c;
2027 }
2028 else {
2029     return Color.lightGray; // soll nie kommen ; )
2030 }

```

```

2031
2032     }
2033
2034     /**
2035      * loesche alle Dateien bis auf den Inhalt
2036      * @param DirName: Name des zu loeschenden Verzeichnisses
2037      * @return boolean: true: wurde geloescht, false: nicht geloescht
2038      */
2039     boolean clearDir(String DirName){
2040         File dir = new File(DirName);
2041         if (dir.isDirectory()){
2042             String[] entries = dir.list();
2043             for (int x=0;x<entries.length;x++){
2044                 File aktFile = new File(dir.getPath()+entries[x]);
2045                 clearDir(aktFile.toString());
2046             }
2047             if (dir.delete())
2048                 return true;
2049             else
2050                 return false;
2051         }
2052         else{
2053             if (dir.delete())
2054                 return true;
2055             else
2056                 return false;
2057         }
2058     }
2059 }
2060
2061 /**
2062  * Umschreiben der Klasse JTextPane, um Bereiche im Textfeld farblich unabhaengig gestallten zu koennen.
2063  */
2064 class AttributedTextPane extends JTextPane{
2065     private static final long serialVersionUID = 1L;
2066     private DefaultStyledDocument m_defaultStyledDocument=new DefaultStyledDocument();
2067
2068     /** constructor*/
2069     public AttributedTextPane(){
2070         this.setDocument(m_defaultStyledDocument);
2071     }
2072 }
2073
2074 /** append text */
2075 public void append(String string,Color color){
2076     try{
2077         SimpleAttributeSet attr=new SimpleAttributeSet();
2078         StyleConstants.setForeground(attr,color);
2079         m_defaultStyledDocument.insertString(m_defaultStyledDocument.getLength(),string,attr);
2080     }
2081     catch(Exception e){
2082         e.printStackTrace();
2083     }
2084 }
2085
2086 /**
2087  * aendern der Farbe eines Textbereiches
2088  * @param offs: Startposition des Farbbereichs
2089  * @param len: Laenge des Farbbereichs
2090  * @param color: Farbe des Farbbereichs

```

```

2091  */
2092  public void changeColor(int offs, int len, Color color){
2093      int max = offs+len;
2094      if (max > this.getText().length()){
2095          System.out.println("-----Einfuerbung schiefgegangen-----");
2096          System.out.println("Werte: \tOffset:" + offs + "\tLaenge:" + len + "\tMaximale Laenge:" + max + "\tTextlaenge:"
              + this.getText().length() + "\nText:\n" + this.getText());
2097      }
2098      else {
2099          try{
2100              String string = this.getText(offs, len);
2101              SimpleAttributeSet attr=new SimpleAttributeSet();
2102              StyleConstants.setForeground(attr, color);
2103              m_defaultStyledDocument.remove(offs, len);
2104              m_defaultStyledDocument.insertString(offs, string, attr);
2105          }
2106          catch(Exception e){
2107              e.printStackTrace();
2108          }
2109      }
2110  }
2111
2112  /** append text in default color */
2113  public void append(String string){
2114      append(string, Color.white);
2115  }
2116  }
2117
2118  /**
2119   * Klasse fuer Obligationsliste
2120   */
2121  class JListModel extends AbstractListModel {
2122
2123      private static final long serialVersionUID = 1L;
2124      private ArrayList<String> data;
2125
2126      public JListModel() {
2127          data = new ArrayList<String>();
2128      }
2129
2130      public int getSize() {
2131          return data.size();
2132      }
2133
2134      public Object getElementAt(int index) {
2135          return data.get(index);
2136      }
2137
2138      public Object getIndex(Object o) {
2139          return data.indexOf(o);
2140      }
2141
2142      public void setElement(String s, int index) {
2143          data.set(index, s);
2144          update(0, this.getSize());
2145      }
2146
2147      public void addElement(String s) {
2148          data.add(s);
2149          update(this.getSize() - 1, this.getSize());

```

```

2150     }
2151
2152     public void removeElement(int index){
2153         data.remove(index);
2154         update(0, this.getSize());
2155     }
2156
2157     public void update(int von, int bis) {
2158         fireIntervalAdded(this, von, bis);
2159     }
2160 }

```

Listing 9.4: NaturalOrderComparator.java

