Fachhochschule Köln

University of Applied Sciences Cologne

Praxissemesterbericht

Thema: Entwicklung des Naproche-Webinterfaces

Student:	Sebastian Zittermann
Matrikelnummer:	11046747
Referent:	Prof. Dr. phil. Gregor Büchel
Betrieb:	Mathematisches Institut der Universität Bonn
	Forschungsgruppe Naproche
Leitung:	Prof. Dr. Peter Koepke

Inhaltsverzeichnis

1 Einführung

1.1 Beschreibung

Im Rahmen des Praxissemsters wurde die Internetseite des Naproche-Projektes (siehe Kapitel 2) unter der Adresse *www.naproche.net* umstrukturiert und das Webinterface (siehe Kapitel 4 und 5.1) um viele neue Funktionen erweitert.
Besucher der Webseite haben die Möglichkeit, mathematische Beweise über das Webinterface prüfen zu lassen. Die Auswertung des Beweises ist nun besser gegliedert als in der früheren Version des Webinterfaces. Es gibt einen Debug-Modus, der primär für die Naproche-Entwickler gedacht ist. Es werden statistische Werte zu dem jeweiligen Beweis ermittelt, die aufgerufen werden können; die einzelnen Beweisschritte kann man während der Laufzeit nachvollziehen.
Ziel des Praxissemesters war es, eine stabile Webseite zu erstellen, die über die gängigen Internetbrowser funktioniert.

1.2 Gliederung

Der Bericht unterteilt sich in folgende wesentliche Bereiche: Nach der Naproche-Projektbeschreibung werden die Realisierungshilfsmittel beschrieben. Anschließend werden die Planungskonzepte dargelegt und die in der Realisierung erstellten Dateien mit ihrer Funktionsweise beschrieben. Es folgt eine Dokumentation der Tests und ein Fazit. Ein abschließender Ausblick beschreibt die in der Zukunft noch zu implementierenden Ansätze und Verbesserungsmöglichkeiten.

1.3 Anforderungen

Die folgenden Anforderungen muss die zu erstellende Internetseite erfüllen:

- Die Inhalte sollen leicht zu pflegen sein.

- Die Positionierung grundlegender HTML-Elemente sollen über eine CSS-Datei geregelt werden.
- Die frühere Version des Webinterfaces muss nach folgenden Gesichtspunkten überarbeitet werden:
  - Benutzerfreundliche Symbole, die mathematische Symbole korrekt in LaTeX-Code an der entsprechenden Position im Textfeld hinzufügen,
  - Einfache Erweiterbarkeit der Symbole,
  - Überprüfung des Textfeldinhaltes auf Formfehler vor dem Start des Beweisprüfungsmoduls,
  - Überprüfung des eingegebenen Beweises im Textfeld auf Korrektheit der LaTeX-Syntax,
  - Generieren eines PDF-Dokumentes aus dem Inhalt des Textfeldes,
  - Erzeugen einer Textausgabe mit dem syntaktischen Aufbau einer Prolog-Liste,
  - Aufruf der Überprüfung des mathematischen Inhalts des eingegebenen Beweises im Textfeld,
  - Erstellung einer geeigneten Textausgabe zur Laufzeit des Beweisprüfungsmoduls mit Kennzeichnung der Richtigkeit des Beweisschrittes,
  - Bei der Textausgabe soll es möglich sein, den ausgewerteten Satz im Eingabe-Textfeld zu markieren,
  - Ermittlung von Statistiken zu dem Beweis und den einzelnen Teilschritten.

2 Projektumfeld

Das Praxissemester fand in der Forschungsgruppe Naproche statt. Naproche steht für Natural Language Proof Checking und ist ein Projekt der Universitäten Bonn und Duisburg/Essen in Zusammenarbeit mit der Fachhochschule Köln. Hier wird

die semi-formale mathematische Sprache nach linguistischen, philosophischen und mathematischen Konzepten analysiert. Aus der Analyse wird eine „controlled natural language“ (CNL) für mathematische Texte entwickelt. Mit Hilfe dieser Sprache können Module aus Prolog einen mathematischen Beweis syntaktisch und logisch auf Korrektheit prüfen, der nach den Regeln der CNL im LaTeX-Stil verfasst wurde. Zur Prüfung wird der eingegebene Text mit Hilfe von linguistischen Methoden in ein „first-order Formular“ (Prädikatenlogik erster Stufe) umgewandelt. Das erzeugte Formular wird an einen automatischen Beweisprüfer übergeben, der schrittweise alle Argumente jeder einzelnen Formel durchgeht und die logischen Zusammenhänge zwischen den einzelnen Formeln und Axiomen herstellt. Die Dokumente, die in diesem Projekt geschrieben wurden und in denen die Einzelheiten genauer beschrieben sind, findet man in [NA09], [DK09] und [MC09].

3 Werkzeuge und Hilfsmittel

In diesem Bereich wird auf Informationen zu den verschiedenen Programmier- und Auszeichnungssprachen, sowie Dateiformate (HTML, CSS, PHP, Javascript, LaTeX, PDF), die bei der Erstellung der Website benötigt wurden, eingegangen. Weiterführende Informationen findet man unter [selfhtml].

3.1 HTML

HTML [HTML] steht für „HyperText Markup Language“ und wird für die Erstellung von Internetseiten genutzt. Die HTML-Seiten sind plattformunabhängig und können mit jedem beliebigen Texteditor erstellt werden. Diese Seiten basieren auf SGML (Standard Generalized Markup Language), welche von IBM entwickelt wurde und ist als ISO-Norm 8879 festgeschrieben.
Eine „Markup-Language“ dient zur logischen Beschreibung der Bestandteile eines textorientierten Dokuments. So ist es für jeden Leser sofort erkennbar, wenn beispielsweise eine Tabelle im Text erscheint.

HTML ist hierarchisch gegliedert. Jedes HTML-Dokument hat einen „Header“ und einen „Body“. Im Header sind HTML-Standartdaten enthalten, während im Body die Inhalte der Seite stehen. Die Inhalte des Bodys bauen sich ebenfalls hierarchisch auf. Wird beispielsweise eine Tabelle auf der Seite dargestellt, müssen die einzelnen Spalten und Zeilen entsprechend strukturiert geschrieben werden. In HTML ist es möglich, Verweise (Hyperlinks) einzubauen. Mit diesen Links gelangt man zu anderen Dokumenten, unabhängig davon, ob sie sich auf der gleichen Webseite oder auf einem anderen Server befinden. Ein Beispiel für eine HTML-Seite sieht wie folgt aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Link - Liste</title>
</head>
<body>
<h1>Eine kleine Verweissammlung</h1>
<a href="http://www.naproche.net/">Naproche</a>
<a href="http://www.fh-koeln.de/">Fachhochschule Köln</a>
<a href="http://www3.uni-bonn.de/">Universität Bonn</a>
</body>
</html>
```

Es wird eine Internetseite „Link - Liste“ erstellt. Der Inhalt des Tags <h1></h1> wird als Überschrift dargestellt. In den <a>-Tags steht die Zieladresse der Links, deren Bezeichnungen auf der Seite zu sehen ist und die durch Anklicken aufgerufen werden können.

3.2 CSS

Cascading Style Sheets [CSS] (abgekürzt CSS)-Dateien bieten die Möglichkeit, ausgewählte Elemente in einem strukturierten Dokument darzustellen. Dies funktioniert mit HTML- und XML-Dateien. Die darzustellenden Inhalte (Tags) können gezielt angesprochen und gestaltet werden. Der Sinn besteht darin, die Inhalte von der Darstellung zu trennen. So braucht ein Designer keine Ahnung von den Textinhalten zu haben, wenn der diese nur auf der Website positionieren will. Andererseits braucht sich ein Texter nicht mit dem Layout zu beschäftigen. Einige Beispiele, die auch bei der Entwicklung der Naproche-Webseite eingesetzt wurden, sind die farbliche Gestaltung der Tags, die Schriftgröße der Inhalte und die Positionierung der Tags sowie den Abstand einzelner HTML-Elemente untereinander.
Ein CSS-Beispiel zu dem im Bereich HTML-erstellten Beispiel:

```
body {
     color: black;
     background-color: white;
}
h1 {
  color: white;
     background-color: black;
     font-size: 14px;
     left: 15%;
}
```

Die Darstellung der im HTML-Dokument definierten Tags <body> und <h1> wird hier eingestellt. Im <body> ist hier die Schriftfarbe schwarz auf weißem Hintergrund, bei der Überschrift ist es umgekehrt. Die Schriftgröße der Überschrift ist 14 Pixel groß, der Abstand zum linken Rand entspricht 15% der Gesamtseitenbreite. Um sich auf die CSS-Datei zu beziehen, muss man im Header der HTML-Datei einen verweis auf die CSS-Datei ergänzen:

```
<link rel="stylesheet" type="text/css" href="beispiel.css">
<style type="text/css">
```

Es besteht außerdem die Möglichkeit, direkt in einem HTML-Dokument Stylesheets zu definieren, wobei zu beachten ist, das die Trennung von Inhalt und optischer Gestaltung nun nicht mehr gegeben ist:

```
<style type="text/css">
  body {
       color: black;
       background-color: white;
  }
 </style>
```

3.3 PHP

PHP [PHP](rekursives Akronym für „Hypertext Preprocessor“) ist eine serverseitig interpretierte Skriptsprache. Sie vereint die Vorteile von ASP (Active Server Pages - siehe [ASP]), welches auf eine einfache Kombination mit HTML-Elementen setzt und die bekannte und leistungsstarke C-Syntax, die auch in ähnlicher Form in Java und Javascript vorkommt. Hier wurde zur Realisierung PHP Version 3.6 genutzt. Ein besonderer Vorteil von PHP ist die einfache Zugriffsmöglichkeit auf Datenbanksysteme (z.B. MySQL, das genau wie PHP für nicht-kommerzielle Zwecke frei zugänglich ist). PHP wird üblicherweise bei Datenbankzugriffen und der Erstellung dynamischer Webseiten benutzt. Wird von einem Anwender eine Anfrage an eine PHP-Seite geschickt, wird der PHP-Code serverseitig interpretiert und die daraus entstandende HTML-Seite dem Anwender zur Verfügung gestellt. Ein Internet-Surfer muss also seinen Browser nicht speziell eingestellt haben, um eine PHP-Seite öffnen zu können. Ein Nachteil von PHP-Seiten ist, dass bei Aktionen auf einer Webseite immer die komplette Seite neu geladen werden muss. Eine PHP-Seite ist im Gegensatz zu Javascript [siehe Kapitel 3.4] allerdings nicht geeignet, während der Laufzeit die

Daten der Seiten zu manipulieren. Ein Beispiel für eine PHP-Seite sieht wie folgt aus:

