# Attempto Controlled English für Amazon Alexa

## Bachelorarbeit

Julia Kübert

| | |
|---|---|
| **Betreuer:** | Prof. Dr. Dietmar Seipel<br>M.Sc. Falco Nogatz |
| **Lehrstuhl:** | Lehrstuhl für Informatik I<br>(Effiziente Algorithmen und wissensbasierte Systeme) |
| **Institut:** | Institut für Informatik |
| **Abgabedatum:** | 13. August 2018 |

## Zusammenfassung

In dieser Arbeit wird eine spezielle Anwendung von Natural Language Processing (NLP), also der Kommunikation zwischen Mensch und Maschine in natürlicher Sprache, behandelt.

Dabei besteht das Ziel darin, mit Amazon Alexa möglichst intuitiv zu sprechen, ohne dabei auf wenige umsetzbare Anwendungsfälle beschränkt zu sein. Um die natürliche Sprache dabei verarbeiten zu können, wird Attempto Controlled English (ACE), eine an der Züricher Universität entwickelte Controlled Natural Language, verwendet. Die von einem Benutzer über eine Amazon-Schnittstelle eingegebenen Informationen sollen mit Prolog so gespeichert und logisch verarbeitet werden können, um auch neues Wissen aus bereits abgelegten Fakten und Regeln abzuleiten.

Diese Arbeit bietet für die genannte Zielsetzung eine erste Lösungsstrategie und einen Ausblick in zukünftige verwandte Problemstellungen.

# Inhaltsverzeichnis

**1. Einleitung** **1**
1.1. Motivation . . . . . 1
1.2. Verwandte Forschung . . . . . 2
1.2.1. Natural Language Processing . . . . . 2
1.2.2. Controlled Natural Languages . . . . . 2
1.2.3. Attempto Controlled English . . . . . 3
1.2.4. Sprachassistenzsysteme . . . . . 3
1.2.5. Entwicklung eines Alexa Skills mit Prolog . . . . . 3
1.3. Zielsetzung . . . . . 4
1.4. Aufbau der Arbeit . . . . . 5

**2. Technologien** **7**
2.1. Amazon Alexa . . . . . 7
2.2. Prolog . . . . . 9
2.2.1. Allgemeine Daten und Entstehung . . . . . 9
2.2.2. Arbeitsweise . . . . . 9
2.2.3. Verwendete Funktionalitäten von SWI-Prolog . . . . . 10
2.2.4. Prolog als Programmiersprache für NLP . . . . . 11
2.3. Attempto Controlled English (ACE) . . . . . 12
2.3.1. Allgemeine Daten und Entstehung . . . . . 12
2.3.2. Zielsetzung . . . . . 12
2.3.3. Funktionsweise von Attempto Controlled English . . . . . 13
2.3.4. Konventionen und Einschränkungen der Sprache . . . . . 15
2.4. Attempto Controlled Englisch Reasoner (RACE) . . . . . 16

**3. Funktionsweise** **19**
3.1. Amazon Alexa . . . . . 21
3.1.1. Entwicklung des Skills . . . . . 21
3.1.2. Ausführung des Skills . . . . . 23
3.2. Prolog . . . . . 24
3.2.1. Prolog Webserver . . . . . 25
3.2.2. Prolog Model . . . . . 26
3.2.3. Prolog *library(race)* . . . . . 29

3.3. ACE Reasoner (RACE) . . . . . . . . . . . . . . . . . . . . . . . . . 30

**4. Evaluierung** **33**
4.1. Qualitätsziele . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.2. Einschränkungen . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**5. Zusammenfassung und Ausblick** **37**
5.1. Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.2. Ausblick . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

**Literaturverzeichnis** **41**

**Erklärung** **45**

**A. Appendix** **i**
A.1. Bildquellen aus Abbildung 3.1 . . . . . . . . . . . . . . . . . . . . . . i
A.2. Wichtige Teile des Programmcodes
der entwickelten Anwendung ***AlexaACE*** . . . . . . . . . . . . . . . i
A.2.1. Extrahieren der Informationen des JSON-Inputs . . . . . . . i
A.2.2. Bearbeitung der einzelnen Intents . . . . . . . . . . . . . . . . ii
A.2.3. Wichtige Hilfsprädikate . . . . . . . . . . . . . . . . . . . . . v

# Abkürzungsverzeichnis

ACE ................ Attempto Controlled English
APE ................ Attempto Parsing Engine
API ................ Application Programming Interface
AVS ................ Amazon Voice Service
CNL ................ Controlled Natural Language
DRS ................ Discourse Representation Structure
DRT ................ Discourse Representation Theory
DSL ................ Domain-Specific Language
GPL ................ General-Purpose Language
HTTP .............. HyperText Transfer Protocol
JSON ............... JavaScript Object Notation
NLP ................ Natural Language Processing
RACE .............. Attempto Controlled English Reasoner
RPC ................ Remote Procedure Call
SOAP .............. Simple Object Access Protocol
SSL ................ Secure Socket Layer
URL ................ Uniform Resource Locator
XML ............... Extensible Markup Language

# Abbildungsverzeichnis

2.1. Bestandteile einer beispielhaften Eingabe in Amazon Alexa . . . . . 8
2.2. Darstellung eines ACE Textes in DRS . . . . . . . . . . . . . . . . . . 14
2.3. Ausgabe der Interpretation eines ACE Textes als Paraphrase . . . . 15
2.4. Überprüfen eines Theorems auf Basis vorhandener Axiome mit RACE 16

3.1. Funktionsablauf der Anwendung ***AlexaACE*** . . . . . . . . . . . . . 20
3.2. Erstellen des Interaction Model von ***AlexaACE*** . . . . . . . . . . . 22
3.3. Ausführen von ***AlexaACE*** über die Testkonsole von Amazon . . . . 24

# Verzeichnis der Quellcodes


2.1. Grundstruktur eines Prolog Webservers . . . . . . . . . . . . . . . . 10

3.1. Erstellung des Skills mithilfe des JSON Editors . . . . . . . . . . . . 22
3.2. JSON Input einer Nutzereingabe . . . . . . . . . . . . . . . . . . . . . 23
3.3. Einrichten eines Proxy-Servers mit nginx . . . . . . . . . . . . . . . . 25
3.4. Prolog Webserver in alexa_mod.pl . . . . . . . . . . . . . . . . . . . . 26
3.5. Umsetzung des Intents „prove“ in alexa_mod.pl . . . . . . . . . . . . 27
3.6. Request an den RACE Webservice für eine beispielhafte Eingabe . . 30
3.7. Reply des RACE Webservices für eine beispielhafte Eingabe . . . . . 31

A.1. Extrahieren der Informationen des JSON-Inputs in alexa_mod.pl . . i
A.2. Umsetzung des Intents „remember“ in alexa_mod.pl . . . . . . . . . ii
A.3. Umsetzung des Intents „question“ in alexa_mod.pl . . . . . . . . . . iv
A.4. Umsetzung nicht zuordenbarer Intents in alexa_mod.pl . . . . . . . v
A.5. Wichtige Hilfsprädikate in alexa_mod.pl . . . . . . . . . . . . . . . . v

# 1. Einleitung

## 1.1. Motivation

Seit ca. Mitte des vergangenen Jahrhunderts gibt es Versuche, Programme zu schreiben, die die Interaktion zwischen dem Menschen und dem Computer in natürlicher Sprache ermöglichen bzw. erleichtern sollen [21]. Eine der größten Herausforderungen des sogenannten *Natural Language Processing* (NLP) stellt dabei, neben dem Erkennen einzelner Wörter aus gesprochenem Text, das semantische Verständnis der Sprache dar [1]. Über die Jahre wurden unterschiedliche Techniken und Sprachen entwickelt, die die natürliche Sprache beispielsweise im Bezug auf Grammatik oder Vokabular eingrenzen und somit eine Analyse und ein Sprachverständnis ermöglichen. Eine solche Sprache wird als *Controlled Natural Language* (CNL) [31] bezeichnet. Eine CNL kann dabei auf ein bestimmtes Problemfeld reduziert sein, wobei sie dann *Domain-Specific Language* (DSL) genannt wird. Kann sie auch in mehr als einem einzigen Bereich angewendet werden, so handelt es sich bei dieser CNL um eine *General-Purpose Language* (GPL) [28]. Ein bekanntes Beispiel dafür ist das an der Universität Zürich entwickelte *Attempto Controlled English*[1] (ACE) [10, 15], welches englische Sätze logisch verarbeiten kann. Jene kann unter anderem mit Prolog[2] [26, 30], einer deklarativen Logikprogrammiersprache mit Top-Down Evaluation, die sich vor allem im Themengebiet des NLP bewiesen hat [20, 27], verwendet werden.

Eine weitere Herausforderung beim NLP ist die Spracherkennung von gesprochenem Text. Hierfür bedarf es Techniken, die es ermöglichen natürliche Sprache zu erfassen und schriftlich festzuhalten [1], sodass sie z. B. durch eine CNL weiter verarbeitet werden kann [28]. Alexa, der Sprachdienst des Unternehmens Amazon, zählt zur Zeit zu den verbreitetsten und am häufigsten genutzten Sprachdiensten [23]. Amazon Developer[3] erlaubt es, über die Standardimplementierung hinaus, sogenannte *Skills* zu entwickeln, mit denen es unter anderem möglich ist, eine spezielle Form des NLP in Verbindung mit Prolog zu entwickeln [2, 24].

[1]Webseite des Attempto Projekts: `http://attempto.ifi.uzh.ch/site/`
[2]SWI-Prolog: `http://www.swi-prolog.org/`
[3]Amazon Developer Plattform: `https://developer.amazon.com/de/alexa`

Das Ziel dieser Bachelorarbeit ist es, eine Controlled Natural Language, im Speziellen ACE, in Prolog einzubinden, sodass sie über Alexa ansprechbar ist. Dadurch soll ermöglicht werden, über Alexa in natürlicher Sprache zu kommunizieren und den vollen Funktionsumfang von Prolog auszunutzen.

## 1.2. Verwandte Forschung

Zu den verschiedenen in dieser Arbeit verwendeten Technologien und erwähnten Gebieten gibt es eine Vielzahl von verwandten Forschungen, wobei keine den speziellen, oben geschilderten Fall betrachtet. So existieren beispielsweise verschiedene Forschungsarbeiten, die sich mit dem Themengebiet des Natural Language Processing im Allgemeinen oder Speziellen beschäftigen. Auch zu Controlled Natural Languages bzw. auch vorhandenen CNL, wie Attempto Controlled English, gibt es eine Vielzahl an Vorarbeiten. In den vergangenen Jahren haben sich auch Sprachassistenzsysteme wie Amazon Alexa für die Forschung etabliert. Des Weiteren existiert bereits ein Projekt über die Entwicklung eines Alexa Skills mit Prolog, worauf im Abschnitt 1.2.5 näher eingegangen wird.

### 1.2.1. Natural Language Processing

Es gibt mehrere Arbeiten, die sich mit der geschichtlichen Entwicklung auseinandersetzen und die Begrifflichkeiten von NLP einführen bzw. differenzieren. So wird in [1] beispielsweise der Unterschied zwischen der Spracherkennung („speech recognition“) und dem Verständnis des Sinns einer Sequenz von Wörtern („language processing“) geklärt. Daneben werden hier auch die Herausforderungen, die sich beim NLP ergeben, aufgezeigt. Die Arbeiten [5, 18] setzen sich damit auseinander, wie man natürliche Sätze logisch darstellen kann. Weitere Forschungen beschäftigen sich mit dem Lösen von Problemen im Bereich NLP mithilfe der Programmiersprache Prolog. Dazu zählt unter anderem [3]. Dabei wird auch darauf eingegangen, warum Prolog sich besonders für Natural Language Processing eignet [26] und welche Möglichkeiten es dabei gibt Prolog einzusetzen [27]. Als Beispiel lässt sich auch [22] nennen, da hier eine auf Prolog basierende Anwendung von IBM präsentiert wird.

### 1.2.2. Controlled Natural Languages

In den Forschungsarbeiten über Controlled Natural Languages, wie beispielsweise [31], wird meist erklärt, was eine CNL ausmacht und was die Zielsetzungen dieser sind. Darüber hinaus wird versucht, diese möglichst genau zu definieren und in

Untergruppen zu klassifizieren [21], welche sich z. B. in der Basissprache oder dem Anwendungsgebiet unterscheiden. Oft werden auch explizit Beispiele für Controlled Natural Languages genannt und eine kurze Einführung gegeben, wie in [28, 29] Attempto Controlled English.