```

1  package net.naproche.GUI;
2
3
4  import java.util.*;
5
6  /**
7   * Diese Klasse bietet die Funktionen zum Sortiern von Objekten in natuerlicher Reihenfolge an
8   */
9  public class NaturalOrderComparator implements Comparator
10 {
11     /** vergleicht die Laenge zweier Strings
12      * @param a: String 1
13      * @param b: String 2
14      * @return int: +1: String 1 groesser, -1: String 2 groesser
15      */
16     int compareRight(String a, String b)
17     {
18         int bias = 0;
19         int ia = 0;
20         int ib = 0;
21
22         // The longest run of digits wins. That aside, the greatest
23         // value wins, but we can't know that it will until we've scanned
24         // both numbers to know that they have the same magnitude, so we
25         // remember it in BIAS.
26         for (;;) ia++, ib++)
27         {
28             char ca = charAt(a, ia);
29             char cb = charAt(b, ib);
30
31             if (!Character.isDigit(ca) && !Character.isDigit(cb))
32             {
33                 return bias;
34             }
35             else if (!Character.isDigit(ca))
36             {
37                 return -1;
38             }
39             else if (!Character.isDigit(cb))
40             {
41                 return +1;
42             }
43             else if (ca < cb)
44             {
45                 if (bias == 0)
46

```



```
47         bias = -1;
48     }
49 }
50 else if (ca > cb)
51 {
52     if (bias == 0)
53         bias = +1;
54 }
55 else if (ca == 0 && cb == 0)
56 {
57     return bias;
58 }
59 }
60 }
61
62 /** vergleicht die Laenge zweier Objekte
63  * @param o1: Objekt 1
64  * @param o2: Objekt 2
65  * @return int: +1: objekt 1 groesser, -1: Objekt 2 groesser
66  */
67 public int compare(Object o1, Object o2)
68 {
69     String a = o1.toString();
70     String b = o2.toString();
71
72     int ia = 0, ib = 0;
73     int nza = 0, nzb = 0;
74     char ca, cb;
75     int result;
76
77     while (true)
78     {
79         // only count the number of zeroes leading the last number compared
80         nza = nzb = 0;
81
82         ca = charAt(a, ia);
83         cb = charAt(b, ib);
84
85         // skip over leading spaces or zeros
86         while (Character.isSpaceChar(ca) || ca == '0')
87         {
88             if (ca == '0')
89             {
90                 nza++;
91             }
92             else
93             {
94                 // only count consecutive zeroes
95                 nza = 0;
96             }
97
98             ca = charAt(a, ++ia);
99         }
100
101         while (Character.isSpaceChar(cb) || cb == '0')
102         {
103             if (cb == '0')
104             {
105                 nzb++;
106             }
```

```
107         else
108         {
109             // only count consecutive zeroes
110             nzb = 0;
111         }
112
113         cb = charAt(b, ++ib);
114     }
115
116     // process run of digits
117     if (Character.isDigit(ca) && Character.isDigit(cb))
118     {
119         if ((result = compareRight(a.substring(ia), b.substring(ib))) != 0)
120         {
121             return result;
122         }
123     }
124
125     if (ca == 0 && cb == 0)
126     {
127         // The strings compare the same. Perhaps the caller
128         // will want to call strcmp to break the tie.
129         return nza - nzb;
130     }
131
132     if (ca < cb)
133     {
134         return -1;
135     }
136     else if (ca > cb)
137     {
138         return +1;
139     }
140
141     ++ia;
142     ++ib;
143 }
144 }
145
146 /** Gibt ein einzelnes Zeichen an einer Position in einem String an
147  * @param s: zu untersuchender String
148  * @param i: Position des Zeichens
149  * @return char: Zeichen
150  */
151 static char charAt(String s, int i)
152 {
153     if (i >= s.length())
154     {
155         return 0;
156     }
157     else
158     {
159         return s.charAt(i);
160     }
161 }
162
163 /** Main zum Testen
164  */
165 public static void main(String[] args)
166 {
```