```
<html>
  <head>
   <title>Hallo-Welt-Beispiel</title>
  </head>
  <body>
    <?
      echo "Hallo Welt!";
    ?>
  </body>
</html>
```

Es handelt sich um eine PHP-Datei, in der in speziellen Bereichen HTML auch unmaskiert eingebettet ist. Der PHP-Code wird hier zwischen den Symbolen „<?“ und „?>“ eingefügt; in diesem Beispiel wird einfach „Hallo Welt“ ausgegeben. Es existieren natürlich auch wesentlich komplexere PHP-Methoden, beispielsweise zur Stringverarbeitung.

3.4 Javascript

Javascript [Javascript] ist eine eigene objektorientierte, dynamische Programmiersprache die für die Optimierung von Webseiten entwickelt wurde. Man kann mit Javascirpt sowohl prozedural als auch funktional programmieren. Mit Hilfe von Javascript können Seiten während der Laufzeit gemäß der Aktionen des Benutzers verändert werden, wo hingegen dies bei HTML/CSS/PHP-Seiten nicht möglich ist. Javascript ist theoretisch browserunabhängig. In der Praxis haben sich aber Unterschiede gezeigt: der InternetExplorer interpretiert einzelne Javascript-Befehle anders als beispielsweise Mozilla-Firefox. So kann es passieren, dass man für ein und dieselbe logische Funktion im schlimmsten Fall für jeden Browser eine separate Funktion programmieren muss. Bei einigen Internetseiten steht daher ein Hinweis, mit welchem

Browser man am besten die Funktionen der Seite aufruft, da die Anpassung an alle Browser oft viel zu zeit- und arbeitsaufwändig ist und zudem von der Zielgruppe der Internetseite meistens nicht genutzt wird. Ein Beispiel für eine HTML-Seite mit Javascript für alle Browser sieht so aus:

```
<html>
<head><title>Test</title>
<script type="text/javascript">
alert("Hallo Welt!");
</script>
</head>
<body>
</body>
</html>
```

Beim Aufruf der Seite erscheint eine Box auf dem Bildschirm, in der „Hallo Welt!“ steht. Diese kann mit Druck auf den Button „OK“ beendet werden.

3.5 LaTeX

LaTeX [LaTeX] ist ein Texteditor, der nicht nach dem „What you see is what you get“-Prinzip arbeitet, sondern auf dem Satzsystem von TeX basiert. Dieses System ist in der Lage ohne großen Aufwand hochwertige Dokumente erstellen zu können. Speziell im wissenschaftlichen Bereich ist LaTeX daher recht verbreitet. Im Dokument selber können über Makros alle möglichen Dokumentstrukturen wie Überschriften, Fußnoten und Tabellen leicht definiert und eingefügt werden. Außerdem ist es problemlos möglich, (mathematische) Symbole und Formeln, wie z.B. „$\in$“ in einen Fließtext einzubauen. Aus jeder TeX-Datei kann unter anderem eine PDF-Datei (siehe Kapitel 3.6) erstellt werden. Hier das Beispiel im LaTeX-code aus dem Webinterface:

```
% Try this example or insert your own text.
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.
```

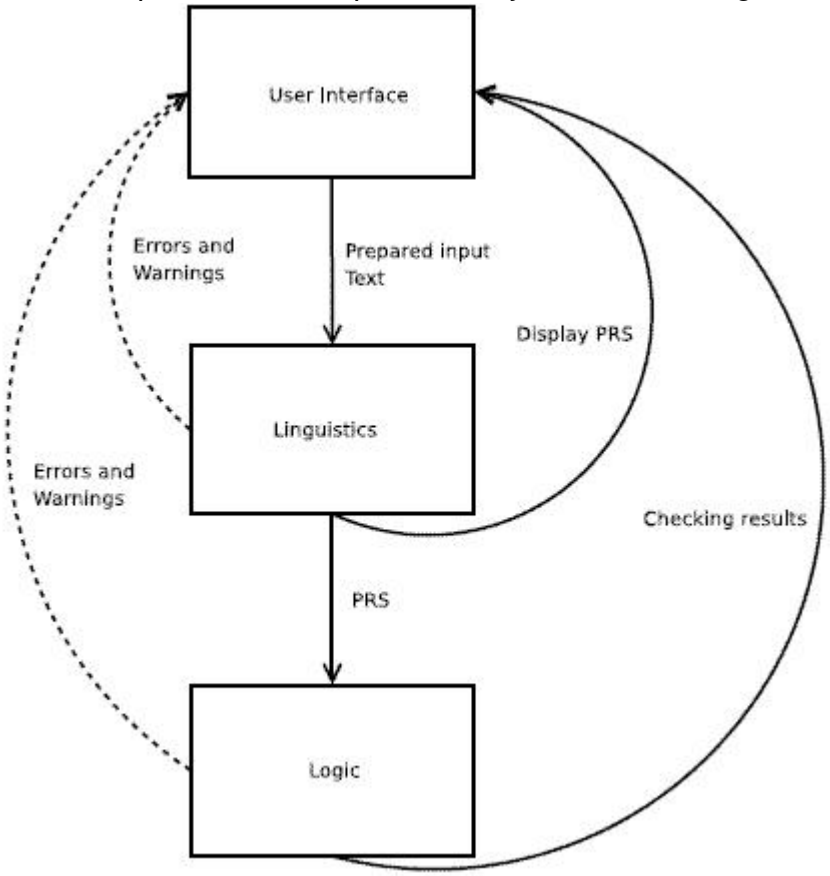
Hinter dem %-Zeichen befindet sich ein einzeiliger Kommentar, der von dem LaTeX-Compiler nicht mitkompiliert wird und im resultierenden Text nicht zu sehen ist. Nach einem Zeilenumbruch ist der Kommentar beendet. Zwischen den $-Zeichen befindet sich ein mathematischer Term, in dem hier sowohl Variablen als auch Symbole eingebunden werden. Aus dem „\in“ wird beim Kompilieren im Text ein „$\in$“.

3.6 PDF

Die Abkürzung PDF [PDF] steht für „Portable Document Format“, was soviel heißt wie transportables Datenformat. Dieses plattformunabhängige Dateiformat wurde von der Firma Adobe Systems entwickelt. Sinn und Zweck einer PDF-Datei ist, dass der Autor bewusst eine Form des Dokumentes gewählt hat, die für jeden Leser gleich ist. Es ist also egal mit welcher Hardware oder geeigneten Software man das Dokument öffnet. Eine PDF-Datei kann sowohl Texte, Bilder, Grafiken als auch Lesehilfen wie anklickbare Inhaltsverzeichnisse beinhalten.

4 Planung

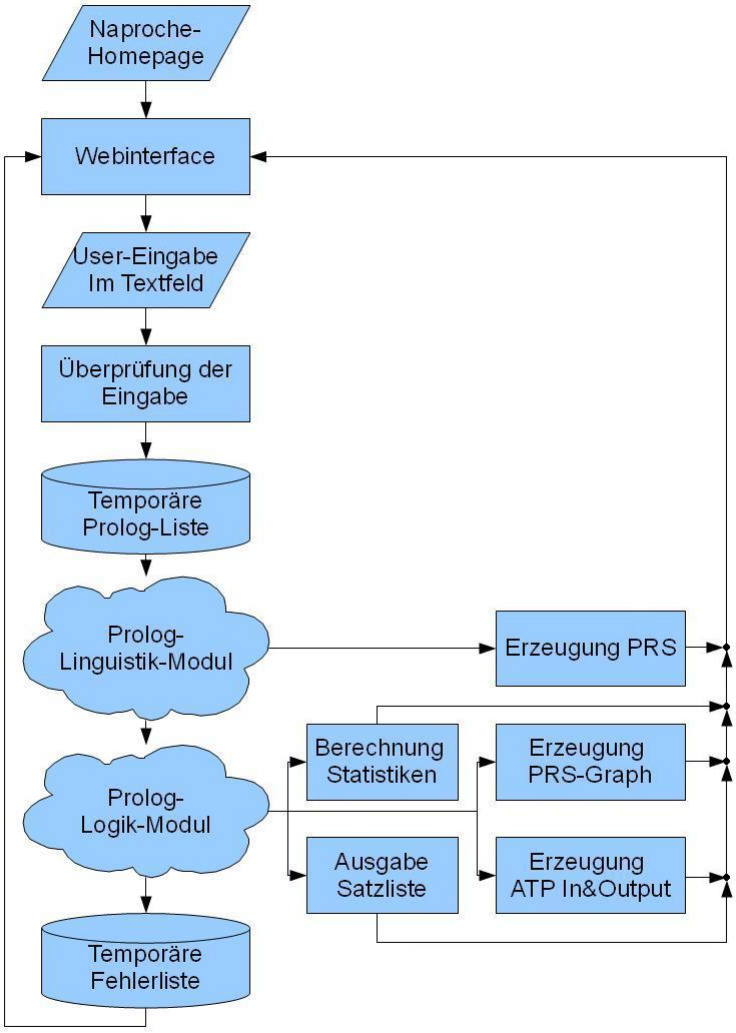
Der Ablaufplan für das Naproche-Projekt sieht wie folgt aus [NA09]:



Der Benutzer gibt einen Text ein. Bevor dieser Text an das linguistische Modul übergeben wird, wird er vorverarbeitet. Das linguistische Modul liefert dem Anwender die PRS zu dem eingegeben Text. Sollten Warnungen oder gar Fehler aufgetreten sein, wird der Anwender davon in Kenntnis gesetzt.
Nach der linguistischen Prüfung wird die PRS an das Logik-Modul übergeben. Das Ergebnis der Überprüfung wird dem Anwender ebenfalls mitgeteilt, genau wie auftretende Warnungen und/oder Fehler.

4.1 Datenflussplan

Zunächst wird hier der konkrete Datenflussplan des Naproche-Webinterfaces gezeigt und näher erläutert:



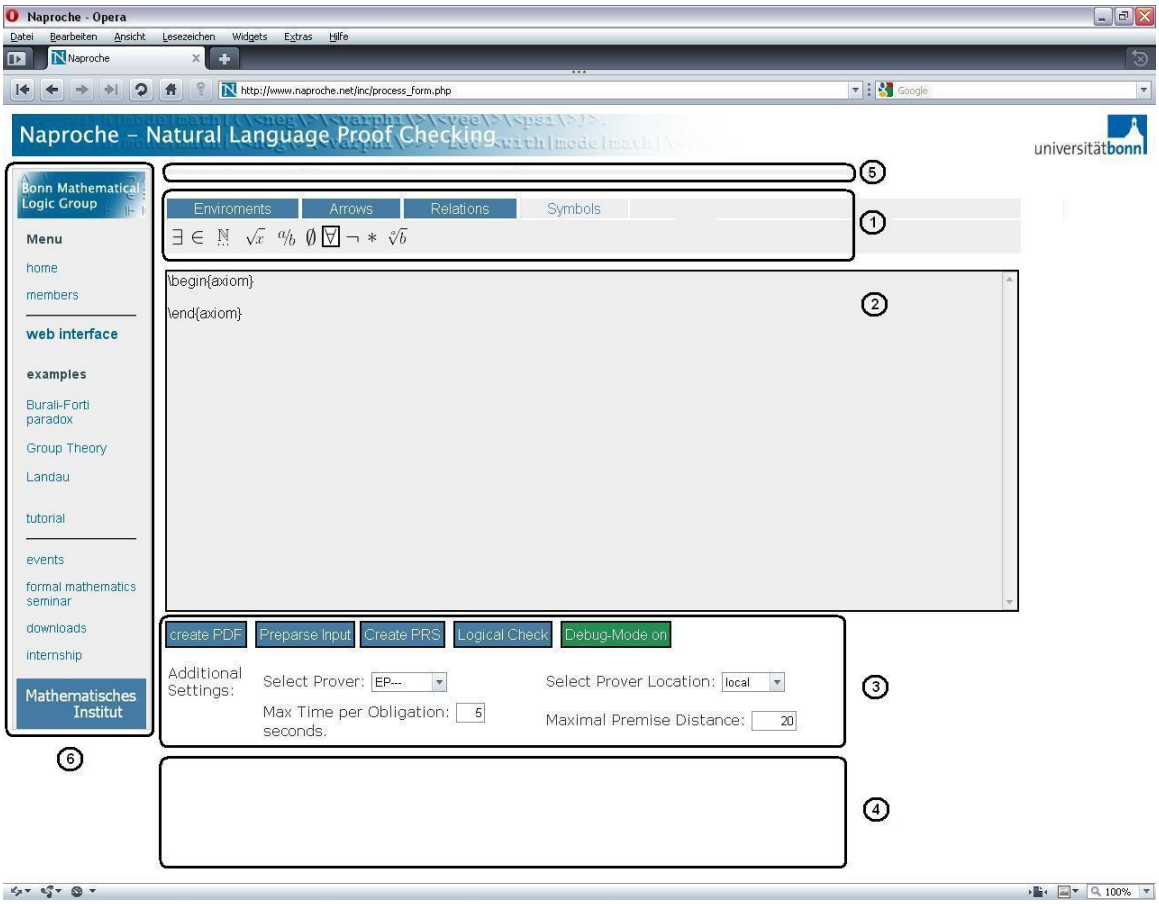
Ein Benutzer gelangt über die Seite *www.naproche.net* (siehe Kapitel 4.3) auf das Webinterface *www.naproche.net/inc/webinterface.php*. Dort findet eine Eingabe im dafür vorgesehenen Textfeld statt. Ein Screenshot des Webinterface mit Markierung des Textfeldes findet man in Kapitel 4.2. Diese Eingabe wird mit Hilfe von PHP-Funktionen überprüft. Bestandteile dieser Überprüfung (siehe Kapitel 5.1.2) sind die Einhaltung der LaTeX-Syntax für mathematische Bereiche sowie formale Kriterien für einen Beweis (z.B. \begin{} \end{} -Konstruktionen). Diese Funktionen sind genauer im Kapitel 5.1.2 beschrieben.

Aus dem Textinhalt wird über Methoden der Datei *init.php* (siehe Kapitel 5.1.1) eine Prolog-Liste erzeugt. Die Liste wird zunächst an ein Prolog-Modul übergeben,

welches den linguistischen Teil des Beweises überprüft. Dieses Modul erzeugt eine PRS (Proof Representation Structure) [NA09].
Konnte die PRS erstellt werden, findet in einem weiteren Prolog-Modul die Überprüfung der Logik statt. Zu der PRS wird dabei ein Graph erstellt. Die Rückmeldung des Moduls erfolgt mit Hilfe von „Sentence“-Objekten (siehe Kapitel 5.1.3) in Form von HTML/PHP Ausgaben auf dem Webinterface. Die einzelnen Sätze des Beweises werden ausgegeben; Informationen zu deren Überprüfung (Statistiken, ATP In-/Output [NA09]) können abgerufen werden. Eventuell auftretende Warnungen oder Fehler werden in der Datei *error.php* gespeichert, und können vom Webinterface ausgegeben werden (siehe Kapitel 5.1.6).

4.2 Webinterface

Das Naproche-Webinterface ist die Schnittstelle zu dem Anwender. Die Menüführung wurde dafür extra angepasst, so dass hier für den Anwender schnell und leicht Beispiele in das Textfeld geladen werden können. Falls Interesse besteht, kann man sich auch das Tutorial ansehen und entsprechende Beispiele erarbeiten.



Der Hauptteil der Seite ist dreigeteilt. In der oberen Zeile befindet sich die Symbolleiste (1). Wählt man zu einem bestimmten Bereich (hier: Symbols) ein Zeichen aus (hier: der Allquantor), erscheint der dafür benötigte LaTeX-Code im Textfeld an der Stelle, wo sich der Cursor vor dem Klicken befand. Die Symbole und ihre Kategorien sind leicht erweiterbar.

Den Hauptteil der Seite nimmt das Textfeld (2) ein, in den man den mathematischen Beweis eingibt. Die „Buttons“ (3), über die man die verschiedenen Funktionen ausführen kann, befinden sich unter dem Textfeld.

Die Fläche (4) unter den „Buttons“ dient während der Laufzeit zur Ausgabe weiterer Informationen. Sie passt sich nach unten hin dynamisch zur benötigten Länge der Ausgabe an.

Die oberste Fläche (5) dient zur Ausgabe von Warungen bzw. Fehlern, falls im Textfeld nach der Betätigung eines „Buttons“ ein Eingabefehler gefunden wurde.

Über die Navigationsleiste (6) kann man die Beispieldokumente (Burali-Forti-paradox,

Group Theory und Landau) ins Textfeld laden, sich das Tutorial durchlesen oder über die Navigation der Webseite zu gewünschten Seite gelangen.

4.2.1 Debug-Modus

Startet man das Webinterface, ist der Debug-Modus auf *off* voreingestellt. Um diesen einzuschalten, muss man den Button „Debug-Mode off“ betätigen, die Bezeichnung des Buttons wird anschließend genau wie der Debug-Modus endsprechend geändert. Wenn der Debug-Modus wie hier aktiv ist, sieht man alle vier anderen Buttons und die „Additional Settings“. Ist er nicht aktiv ist nur „Create PDF“, „Logical Check“ und eben der Debug-Modus Button zu sehen.

4.2.2 Create PDF

Mit den Button „Create PDF“ wird aus dem Inhalt des Textfeldes ein PDF-Dokument erstellt. Folgendes erscheint nach Betätigung des Knopfes „Create PDF“:

Die Ausgabe wird automatisch von dem LaTeX-Compiler erzeugt und bewusst ausgegeben, sodass man den Vorgang besser nachvollziehen kann. Am unteren Ende der Seite befindet sich anschliessend der Link, mit dem man sich das erzeugte PDF-Dokument ansehen kann.

4.2.3 Preparse Input

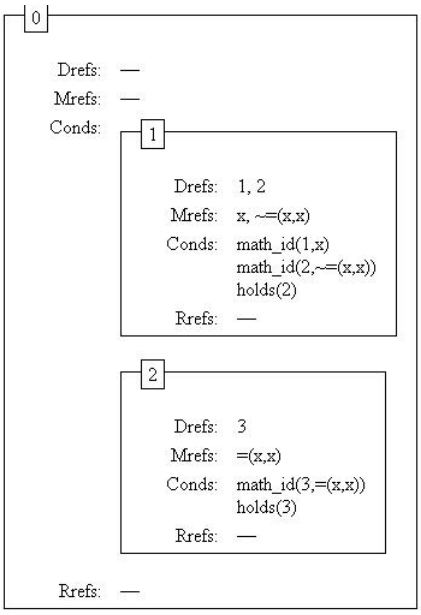
Drückt man „Preparse Input“ wird der Textinhalt in eine Prolog-Liste umgewandelt und diese auf dem Webinterface ausgegeben. Diese Prolog-Liste dient vor allem zur Vereinfachung der Prolog-Modultests. Hier sieht man die Ausgabe des Beispieltextes, der ebenfalls unter den Knöpfen angezeigt wird:

```
Prolog readable input:
[
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'])
]
```

Dieser Text kann problemlos kopiert und in jedes Prologmodul eingefügt werden um dann daran linguistische oder auch logische Tests durchzuführen.

4.2.4 Create PRS

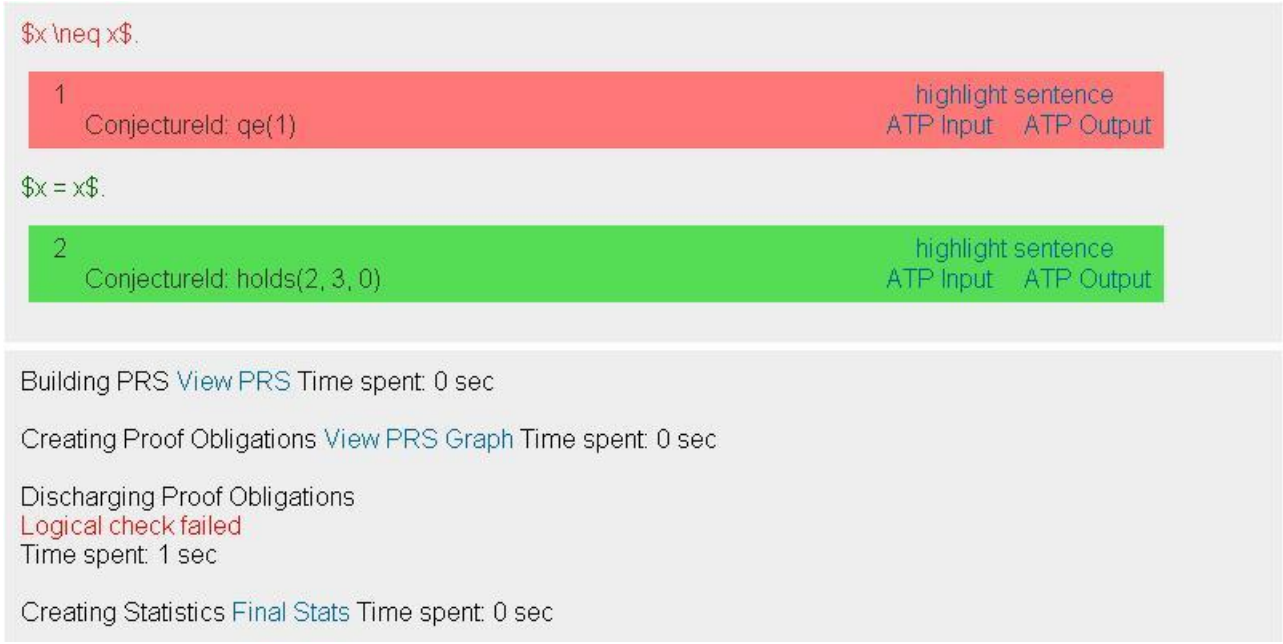
Der Textfeldinhalt wird hier zunächst in eine temporäre Prolog-Liste umgewandelt (siehe Kapitel 5.1.1). Das linguistische Prolog-Modul (siehe Kapitel 4.1) erstellt aus der Liste dann die PRS. Bei einem Textfeldinhalt von „$x \neq x$. $x = x$.“ wird aus dem Prologmodul diese PRS erzeugt und in einem separatem Browsertabulator angezeigt:



Sollten Warnungen oder Fehler beim Erstellen der PRS auftreten, werden diese oberhalb der eigentlichen Darstellung aufgelistet.

4.2.5 Logical Check

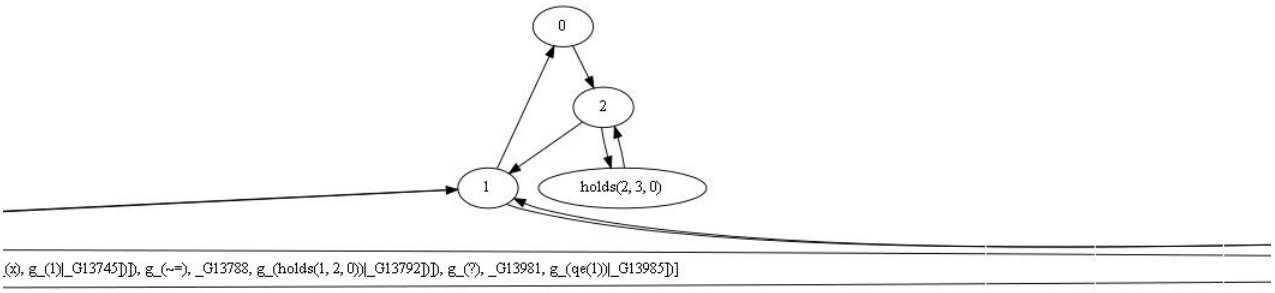
Bei einer Betätigung des Buttons „Logical Check“ bekommt das Prolog-Modul für die Überprüfung der Logik die temporäre Satzliste des Textfeldinhaltes übergeben. Zur Laufzeit werden unter dem Bereich (3) der Grafik auf Seite 16, dem „Output“-Bereich, Ausgaben erzeugt. Wenn im Textfeld der Text „$x \neq x$. $x = x$.“ steht, wird dieser Output beim „Logical Check“ erzeugt:



Zunächst wird im Outputbereich eine komplette Liste aller im Textfeld vorhandenen Sätze geschrieben. Jeder Satz, der überprüft wurde, wird nach seiner Überprüfung farblich markiert („$x = x$.“ ist grün für bestätigt, „$x \neq x$.“ ist rot für nicht nachvollziehbar) und damit verlinkt. Klickt man auf einen Satz, erscheinen die farblichen Balken mit den Links „Highlight Sentence“, „ATP-Input“ und „ATP-Output“.
Highlight Sentence markiert den Satz im Fliesstext.
ATP-In/Output sind lediglich Textausgaben, mit denen rund um ATP weiter gearbeitet werden kann. Der mit dem Prolog-Logik-Modul erzeugte ATP-Input sieht zu dem Beispiel so aus:

```
fof('qe(1)', conjecture, ?[Vd1]:(~ (Vd1 = Vd1))).
```

Unter der Satzliste stehen ebenfalls noch im „Output-Bereich“ (Bereich (4) der Grafik auf Seite 16) allgemeine Informationen zu dem kompletten Textinhalt. Man kann sich die erzeugte PRS mit einem Klick auf den Link „view Graph“ anzeigen lassen (siehe Kapitel 4.2.4), genau wie den erzeugten Graphen zur PRS:



Farblich markiert ist auch das Endresultat des Textinhaltes. In dem Beispiel steht dort in rot „Logical check failed“. Mit dem untersten Link „Final Stats“ sieht man die Gesamtstatistiken zu dem erfassten Beweis:

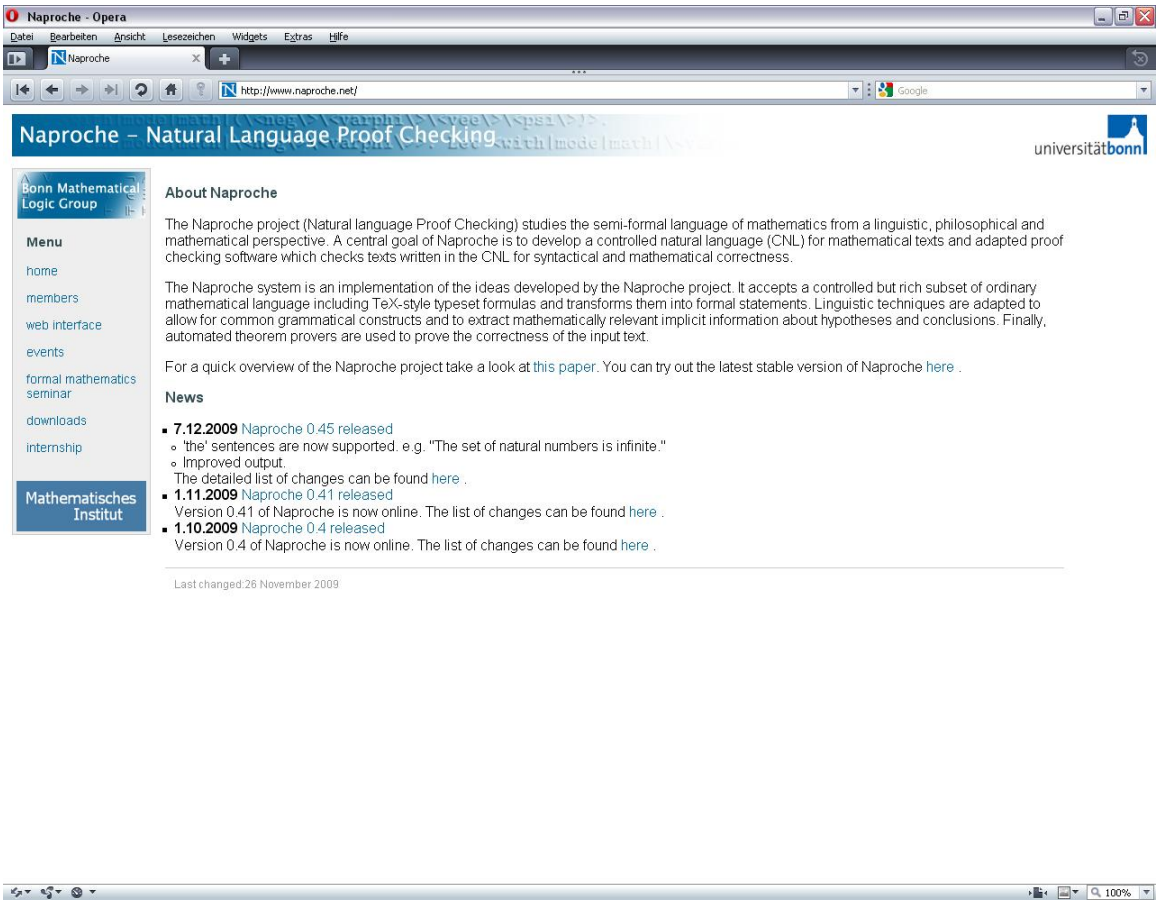
statistical analyses:

general stats	total	theorem	no proof
number of obligations	2	1	1
number of all premises in all obligations	1	1	0
average number of premises per obligation	0.5	1	0
average number of used premises per obligation	0	0	0
average number of non-used premises per obligation	0.5	1	0
average distance of premises	1	1	N/A
average distance of used premises	N/A	N/A	N/A
average distance of non-used premises	1	1	N/A
median distance of obligation	1	1	N/A
median distance of used obligation	N/A	N/A	N/A
median distance of non-used obligation	1	1	N/A
minimal distance of obligations	1	1	N/A
minimal distance of used obligations	N/A	N/A	N/A
minimal distance of non-used obligations	1	1	N/A
maximal distance of obligations	1	1	N/A
maximal distance of used obligations	N/A	N/A	N/A
maximal distance of non-used obligations	1	1	N/A

Diese Statistiken geben Auskunft darüber, wie viele Beweisschritte pro Annahme für die Überprüfung benötigt wurden. So erhält man einen kurzen und schnellen Überblick des Durchlaufs. Die Einzelstatistiken zu jedem Beweisschritt sind im Debugmodus verfügbar.

4.3 Webseitenaufbau

Um möglichst vielen Benutzern zu ermöglichen, mit dem Proof-Checker zu arbeiten, wurde entschieden, auf der Webseite *www.naproche.net* das User-Interface zu implementieren, das von der nachfolgend abgebildeten Naproche-Homepage aufgerufen wird. Im Rahmen dieser Umsetzung sollte auch die aktuelle Webseite angepasst werden:

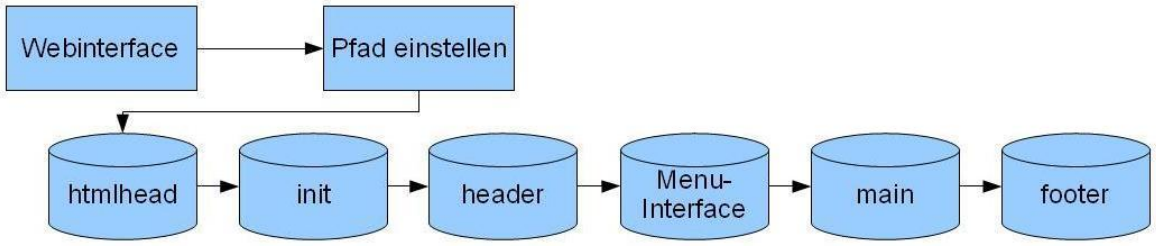


Die obere Zeile ist für das Naproche-Logo und das Logo der Universität Bonn gedacht. Links befindet sich die Navigationsleiste. Der restliche Bereich dient zur Darstellung der Inhalte des ausgewählten Menüpunktes.

5 Realisierung

5.1 Webinterface

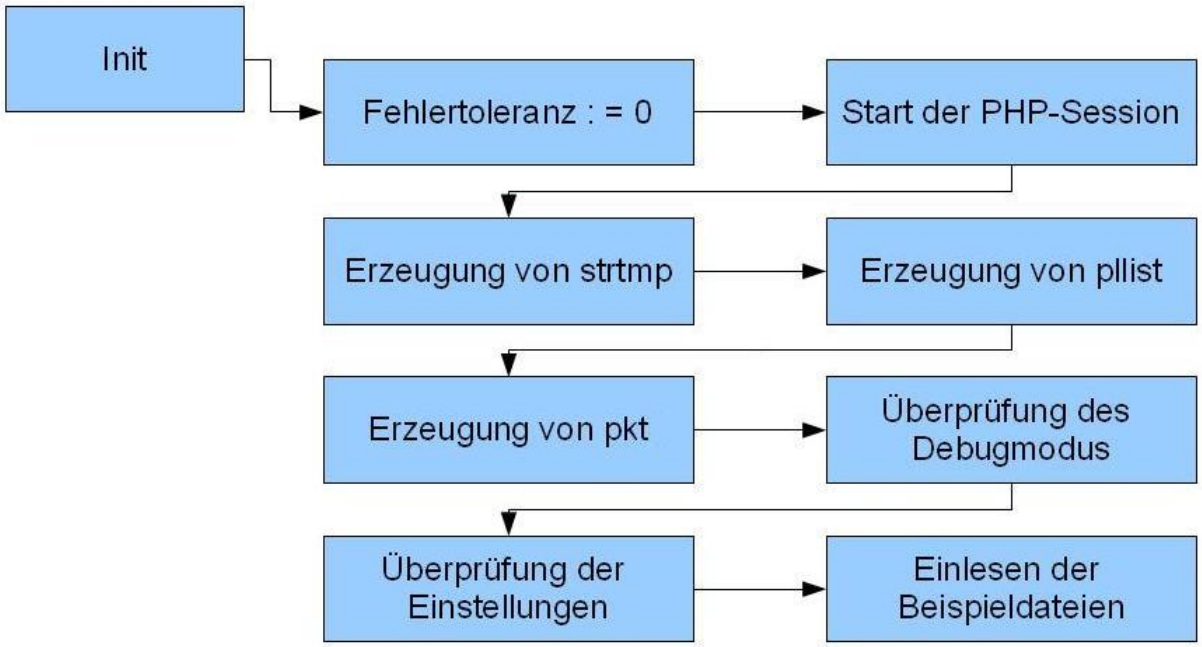
Das Webinterface wird über die Datei *webinterface.php* aufgerufen und gestartet.



In der PHP-Datei wird die Variable *pfad* an die Ordnersturktur angepasst und gesetzt. Anschließend werden die Dateien *htmlhead.php*, *init.php*, *header.php*, *menuInterface.php main.php* und *footer.php* eingebunden. Diese benötigen die Informationen des Pfades, um die HTML-Objekte mit der CSS-Datei in Verbindung bringen zu können. Ein weiteres wichtiges PHP-Modul ist die Datei *process_form.php*, die aus dem main-Modul aufgerufen wird.

5.1.1 Initialisierung der PHP-Variablen

Die Datei *init.php* initialisiert alle PHP-Variablen, die für die Verarbeitung der Inhalte der HTML-Textarea benötigt werden. Sie wird in den Dateien *webinterface.php* und *process_form.php* eingebunden.



Im Folgenden werden die einzelnen Teilschritte näher erläutert:

- Setzen der Fehlertoleranz auf 0:
  Allgemein gibt die Fehlertoleranz an, welcher Art und in welchem Umfang PHP-Fehlermeldungen auf der Webseite ausgegeben werden sollen. Der Wert „Null“ hier bedeutet, dass alle Fehlermeldungen ausgegeben werden.

- Start der PHP-Session:
  Sie stellt temporäre Werte (wie z.B. die ID der jeweiligen Session) zur Verfügung. Zu jeder Sitzung wird automatisch ein neuer Ordner auf dem Server mit der ID der Sitzung erstellt, in dem die zur Sitzung benötigten temporären Daten und Dateien geschrieben werden.

- Erzeugung der Variablen *strtmp*, *pkt* und *pllist*:
  Die Variable *strtmp* enthält den aktuellen Inhalt des Textfeldes. Wird die Seite das erste Mal aufgerufen, erscheint ein Beispieltext, der in der Initialisierung gesetzt wird. Ist bereits ein Inhalt im Textfeld vorhanden, wird dieser näher durch die dann eingebundene Datei *check.php* analysiert (siehe 5.1.2). Werden keine Fehler gefunden, werden die Kommentare in LaTeX-Syntax (% für

einzeilige Kommentare, oder /* */ für Bereiche) für die Weiterverarbeitung nicht weiter berücksichtigt.

Aus dem Inhalt des Textfeldes wird über die in PHP zur Verfügung gestellten Bearbeitungsmethoden für Variablen ein String zusammengesetzt, der einer Prolog-Liste entspricht. Für eine solche Liste gilt es folgendes zu beachten: Feststehende Ausdrücke, wie *i.e.*, in denen Punkte vorkommen, werden temporär durch Zeichenfolgen ohne Punkte ersetzt, um dann das Array *pkt* mit den Positionen aller Satz-Punkte zu füllen. Diese Liste wird mit den Positionen der Kommentare verglichen und angepasst. Weitere feststehende Ausdrücke und Symbole im Text werden durch geeignete Prolog-Syntax ersetzt (z.B. wird aus einem einfachen Hochkommata (') *succ*).

Die einzelnen Sätze werden nun separat in Prolog-Syntax umgewandelt. Eine Besonderheit in dieser Umwandlung der jeweiligen Satzinhalte stellt der spezielle Modus in LaTeX für mathematische Terme dar, der in dem LaTeX-Stil des Textfeldes durch *$ (mathematischer Inhalt) $* gekennzeichnet ist. Dieser muss für die Prolog-Module in *math(["inhalt"])* umgewandelt werden. In diesem mathematischen Bereich können auch Symbole in der üblichen LaTeX-Schreibweise (z.B. „\in“) hinzugefügt werden, die auch in der Liste entsprechend prologkompatibel in Anführungszeichen gesetzt werden. Alle anderen Zeichen werden als einzelne Variablen gewertet und automatisch voneinander getrennt. Zusätzlich werden auch alle großgeschriebenen Werte im mathematischen Beriech in Anführungszeichen geschrieben.

Die Prolog-Liste steht nun in der Variablen „pllist“ und enthält folgende Struktur (Das Beispiel ist die resultierende Prologliste aus dem Standardtext des Webinterfaces):