### 1.2.3. Attempto Controlled English

Zu ACE existieren viele Forschungsarbeiten, wobei diese unterschiedliche Aspekte der speziellen CNL beleuchten. In [15] wird zunächst die Motivation zur Entwicklung von ACE genannt, sowie die Arbeitsweise des Systems und ein Überblick über die Sprache. Dabei wird vor allem darauf eingegangen, wie ACE Texte in andere Strukturen umgewandelt werden, sodass die Informationen daraus logisch verarbeitet werden können [16]. In den verschiedenen bereitgestellten Manuals [13, 17] wird genau auf die Syntax, insbesondere die Konstruktionsregeln und Konventionen, eingegangen. Dabei werden außerdem die Verbesserungen des Systems zwischen den einzelnen Versionen deutlich. Zudem werden verschiedene Einsatzgebiete von ACE vorgestellt, wie als „Semantic Web Language“ [19], oder als Sprache für „Knowledge Representation“ [10]. Weitere Forschungsarbeiten im Themenbereich Attempto Controlled English gehen auf neu bereitgestellte Tools [11] ein, beispielsweise den Reasoner [8, 9], der unter anderem Theoreme anhand bekannter Axiome ableiten kann. Dieser kann unter Prolog mithilfe einer Bibliothek aufgerufen werden [25], welche auch in dieser Arbeit genutzt wird.

### 1.2.4. Sprachassistenzsysteme

Im Zusammenhang mit Natural Language Processing und der Spracherkennung gibt es auch interessante Literatur zum Thema Sprachassistenzsysteme. Neben Arbeiten, die verschiedene Assistenzsysteme gegenüberstellen [23] und deren jeweilige Vor- und Nachteile evaluieren, wird auch erklärt, dass man diese Systeme selbst durch eigene Programme ergänzen kann. Hierbei wird darauf eingegangen, was für eine Erweiterung des Angebots von Smarthome Geräten [7], beispielsweise mit Amazon Alexa, beachtet werden muss. Zudem wird in [4] ein Anwendungsbereich genannt, in dem eigens entwickelte Programme eingesetzt werden können.

### 1.2.5. Entwicklung eines Alexa Skills mit Prolog

Zwar existiert auch bereits ein kleines Projekt [2, 24] über die Entwicklung von Alexa Skills mit Prolog, dieses ist allerdings lediglich ein Beispiel mit wenigen, manuell erstellten denkbaren Regeln und Fakten. Durch diese Einschränkung können nur

vorher festgelegte Sätze über eine Amazon-Schnittstelle eingegeben und in Prolog abgespeichert werden. Weicht man von den möglichen Eingaben ab oder gibt einen komplett anderen Inhalt ein, kann dies das Programm nicht verarbeiten und wirft einen Fehler. Dementsprechend ist es damit noch nicht möglich, natürliche Sprache direkt zu verarbeiten und die resultierende Datenbasis weiterzuverwenden.

## 1.3. Zielsetzung

Die Arbeit setzt sich zum Ziel, die im Abschnitt 1.2.5 beschriebene Beschränkung auf eine kleine Anzahl verwendbarer Fakten und Regeln zu umgehen, indem eine Controlled Natural Language eingebunden wird. Eine mögliche Sprache dafür ist Attempto Controlled English, welche über die Schnittstelle Alexa erreichbar sein soll.

Nach der Einarbeitung in die verwendeten Programmiersprachen und den Ablauf der Entwicklung eines Skills, soll es zunächst ermöglicht werden, dass ein Skill in Prolog implementiert wird. Danach folgt die Einbindung einer CNL, bzw. im Speziellen ACE, auf die in diesem Alexa Skill zugegriffen werden soll. Damit soll es ermöglicht werden, mit Alexa in natürlicher Sprache zu kommunizieren und die Eingaben logisch zu verarbeiten. Dabei beschränkt sich die Arbeit aufgrund des eingebundenen ACE auf englische Sätze, da dessen Grammatik und Vokabular lediglich auf diese Sprache ausgelegt ist.

Diese Arbeit soll dabei die Frage beantworten, ob eine Implementierung entwickelt werden kann, welche die mit Amazon Alexa eingegebene natürliche Sprache in Prolog sinnvoll verarbeiten kann. Genauer wird hierbei auf die Umsetzung von Prolog als Webserver und die Verarbeitung der mit ACE extrahierten Fakten und Regeln in einer Datenbasis eingegangen. Dabei wird auch auf die Anwendung des auf ACE gestützten RACE Webservices eingegangen.

Des Weiteren sollen im Zuge der Anwendungsentwicklung weitere zukünftige Forschungsfragen und Problemstellungen aufgeworfen werden. Beispielhaft zu nennen ist hier die Diskussion, ob die verwendeten Technologien geeignet sind oder ob es hierbei Einschränkungen gibt, die man durch die Benutzung anderer Techniken umgehen könnte. Außerdem sollen abschließend auch andere Problemstellungen festgehalten werden, die sich während der Entwicklung der Anwendung herausstellten.

Die Implementierung des Projekts ***AlexaACE*** soll außerdem einigen Qualitätszielen genügen:

- **Benutzerfreundlichkeit:** Die Anwendung soll von Menschen ohne große Vorkenntnisse über ***AlexaACE*** benutzt werden können. Dabei soll die Bedienung in nahezu natürlicher Sprache möglich sein.
- **Zuverlässigkeit:** ***AlexaACE*** soll den oben geschilderten Anforderungen genügen und alle geforderten Kriterien zuverlässig erfüllen.
- **Performanz:** Die Performanz der Anwendung sollte im Bezug auf die Eingabe möglichst gut sein, da ***AlexaACE*** in Echtzeit ausgeführt werden soll.
- **Austauschbarkeit von Komponenten:** Für die Ermöglichung einer Weiterentwicklung des Projekts ***AlexaACE*** soll der leichte Austausch von Komponenten sichergestellt werden. Das schließt zum einen eine einfache Veränderbarkeit des Models oder des PROLOG Webservers ein, zum anderen sollen aber auch Umstellungen bei der externen Software vorgenommen werden können. So soll es beispielsweise nach den Ergebnissen der Evaluierung möglich sein, dass nicht zwingend Amazon Alexa als Sprachdienst und ACE als CNL genutzt werden müssen.
- **Lesbarkeit und Verständlichkeit des Codes:** Um die Wartbarkeit zu erleichtern, soll der Quellcode übersichtlich und lesbar bleiben. Um dies zu erreichen, soll neben einer geeigneten Benennung von Variablen und Prädikaten außerdem auf die lückenlose Kommentierung des Codes geachtet werden.

## 1.4. Aufbau der Arbeit

Die Arbeit gliedert sich dabei in mehrere Teile. Zunächst werden im sich anschließenden Kapitel 2 die wichtigsten verwendeten Technologien vorgestellt, nämlich Amazon Alexa, PROLOG, Attempto Controlled English und der darauf basierende Reasoner RACE.

In Kapitel 3 schließt sich die Betrachtung der Funktionsweise an. Zunächst wird der allgemeine Ablauf der Anwendung anhand einer Abbildung veranschaulicht. Dabei werden die einzelnen Komponenten, sowie der Austausch von Informationen zwischen und innerhalb selbiger grob beschrieben. Die genaue Vorstellung der Funktionsweise folgt in den Unterkapiteln der jeweiligen Komponenten, wobei im Abschnitt 3.2 außerdem Teile der Implementierung vorgestellt werden.

Die im Abschnitt 1.3 formulierten Qualitätsziele werden im Kapitel 4 auf ihre Umsetzung überprüft. Zudem erfolgt im Zusammenhang der Evaluierung auch ein Überblick über die Einschränkungen des Programms bzw. die bislang nicht umsetzbaren Möglichkeiten.

Die Arbeit endet in Kapitel 5 mit einer Zusammenfassung der wichtigsten Aspekte der Arbeit und einem Ausblick auf zukünftige Problemstellungen und Möglichkeiten der Anwendungsoptimierung.

# 2. Technologien

Um eine Anwendung mit den bereits genannten Qualitätszielen und Funktionsanforderungen umsetzen zu können, werden mehrere Technologien benötigt. Zunächst wird in Abschnitt 2.1 der Sprachdienst Amazon Alexa erklärt, welcher sowohl als Schnittstelle zwischen Nutzer und Software fungiert, als auch die Spracherkennung als solches übernimmt. Die im Programmcode der entwickelten Applikation verwendete Sprache ist PROLOG, auf welche unter Abschnitt 2.2 auch genauer eingegangen wird. Als Controlled Natural Language, die Informationen aus natürlicher Sprache extrahiert, wird, wie bereits erwähnt, Attempto Controlled English verwendet, welches unter Abschnitt 2.3 erklärt wird. Diese bietet außerdem die Basis für den in der Arbeit eingesetzten, und unter Abschnitt 2.4 vorgestellten, RACE Webservice.

## 2.1. Amazon Alexa

Im Zuge der steigenden Beliebtheit von sprachgesteuerten Assistenzsystemen, wie beispielsweise Siri[4] von Apple oder Cortana[5] von Microsoft, hat auch Amazon mit „Alexa“ ein Spracherkennungssystem entwickelt [23]. Viele dieser Systeme besitzen, neben einer Menge an bereits vorhandenen Fähigkeiten, die Möglichkeit eigene Funktionalitäten zu entwickeln [7]. Auch Amazon Alexa bietet durch das Alexa Skill Kit[6] eine Plattform mit API, Dokumentationen und Codebeispielen, um selbst sogenannte *Skills* zu programmieren. Dabei handelt es sich um Applikationen, die von Amazon-Nutzern erstellt werden können und danach außerdem durch eine Veröffentlichung auch anderen Nutzern von Amazon Smarthome-Geräten mithilfe der Amazon Alexa App zur Verfügung gestellt werden können. Für den Gebrauch muss nur ein von Amazon bereitgestelltes Interface eingebunden werden. Dazu kann für das Aufrufen von Skills, neben der Testkonsole der Developerplattform, auch ein Testing Tool[7] genutzt werden, welches eigens entwickelt wurde, um Applikationen

[4]Apple Siri: `https://www.apple.com/de/ios/siri/`
[5]Microsoft Cortana: `https://www.microsoft.com/de-de/windows/cortana`
[6]Amazon Developer Plattform: `https://developer.amazon.com/de/alexa`
[7]Echosim: `https://echosim.io/`

auch ohne der von Amazon vertriebenen Hardware testen zu können. Gewöhnlicher ist aber die Nutzung der speziellen Endgeräte[8], wie des Amazon Echo oder des Amazon Echo Dot. Dabei handelt es sich um zylinderförmige Geräte, die mit ihrem Mikrofon eine Nutzereingabe entgegennehmen und diese über den vorhandenen Internetzugang an die Amazon Server weiterleiten [4]. Mithilfe des sogenannten *Amazon Voice Service* (AVS)[9] wird eine Spracherkennung durchgeführt und nach der weiteren Bearbeitung des Skills eine dadurch generierte Antwort zurück geschickt und durch den Lautsprecher dem Nutzer zugänglich gemacht.

Um die einzelnen Bestandteile einer typischen Nutzereingabe besser verstehen zu können, wird eine beispielhafte Anweisung in Abbildung 2.1 betrachtet.

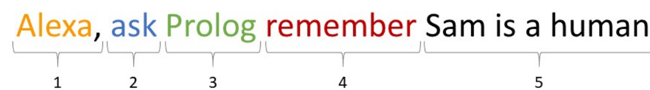


Abbildung 2.1.: Bestandteile einer beispielhaften Eingabe in Amazon Alexa

Dabei haben die einzelnen Bestandteile spezielle Bezeichnungen und Funktionen im Kontext des Skills[10] [4, 7, 24]:

1. **Aktivierungswort:** Durch das Nennen dieses Wortes verlässt das Endgerät seinen Ruhezustand und erwartet die restliche Nutzereingabe.

2. **Phrase zum Starten eines Skills:** Um einen Skill aufzurufen und ihn zu verwenden, benutzt der Nutzer eine der von Amazon festgelegten Phrasen. Im Englischen stehen dafür beispielsweise „ask“, „start“ oder „open“ zur Verfügung.

3. **Invocation Name des Skills:** Der vom Entwickler vorab festgelegte Invocation Name identifiziert einen Skill eindeutig, sodass dieser darüber gestartet wird.

4. **Intentname:** Ein Intent ist ein Teil des Skills, in dem eine Funktionalität der Anwendung definiert wird. Der im Beispiel aus Abbildung 2.1 aufgerufene Intent „remember“ realisiert im Zusammenhang mit ***AlexaACE*** die Möglichkeit, Fakten einzulesen und abzuspeichern. Jeder Skill besitzt von Amazon vordefinierte Intents (etwa zum Beenden des Skills) und kann darüber hinaus durch eigene ergänzt werden, um die gewünschten Methoden umzusetzen.

[8] Hochrechnung der Verkaufszahlen von Amazon Endgeräten in der USA: `https://www.amazon-watchblog.de/sortiment/846-hochrechnung-amazon-8-millionen-echos-usa.html`

[9] Amazon Voice Service: `https://developer.amazon.com/alexa-voice-service`

[10] Tutorial zum Erstellen eines Skills auf der Amazon Developer Plattform: `https://developer.amazon.com/de/alexa-skills-kit/tutorials/den-ersten-alexa-skill-entwickeln`, Dokumentation zum Erstellen eines Skills auf der Amazon Developer Plattform: `https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html`

5. **Eingabewert des Intentslots:** Ein Intent kann optional einen oder mehrere Slots enthalten. Diese Platzhalter erwarten für die weitere Skillbearbeitung eine Eingabe, wobei dafür ein Datentyp festgelegt wird. Auch hierfür bietet Amazon bereits eingestellte Optionen, z. B. AMAZON.NUMBER für Zahlen, welche durch eigene Festlegung von Slottypen erweitert werden können.