```

167     String[] strings = new String[] { "1-2", "1-02", "1-20", "10-20", "fred", "jane", "pic01",
168     "pic2", "pic02", "pic02a", "pic3", "pic4", "pic 4 else", "pic 5", "pic05", "pic 5",
169     "pic 5 something", "pic 6", "pic 7", "pic100", "pic100a", "pic120", "pic121",
170     "pic02000", "tom", "x2-g8", "x2-y7", "x2-y08", "x8-y8" };
171
172     List orig = Arrays.asList(strings);
173
174     System.out.println("Original: " + orig);
175
176     List scrambled = Arrays.asList(strings);
177     Collections.shuffle(scrambled);
178
179     System.out.println("Scrambled: " + scrambled);
180
181     Collections.sort(scrambled, new NaturalOrderComparator());
182
183     System.out.println("Sorted: " + scrambled);
184 }
185

```

Listing 9.5: PrintTool.java

```

1 package net.naproche.GUI;
2 // Quelle: http://www.tutorials.de/swing-java2d-3d-swt-jface/242065-printtool.html
3
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.awt.print.*;
7 import java.io.BufferedReader;
8 import java.io.File;
9 import java.io.FileNotFoundException;
10 import java.io.FileReader;
11 import java.io.IOException;
12 import java.io.PrintStream;
13 import java.util.ArrayList;
14 import java.util.LinkedList;
15 import java.util.List;
16
17 import javax.swing.*;
18 import javax.swing.text.*;
19
20 /**
21  * Diese Klasse bietet die Funktionen zum Drucken eines Strings an.
22  */
23 public class PrintTool implements Printable {
24
25     private PrintStream o = System.out;
26
27     private String textTotal;
28     private PageFormat pageFormat;
29     private Font fontForPrint;
30     private String textPassage;
31     private List<String> textPassages = new ArrayList<String>();
32
33     private JWindow windowForPrint = new JWindow();
34     private JTextArea textAreaForPrint = new JTextArea();
35     AttributedTextPane txt = new AttributedTextPane();
36     private FontMetrics fontMetrics;
37     private Dimension pageDim;
38     private BufferedImage bufferedImage;

```

```

39
40 private static final int CONS = 2;
41 private int linesTotal = 0;
42 private int linesMaxOnPage = 0;
43 private int numberOfPages = 0;
44 private int pageBorders[][] = new int[999][2];
45
46 private String proofname = "tmp"; // TBC
47 private boolean colorful = false;
48 private int currentNumberOfSigns = 0;
49 private int maxNumberOfSigns = 0;
50 private int stillToPrintSigns = 0;
51 Color[] all = new Color[6];
52 Color glRed = new Color(205,0,0); // farbe 1
53 Color glOrange = new Color(254,158,0); // farbe 2
54 Color glGreen = new Color(80,204,80); // farbe 3
55 Color glGray = new Color(111,111,111); // farbe 4
56 Color glBlue = new Color(0,0,255); // farbe 5
57
58 private PrinterJob printerJob = PrinterJob.getPrinterJob();
59
60
61
62 public PrintTool(String textToPrint, PageFormat pageFormat, Font font, String proofname, boolean colorful) {
63     // Initialize parameters
64     this.textTotal = textToPrint;
65     this.pageFormat = pageFormat;
66     this.fontForPrint = font;
67     this.maxNumberOfSigns = textToPrint.length();
68     this.stillToPrintSigns = textToPrint.length();
69     this.proofname = proofname;
70     this.colorful = colorful;
71
72     if (textTotal == null)
73         textTotal = textForTestpage;
74
75     if (this.pageFormat == null)
76         this.pageFormat = new PageFormat();
77
78     if (fontForPrint == null)
79         fontForPrint = new Font("Arial", Font.PLAIN, 16 * CONS);
80
81     else
82         fontForPrint = new Font(font.getFamily(), font.getStyle(), font.getSize() * CONS);
83     pageDim = new Dimension((((int) this.pageFormat.getImageableWidth() - 10) * CONS, ((int) this.pageFormat.
84         getImageableHeight()) * CONS);
85
86     // Prepare textarea for Print and Preview
87     textareaForPrint.setFont(fontForPrint);
88     fontMetrics = textareaForPrint.getFontMetrics(fontForPrint);
89     textareaForPrint.setLineWrap(true);
90     textareaForPrint.setWrapStyleWord(true);
91     textareaForPrint.setPreferredSize(pageDim);
92     textareaForPrint.setTabSize(4);
93     textareaForPrint.setText(textTotal);
94     txt.setFont(fontForPrint);
95     txt.setPreferredSize(pageDim);
96     txt.setText(textTotal);
97
98     // Add on JWindow

```