```
[
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'])
]
```

Diese Liste wird in der Variablen `pllist` gespeichert und steht so den Modulen zur Weiterverarbeitung zur Verfügung.

- Überpüfung des Debugs-Modus (*on* oder *off*):
  Wenn der Debug-Modus nicht explizit eingestellt wurde, ist er *off*. Der aktuelle Wert steht in der Variable *debugMode*.

- Setzen der Einstellungen (*additional Settings*):
  Standarteinstellungen der Werte sind:

  – Select Prover: EP—

  – Select Prover Location: local

  – Max Time per Obligation: 5 seconds

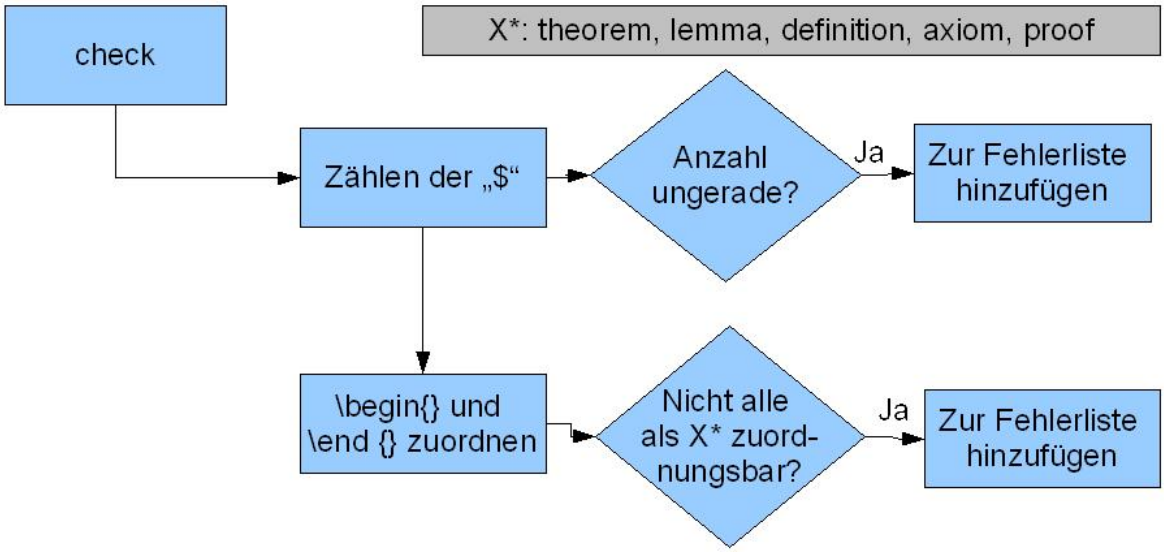
  – Maximal Permise Distance: 20

  Die Einstellungen werden in entsprechenden Variablen gespeichert. Die Werte werden an die beiden Prolog-Module übergeben, die dann entsprechend den Einstellungen ihre Berechnungen druchführen.

- Einlesen der Beispieldateien:
  Wurde ein Beispiel aus dem Menu ausgewählt, wird es in die Textfeld geladen.

5.1.2 Überprüfung der Werte

Die Datei *check.php* überprüft, ob die eingegebenen Daten des Textfeldes formale Fehler enthalten. Sie wird von der Datei *init.php* aufgerufen.



Es können Fehler aus einer ungeraden Anzahl von $-Zeichen entstehen, die den Anfang und das Ende eines mathematischen Bereiches kennzeichnen. Weiterhin können die \begin{} \end{} -Konstruktionen nicht richtig abgeschlossen worden sein. Erlaubt sind solche Konstruktionen nur, wenn es sich dabei um folgende Einträge handelt:

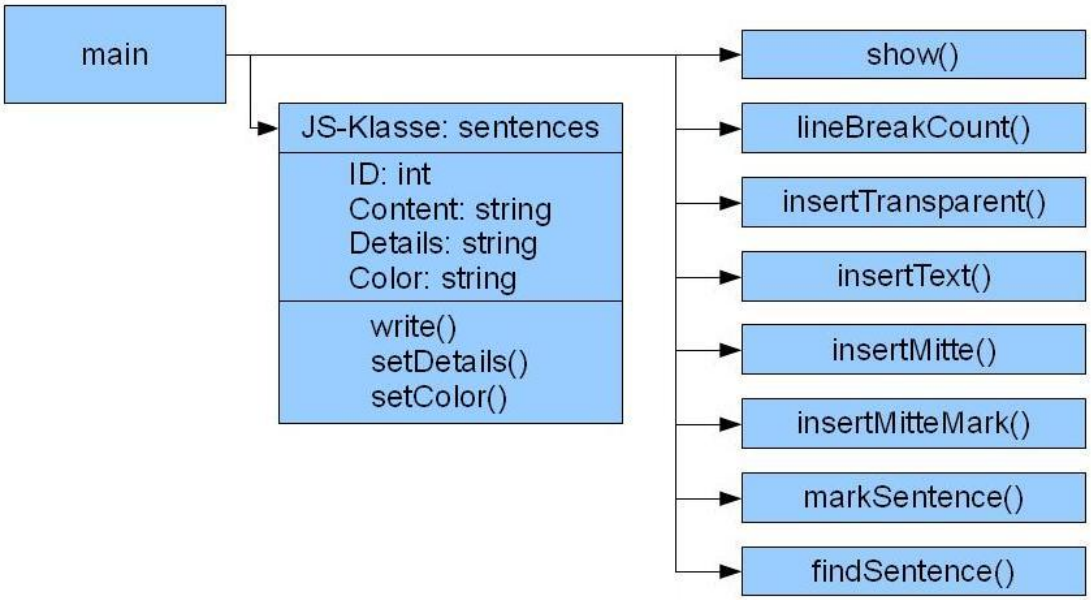
- theorem
- lemma
- definition
- axiom
- proof

Alle anderen \begin{} \end{} -Konstruktionen verursachen einen Fehler.
Neben den Fehlern gibt es noch die Möglichkeit, eine Warnmeldung auszugeben. Sollte der Gesamttext nicht mit einem Punkt abgeschlossen worden sein, wird der Bereich, der keinem Satz zugeordnet werden konnte, in einer Warnung deutlich

sichtbar erscheinen. Alle Ausgaben werden in dem Bereich (5) der Grafik von Kapitel 4.2 angezeigt.

5.1.3 Main

Die Datei *main.php* ist die sichtbare Schnittstelle zwischen den Prolog-Modulen und dem Anwender. Ein wesentlicher Bestandteil dieser Datei sind die verschiedenen Javascript-Funktionen, auf die im Programmablauf zugegriffen wird:



- Klasse *sentences*

  Attribute:

  - *ID*: Die ID ist die eindeutige Nummer des jeweiligen Satzes.
  - *content*: Der Satzinhalt, wie er in der Textarea steht.
  - *details*: HTML-Code, der die Verlinkung zu den ATP-In-/Output, Highlight Sentence und Statistiken beinhaltet, wenn der Satz überprüft wurde. Ansonsten sind die *details* leer.
  - *color*: Farbe, in der die *details* dargestellt werden sollen.

  Methoden:

  - *write()*: Schreibt das Tag „details“ des Satzes mit seiner ID im HTML-Stil auf eine Website.
  - *setDetails()*: Schreibt den übergebenen HTML-Code in das Attribut *details* und regelt die Darstellung in Abhängigkeit vom Debug-Modus. Ist der Debugmodus ausgeschaltet, sind die Details beim Aufruf von *write()* nicht sichtbar; ist er eingeschaltet sind sie sichtbar.
  - *setColor()*: Setzt das Attribut *color* auf den übergebenen Wert. Mögliche Farbwerte sind schwarz, grün und rot.

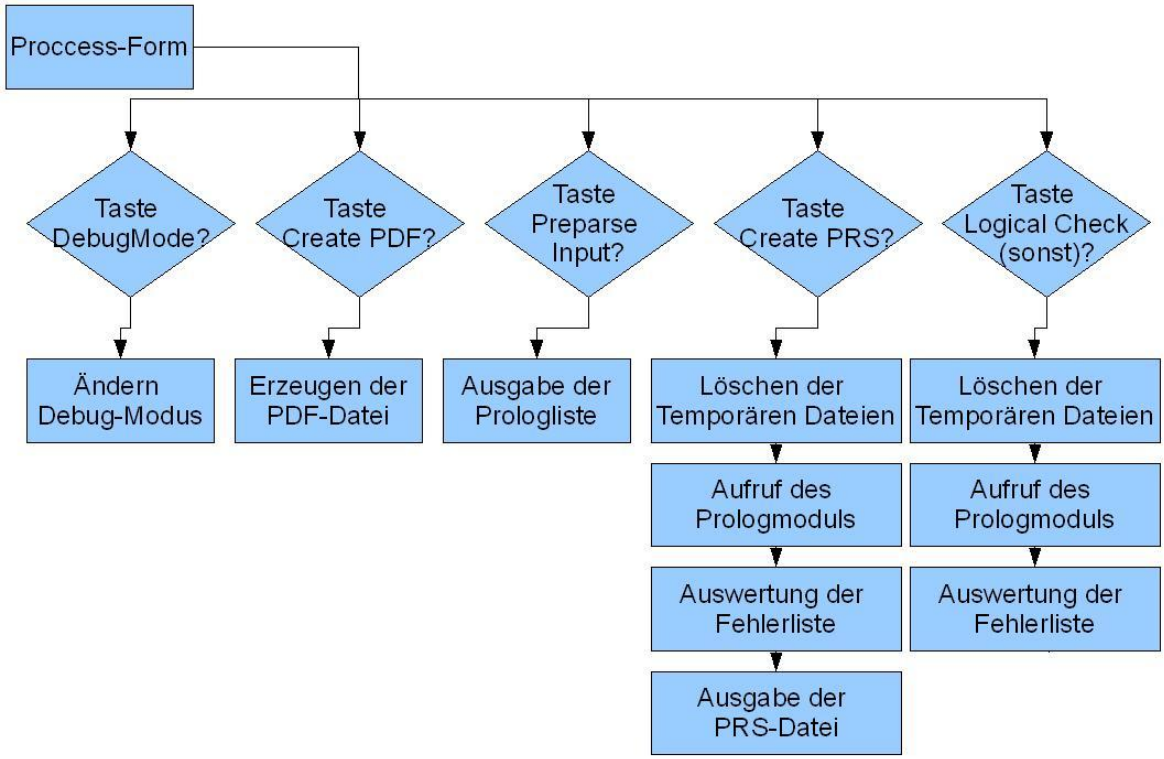
  Die einzelnen Objekte der Klasse *sentences* werden von dem Logik-Modul erzeugt. So wird nach Betätigung des Buttons „Logical Check“ zunächst die Satzliste im Outputbereich (siehe Kapitel 4.2.5) erstellt und während der Überprüfung der Beweisschritte der jeweilige Schritt eingefärbt und verlinkt.

- Funktionen:
  - *show()*: Ändert die Anzeige der *details* (sichtbar / unsichtbar) zu einem ausgewählten Satz.
  - *lineBreakCount()*: Zählt die manuellen Zeilenumbrüche in dem übergebenen String.
  - *openRibbonMenu()*: Regelt die Anzeige des Symbol-Menüs: Das aktuell markierte sieht anders aus als die nicht markierten Symbole; der Inhaltsbereich wird mit den Inhalten des markierten Bereiches gefüllt.
  - *insertTransparent()*: Fügt die transparenten Bilder im Symbolmenü ein. Je nach Funktion, die hinter dem Bild steht wenn es angeklickt wird, wird eine der drei “insert”-Methoden aufgerufen.
  - *insertText()*: Funktion für das Einfügen von Strings in das Textfeld. Der Textcursor wird hinter das letzte eingefügte Zeichen gesetzt. Beispiel: Bei einem Klick auf $\forall$ wird “\forall” eingefügt.

- *insertMitte()*: Funktion für das Einfügen von Strings in die Textarea, bei denen der Textcursor nicht ans Ende gesetzt werden soll. Beispiel: Bei einem Klick auf *Axiom* wird “\begin{Axiom}” [TextCursor] “\end{Axiom}” eingefügt.
- *insertMitteMark()*: Funktion für das Einfügen von Strings in die Textarea, bei denen der Textcursor nicht ans Ende gesetzt werden soll, und ein Teil des einzufügenden Inhalts markiert wird. Beispiel: Bei einem Klick auf $\mathbb{N}$ wird “\mathbb{” [markiertes N] “}” eingefügt.
- *markSentence()*: Funktion, um einen einzelnen Satz im Textfeld zu markieren. Es wird eine SatzID mit übergeben, die der Position des Satzes im Text entspricht. Vom Ende des letzten Punktes bis zum Ende dieses Satzes wird alles markiert. Zu beachten: In verschiedenen Browsern werden die Zeilenumbrüche unterschiedlich verarbeitet. Diese Funktion ist für den Browser “Opera” optimiert. Die Browser Firefox und Internet Explorer funktionieren mit einer gewissen Toleranz.
- *markArea()*: Funktion, um einen Bereich im Textfeld zu markieren. Übergabewerte sind der Start und der Endwert der Markierung als Position des Zeichens im Textfeld. Diese Funktion ist für den Browser „Opera“ optimiert. Die Browser Firefox und Internet Explorer funktionieren mit einer gewissen Toleranz.

5.1.4 Process-Form

Die Datei *process_form.php* wird aufgerufen, sobald ein Button im Main-Bereich (Bereich (3) im Webinterface, siehe Grafik S. 16) gedrückt wurde. Sie dient zur Weiterverarbeitung der Eingabewerte.



Zuerst wird die Datei *init.php* geladen und anschließend der Debug-Modus überprüft. Wenn die Taste des Debug-Modus gedrückt wurde, wechselt dieser von *on* nach *off* oder umgekehrt.

Wurde *Create PDF* gedrückt, wird neben dem HTML-Header auch die Datei *pdf.php* geladen, die dann die weitere Verarbeitung regelt (siehe Kapitel 5.1.5).

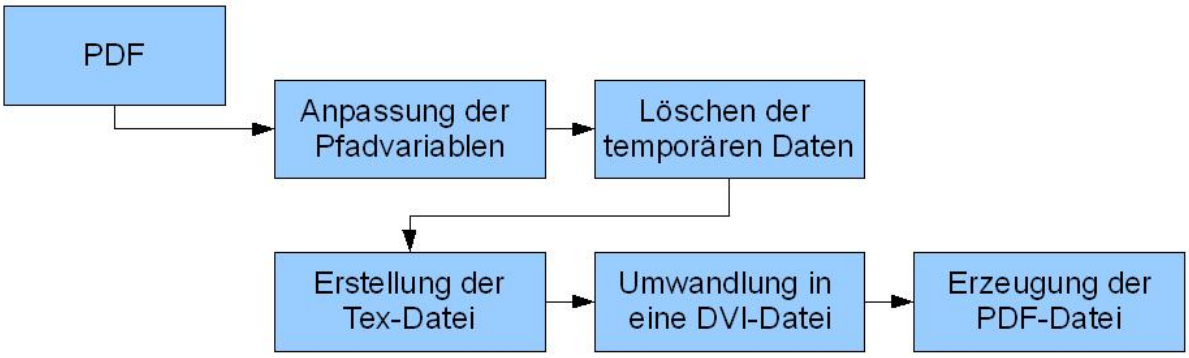
Beim Druck auf den Knopf *Preparse Input* wird die in der Datei *init.php* vorbereitete Prologliste unter dem Main-Bereich ausgegeben.

Sollte *Create PRS* das Formular ausgelöst haben, wird im Verzeichnis „tmp“ zunächst der Inhalt des temporären Ordners der aktuellen Session-ID gelöscht. Anschliessend wird ein String zusammengestellt, der hinter dem vorverarbeiteten Inhalt des Textfeldes aus der Datei *init.php* die eingestellten Werte des Formulars enthält. Das Prologmodul wird über einen System-Aufruf gestartet und bekommt den zusammengesetzten String übergeben und erzeugt daraus die Datei *prsdiv.html*. In der Datei befindet sich die erzeugte PRS in Form einer HTML-Datei, die anschliessend in einem neuen Tabulatorfenster des gleichen Browsers geöffnet wird. Ebenfalls in die neue HTML-Datei wird die Datei *prsErrorHandling.php* eingebunden, damit even-

tuell auftretende Warnungen oder Fehler angezeigt werden können. Konnte keine PRS erstellt werden, erscheint auf dem Bildschirm das Fenster, von dem aus versucht wurde, die PRS zu erstellen mit einer zusätzlichen Fehlermeldung, warum die PRS nicht erstellt werden konnte.