Der Entwickler muss für den Skill mehrere wichtige Definitionen vornehmen und ein sogenanntes *Intent Schema* anlegen. Darin werden, wie bereits beschrieben, die einzelnen Methoden erstellt, welche die Applikation ausführen können soll. Auf die technischen Details dazu wird in Abschnitt 3.1 eingegangen.

## 2.2. Prolog

Der im Zuge dieser Arbeit entwickelte Skill wird auf einem PROLOG Webserver ausgeführt, wobei das Programm auch in der Sprache PROLOG realisiert wird. An dieser Stelle soll lediglich eine Einführung mit den wichtigsten Aspekten der Programmiersprache stattfinden, eine detailliertere Beschreibung ist unter anderem in [26] zu finden.

### 2.2.1. Allgemeine Daten und Entstehung

Die erste Version dieser von Alain Colmerauer und seinem Team entwickelten Programmiersprache wurde im Jahre 1972 veröffentlicht. Der Name setzt sich zusammen aus „*PRO*grammation en *LOG*ique“ [6], wodurch schon deutlich wird, dass es sich hierbei um eine Logikprogrammiersprache handelt, welche auf first-order Prädikatenlogik basiert. Außerdem setzt sie das deklarative Programmierparadigma um, bei dem es, im Gegensatz zu prozeduralen Sprachen, wichtig ist, das vom Programm zu lösende Problem festzulegen, weniger aber wie die genaue Umsetzung aussieht [30].

### 2.2.2. Arbeitsweise

Im Quellcode werden in PROLOG sowohl Fakten als auch Regeln festgelegt, welche auch als Klauseln bezeichnet werden [26]. Nach dem Laden des Programms mit dem Befehl `?- consult(...)`, kann man beispielsweise testen lassen, ob ein eingegebener Fakt, auch Goal genannt, aus den festgelegten Klauseln ableitbar ist. Dafür wird die sogenannte Top-Down Evaluation [30] verwendet. Zunächst wird dabei überprüft, ob das Goal bereits in den Fakten des Programms enthalten ist. Anschließend wird unter Zuhilfenahme der Fakten und Regeln eine Unifikation durch das Substituieren

von Variablen durchgeführt, wobei versucht wird, durch Belegungen von Variablen mit Konstanten das angefragte Goal abzuleiten. Bei der Top-Down Evaluation wird zudem das Prinzip des Backtrackings verwendet. Hat das Programm an einer Stelle bei der Unifikation mehrere anwendbare Klauseln, so erfolgt nach dem vollständigen Ausführen des ersten Falles das Zurückspringen zu diesem Punkt, wo die nächste mögliche Unifikation durchgeführt wird. Am Ende wird als Ergebnis entweder `false` ausgegeben, falls das Goal nicht abgeleitet werden konnte, oder die entsprechenden Variablenbelegungen.

### 2.2.3. Verwendete Funktionalitäten von SWI-Prolog

Für den Quellcode von ***AlexaACE*** wird SWI-PROLOG[11] verwendet. Dabei handelt es sich um eine Programmierumgebung, mit der PROLOG frei genutzt werden kann. SWI-PROLOG bietet über die Standardimplementierung hinaus zusätzliche Pakete und Funktionalitäten, die beispielsweise für die Implementierung eines PROLOG Webservers genutzt werden können[12].

**HTTP Server Bibliotheken.** Um die Kommunikation zwischen einem Webclient, im Falle von ***AlexaACE*** die Amazon Server, und einem Webserver, hier dem PROLOG Webserver, zu ermöglichen, wird häufig das *HyperText Transfer Protocol* (HTTP) eingesetzt, wobei PROLOG dafür eine Bibliothek anbietet[13].

Listing 2.1: Grundstruktur eines PROLOG Webservers

```
1 :- use_module(library(http/thread_httpd)).
2 :- use_module(library(http/http_dispatch)).
3
4 server(Port) :-
5   http_server(http_dispatch, [port(Port)]).
6
7 :- http_handler(root(.), entry_page, []).
8 ...
9
10 entry_page(Request) :-
11        ...
```

Die Standardimplementierung[14], die in Quellcode 2.1 zu sehen ist, umfasst dabei mehrere Teile. Zuerst werden die erforderlichen Bibliotheken `http/thread_httpd`

[11]SWI Prolog: `http://www.swi-prolog.org/`
[12]Tutorial zum Erstellen von Web Applikationen in SWI-PROLOG: `http://www.pathwayslms.com/swipltuts/html/`
[13]SWI-PROLOG HTTP Support: `http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/http.html%27)`
[14]Grundgerüst einer PROLOG Webserver Implementierung: `http://www.swi-prolog.org/pldoc/man?section=httpserver`

und `http_dispatch` eingebunden, um die darin bereitgestellten Funktionalitäten nutzen zu können. So verwendet das Prädikat `server(Port)` das vordefinierte Prädikat `http_server/2`. Damit wird ein Server erstellt, wobei dieser auf einem vom Nutzer festgelegten Port laufen soll. Damit die Implementierung des Models unabhängig vom Webserver ist, wird mithilfe von `http_handler(+Path, :Closure, +Options)` und der Belegung von `Closure` festgelegt, in welchem Prädikat die Antwort berechnet werden soll. Dies geschieht im Quellcode 2.1 beispielsweise im Prädikat `entry_page(Request)`.

**Dicts.** Des Weiteren werden in der Implementierung von ***AlexaACE*** sogenannte *Dicts*[15] verwendet. Dabei handelt es sich um eine Datenstruktur, die in PROLOG seit Version 7 verfügbar ist und ihren Einsatz vor allem findet, wenn kein anderer Datentyp verwendet werden kann oder nicht klar ist, wie viele Argumente diese beinhaltet. In ***AlexaACE*** werden Dicts verwendet, um die von den Amazon Servern erhaltene Informationen auszuwerten. Dafür wird das Prädikat `get_dict(?Key, +Dict, -Value)` verwendet, welches durch Unifizierung den Wert, der mit dem Key assoziiert ist, in der Variable Value ausgibt.

**JSON.** Wie später in der Arbeit unter Abschnitt 3.2.1 zu sehen, ist es außerdem notwendig eine Bibliothek einzubinden, um sowohl JSON Inputs auswerten zu können, zum anderen auch JSON Outputs als Antwort an die Amazon Server zu generieren. Bei *JSON* (JavaScript Object Notation)[16] handelt es sich um ein Datenaustauschformat, welches unter anderem in ***AlexaACE*** für die Kommunikation zwischen Amazon und dem PROLOG Webserver dient. Dafür wird die Bibliothek `http/http_json`[17] eingebunden. Diese stellt zum einen das Prädikat `http_read_json_dict(+Request, -Dict)` zur Verfügung, mit dem aus einem erhaltenen JSON Input die Daten extrahiert und in einem Dict gespeichert werden können. Ebenfalls existiert ein Prädikat `reply_json/1`, welches einen JSON Output generiert, der über HTTP verschickt werden kann.

### 2.2.4. Prolog als Programmiersprache für NLP

Neben den mittlerweile etablierten Anwendungsgebieten, wie Datenbankprogrammierung oder allgemein Programmierung im Zusammenhang mit formaler Logik, wurde PROLOG ursprünglich zu einem anderen Zweck entwickelt [26]: „Prolog was originally intended for the writing of natural language processing applications.“ Deshalb dürfte es nicht weiter verwundern, dass bereits das erste bedeutende, in PROLOG realisierte, Projekt dem Bereich NLP zugeordnet wird [20]. Grundsätzlich wird die

[15] SWI-PROLOG Dicts: `http://www.swi-prolog.org/pldoc/man?section=dicts`
[16] Einführung in JSON: `https://www.json.org/json-de.html`
[17] SWI-PROLOG JSON Support: `http://www.swi-prolog.org/pldoc/man?section=jsonsupport`

Programmiersprache bevorzugt für Natural Language Processing eingesetzt [3], da die Prägnanz, zugleich aber Ausdrucksstärke, der Sprache vorteilhaft ist [14, 22].

Abschließend fasst der Entwickler Colmerauer den Nutzen von Prolog in [6] folgendermaßen zusammen: „[...] successful marriage between natural language processing and automated theorem-proving“. Dies zeigt auch, dass Prolog für die Forschung im Bereich NLP sehr geeignet ist. Auch die Controlled Natural Language ACE, die für den im Zuge dieser Arbeit entwickelten Skill verwendet wird, bzw. deren Parsing Engine (APE) sowie der Reasoner (RACE) sind in Prolog realisiert [11, 12].

## 2.3. Attempto Controlled English (ACE)

An dieser Stelle soll eine kurze Einführung in die wichtigsten Daten, Arbeitsweisen und Strukturen erfolgen. Diese dient lediglich als Überblick, wobei detailliertere Beschreibungen unter anderem in den Manuals, wie [17], oder Vorstellungen dieser speziellen CNL, wie in [10] oder [15], zu finden sind.

### 2.3.1. Allgemeine Daten und Entstehung

Attempto Controlled English ist eine seit 1995 an der Züricher Universität entwickelte Controlled Natural Language [15, 19]. Genauso wie allgemein jede CNL, sollte auch ACE der Anforderung genügen, dass die Sprache sowohl für den Menschen nahezu natürlich benutzt werden kann und trotzdem präzise genug ist, um von einem Computer verarbeitet werden zu können [10]. In einer der Veröffentlichungen [12] der Entwickler wird Attempto Controlled English mit folgender Aussage beschrieben: „ACE is a logic language with an English syntax.“ Auch wenn diese oft als natürliche Sprache angesehen wird, handelt es sich dabei aber um eine formale Sprache, deren Syntax bzw. Konstruktionsregeln vor der Benutzung erlernt werden müssen.

### 2.3.2. Zielsetzung

Die ursprüngliche Zielsetzung von ACE war die Entwicklung einer Sprache, die für die Spezifikation von Softwareanforderungen verwendet werden kann [11].

Da die natürliche Sprache aufgrund der einfachen Nutzung, des leichten Verständnisses und der Ausdruckskraft sehr benutzerfreundlich ist, waren die Entwickler bestrebt, dass sich ACE möglichst an natürlichem Englisch orientiert [16]. Gleichzeitig sollten die Vorzüge einer formalen Sprache berücksichtigt werden. Diese bestehen darin, dass sich jene auf eine eindeutige und wohldefinierte Syntax stützen und für

den Computer nutzbar sind. ACE kombiniert die Vorteile von natürlicher und formaler Sprache durch Einschränkungen im Bereich der Grammatik und des Vokabulars [15]. Die resultierende Sprache ist weniger komplex als eine formale Sprache und enthält deutlich weniger Mehrdeutigkeit als eine ausschließlich natürliche Sprache, wodurch ein Programm sie besser verarbeiten kann [12, 19].

Dabei sollte ACE, wie jede Controlled Natural Language, in der Lage sein, neben simplen Sätzen der Form Subjekt, Prädikat, Objekt, auch zusammengesetzte Sätze wie Relativsätze, Vergleichskonstrukte, if-then Sätze, Negationen oder Fragen umzusetzen [14]. Dass dies den Entwicklern gelungen ist, wird an einem Zitat [29] deutlich: „[...]Attempto Controlled English (ACE) is a subset of natural language that can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage.“

### 2.3.3. Funktionsweise von Attempto Controlled English

Für die Umsetzung der Zielanforderungen verwendet das Attempto System die sogenannte *Attempto Parsing Engine* (APE), welche den eingegebenen ACE Text eindeutig in *Discourse Representation Structure* (DRS)[18], sowie optional in PROLOG, umwandelt [12, 15]. Das Ziel von DRS ist dabei, Sätze bzw. deren Aussage logisch darzustellen. Dabei wird der Zusammenhang durch Prädikate und die Belegung dieser durch Variablen und Konstanten ausgedrückt. Dadurch wird die semantische Struktur dargestellt, wobei DRS dabei die Kontextabhängigkeiten des gesamten eingegebenen Textes berücksichtigt [10]. Die Arbeitsweise der APE, welcher über den Webservice[19] aufgerufen wird, wird anhand der Ausgabe für das in Abbildung 2.2 eingegebene Beispiel verdeutlicht.

ACE ist so realisiert, dass sich die zuletzt eingegebenen Sätze immer auf die vorherigen Nutzereingaben beziehen, da die Sprache keine eigens definierte Wissensdatenbasis bietet. Um dabei Fehlinterpretationen bei mehrdeutigen Sätzen zu vermeiden, wurden drei Strategien entwickelt dies zu umgehen [10, 12]:

- **Beschränkung der Sprache:** Wie bereits erwähnt, wird die englische Sprache mithilfe von Konventionen und Konstruktionsregeln in Vokabular und Grammatik eingeschränkt. Dadurch wird bereits ein Großteil an möglichen Mehrdeutigkeiten ausgeschlossen.