```

98     windowForPrint.add(textareaForPrint);
99     windowForPrint.pack();
100     // Wrapp text and give to TextArea
101     textareaForPrint.setText(this.getWrappedText(textareaForPrint));
102     pageDim = new Dimension((int) this.pageFormat.getImageableWidth() * CONS, (int) this.pageFormat.getImageableHeight() *
        CONS);
103     textareaForPrint.setPreferredSize(pageDim);
104     windowForPrint.pack();
105
106     // txtpane
107     windowForPrint.add(txt);
108     windowForPrint.pack();
109     txt.setText(this.getWrappedText(textareaForPrint));
110     txt.setPreferredSize(pageDim);
111
112
113     // Calculate specifications of TextArea
114     linesMaxOnPage = this.getMaxLines();
115     linesTotal = textareaForPrint.getLineCount();
116     numberOfPages = this.getNumberOfPages();
117
118     // Calculate Start and End of the pages and store in pageBorders
119     // And split text in passages and store in textPassages
120
121     try {
122         for (int i = 0; i < numberOfPages; i++)
123             pageBorders[i][0] = textareaForPrint.getLineStartOffset(i * linesMaxOnPage);
124         for (int i = 0; i < numberOfPages - 1; i++)
125             pageBorders[i][1] = pageBorders[i + 1][0] - 1;
126         pageBorders[numberOfPages - 1][1] = textareaForPrint.getLineEndOffset(linesTotal - 1);
127         for (int i = 0; i < numberOfPages; i++)
128             textPassages.add(textareaForPrint.getText(pageBorders[i][0], pageBorders[i][1] - pageBorders[i][0]));
129     } catch (BadLocationException e) {
130         e.printStackTrace();
131     }
132
133     // initialisierung der Farbwerte
134     all[0] = new Color(0,0,0);
135     all[1] = new Color(205,0,0);
136     all[2] = new Color(254,158,0);
137     all[3] = new Color(80,204,80);
138     all[4] = new Color(111,111,111);
139     all[5] = new Color(0,0,255);
140
141 }
142
143 public boolean printPage(int page) {
144     if (page < 0 | page > numberOfPages - 1)
145         return false;
146     printerJob.setPrintable(this, pageFormat);
147     textPassage = textPassages.get(page);
148     try {
149         printerJob.print();
150     } catch (PrinterException e) {
151         e.printStackTrace();
152         return false;
153     }
154     return true;
155 }
156

```

```

157 public void printAllPages() {
158     // Zuerst Dialog, dann alles drucken
159     printerJob.setPrintable(this, pageFormat);
160
161     if (printerJob.printDialog()) {
162         try {
163             for (int i = 0; i < numberOfPages; i++)
164                 printPage(i);
165         } catch (Exception pe) {
166             System.out.println("Error printing: " + pe);
167         }
168     }
169 }
170
171 public int print(Graphics g, PageFormat pFormat, int pageIndex)
172     throws PrinterException {
173     if (pageIndex > 0)
174         return Printable.NO_SUCH_PAGE;
175
176     Graphics2D g2 = (Graphics2D) g;
177     g2.translate((int) pFormat.getImageableX()+10, (int) pFormat.getImageableY());
178     g2.scale(1.0 / CONS, 1.0 / CONS);
179     txt.setText(textPassage);
180     currentNumberOfSigns = textPassage.length();
181     stillToPrintSigns = stillToPrintSigns - (currentNumberOfSigns/2); // durch 2 dann stimmt's, warum auch immer
182
183     if (colorful == true) {
184         checkColors();
185     }
186
187     txt.setBorder(BorderFactory.createMatteBorder(1, 1, 1, 1, Color.gray)); // test
188     bufferedImage = null;
189     bufferedImage = new BufferedImage(pageDim.width-10, pageDim.height, BufferedImage.TYPE_BYTE_GRAY);
190     txt.paint(bufferedImage.getGraphics());
191     g2.drawImage(bufferedImage, 0, 0, txt);
192
193     g2.setColor(new Color(0,0,0)); // Test: Ausgabe
194     g2.setFont(fontForPrint);
195     g2.dispose();
196
197     return Printable.PAGE_EXISTS;
198 }
199
200 public BufferedImage getPreviewOfPage(int pageI) {
201     textareaForPrint.setText(textPassages.get(pageI));
202
203     bufferedImage = null;
204     bufferedImage = new BufferedImage(pageDim.width, pageDim.height,
205         BufferedImage.TYPE_BYTE_GRAY);
206     textareaForPrint.paint(bufferedImage.getGraphics());
207
208     try {
209         return bufferedImage;
210     } finally {
211         bufferedImage = null;
212     }
213 }
214
215 private String getWrappedText(JTextComponent c) {
216     int len = c.getDocument().getLength();
217     int offset = 0;

```