Wurde keine der bisher beschriebenen Tasten betätigt, wird der *Logical Check* ausgeführt. Nun werden nach dem Löschen der temporären Sitzungsdateien die Werte der Einstellungen und der Inhalt des Textfeldes an das Prolog-Modul *process_input.pl* übergeben. Das Prolog-Modul ist für die Ausgabe im HTML-Code verantwortlich (anklickbarer Fliesstext des Eingabetextes mit Verlinkung zu Statistiken, ATP In-/Output und Markierung des Satzes in dem Textfeld) und erstellt weiterhin die Datei *error.php*, die in das Script geladen wird. Sollte die Datei leer sein, findet keine zusätzliche Ausgabe statt; sollte sie nicht leer sein, wird eine Tabelle mit den aufgetretenen Fehlern inklusive Beschreibung und Fehlerart ausgegeben. Bei einem Klick auf einen Fehler in der Tabelle wird über die Methode *findSentence()* (siehe Kapitel 5.1.3) der fehlerhafte Satz im Textfeld markiert.

5.1.5 PDF

Die Datei *pdf.php* wandelt die Daten des Textfeldes in eine PDF-Datei um.

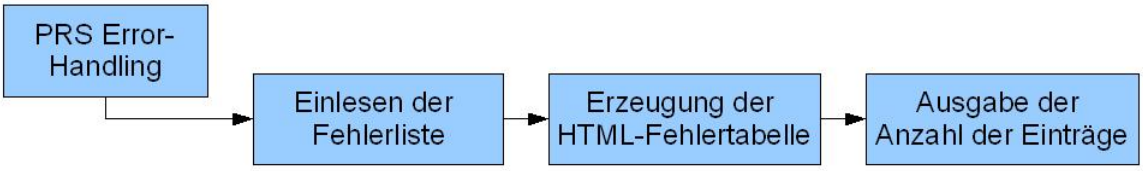


Dazu werden zunächst die Dateien eingebunden und die pfad-Variable entsprechend gesetzt, was im Kapitel 5.1.7 näher erläutert wird. Falls bereits Daten im Rahmen einer PDF-Erstellung in der Session erzeugt wurden, werden diese gelöscht. Anschliessend wir eine neue TEX-Datei mit dem Namen der ID der Session erstellt.
Die Tex-Einstellungen werden als String vor dem variablen Inhalt des Textfeldes ge-

schrieben, bevor das Dokument über einen ebenfalls konstanten String beendet wird. Es folgt das Schreiben der Inhalte in die temporäre Datei mit dem Namen *$ses*, aus der durch den PHP-Systemaufruf *system(“dvipdf “. $_SERVER[“ DOCUMENT_ROOT “] . “/naproche/tmp/“ .$ses. “/“ .$ses. “.dvi“)* eine DVI-Datei selben Namens wird. Die Ausgabe, die bei dieser Umwandlung gewöhnlich in der Konsole erscheint, wird auf der Website angezeigt. Ebenfalls sichtbar ist die Umwandlung der DVI- in eine PDF-Datei, die durch einen weiteren ähnlichen Systemaufruf erstellt wird. Die bei beiden Umwandlungen erzeugten temporären Dateien werden bis auf die PDF-Datei sofort wieder gelöscht.
Auf der Website erscheint nun deutlich sichtbar ein Link, über den man das PDF-Dokument begutachten kann. Konnte die PDF-Datei nicht erstellt werden, sieht man anstelle des Links eine entsprechende rot hervorgehobene Meldung.

5.1.6 PRS Error Handling

Die Datei *prsErrorHandling.php* analysiert die von dem Prolog-Lingustik-Modul automatisch erstellte Fehlerliste. Diese Fehlerliste wird immer erstellt, sobald das Modul aufgerufen wird; also sowohl bei *Create PRS* als auch beim *Logical Check*. Der Inhalt eines Eintrags der Liste kann entweder eine Warnung oder ein Fehler sein.



Die unter Prolog erzeugte Fehlerliste wird zunächst eingelesen. Es wird eine HTML-Tabelle mit den Spalten *Type*, *Location* und *Description* erstellt. Für jeden Eintrag der Fehlerliste wird eine Tabellenzeile angelegt; Warnungen haben eine gelbe, Fehler eine rote Hintergrundfarbe. Sollte es sich bei dem Tabelleneintrag um einen Fehler vom Typ “naproche_text” handeln, wird der Wert der “Location” näher analysiert:
Die „Location“ beinhaltet zwei Zahlenwerte. Der erste entspricht der SatzID, der zweite der Posititon des Wortes (von hinten gezählt), das den Fehler verursacht hat. Ein mathematischer Bereich, der im Textfeld durch ein öffnendes und ein schlies-

sendes $-Zeichen gekennzeichnet ist, zählt hierbei als ein Wort, unabhängig von der Anzahl der Leerzeichen, die in ihm vorhanden sind. Mit Hilfe des *pkt*-Arrays, das in der Datei *init.php* (siehe Kapitel 5.1.1) mit den Positionen der Satzanfänge gefüllt wurde, wird der fehlerhafte Satz lokalisiert. In dem Satz werden die Wörter von hinten nach vorne durchgezählt, bis das fehlerhafte Wort gefunden wurde. Der Positionswert des Anfangs und des Endes des fehlerverursachenden Wortes werden der JavaScript-Funktion *findSentence()* übergeben, die dann gezielt das Wort markiert. Sollte die Position des letzten Wortes gleich Null sein, wird das letzte Wort mit dem beendenden Punkt des Satzes markiert. Unter der Tabelle wird die Anzahl der Einträge der Fehlerliste ausgegeben.

5.1.7 Statistiken

Die Datei *stats.php* wurde ebenfalls erstellt. Sie dient zur Berechnung der Statistiken aller bzw. einzelner Obligationen. Die Methoden der Datei werden von dem Prolog-Logik-Modul beim Setzen der Details (siehe Kapitel 5.1.3) aufgerufen. Ein Screenshot der Ausgabe der Statistiken ist in Kapitel 4.2.5 abgebildet. Sie enthält folgende Funktionen:

- showStats(): Berechnet alle Statistiken über alle Obligationen.
- compareStats(): Berechnet Statistiken zu einer ausgewählten Obligation, Vergleicht diese mit Sollwerten aus einer Datei und zeigt alles anschliessend mit der Information an, wieviele Fehler gefunden wurden.
- showObligationStats(): Berechnet alle Statistiken über eine spezielle Obligation.
- getMedian(): Berechnet den Median eines übergebenen Arrays.

5.1.8 Sonstige PHP-Dateien

Folgende Dateien wurden rund um das Webinterface erstellt, die die jeweiligen HTML-Elemente definieren und anzeigen:

- htmlhead.php: beinhaltet den HTML-Header mit seinen Einstellungen
- header.php: beinhaltet die Bild-Zeile in der Webdarstellung mit den Icons von Naproche und der Uni-Bonn
- menuinterface.php: das Navigationsmenü des Webinterfaces
- menu.php: alternatives Menü innerhalb der Webseite, welches von allen PHP-Seiten eingebunden wird, die sich nicht auf das Webinterface beziehen
- footer.html: schliesst die Tags eines HTML-Dokumentes

Bei allen PHP-Dokumenten werden die HTML-Tags mit der Datei *htmlhead.php* aufgemacht und mit der Datei *footer.html* abgeschlossen. Dazwischen werden die Bilder der Kopfzeile am oberen Bildschirmrand und das Menü am linken Bildschirmrand eingefügt, wobei das Webinterface ein erweitertes Menü besitzt (siehe Grafik Kapitel 4.2 Bereich (6)).

5.2 Webseite

Die Naproche-Webseite wurde zusammen mit einer CSS-Datei (siehe Kapitel 2.3) entwickelt. Die bereits vorhandenen Inhalte der einzelnen Webseiten wurden übernommen. Die Ordnerstruktur der Webseite sieht nun wie folgt aus:

- Hauptordner
  - index.php: von der aus die Webseite gestartet wird
  - robots.txt: Einstellungen für Suchmaschinen
  - favicon: Naproche-Icon [Rahn] in der Menüzeile des Browsers
- inc: beinhaltet alle PHP/HTML-Dateien
- css: dort sind die CSS-Dateien für die Webseite und für dir PRS-Darstellung [Kolev] hinterlegt

- ex: die Webinterfacebeispiele, die über das Menü ausgewählt werden können, als TXT-Dateien
- tmp: hier werden die im Laufe einer aktuellen Sitzung benötigten temporären Dateien in einem seperaten Ordner, der den Namen der Sitzungs-ID trägt, ausgelagert
- cgi-bin: dieser Ordner beinhaltet die Prolog-Module
- download: enthält alle downloadbaren Dateien
- img: beinhaltet die Bilder der Webseite mit einem weiteren Unterordner für die Symbole der Eingabehilfe

Die schon vorhandenen Dateien wurden inhaltlich nicht verändert, aber die Verknüpfung zu der aktuellen CSS-Datei sowie der geänderten Ordnerstruktur durch eine globale dynamische PHP-Variable ($pfad) in der jeweiligen Datei angepasst. Da Beispielsweise die Datei *index.php* auf einer anderen Ordnerebene verglichen zur Datei *main.php* liegt, unterscheidet sich der Wert der Pfad-Variablen dieser Seiten. Optische Änderungen können nun in der CSS-Datei vorgenommen werden, inhaltliche in der zugehörigen Datei, die geändert werden soll. Beispiele der Pfad-Einstellungen sehen konkret wie folgt aus:

- Hauptverzeichnis: $pfad = ““
- Inc-Ordner: $pfad = “../“
- Seminar-Ordner im Inc-Ordner: $pfad = “../../“

Es wird für jede höhere Ordner-Ebene ein “../“ hinzugefügt. Es muss also nur nachgesehen werden, wo sich die Dateien befinden.

5.3 CSS

Die im Rahmen des Projektes erstellte CSS-Datei ist wie eine im Kapitel 3.2 beschriebene typische CSS-Datei aufgebaut. Sie enthält zwei verschiedene Bereiche:

Die HTML-Tags und die eigenen Tags.

5.3.1 HTML-Tags

Es werden den bekannten HTML-Elementen wie „body“ die gewünschten Werte zugewiesen:

```
body{
  font-size: 16px;
font-family: Arial, sans-serif;
color: #333333;
line-height: normal;
margin: 0px;
padding: 0px;
clear: none;
float: none;
height: auto;
width: 100%;
border-right-width: 0px;
border-right-style: none;
overflow: visible;
visibility: visible;
padding-top: 10px;
position: absolute;
}
```

Dem Body wurde eine einheitliche Schrift, eine Schriftgröße und eine Schriftfarbe zugewiesen. Neben einigen Standarteinstellung, die der Vollständigkeit halber enthalten sind (u. a. „width:100%;“ was soviel bedeutet wie „nutze die komplette zur Verfügung stehende Breite aus“), wird der Abstand nach oben auf 10 Pixel gesetzt, um ein wenig Platz zwischen dem Seitenanfang und den Logos zu haben.

5.3.2 Eigene Tags

Ein Beispiel für einen eigenen Tag ist der „Output-Bereich“ (Punkt (4) der Grafik auf Seite 16). Die Ausgaben erscheinen auf dem Webinterface, wenn einer der „Buttons“ (außer dem Debug-Mode) gedrückt wurde. Dieser Ausgabebereich ist wie folgt gestaltet:

```
div#output {
width: 95%;
height: 100%;
min-height:0px;
min-width: 504px;
margin: 5px 0 0 0;
padding: 10px 0 10px 10px;
color: #000000;
background-color:#EEEEEE;
}
```

Aus optischen Gründen wird die volle Höhe, aber nur 95 Prozent der Breite ausgenutzt, wobei die mindestbreite 504 Pixel beträgt. Die Außen- und Innenabstände sind ebenso wie die Schrift- und Hintergrundfarben definiert.
Weiterhin werden die Tags, die durch die erzeugten HTML-Ausgaben des Logik-Prolog-Moduls entsprechen (z.B. „green“), ebenfalls in dieser CSS-Datei definiert:

```
#green{
float:left;
color: #008800;
}
```

6 Browserkompabilitätstest

Die Webseite mit Ihrem Webinterface ist im Laufe der Praxissemesterzeit recht komplex geworden. Das zwischenzeitlich erhoffte Ziel, komplett auf Javascript zu verzichten, konnte durch die in der Zwischenzeit erweiterten Funktionen der Prolog-Module nicht erreicht werden, was einige Konsequenzen hat: Die Bereiche, die ausschließlich aus einer Kombination aus HTML, CSS und PHP bestehen, funktionieren in allen gewünschten Browsern. Sobald Javascript dazu kommt, verhalten sich fast alle Browser unterschiedlich:

- Opera (Pflicht): Der Browser mit einem voll funktionsfähigem Webinterface.
- Mozilla Firefox(Pflicht): Sobald man über eine erzeugte Ausgabe im „Outputbereich“ (4) versucht, einen ausgewählten Satz in dem eingegebenen Textfeld zu markieren, können Fehler auftreten. Diese Fehler sind abhängig von der Verarbeitung der Zeilenumbrüche. Es gibt zwei Arten von Zeilenumbrüchen: den erzwungenen (Betätigung von Enter) und den automatischen Umbruch (Zeile bis zum Ende geschrieben, nachfolgender Text wird in der nächsten Zeile angezeigt). Die Kombination aus beiden verursacht Schwierigkeiten, die leider nicht immer reproduzierbar waren. Dadurch ist diese Markierungsfunktion für den Firefox optimierungsbedürftig. Die restliche Funktionalität ist vorhanden.
- Safari (optional): Funktioniert genauso wie der Firefox.
- InternetExplorer (optional): Hier unterscheiden sich auch die einzelnen Versionen von einander, obwohl es sich um ein und den selben Quelltext handelt:
  - IE 6: In dem Browser macht schon das Symbol-Menu Probleme. Die selbsterstellten Icons sind deutlich zu erkennen, die anderen jedoch nicht. Klickt man darauf, passiert auch das Gewünschte (Einfügen eines Textes im Textfeld), aber der Coursor springt zurück zum ersten Zeichen in diesem Textfeld.

- IE 7: Die Probleme des Symbol-Menus sind geringer geworden. Alle Icons sind deutlich zu erkennen, bei einem Klick darauf werden sie auch eingefügt, aber nicht immer an der gewünschten Stelle. Die Funktionsknöpfe unter dem Textfeld sind vorhanden, aber nicht lesbar. betätigt man diese wird offensichtlich die maximalbreite der Seite nicht mehr voll ausgenutzt; aber dafür sind die Knöpfe jetzt lesbar.
- IE 8: In der 8er-Version hat das Symbol-Menu die selben Probleme wie die 7er-Version des IE. Die Knöpfe sind hier deutlich zu erkennen, aber bei Betätigung eines Knopfes verschwindet der Scrollbalken am rechten Rand. Die Folge ist, dass man nur mit Hilfe eines Markierungstricks zum unteren Ende der Seite gelangen kann.

Wer den vollen Umfang des Webinterfaces mit präzisen Markierungen haben möchte, kommt um den Browser Opera nicht herum. Im Firefox und Safari bietet die Seite auch einen einsetzbaren vernünftigen Funktionsumfang. Der Internetexplorer ist leider nicht für diese Seite geeignet.

7 Ausblick

In Zukunft, wenn auch die Prologmodule weiterentwickelt wurden, sollte man die komplette Struktur des Webinterfaces entsprechend anpassen und eventuell geeignetere Methoden zur Darstellung diverser Ausgaben der Prolog-Werte finden.
Eine mögliche und sinnvolle Arbeit wäre die Anpassung des Javascriptcodes an die verschiedenen Browser, was im zeitlichen Rahmen des Praxissemesters nicht möglich war.

8 Fazit

Das Praxissemester verlief organisatorisch und inhaltlich sehr erfreulich. Bei den gemeinsamen Besprechungen der Arbeitsabläufe habe ich verschiedene interessante Themengebiete der Mathematik kennengelernt, die mir bis jetzt im Rahmen des Studiums noch nicht begegnet sind.
Die Aufgaben des Webinterfaces wuchsen parallel zur Weiterentwicklung der beiden Prologmodule. Dabei mussten einige Kompromisse (z.B. der Gebrauch von Javascript) eingegangen werden, um trotzdem eine ansprechende und benutzerfreundliche Ausgabe erzeugen zu können.

9 Quellen

- [NA09] M. Cramer, P. Koepke, D. Kühlwein, and B. Schröder, The Naproche System, paper for the Calculemus 2009.
- [DK09] Daniel Kuehlwein, Diploma thesis, A calculus for Proof Representation Structures, explains the theoretical background of the checking algorithm 2009.
- [MC09] Marcos Cramer, Master thesis, Mathematisch-logische Aspekte von Beweisrepräsentationsstrukturen, explains the mathematical and logical aspects of Proof Representation Structures 2009.
- [Kolev] Nickolay Kolev, Magister thesis, Generating Proof Representation Structures in the Project NAPROCHE, defines PRSs and explains how to create them from a Naproche text 2008.
- [Rahn] Mona Rahn Internship Report 2009
- [selfhtml] http://de.selfhtml.org/; SELFHTML: Version 8.1.2 vom 01.03.2007
  - [HTML] http://de.selfhtml.org/html/index.htm; HTML
  - [CSS] http://de.selfhtml.org/css/index.htm; CSS

  - [PHP] http://de.selfhtml.org/php/index.htm; PHP
  - [Javascript] http://de.selfhtml.org/javascript/index.htm; Javascript

- [LaTeX] http://de.wikipedia.org/wiki/LaTeX; 11. Januar 2010
- [PDF] http://de.wikipedia.org/wiki/PDF; 10. Januar 2010