[18] Alistair Knott. Introduction to Discourse Representation Theory (DRT): `http://www.cs.otago.ac.nz/staffpriv/alik/papers/altss-talk.pdf`
[19] APE Webservice: `http://attempto.ifi.uzh.ch/ape/`

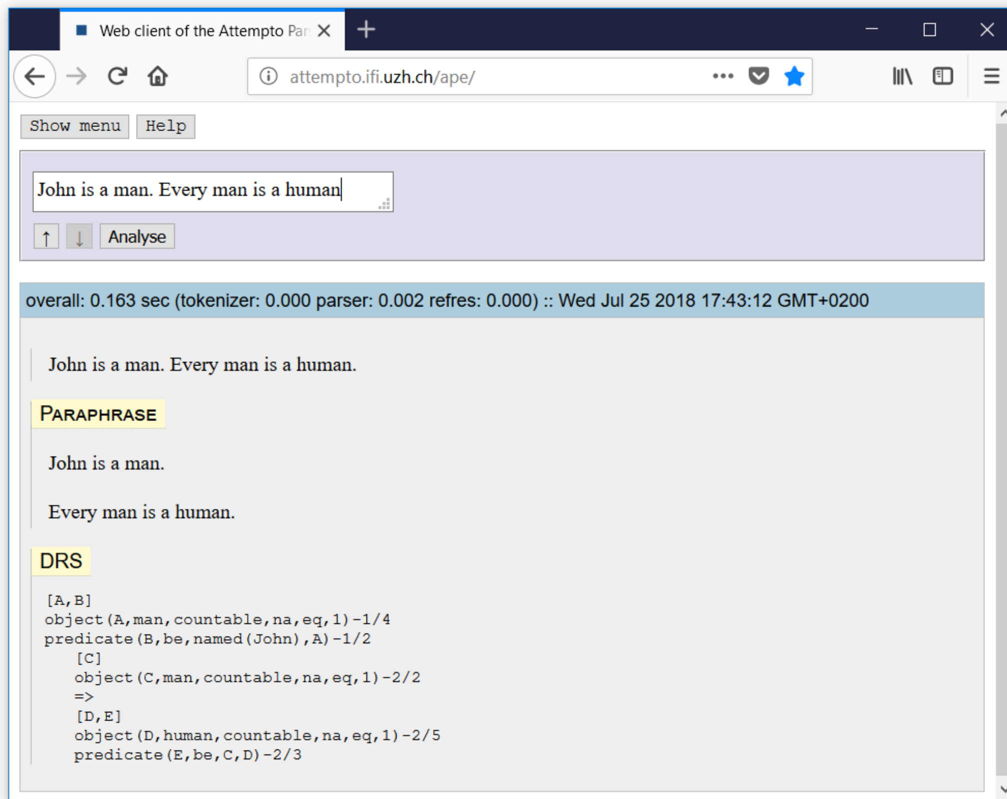


Abbildung 2.2.: Darstellung eines ACE Textes in DRS

- **Nutzung von Interpretationsregeln:** ACE besitzt zudem eine Reihe an Regeln, welche bei möglichen unklaren Fällen eine Interpretation festlegen. Beispielsweise ist beim Satz „The customer who inserts a card slowly enters a code“ [13] nicht eindeutig klar, ob sich das Wort „slowly“ auf „inserts“ oder „enters a code“ bezieht. Eine der Interpretationsregeln legt allerdings fest, dass das unklare Wort zum jeweils vorangegangenen Teilsatz gezählt wird, hier also ein Bezug auf „inserts“ festgelegt wird.

- **Ausgabe einer Paraphrase:** Der letzte Schritt eine Fehlinterpretation zu erkennen, liegt auf Seite des Nutzers. APE gibt zu jeder Eingabe eine Paraphrase aus, an der deutlich wird, wie der eingegebene Text interpretiert wurde. Erkennt der Nutzer daran, dass der Sinn des Satzes nicht wie gewünscht aufgenommen wurde, ist es ihm möglich den Satz umzuformulieren und nochmal einzugeben.

  Bespielsweise wird, wie in Abbildung 2.3, für den bereits genannten Satz „The customer who inserts a card slowly enters a code“ an der Paraphrase deutlich, dass die oben genannte Interpretationsregel genutzt wurde.

Web client of the Attempto Par

attempto.ifi.uzh.ch/ape/

The customer who inserts a card slowly enters a code.

↑ ↓ Analyse

overall: 0.197 sec (tokenizer: 0.000 parser: 0.001 refres: 0.000) :: Wed Jul 25 2018 17:54:05 GMT+0200

| | Type | Sentence | Problem | Suggestion |
|---|---|---|---|---|
| warning | anaphor | 1 | The definite noun phrase 'the customer' does not have an antecedent and thus is not interpreted as anaphoric reference, but as a new indefinite noun phrase. | If the definite noun phrase 'the customer' should be an anaphoric reference then you must introduce an appropriate antecedent. |

The customer who inserts a card slowly enters a code.

PARAPHRASE

There is a customer X1.
The customer X1 enters a code.
The customer X1 inserts a card slowly.

DRS

```
[A,B,C,D,E]
modifier_adv(E,slowly,pos)-1/7
predicate(E,insert,C,D)-1/4
object(D,card,countable,na,eq,1)-1/6
object(C,customer,countable,na,eq,1)-1/2
object(A,code,countable,na,eq,1)-1/10
predicate(B,enter,C,A)-1/8
```

Abbildung 2.3.: Ausgabe der Interpretation eines ACE Textes als Paraphrase

### 2.3.4. Konventionen und Einschränkungen der Sprache

Wie bereits erwähnt, ist die Umsetzung von ACE und die Vermeidung der meisten Fälle von Mehrdeutigkeit nur möglich, indem sowohl Vokabular als auch Grammatik bzw. das Bilden von Sätzen einiger Konstruktionsregeln unterliegen. Die Einschränkungen umfassen unter anderem die Konventionen, dass ein Satz immer mit einem Punkt, Fragezeichen [13] oder, ab Version 6 des Systems, auch mit einem Ausrufezeichen enden muss [11]. Des Weiteren müssen gewöhnliche Nomen immer in Kombination mit einem Artikel oder Quantifizierer verwendet werden. Durch die Weiterentwicklung wurden allerdings auch anfängliche Konstruktionsregeln überflüssig, wie beispielsweise die Beschränkung der Verbformen auf Indikativformen der 3. Person Singular im Aktiv und simple present [13]. Seit Version 4 ist es beispielsweise über eine spezielle Definition der DRS möglich, auch Pluralkonstruktionen zu verwenden [11, 17]. Auf die genauen Konventionen der aktuellen Version 6.7 des Systems wird nicht weiter eingegangen, diese können aber in der Dokumentation[20] auf der offiziellen Webseite des Attempto Projekts nachgelesen werden.

[20]Dokumentation ACE 6.7: `http://attempto.ifi.uzh.ch/site/docs/ace_nutshell.html`

## 2.4. Attempto Controlled Englisch Reasoner (RACE)

Eines der von ACE bereitgestellten Tools ist der sogenannte *ACE Reasoner* (RACE), der auf ACE Texte angewendet werden kann [16]. Ähnlich wie oben beschrieben, werden auch hier die Eingaben in DRS und anschließend in first-order logic Klauseln übersetzt, mit welchen Wissen abgeleitet werden kann [12]. Zunächst sollte mit dem ACE Reasoner lediglich ermöglicht werden, Theoreme aus bereits bekannten Axiomen abzuleiten. Dies wurde allerdings erweitert, sodass neben der Überprüfung von Theoremen auch die Möglichkeit besteht, ACE Texte auf Konsistenz zu überprüfen oder Fragen auf Basis bekannter Fakten beantworten zu lassen [8, 11]. Auch hierfür bietet die Züricher Universität einen Webclient[21], mit dem man das System testen kann [9]. Eine Beispieleingabe ist in Abbildung 2.4 zu sehen.

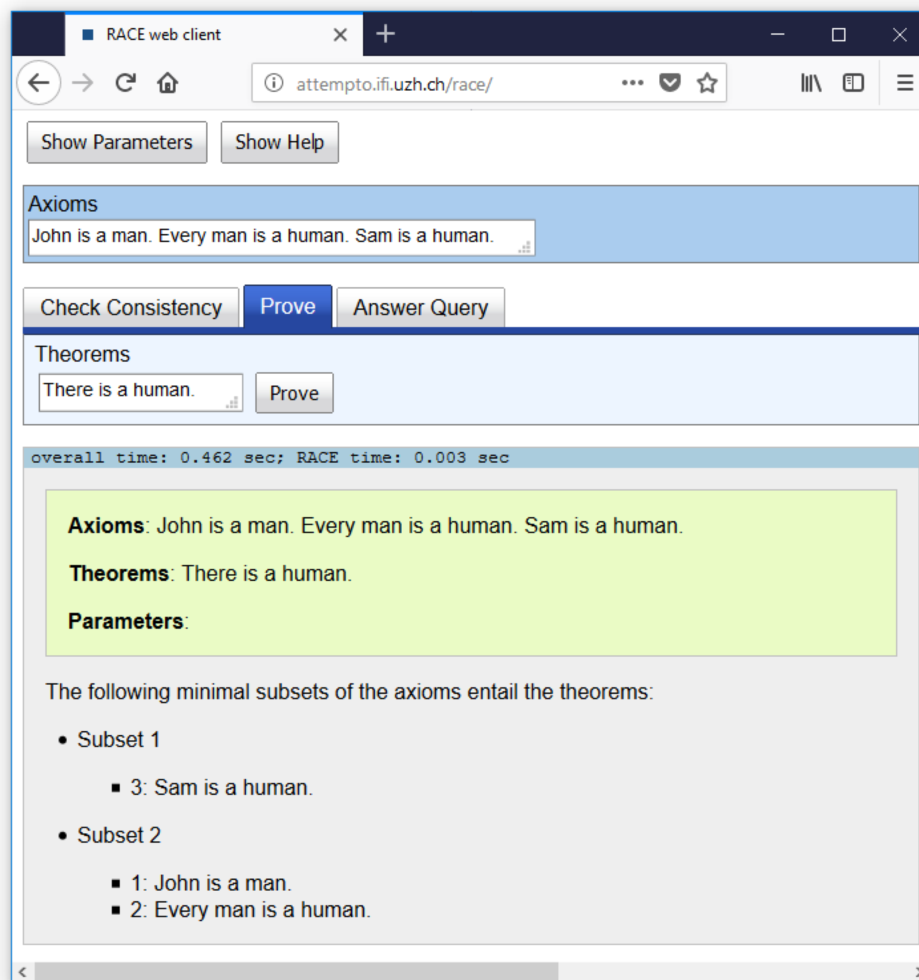


Abbildung 2.4.: Überprüfen eines Theorems auf Basis vorhandener Axiome mit RACE

[21]RACE Webclient: `http://attempto.ifi.uzh.ch/race/`

Hierbei wird vor allem deutlich, dass RACE für Theoreme, die über mehrere Wege abgeleitet werden können, auch alle möglichen Beweise ausgibt. So wird das Goal „There is a human." zum einen über das Axiom „Sam is a human" nachgewiesen, zum anderen noch über die Axiome „John is a man." und „Every man is a human.".

Neben der im Browser aufrufbaren Webanwendung, gibt es auch noch den RACE Webservice[22], der über einen *Remote Procedure Call* (RPC) aufgerufen wird. Dabei stellt ein Nutzer bzw. eine von ihm gestartete Anwendung einen Request an den RACE Webservice, wo die Anfrage bearbeitet wird. Anschließend wird die berechnete Antwort an den Nutzer zurückgeschickt. Die Kommunikation findet dabei über das *Simple Object Access Protocol* (SOAP)[23] statt. Dieses nutzt dafür XML[24] und muss eine spezielle Struktur einhalten, welche allerdings in diesem Zusammenhang nicht näher beschrieben wird. Der RACE Webservice wird auch für die im Zuge dieser Arbeit entwickelten Anwendung verwendet.

[22]RACE Webservice: `http://attempto.ifi.uzh.ch/site/docs/race_webservice.html`
[23]SOAP 1.1: `https://www.w3.org/TR/soap11/`
[24]XML: `https://www.w3.org/XML/`

# 3. Funktionsweise

Die Funktionsweise bzw. der Ablauf der entwickelten Anwendung wird anhand des Schaubilds in Abbildung 3.1 erklärt.