```

217     StringBuffer buf = new StringBuffer((int) (len * 1.30));
218     String s = "";
219     try {
220         while (offset < len) {
221             int end = Utilities.getRowEnd(c, offset);
222             if (end < 0) {
223                 break;
224             }
225             end = Math.min(end + 1, len);
226             s = c.getDocument().getText(offset, end - offset);
227             buf.append(s);
228             if (!s.endsWith("\n")) {
229                 buf.append('\n');
230             }
231             offset = end;
232         }
233     } catch (BadLocationException e) {
234         e.printStackTrace();
235     }
236     try {
237         return buf.toString();
238     } finally {
239         buf = null;
240         s = null;
241     }
242 }
243
244 public int getNumberOfPages() {
245     int max = this.getMaxLines();
246     int total = textareaForPrint.getLineCount();
247     int pages = (int) Math.ceil((double) total / (double) max);
248     return pages;
249 }
250
251 private int getMaxLines() {
252     return textareaForPrint.getHeight() / fontMetrics.getHeight();
253 }
254
255 // Methode um das globale Array zu fuellen
256 private void checkColors() {
257     LinkedList<String> tmp = loadFileToString(new File (proofname + "/ColorKoord.txt"));
258     for (int i = 0; i < tmp.size(); i++) {
259         //System.out.println(i + ": " + tmp.get(i));
260         String[] x = tmp.get(i).split("!");
261         int start = Integer.parseInt(x[0]);
262         int ende = Integer.parseInt(x[1]);
263         int off = ende - start;
264         int col = Integer.parseInt(x[2]); // wenn alles auf einer Seite ist, faerbe die Teile der Seite
265         if (ende - 1 < txt.getText().length()) {
266             txt.changeColor(start, off, all[col]);
267             System.out.println("Von " + start + " bis " + ende + " in Farbe " + all[col] + " gefaerbt!");
268         }
269     }
270 }
271
272 private LinkedList<String> loadFileToString(File file) {
273     //StringBuffer buf = new StringBuffer();
274     LinkedList<String> erg = new LinkedList<String>();
275     if (file.exists()) {
276         try {

```

```

277         BufferedReader reader = new BufferedReader(new FileReader(file));
278         String line = "";
279         while((line = reader.readLine()) != null){
280             //buf.append(line+"\n");
281             erg.add(line);
282         }
283         reader.close();
284     }
285     catch (FileNotFoundException e) {
286         e.printStackTrace();
287     }
288     catch (IOException e) {
289         e.printStackTrace();
290     }
291 }
292 else {
293     System.out.println("NICHT DA!");
294 }
295 //return (buf.toString());
296 return (erg);
297 }
298
299 private static final String textForTestpage = "Kein Text im Textfeld vorhanden!";
300
301 }

```

Listing 9.6: Sentence.java

```

1 package net.naproche.preparser;
2 import java.util.LinkedList;
3
4 /**
5  * Represents a single Sentence, parsed by input_parser.pl
6  * @author Julian Schloeder
7  * modified: Sebastian Zittermann
8  */
9 public class Sentence{
10     //General note for debugging:
11     //All the ugly substringy-stuff must be seen on an example to make sense. Don't try to understand it purely by reading
12     //this code.
13     //This whole module is actually more the result of a LOT try and error than of sane coding.
14
15     //Each sentence has an unique id, the index of the starting character and the index of the last character (usually .
16     //or :).
17     public int id, start, end;
18
19     public LinkedList<Word> content;
20
21     // Constructor with all Attributes
22     public Sentence(int i, int s, int e, LinkedList<Word> c){
23         this.id = i;
24         this.start = s;
25         this.end = e;
26         this.content = c;
27     }
28
29
30     // inString is one substring of the return value of create_naproche_input which represents one sentence

```



```

31 public Sentence(String inString){
32     boolean foundPos=false, foundCon=false;
33
34     int pairs=0,posStart=0,conStart=0;
35
36     // we iterate over every single character.
37     // "pairs" is the number of open parentheses with no corresponding closing one.
38
39     // parentheses are not to be counted if they are atoms themselves therefore they need not to be surrounded by
40     // but if they are it could be a construct like '(atom)' (the parentheses is surrounded by ' but not an atom
41     // so this case is explicitly excluded
42     //
43     // input_parser.pl guarantees that if ( ) are part of atoms, they are always the only character of this atom
44     // should this change: good luck.
45     for (int i=0; i<inString.length(); i++){
46         if (inString.substring(i,i+1).equals("(")
47             && !(inString.substring(i+1,i+2).equals("' ")
48                 && inString.substring(i-1,i).equals("' ")
49                 && !inString.substring(i-2,i-1).equals(".")
50                 && inString.substring(i-3,i-2).equals("' ")
51             )
52         )
53             pairs++;
54         else if (inString.substring(i,i+1).equals(")")
55             && !(inString.substring(i-1,i).equals("' ")
56                 && inString.substring(i+1,i+2).equals("' ")
57                 && !inString.substring(i-2,i-1).equals(".")
58                 && inString.substring(i-3,i-2).equals("' ")
59             )
60         )
61             pairs--;
62         // A list is found (starts with '.',), the only open parenthesis is the one in "sentence(" and the
63         // position of the List containing the wordpositions is not found yet. -> The list containing the
64         // wordpositions is found.
65         else if (inString.substring(i,i+1).equals(".") &&
66             inString.substring(i-1,i).equals("' ") &&
67             inString.substring(i+1,i+2).equals("' ") &&
68             pairs == 1 &&
69             foundPos == false ){
70             foundPos = true;
71             posStart = i-1;
72         }
73         // A list is found (starts with '.',), the only open parenthesis is the one in "sentence(" and the
74         // position of the list containing the word-positions is already found. -> It must be the list
75         // containing the wordcontent.
76         else if (inString.substring(i,i+1).equals(".") &&
77             inString.substring(i-1,i).equals("' ") &&
78             inString.substring(i+1,i+2).equals("' ") &&
79             pairs == 1 &&
80             foundCon == false &&
81             foundPos == true ){
82             foundCon = true;
83             conStart = i-1;
84         }
85     }
86
87     // empty sentence case. We're mathematicians here, so the empty sentence is a sentence too.
88     if (posStart==0 || conStart==0){
89         posStart = inString.split("\\[\\]") [0].length(); // number of characters until [ (position list)

```