Über eine Schnittstelle, beispielsweise der Testkonsole von Amazon Developer[25], oder aber, wie im Schaubild dargestellt, einem Amazon Echo wird ein Skill gestartet. Wie unter Abschnitt 2.1 beschrieben, muss der Nutzer dafür eine bestimmte Struktur einhalten und mehrere wichtige Informationen nennen. Der Nutzer muss das Gerät dafür zunächst mit „Alexa“ aus dem Ruhemodus wecken und danach den entsprechenden Skill mit „ask *Invocation Name*“ aufrufen. In dem vorliegenden Fall lautet der Invocation Name, das Schlüsselwort welches den Skill startet, „Prolog“ und der Skill umfasst drei Intents, das sind bestimmte Methoden des Programms, die sich wiederum durch ein spezielles Stichwort aufrufen lassen. So ist es möglich mit „remember ...“ Wissen abzuspeichern, mit „question ...“ Fragen zu stellen und mit „prove ...“ ein Theorem überprüfen zu lassen. Die jeweilige Freitextantwort, die sich als Wert des Slots an einen Intent Name anschließt, muss dafür ein ACE Text sein. Das umfasst zum einen die Beschränkung auf englische Sätze, sowie die Verwendung der Grammatik- und Konstruktionsregeln dieser CNL.

Diese Anfrage wird als User-Request an die Amazon Server weitergeleitet, wo mithilfe des Amazon Voice Services die Sprache erkannt, und mit dem genannten Invocation Name der entsprechende Skill aufgerufen wird. Dieser enthält neben der Definition der einzelnen Intents vor allem die Information über den Endpoint, das heißt, die URL des Servers, zu dem die Nutzereingabe als JSON Input über ein HTTP POST-Request weitergeleitet wird, um dort bearbeitet zu werden.

Nach der Übermittlung der Daten an den Prolog Webserver, welcher in dieser Anwendung verwendet wird, werden die für die Bearbeitung relevanten Informationen des JSON Inputs in der Variable „DictIn“ gespeichert, sodass diese im Model zur Verfügung stehen. Dort ist für jeden einzelnen Intent jeweils ein Prädikat implementiert, das die Verarbeitung ermöglicht. Unabhängig um welche der drei in der Anwendung realisierten Methoden es sich bei der Nutzereingabe handelt, nutzt jede das bereits in der Datenbasis abgelegte Wissen für die weitere Bearbeitung des

[25]Amazon Developer Plattform: `https://developer.amazon.com/de/alexa`

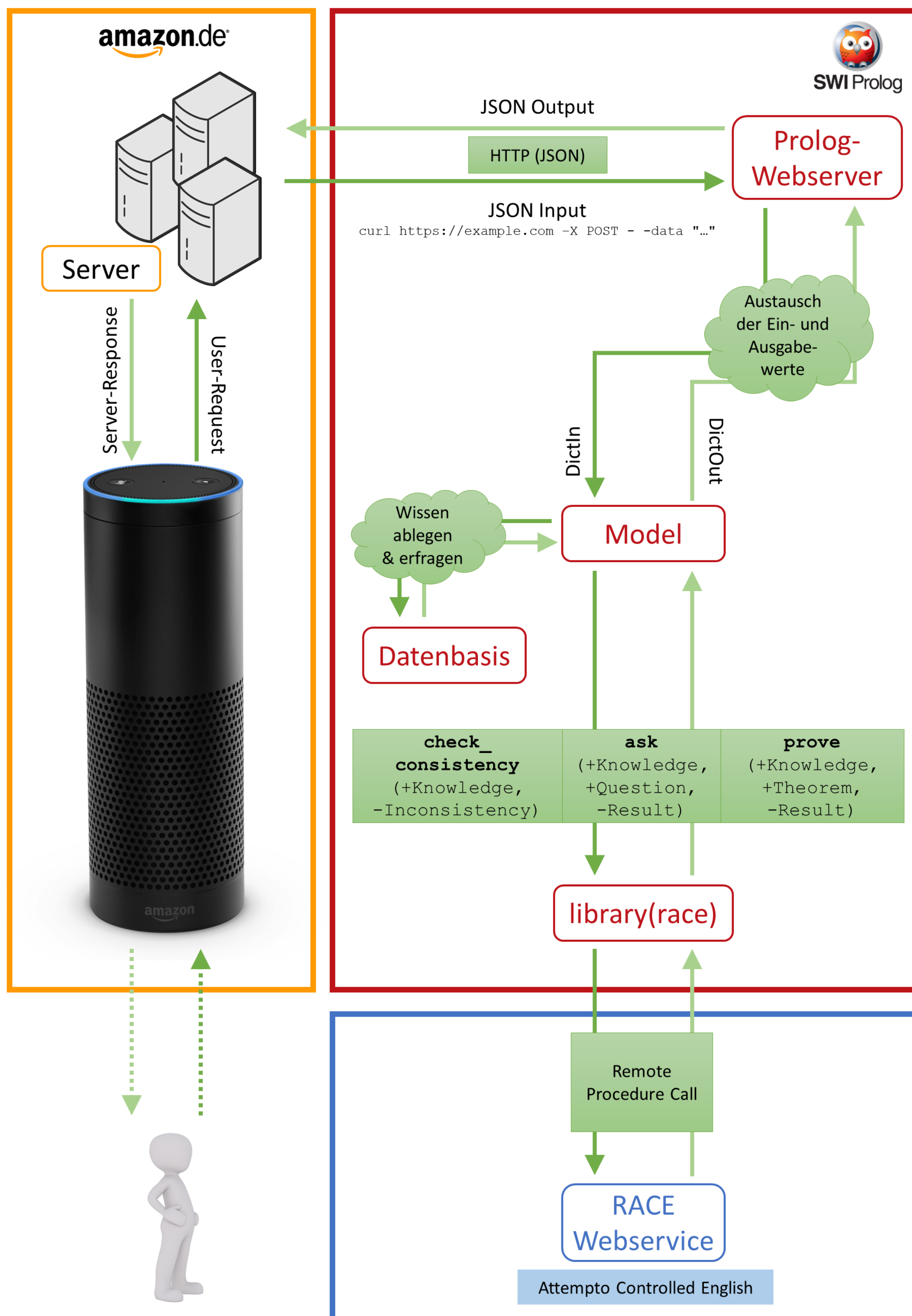


Abbildung 3.1.: Funktionsablauf der Anwendung (Bildquellen siehe Anhang A.1)

Intents. Auch dieser Austausch erfolgt PROLOG intern mit einem Prädikat und einer Variablenbelegung als Rückgabewert. Zusammen mit der vorhandenen Fakten-

und Regelbasis wird die Nutzereingabe an den Attempto Controlled English Reasoner (RACE) weitergeleitet, wobei hierfür die drei durch die genutzte Bibliothek *library(race)* [25] bereitgestellten Prädikate verwendet werden.

Die Bibliothek dient als Schnittstelle zwischen der Prolog Implementierung und dem auf Attempto Controlled English basierenden RACE Webservice. Da dieser nur über XML-Schema aufgerufen werden kann, wird in der eingebundenen *library(race)* die Belegung der Prolog Variablen entsprechend in das erforderliche XML-Format umgewandelt und über einen Remote Procedure Call an RACE geschickt.

Nach der Bearbeitung der angefragten Methode schickt der Webservice eine Antwort über die Bibliothek an das Model. Dort erfolgt die Generierung der Antwortphrase und das Speichern in der Variable „DictOut“, welche dem Prolog Webserver übergeben wird. Die Amazon Server erhalten danach den JSON Output, welcher im letzten Schritt über die verwendete Schnittstelle dem Nutzer übermittelt wird.

## 3.1. Amazon Alexa

Die erste Komponente, die die Nutzeranfrage erhält, ist Amazon Alexa. Wird ein Skill, wie unter Abschnitt 2.1 beschrieben, aufgerufen, so leitet die Schnittstelle, in dem vorliegenden Fall der Amazon Echo, die eingegebenen Informationen an die Amazon Server weiter. Der Amazon Voice Service erhält so den Invocation Name, den Intent Name und den Inhalt des Slots. Diese Spracheingaben wandelt der AVS in Text um, welcher von den Amazon Servern ausgewertet wird, sodass der richtige Skill erkannt und ausgeführt werden kann.

### 3.1.1. Entwicklung des Skills

Die Skillentwicklung besteht aus mehreren Schritten, die durchlaufen werden müssen[26]. Voraussetzung für das Erstellen eines neuen Skills ist ein Amazon Konto, mit dem man sich bei Amazon Developer anmelden muss. Nachdem man einen Namen für die Applikation eingegeben und die Sprache ausgewählt hat, sollte man sich bei der Wahl des Models für einen sogenannten „Custom Skill“ entscheiden. Dieser ermöglicht es, einen komplett eigenen Skill ohne bereits vordefinierte Teile zu entwickeln. Anschließend muss man eine Checkliste mit vier Punkten abarbeiten, die unter anderem das Eingeben des Invocation Names und das Festlegen eines Endpoints umfasst. Der Server, dessen URL man dort eingibt, wird die Anwendung bearbeiten und muss außerdem ein gültiges SSL-Zertifikat besitzen.

[26] Dokumentation zum Erstellen eines Skills auf der Amazon Developer Plattform: `https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html`

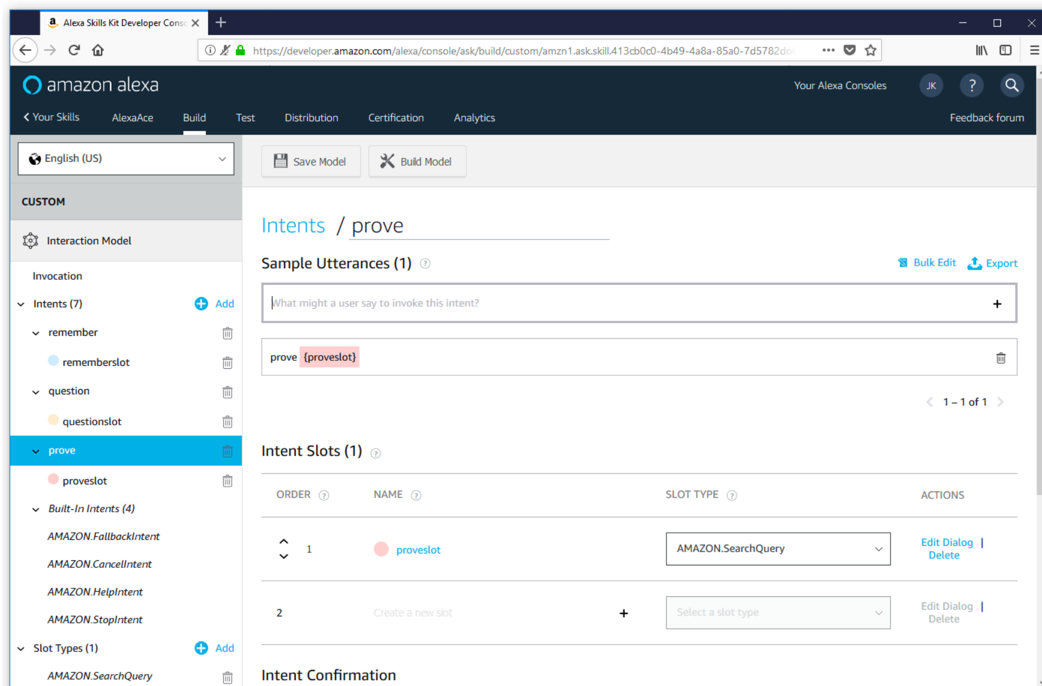


Abbildung 3.2.: Erstellen des Interaction Model von ***AlexaACE***

Am wichtigsten für die Funktionalität des Skills ist das Erstellen der einzelnen Intents im sogenannten *Interaction Model*. Dafür muss man, wie in Abbildung 3.2 zu sehen, sowohl den Intent Namen, als auch Phrasen eingeben, die diesen Intent aufrufen sollen. Für den Skill ***AlexaACE*** werden die Intents „remember“, „question“ und „prove“ mit ihren entsprechenden Slots erstellt. Die Slots dienen dazu, dass der Nutzer eine Eingabe machen kann, die von der Applikation für das Bearbeiten zwingend benötigt wird. Als Datentyp des Slots wird bei ***AlexaACE*** „AMAZON.SearchQuery“ verwendet, welcher ermöglicht, einen Satz einzugeben, ohne dass er irgendwelchen Konventionen genügen muss. Alternativ kann man die Erstellung der Intents auch als JSON Code in den JSON Editor eingeben, wobei das Grundgerüst dafür in Listing 3.1 zu sehen ist.

Listing 3.1: Erstellung des Skills mithilfe des JSON Editors

```
1 {
2   "interactionModel": {
3     "languageModel": {
4       "invocationName": "prolog",
5       "intents": [
6         { "name": "remember", ... },
7         { "name": "question", ... },
8         { "name": "prove",
```

```
9           "slots": [
10            { "name": "proveslot",
11              "type": "AMAZON.SearchQuery" } ],
12          "samples": [ "prove {proveslot}" ],
13          ...
14  } ] } } }
```

Da dies für unerfahrene Nutzer allerdings einige Fehlerquellen birgt, wird empfohlen, die grafische Benutzeroberfläche für die Definition der Intents zu verwenden. Als letzten Punkt der Checkliste muss man dann das Interaction Model speichern und erstellen lassen. Danach ist die Skillerstellung über die Amazon Webseite abgeschlossen, wobei die Funktionalität selbst natürlich erst auf dem Server programmiert werden muss.

### 3.1.2. Ausführung des Skills

Möchte man den Skill nun ausführen, so muss zuvor der Server, auf dem die Bearbeitung von ***AlexaACE*** stattfindet, gestartet werden. Anschließend kann der Nutzer über eine Schnittstelle den Skill starten und eine Eingabe tätigen. Diese wird nach der Spracherkennung durch Amazon Alexa als JSON Input an den zuvor festgelegten Endpoint geschickt, wobei die Struktur der übermittelten Nachricht in Listing 3.2 dargestellt ist. Beispielsweise möchte der Nutzer mit der dargestellten Eingabe überprüfen, ob mit der vorab über den Skill eingegebenen Datenbasis das Theorem „Sam is a human" bewiesen werden kann. Dafür wird zunächst der Intent „prove" aufgerufen und der Inhalt des Slots ist der zu überprüfende Satz.

Listing 3.2: JSON Input einer Nutzereingabe

```
1  {
2    "version": "1.0",
3    "session": { ... }
4    "request": {
5      "type": "IntentRequest",
6      ...
7      "intent": {
8        "name": "prove",
9        "confirmationStatus": "NONE",
10       "slots": {
11         "proveslot": {
12           "name": "proveslot",
13           "value": "Sam is a human",
```

```
14          "confirmationStatus": "NONE"
15 } } } } }
```

Nach der Ausführung auf dem PROLOG Webserver wird wieder ein JSON Output an Amazon zurückgeschickt, welcher dem Nutzer über die entsprechende Schnittstelle übermittelt wird. In der Testkonsole wird die Antwort, wie in Abbildung 3.3 dargestellt, empfangen und entsprechend ausgegeben. Dabei wird deutlich, dass das eingegebene Theorem abgeleitet werden konnte. Darüber hinaus gibt ***AlexaACE*** noch die für die Herleitung genutzten Axiome an, wobei nur der erste Lösungsweg angegeben wird, falls mehr als eine Ableitung existiert. Das Erstellen der getätigten Ausgabe findet auf dem PROLOG Webserver statt, worauf im nächsten Absatz noch genauer eingegangen wird.

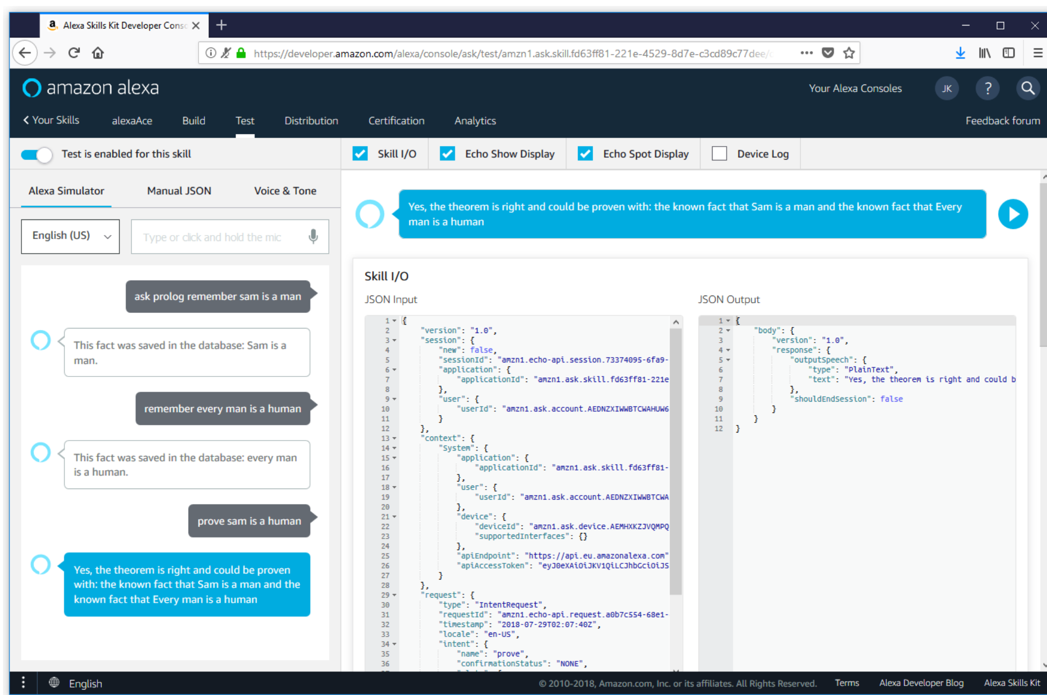


Abbildung 3.3.: Ausführen von ***AlexaACE*** über die Testkonsole von Amazon

## 3.2. Prolog

Die Anwendung ***AlexaACE*** wird mithilfe von PROLOG realisiert, wobei Teile der Implementierung auf dem bereits erwähnten Beispiel einer PROLOG Anwendung mit Amazon Alexa [2] basieren. Dabei gliedert sich das Programm, von welchem die wichtigsten Teile im Anhang A.2 zu finden sind, in zwei Dateien. In `run.pl` wird der erste Teil der Implementierung des PROLOG Webservers umgesetzt und ***AlexaACE***

mithilfe des Prädikats `main :- server(8080)` so gestartet, dass sie auf dem Port 8080 Eingaben erwartet. Die zweite Datei `alexa_mod.pl` enthält neben dem Rest des für die Umsetzung des PROLOG Webservers erforderlichen Quellcode die Prädikate, die für das Bearbeiten des Skills benötigt werden.

### 3.2.1. Prolog Webserver

Um mit Amazon kommunizieren zu können, muss ein Webserver eingerichtet werden, wobei sich die Arbeit auch an dem bereits erwähnten Beispiel einer PROLOG Anwendung mit Amazon Alexa [24] orientiert. Amazon sendet die Requests an den standardmäßig für HTTPS verwendeten Port 443. Diese Eingabe muss dann über einen Reverse Proxy[27] an einen Port weitergeleitet werden, auf dem die Anwendung lauscht. Für die Umsetzung wurde bei ***AlexaACE*** nginx[28] als Proxy-Server verwendet. Teile der Konfiguration davon können im folgenden Codefragment 3.3 betrachtet werden.

Listing 3.3: Einrichten eines Proxy-Servers mit nginx

```
1  server {
2     % Receiving messages on the HTTPS default port
3     listen 443 ssl http2;
4     listen [::]:443 ssl http2;
5     ...
6     % Set the required ssl certificate
7     ssl_certificate...
8     ...
9     location / {
10       ...
11       % Forward the message from port 443 to port 8080
12       proxy_pass        http://127.0.0.1:8080;
13    ...
14 }
```

Hier wird deutlich, dass die erhaltenen Informationen über Port 443 an den Port 8080 weitergeleitet werden. Dort muss außerdem der PROLOG Webserver gestartet werden. Die Implementierung stützt sich auf das in Listing 2.1 vorgestellte Gerüst, wobei es auf ***AlexaACE*** angepasst und auf zwei PROLOG Dateien aufgeteilt wird. Der erste Teil befindet sich in `run.pl`.

[27] Reverse Proxy mit nginx: `https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/`
[28] Webseite nginx: `https://www.nginx.com/`

Dieser unterscheidet sich nur im `http_handler/3` von der unter Listing 2.1 erwähnten Standardimplementierung. Dort wird als Belegung der Variable `Closure` das Prädikat `alexa` eingegeben, unter dem der zweite Teil des Webservers abläuft. Des Weiteren werden an der dritten Stelle des Prädikats `http_handler/3` Optionen für die Konfiguration des Webservers eingegeben, mit denen beispielsweise ermöglicht wird, POST-Requests zu verarbeiten.

In der Datei `alexa_mod.pl` wird die Implementierung des Prädikats `alexa` realisiert, welche im Quellcode 3.4 zu sehen ist. Dabei wird zunächst auf den empfangenen Request von Amazon, welcher im JSON-Format übermittelt wird, das Prädikat `http_read_json_dict/2` angewendet. Dadurch werden die wichtigen Informationen in einem Dict gespeichert, um sie später unter PROLOG weiter verwenden zu können. Das Prädikat `handle_dict/2` wird im Model definiert und dient dazu, die Eingabe in `DictIn` zu verarbeiten und das Ergebnis zu berechnen, welches in `DictOut` gespeichert wird. Zum Schluss wird dieses durch das von den eingebundenen Bibliotheken bereitgestellte Prädikat `reply_json/1` wieder ins JSON-Format konvertiert, um es an die Amazon Server zurückzuschicken.

Listing 3.4: PROLOG Webserver in alexa_mod.pl

```
1 % Generate the JSON output in the variable DictOut
2 alexa(Request) :-
3   % Extract the information of the received JSON input and store it in the
        variable DictIn
4   http_read_json_dict(Request, DictIn),
5   % Compute DictOut depending on the given DictIn
6   handle_dict(DictIn, DictOut),
7   % Generate the JSON reply
8   reply_json(DictOut).
```

Um den Server dann zu starten, sodass ***AlexaACE*** verwendet werden kann, muss lediglich die main-Methode der Datei `run.pl` auf der PROLOG Konsole ausgeführt werden, damit der Webserver auf dem Port 8080 lauscht.

### 3.2.2. Prolog Model

Sobald der PROLOG Webserver eine Eingabe erhalten hat, wird diese im Model verarbeitet. Dieser Vorgang wird in der PROLOG Datei `alexa_mod.pl` durchgeführt, wobei dafür mehrere Prädikate verwendet werden. Nachdem die Daten des JSON Inputs, wie in Listing 3.4 dargestellt, in der Variable DictIn gespeichert sind, werden die einzelnen Informationen in entsprechenden Variablen abgelegt. Beispielsweise kann so über `IntentName` herausgefunden werden, um welchen der drei in ***AlexaACE***

definierten Intents es sich handelt. Mithilfe des Prädikats `handle_dict/2`, welches auch im Zuge der Implementierung des Webservers aufgerufen wird, wird eines der implementierten Prädikate `intent_dictOut/3` aufgerufen. Diese behandeln die drei Intents, sowie den Fall, dass der Wert aus `IntentName` mit keinem der möglichen Intent Namen übereinstimmt. Falls dies eintritt, wird die Ausgabe `'Error Parsing'` an die Amazon Server zurückgegeben. Im vorliegenden Quelltext 3.5 ist das entsprechende Prädikat für den Intent „prove“ dargestellt.

Listing 3.5: Umsetzung des Intents „prove“ in alexa_mod.pl

```
1  % Execute the prove Intent of the skill
2  intent_dictOut("prove", DictIn, DictOut) :-
3     % Extract important infomation of the DictIn by using the predefined
          predicate get_dict/3
4     ...
5     get_dict(request, DictIn, RequestObject),
6     get_dict(intent, RequestObject, IntentObject),
7     get_dict(slots, IntentObject, SlotsObject),
8     get_dict(proveslot, SlotsObject, MySlotObject),
9     get_dict(value, MySlotObject, ValueMaybeDot),
10    % Add a dot to a sentence, if there is no dot available in the inserted
          sentence; this is required for RACE
11    maybe_dot(ValueMaybeDot, Value),
12    ...
13    % Get the whole knowledge which is stored in the database
14    get_whole_database(WholeDatabase),
15    ...
16    % Solve if the inserted theorem can be proven with the available
          knowledge by using prove_with_answers/3 of the library(race)
17    prove_with_answers(WholeDatabase, Value, Result),
18    ...
19    % Check if the received result contains the term 'results'
20    (Result = results([ResultsString|_]) ->
21       % if there are results print the first derivation
22       ...
23       % Concat the answer
24       string_concat('Yes, the theorem is right and could be proven with: ',
             ResultsString, AnswerProof)
25    ;
26       % else (if the theorem could not be proven)
27       ...
28       % Concat the answer
29       atom_concat('The theorem could not be proven. ', 'Try again',
             AnswerProof)
30    ),
```

```
31    % Generate DictOut
32    my_json_answer(AnswerProof, DictOut).
```

Dieser Intent prüft, ob ein Theorem aus bereits vorhandenem Wissen abgeleitet werden kann. In der Implementierung dafür werden zunächst alle wichtigen Informationen aus `DictIn` extrahiert und, wie beispielsweise der Wert des Slots, in eigenen Variablen abgelegt. Nach der Sicherstellung, dass der eingegebene Satz durch einen Punkt beendet wird, mithilfe des Hilfsprädikats `maybe_dot/2`, wird der Inhalt der Datenbasis über das eigens programmierte `get_whole_database/1` in der Variable `WholeDatabase` gespeichert. Zusammen mit dem eingegebenen Theorem wird diese für die Ausführung von `prove_with_answers/3`[29] benötigt. Dabei handelt es sich um ein von der Bibliothek *library(race)* [25] bereitgestelltes Prädikat. Darüber wird die entsprechende Funktionalität des RACE Webservices ausgeführt. Das Ergebnis, das in `Result` gespeichert ist, enthält entweder eine Liste von Ableitungswegen oder aber ist leer, falls das Theorem nicht bewiesen werden konnte. Bei letzterem Fall generiert das Model eine Antwort aus der ersichtlich wird, dass die Eingabe nicht abgeleitet werden konnte. Enthält das von RACE erhaltene Ergebnis aber mindestens eine Herleitung, so wird diese zusammen mit der Ausgabe **`'Yes, the theorem is right and could be proven with:'`** in der Variable `DictOut` abgelegt und in JSON-Format gebracht.