```

186         conStart = inString.split("\\[\\]") [1].length()+posStart+2; // "[" is not included in split, so add 2
187     }
188
189     // first 9 characters are "sentence(", following are id, start and end, seperated by ,.
190     String[] idStartEnd = inString.substring(9,posStart).split(",");
191
192     this.id = Integer.parseInt(idStartEnd[0].trim());
193     this.start = Integer.parseInt(idStartEnd[1].trim());
194     this.end = Integer.parseInt(idStartEnd[2].trim());
195
196     // Strings representing the lists containing wordpositions and wordcontent
197     String pos = inString.substring(posStart,conStart-1);
198     String con = inString.substring(conStart);
199
200     //Lists containing the strings representing the wordpositions and wordcontent
201     LinkedList<String> tmpPositions = convertDotNotation(pos);
202     LinkedList<String> tmpContent = convertDotNotation(con);
203
204     //List containing triples containing [type, start, end] for each word.
205     LinkedList<LinkedList<String>> positions = Word.convertWord(tmpPositions);
206
207     content = new LinkedList<Word>();
208     Word tmpWord;
209
210     // Constructing individual words.
211     // Assumption: Elements in positions and elements in tmpContent are exactly bijective
212     for(int i = 0; i < positions.toArray().length; i++){
213         tmpWord = new Word(positions.get(i).get(1),
214                             positions.get(i).get(2),
215                             positions.get(i).get(0),
216                             tmpContent.get(i));
217         content.add(tmpWord);
218     }
219 };
220
221 /**
222  * Returns a String which reads (or it should read) exactly like the sentence in the swipl interpreter.
223  * @return String
224  */
225 public String toString(){
226     //Everything here is straight-forward. Really.
227     String retVal = "sentence(";
228     retVal = retVal+id+", "+start+", "+end+", ";
229     retVal = retVal.concat("[");
230     for (Word p : this.content){
231         retVal = retVal.concat(p.type);
232         retVal = retVal.concat("(");
233         retVal = retVal.concat(String.valueOf(p.start));
234         retVal = retVal.concat(", ");
235         retVal = retVal.concat(String.valueOf(p.end));
236         retVal = retVal.concat(")");
237         retVal = retVal.concat(", ");
238     }
239     retVal = retVal.substring(0,retVal.length()-2);
240     retVal = retVal.concat("], ");
241     retVal = retVal.concat(content.toString());
242     retVal = retVal.concat(")");
243     return retVal;
244 }
245

```

```

146 /**
147  * Takes a String representing a Prolog-list in recursive .-Notation (so it has to start with '.'( )
148  * Does not recurse into nested lists. So if you have a list containing more lists, you will get
149  * a LinkedList containing Strings which represent the nested lists. You then may proceed to convert
150  * these with this method again.
151  * DEBUG: This is a static utility-procedure an may be moved.
152  * @param inString
153  * @return LinkedList<String>
154  */
155 public static LinkedList<String> convertDotNotation(String inString){
156     // please note: This Method was written with the output from input_parser in mind
157     // and may break down horribly when used on general lists (it's especially tacky with parentheses).
158     // For instance a List containing an atom like 'ab(c' will cause this to malfunction.
159     int l = inString.length();
160     int pairs = 0;
161     int j=1;
162     if (l < 3 || !inString.substring(0,3).equals("'.'")){
163         return new LinkedList<String>();
164     }
165     // comma on parentheses-level 1 indicates a new member of the list.
166     for (int i = 0; i<l; i++){
167         // parentheses are not to be counted if they are atoms themselves
168         // therefore they need not to be surrounded by '
169         // but if they are it could be a construct like '.'('atom'
170         // so this case is excluded. For the general idea see above in the constructor.
171         if (inString.substring(i,i+1).equals(" ( ")
172             && !( inString.substring(i+1,i+2).equals("' ")
173                 && inString.substring(i-1,i).equals("' ")
174                 && !inString.substring(i-2,i-1).equals(".")
175                 && inString.substring(i-3,i-2).equals("' ")
176             )
177         )
178             pairs++;
179         else if (inString.substring(i,i+1).equals(" ) ")
180             && !( inString.substring(i-1,i).equals("' ")
181                 && inString.substring(i+1,i+2).equals("' ")
182                 && !inString.substring(i-2,i-1).equals(".")
183                 && inString.substring(i-3,i-2).equals("' ")
184             )
185         )
186             pairs--;
187         // ', ' could be an atom, so this case is excluded.
188         // Also commas which separte members always have a following space (hopefully.)
189         else if (inString.substring(i,i+1).equals(" , ")
190             && !( inString.substring(i-1,i).equals("' ")
191                 && inString.substring(i+1,i+2).equals("' ")
192             )
193             && inString.substring(i+1,i+2).equals(" ")
194             && pairs == 1){
195             j = i+2;
196             break;
197         }
198     }
199     // recurses on the tail of the string after a comma is found
200     LinkedList<String> retVal = new LinkedList<String>(convertDotNotation(inString.substring(j,l-1)));
201     // adding found element (before the comma)
202     retVal.addFirst(inString.substring(4,j-2));
203     return retVal;
204 }
205

```