Die Implementierung für den Intent „question“ ist sehr ähnlich zu dem zuvor beschriebenen. Dabei ändert sich lediglich die Überprüfung des Satzzeichens auf ein Fragezeichen mithilfe von `maybe_questionmark/2` und das Einbinden des entsprechenden Prädikates von RACE zu `ask_with_answers/3`. Der genaue Quellcode dieses Intents ist zum Nachlesen auch im Anhang A.2 unter Listing A.3 zu finden.

Im Gegensatz zu den zuvor beschriebenen Intents, erwartet „remember“ keine Herleitung eines Faktes oder die Beantwortung einer Frage, sondern dient in erster Linie dazu Wissen abzuspeichern. Damit die Datenbasis keine widersprüchlichen Fakten ablegt, erfolgt in diesem Prädikat die Überprüfung auf Inkonsistenzen. Nachdem auch hier, wie bei „prove“ genauer beschrieben, wichtige Informationen extrahiert wurden und sichergestellt wurde, dass die Eingabe am Ende des Satzes einen Punkt besitzt, wird die Konsistenz der Eingabe überprüft. Dafür wird der abzuspeichernde Fakt in das aus *library(race)* bereitgestellte Prädikat `check_consistency/3` eingesetzt. Falls die Rückgabe eine leere Liste beinhaltet, liegt keine Inkonsistenz vor und der Fakt wird anschließend im Zusammenhang mit der bereits vorhandenen Datenbasis auf Widersprüche überprüft. Auch dies geschieht über das Prädikat `check_consistency/3`. Liegen auch hier keine Inkonsistenzen vor, wird der neue

[29] Zu Gunsten der einfacheren Darstellung wurde in Abbildung 3.1 `prove/3` statt `prove_with_answers/3` verwendet. Analog dazu `ask/3` statt `ask_with_answers/3`

Fakt zur Datenbasis hinzugefügt und eine Ausgabe generiert, die den Nutzer über das erfolgreiche Speichern informieren soll. Liegt in einer der beiden Prüfungen auf Widersprüche eine Inkonsistenz vor, so wird der von RACE übermittelte Grund, das heißt der inkonsistente Fakt oder ein Paar von widersprüchlichen Fakten, in der Ausgabe gespeichert. Dies gibt dem Nutzer die Möglichkeit nachzuvollziehen, warum das eingegebene Wissen nicht abgespeichert werden konnte.

### 3.2.3. Prolog *library(race)*

Als Schnittstelle zwischen der PROLOG Implementierung und dem RACE Webservice dient die Bibliothek *library(race)* [25]. Diese übernimmt die Kommunikation des Remote Procedure Calls über SOAP und stellt dem PROLOG Entwickler Prädikate zur Verfügung, um die unter Abschnitt 2.4 vorgestellten Funktionalitäten nutzen zu können. Diese sind folgendermaßen definiert:

- `check_consistency(+Knowledge, -Inconsistencies)`: Dieses Prädikat erhält als Eingabe eine Menge an Fakten bzw. zu überprüfenden ACE Sätzen und ruft damit die entsprechende Funktion des RACE Webservices auf. Liegt keine Inkonsistenz vor, so enthält das Argument `Inconsistencies` eine leere Liste. Bestehen innerhalb der eingegebenen Faktenmenge Widersprüche, so wird der inkonsistente Fakt oder ein Paar von Fakten, welches die Inkonsistenz hervorruft, in der Variable `Inconsistencies` gespeichert. Damit lässt sich nachvollziehen, an welcher Stelle ein Widerspruch in der Wissensdatenbasis vorliegt. Ist dies nicht erwünscht, so bietet *library(race)* noch das Prädikat `check_consistency(+Knowledge)`, welches lediglich einen Wahrheitswert zurück liefert.

- `ask_with_answers(+Knowledge, +Question, -Result)`: Mithilfe dieses Prädikats wird versucht, eine Frage aufgrund einer vorhandenen Datenbasis zu beantworten. Die unter `Knowledge` eingegebenen bekannten Fakten werden zusammen mit der Frage in der Variable `Question` an den RACE Webservice übermittelt und dort das berechnete Ergebnis im Argument `Result` gespeichert. Enthält dieses lediglich eine leere Liste, so konnte die Frage nicht beantwortet werden, während dagegen sonst die möglichen Ableitungen gespeichert werden, die zur Beantwortung nötig waren. Diese sind, um die Ausgabe zu erleichtern, bereits als englische Sätze abgelegt. Sind detailliertere Informationen über die Substitution und die verwendeten Fakten und Regeln erwünscht, so bietet *library(race)* auch das Prädikat `ask(+Knowledge, +Question, -Result, +Options)`.

- `prove_with_answers(+Knowledge, +Theorem, -Result)`: Dieses Prädikat dient zur Überprüfung eines Theorems auf Basis einer bekannten Faktenmenge. Das Wissen wird dabei dem Argument `Knowledge` übergeben, der zu beweisende Satz der Variable `Theorem`. Die Ausgabe der Herleitung erfolgt über `Result`, wobei das Theorem nicht bewiesen werden konnte, falls die Liste leer ist. Ansonsten enthält diese allerdings alle möglichen Ableitungen der zu beweisenden Eingabe in englischen Sätzen. Analog zu dem unter `ask_with_answers/3` beschriebenen zusätzlichen Prädikat, gibt es auch für das Überprüfen von Theoremen ein Prädikat `prove(+Knowledge, +Theorem, -Result, +Options)`, das eine genauere Darstellung als die textuelle Ausgabe anbietet.

Um die Bibliothek zu nutzen, muss man, der Anleitung [25] folgend, zunächst erforderliche Pakete installieren. Anschließend kann man nach dem Aufnehmen der Codezeile `use_module(library(race))` die Prädikate von *library(race)* nutzen.

## 3.3. ACE Reasoner (RACE)

RACE bietet drei mögliche Funktionalitäten an, die mithilfe des Webservices über einen Remote Procedure Call ausgeführt werden können. Neben dem Prüfen eines Theorems und dem Beantworten von Fragen aufgrund von vorhandenem Wissen, können auch Eingaben auf Konsistenz überprüft werden, wobei die *library(race)*, wie unter Abschnitt 3.2.3 beschrieben, jeweils das entsprechende PROLOG Prädikat zur Verfügung stellt.

Die Bibliothek *library(race)* ermöglicht außerdem die Kommunikation zwischen der PROLOG Komponente und dem Webservice des Attempto Controlled English Reasoners. Dafür wird ein Remote Procedure Call verwendet, worüber eine Anfrage an RACE gestellt wird und nach der Bearbeitung die Antwort übermittelt wird. Da SOAP als Protokoll für die Kommunikation genutzt wird, müssen sowohl Request als auch Reply in einer bestimmten Form, die in XML-Format codiert ist, vorliegen.

In dem von der Webseite[30] entnommenen Listing 3.6 ist beispielsweise dargestellt, wie eine Anfrage aufgebaut ist. In diesem Beispiel wird die Konsistenz des eingegebenen ACE Textes „John is a man. Mary is a woman. John is not a man.“ überprüft.

Listing 3.6: Request an den RACE Webservice für eine beispielhafte Eingabe

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
3   <env:Body>
```

[30]RACE Webservice: `http://attempto.ifi.uzh.ch/site/docs/race_webservice.html`

```
4         <race:Request xmlns:race="http://attempto.ifi.uzh.ch/race">
5             <race:Axioms>John is a man. Mary is a woman. John is not a man.</
                  race:Axioms>
6             <race:Mode>check_consistency</race:Mode>
7         </race:Request>
8     </env:Body>
9 </env:Envelope>
```

RACE soll nun testen, ob das übermittelte Wissen widersprüchlich ist. Dafür wird, wie in den Kapiteln 2.3.3 und 2.4 beschrieben, der ACE Text in DRS und first-order logic Klauseln umwandelt. Damit kann dann überprüft werden, ob die Eingabe konsistent ist oder nicht. Die genaue Ausführung ist für den Nutzer leider nicht nachvollziehbar, es ist lediglich bekannt, dass dafür im Hintergrund mehrere Hilfsprädikate nötig sind [12].

Nach der Bearbeitung werden die Ergebnisse analog zur Anfrage über SOAP zurück an die Prolog Komponente übermittelt. Dabei dient die Bibliothek *library(race)* erneut dazu, die Kommunikation zu übernehmen und wandelt die, unter Listing 3.7 dargestellte, erhaltene Ausgabe um, sodass sie unter Prolog verarbeitet werden kann. Im vorliegenden Beispiel besteht das Ergebnis aus einem Paar von inkonsistenten Axiomen, da „John is a man.“ dem weiteren Fakt „John is not a man.“ widerspricht. Diese Inkonsistenz wird von der Bibliothek *library(race)* mithilfe einer Liste an das Prolog Model übermittelt, wo die Ausgabe generiert wird, um sie an Amazon und anschließend an den Nutzer weiterzuleiten.

Listing 3.7: Reply des RACE Webservices für eine beispielhafte Eingabe

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <env:Envelope
3     xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
4     xmlns:race="http://attempto.ifi.uzh.ch/race">
5   <env:Body>
6     <race:Reply>
7       <race:Runtime>30</race:Runtime>
8       <race:Proof>
9         <race:UsedAxioms>
10          <race:Axiom>John is not a man.</race:Axiom>
11          <race:Axiom>John is a man.</race:Axiom>
12        </race:UsedAxioms>
13        <race:UsedAuxAxioms/>
14      </race:Proof>
15    </race:Reply>
16  </env:Body>
17 </env:Envelope>
```

# 4. Evaluierung

Nach der Beschreibung der Funktionsweise der Anwendung ***AlexaACE*** soll nun evaluiert werden, wie gut die Zielsetzungen dafür umgesetzt sind. Das Hauptziel dabei war die Beantwortung der Frage, ob und wie eine Implementierung in PROLOG entwickelt werden kann, die es ermöglicht mit Amazon Alexa unter Einbindung der CNL Attempto Controlled English in natürlicher Sprache zu kommunizieren. Das entwickelte ***AlexaACE*** stellt hierfür einen ersten Lösungsansatz dar. Wie im Kapitel 3 beschrieben, ist es also möglich, diese Form von Natural Language Processing umzusetzen, wobei das Ergebnis in diesem Kapitel genauer beurteilt werden soll. Für die genaue Evaluierung werden sowohl die unter Abschnitt 1.3 aufgelisteten Qualitätsziele betrachtet, sowie im Absatz 4.2 aufgeworfene Problemstellungen und vorhandene Begrenzungen von ***AlexaACE*** aufgezeigt.

## 4.1. Qualitätsziele

Neben der Funktionstüchtigkeit einer Anwendung und der Erfüllung des geforderten Funktionsumfangs, sollte eine Software auch den aufgestellten Qualitätszielen genügen.

- **Benutzerfreundlichkeit:** Die intuitive Bedienung von ***AlexaACE*** sollte auch Menschen ohne großen Vorkenntnissen zur Applikation und Erfahrungen in diesem Bereich ermöglicht werden. Möchte ein Nutzer die Anwendung verwenden, so muss er sich zum einen mit der zum Skillaufruf erforderlichen Struktur beschäftigen, zum anderen mit den Konventionen von ACE Texten. Ersteres ist relativ leicht zu erlernen, da sich die Struktur, wie in Abschnitt 2.1 beschrieben, an der natürlichen Sprache orientiert. Das Erlernen der von Attempto Controlled English erlaubten Satzstrukturen ist etwas anspruchsvoller. In einer der Veröffentlichungen [10] der Entwickler wird der aufzubringende Aufwand, der für die Nutzung dieser CNL ausreichend ist, auf ca. 1-2 Tage geschätzt. Insgesamt ist ***AlexaACE*** also eine annähernd intuitive und in nahezu natürlicher Sprache bedienbare Anwendung.