```

206 /**
207  * Wrappermethod for constructing the complete list of all sentences in the input-String.
208  * DEBUG: This is a static utility-procedure and may be moved.
209  * @param inString
210  * @return LinkedList<Sentence>
211  */
212     public static LinkedList<Sentence> convertSentences(String inString){
213         LinkedList<String> temp = convertDotNotation(inString);
214         LinkedList<Sentence> retVal = new LinkedList<Sentence>();
215         for (String sentence : temp)
216             retVal.add(new Sentence(sentence));
217         return retVal;
218     }
219 }

```

Listing 9.7: Word.java

```

1 package net.naproche.preparser;
2 import java.util.LinkedList;
3
4 /**
5  * Containerclass for a single word. Can be seen as an analogous concept to a "struct" in C-style languages.
6  * @author Julian Schloeder
7  * modified: Sebastian Zittermann
8  */
9 public class Word{
10     // Start and End of the word (absolute values in respect of the whole text)
11     public int start, end;
12
13     // either "word" or "math"
14     public String type;
15
16     // empty when type=="math"
17     public String wordContent;
18
19     // empty when type=="word"
20     public LinkedList<String> mathContent;
21
22     public Word(int s, int e, String t, String wContent, LinkedList<String> mContent){
23         this.start = s;
24         this.end = e;
25         this.type = t;
26         this.wordContent = wContent;
27         this.mathContent = mContent;
28     }
29
30
31     // inString is either the representation of a PROLOG-list containing the math-elements or the word itself
32 /**
33  * Constructs a word from a String and the properties of the word, the String may be either the word itself or a
34  * representation of a Prolog-style list containing the math-elements.
35  * @param start
36  * @param end
37  * @param type
38  * @param inString
39  */
39     public Word(String start, String end, String type, String inString){
40         this.start = Integer.valueOf(start);
41         this.end = Integer.valueOf(end);
42         this.type = type;

```

```

43         if (type.equals("math")){
44             mathContent = Sentence.convertDotNotation(inString.substring(5,inString.length()-1)); //Strip "math("
45             wordContent = "";
46         }
47         else{
48             wordContent=inString;
49             mathContent=new LinkedList<String>();
50         }
51     }
52
53 /**
54  * Converts a List containing Prolog-predicates containing type, start and end of words to a List containing triples (
55  *   LinkedLists with exactly 3 elements) containing type, start end end of words; please note: has nothing to do with the
56  *   word-content.
57  * DEBUG: This is a static utility-procedure and may be moved.
58  * @param inList
59  * @return LinkedList<LinkedList<String>>
60  */
61 public static LinkedList<LinkedList<String>> convertWord(LinkedList<String> inList){
62     LinkedList<LinkedList<String>> retVal = new LinkedList<LinkedList<String>>();
63     String[] splitspace = new String[3];
64     for (String word : inList){
65         LinkedList<String> tmp = new LinkedList<String>();
66         // Assumption: types have exactly 4 characters (currently true as there are only "math" and "word")
67         splitspace = word.substring(5,word.length()-1).split(",");
68         if (word.startsWith("word"))
69             tmp.add("word");
70         else if (word.startsWith("math"))
71             tmp.add("math");
72         tmp.add(splitspace[0].trim());
73         tmp.add(splitspace[1].trim());
74         retVal.add(tmp);
75     }
76     return retVal;
77 }
78 /**
79  * Returns a String containing the content of the word, NOT the type, start or end.
80  * @return String
81  */
82 public String toString(){
83     String retVal="";
84     if (type.compareTo("word") == 0)
85         retVal = wordContent;
86     else if (type.compareTo("math") == 0)
87         retVal = "math(" + mathContent.toString() + ")";
88     return retVal;
89 }
90
91 /**
92  * Returns a string with all values
93  * @return String
94  */
95 public String showString(){
96     String retVal="";
97
98     retVal="Start: " + this.start + " End: " + this.end + " Type: " + this.type + " " + this.type + "-
99     Content: ";
100     if (type.compareTo("word") == 0)

```

```
100         retVal = retVal + wordContent;
101     else if (type.compareTo("math") == 0)
102         retVal = retVal + mathContent.toString();
103
104     return retVal;
105 }
106
107 }
```