- **Zuverlässigkeit:** Die entwickelte Anwendung sollte zuverlässig arbeiten und die unter Kapitel 1.3 definierten Ziele erfüllen. Wie anhand der beschriebenen Funktionsweise von ***AlexaACE*** im Kapitel 3 ersichtlich, ist es möglich über Amazon Alexa in natürlicher Sprache zu kommunizieren und die mithilfe von ACE bzw. RACE gewonnenen Informationen unter PROLOG zu verarbeiten. Dabei wird die Beschränkung auf eine geringe Menge an möglichen abzuspeichernden Fakten, wie im anfangs beschriebenen Beispiel einer derartigen Anwendung [2], aufgelöst. In ***AlexaACE*** kann man also jegliches Wissen ablegen, solange die Eingaben die Konventionen von ACE erfüllen. Da Amazon Alexa allerdings manche Spracheingaben nicht korrekt versteht, kommt es an dieser Stelle gelegentlich zu Problemen in der Ausführung. Ein falsch erkannter und an RACE weitergeleiteter Satz könnte dementsprechend gegen die Konstruktionsregeln von Attempto Controlled English verstoßen und somit nicht richtig verarbeitet werden. Genauso wurden während der Testphase der Anwendung Probleme bei der Testkonsole festgestellt. So kommt es teilweise zu nicht reproduzierbaren und nicht nachvollziehbaren Fehlermeldungen, die die Eingabe nicht zulassen oder nicht bearbeiten.

- **Performanz:** Die Performanz sollte im Bezug auf die Eingabe möglichst gut sein. Funktionieren die Schnittstellen von Amazon zuverlässig, so ist es auch möglich, eine gute Performanz zu erreichen. ***AlexaACE*** kann in Echtzeit verwendet werden, die Bearbeitungszeit liegt im niedrigen einstelligen Sekundenbereich. Diese verlängert sich bei einer entsprechend größeren Datenbasis, aufgrund des erhöhten Aufwands bei der Übertragung. Da durch die Einbindung der Bibliothek *library(race)*, und der damit verbundenen Nutzung des RACE Webservices, noch ein weiterer entfernter Server angesprochen werden muss, verlängert sich die Übertragungszeit gegenüber dem lokalen Aufruf auf dem Server der Anwendung entsprechend. Werden zu viele Fakten eingegeben, wird ein Timeout von Alexa erreicht, wobei dies oft leider schon ab ca. 15 abgespeicherten Fakten der Fall ist. Zudem besitzt auch RACE ein Zeitlimit [9], das erreicht ist, wenn die Bearbeitung nicht innerhalb von 10 Sekunden abgeschlossen werden konnte.

- **Austauschbarkeit von Komponenten:** Bei der Entwicklung sollte darauf geachtet werden, dass die einzelnen Komponenten von ***AlexaACE*** gut ausgetauscht werden können. Zum einen ist es möglich, die Funktionalität des PROLOG Models unabhängig vom Webserver zu ändern, da die Implementierungen gut getrennt werden. Die Verbindung von Webserver und Model besteht dabei lediglich über das Prädikat `handle_dict/2`. Viel wichtiger ist aber die Ermöglichung des Austausches der externen Software. Da auch hier die Schnittstellen über PROLOG Prädikate klar definiert sind, können sowohl Amazon Alexa, als

auch ACE durch andere Komponenten ersetzt werden, ohne große Änderungen am PROLOG Code vornehmen zu müssen.

- **Lesbarkeit und Verständlichkeit des Codes:** Für eine mögliche Änderung oder Erweiterung der Anwendung, sollte auch auf die Übersichtlichkeit des Quellcodes geachtet werden. Dieser ist, wie im Anhang A.2 zu sehen, zufriedenstellend und vor allem lückenlos kommentiert, sodass die Funktionalität der einzelnen PROLOG Prädikate deutlich wird. Außerdem geben die Bezeichnungen der Prädikate und Variablen bereits Aufschluss darüber, welche Bedeutung die jeweilige Struktur im Zusammenhang hat. Dadurch wird es dementsprechend erleichtert, den Code zu verstehen.

## 4.2. Einschränkungen

Auch wenn ***AlexaACE***, im Gegensatz zum unter Abschnitt 1.2.5 beschriebenen Projekt, die Reduktion auf wenige vorher festgelegte Fakten und Regeln aufhebt, ergeben sich auch hier Beschränkungen.

Zunächst zu nennen ist die Einschränkung auf englische Sätze, die sich durch die Nutzung von ACE unweigerlich ergibt. Diese CNL ist durch den englischen Wortschatz und ihre Grammatik lediglich in der Lage, Englisch zu verarbeiten. Dies umfasst auch die Verwendung von Eigennamen, die nicht aus dem englischen Wortschatz stammen, welche somit nicht als solche erkannt werden. Gegebenenfalls kann ***AlexaACE*** diesbezüglich aber erweitert bzw. verändert werden, da, wie unter Abschnitt 4.1 beschrieben, der Austausch der verwendeten Controlled Natural Language leicht umgesetzt werden kann. Des Weiteren könnte das Vokabular beispielsweise um Eigennamen ergänzt werden, wenn es möglich wäre, ACE direkt auf dem Server, auf dem ***AlexaACE*** läuft, lokal einzubinden.

Da der Quellcode von ACE allerdings nicht zur Verfügung steht, kann ***AlexaACE*** die CNL nicht direkt einbinden, weswegen der Umweg über RACE notwendig ist. Dieser hat auch zur Folge, dass große Eingaben aufgrund des Timeouts des ACE Reasoners nicht verarbeitet werden können. Dementsprechend kann über die entwickelte Anwendung keine beliebig große Faktenmenge eingegeben werden. Möglich wäre ein Umgehen dieser Einschränkung durch das direkte Einbinden der CNL bzw. des Reasoners auf dem Server der Anwendung. So könnte eine Übertragung auf einen entfernten Server vermieden werden, was sich positiv auf die Bearbeitungszeit auswirken würde und womit eine Reduktion auf eine geringe Faktenmenge umgangen werden könnte.

Von den durch RACE ausgegebenen Herleitungen bei den Intents „question“ und „prove“ wird, wie im Abschnitt 3.1.2 beschrieben, lediglich die erste in der durch das Prolog Model erhaltenen Liste zurückgegeben. Dies dient hauptsächlich der Übersichtlichkeit, da oft mehrere sehr lange Herleitungen existieren. Eine Möglichkeit dem interessierten Nutzer von ***AlexaACE*** diese Informationen trotzdem zur Verfügung zu stellen, wäre eine weitere Nutzereingabe zu erlauben, die die Aufforderung zur weiteren Ausgabe enthält. Zur Zeit ist diese Funktionalität allerdings nicht umgesetzt.

Außerdem ist die unter Abschnitt 4.1 beschriebene natürliche Nutzung durch die Konventionen von ACE und das feste Eingabeschema von Amazon Alexa eingeschränkt. Vollkommen freie, von festen Strukturvorgaben losgelöste, Eingaben sind demnach leider noch nicht möglich.

Vor allem aber beschränkt sich die Anwendung ***AlexaACE*** auf die drei beschriebenen Funktionalitäten Ablegen von Wissen nach einer Konsistenzprüfung, Beantworten von Fragen und Überprüfen von Theoremen. Auch hier könnten die Funktionalitäten aber durch den Austausch von Komponenten und einer Veränderung des Models geändert werden.

# 5. Zusammenfassung und Ausblick

Als Abschluss werden in diesem Kapitel unter Abschnitt 5.1 die wichtigsten Aspekte der Arbeit zusammengefasst. Zudem gibt das letzte Unterkapitel 5.2 einen Ausblick in mögliche Erweiterungen von ***AlexaACE*** und neue Einsatzmöglichkeiten.

## 5.1. Zusammenfassung

Diese Arbeit präsentiert einen Lösungsansatz für eine spezielle Form des Natural Language Processing. Dabei soll die nahezu natürliche Kommunikation eines Nutzers mit einem Prolog Server über Amazon Alexa und unter Einbindung der speziellen Controlled Natural Language Attempto Controlled English ermöglicht werden. Es wurde gezeigt, dass eine derartige Anwendung entwickelt werden kann.

Hierfür wurden zunächst in Kapitel 2 die wichtigsten verwendeten Technologien vorgestellt. Neben Amazon Alexa sind hier vor allem Prolog und ACE zu erwähnen. Ersteres diente bei der Entwicklung von ***AlexaACE*** als Programmiersprache für den Webserver, auf dem die Anwendung läuft. Um die Nutzeranfragen bearbeiten zu können, wird RACE verwendet, wobei dieses Tool auf Attempto Controlled English basiert.

Im anschließenden Kapitel 3 wurde die Anwendung ***AlexaACE*** und deren Funktionsablauf detailliert vorgestellt. Wichtig ist dabei der Aufbau, der sich aus drei Komponenten zusammensetzt:

- Amazon Alexa dient als Schnittstelle zwischen Nutzer und dem Server, auf dem die Anwendung ausgeführt wird. Dabei übernimmt diese Komponente die Spracherkennung und die Weiterleitung der Eingabe an den Prolog Webserver. Nach der Bearbeitung der Anwendung wird die erhaltene Antwort außerdem wieder über die genutzte Schnittstelle von Amazon Alexa an den Nutzer übermittelt.

- Die Prolog Komponente setzt sich aus mehreren Teilen zusammen. Zum einen wird für die Verbindung zu Amazon Alexa ein Webserver benötigt. Dieser übernimmt das Empfangen von Nutzereingaben und leitet diese an das

PROLOG Model weiter. Außerdem sendet er die von der Anwendung generierte Antwort wieder zurück an Amazon Alexa. Die Bearbeitung findet dabei im Model statt, welches auf eine Datenbasis zugreifen kann und die bereitgestellten Prädikate der Bibliothek *library(race)* einsetzt. Diese dient als Schnittstelle zwischen dem Model und Attempto Controlled English.

- Das von ACE bereitgestellte Tool RACE erhält von der PROLOG Komponente über einen Remote Procedure Call übermittelte Anfragen an das System. Diese werden durch den RACE Webservice bearbeitet und die Ergebnisse wieder über die Bibliothek *library(race)* als Schnittstelle zurück gesendet.

In der Evaluation, welche im Kapitel 4 vorgenommen wurde, konnte festgestellt werden, dass die entwickelte Anwendung die geforderten Ziele größtenteils erfüllt. Daneben wurden die Einschränkungen von ***AlexaACE*** aufgezeigt, die im momentanen System nicht gelöst werden konnten.

## 5.2. Ausblick

In der Weiterentwicklung könnte versucht werden, die beschriebenen Beschränkungen zu umgehen. Diese basieren vor allem auf Einschränkungen der externen Systeme Amazon Alexa und Attempto Controlled English. Durch die Einbindung einer auf einer anderen natürlichen Sprache, wie beispielsweise Deutsch, basierenden CNL, könnte man die Reduktion auf englische Sätze umgehen. Auch könnte mithilfe eines Austausches der Komponente ACE bzw. RACE die Beschränkung auf die Anzahl der eingegebenen Fakten aufgelöst werden. Des Weiteren könnte statt Amazon Alexa eine andere Spracherkennungssoftware verwendet werden, die weniger auf vordefinierte Strukturen festgelegt ist. In diesem Zuge könnte versucht werden, eine eher gesprächsähnliche Ausführung der Anwendung umzusetzen, um beispielsweise nach einer Nutzeraufforderung alle Herleitungen eines Theorembeweises ausgeben zu lassen. Wäre das System in der Lage eine Nutzerauthentifizierung durchzuführen, könnte man möglicherweise auch Nutzerkonten verwalten und von einem Nutzer bereits eingegebene Fakten speichern und zu einem späteren Zeitpunkt wieder abrufen.

Aus diesen beschriebenen Erweiterungen ergäben sich auch weitere Anwendungsbereiche derartiger Systeme. Beispielsweise könnten diese, wie in [4] beschrieben, im Bereich E-Learning eingesetzt werden, wobei persönliche Fortschritte gespeichert und die Informationen darüber für angepasstere Übungsaufgaben und ähnliches genutzt werden könnten. Auch in der Beratung, wie bereits in Supermärkten[31], könnten ver-

[31] Artikel über den Einsatz von Robotern in Supermärkten: `https://www.handelsblatt.com/technik/forschung-innovation/einzelhandel-wenn-der-roboter-beim-einkauf-hilft/20539752.html?ticket=ST-1881289-t1XuVLeo7CIJXNiSxSXx-ap3`

gleichbare Anwendungen genutzt werden. Hier gibt es zum Teil bereits interaktive Roboter, die versuchen den Kunden zu helfen, Produkte zu finden oder Informationen darüber auszugeben. Hätte man die Möglichkeit einzelne Nutzerkonten zu verwalten, so könnten Fakten und Regeln eingegeben werden, wie beispielsweise Lebensmittelunverträglichkeiten oder persönliche Vorlieben, und zusammen mit vom Supermarkt eingegebenen aktuellen Rabattaktionen kundenspezifische Angebote errechnet werden. Dabei könnte jeder Kunde in dem Geschäft diesen Service nutzen, wenn die Kommunikation in natürlicher Sprache abliefe, und das System so intuitiv und ohne Vorwissen bedienbar wäre.

Wie diese Beispiele zeigen, wird Natural Language Processing sicherlich auch in Zukunft eine große Rolle in der Informatik spielen, gerade im Bezug auf sprachgesteuerte Assistenzsysteme. Als Schlusswort dieser Arbeit soll folgendes Zitat [1] dienen: „[...] many people seem to think it should be easy for computers to deal with human language, just because they themselves do so easily. “ Die Herausforderungen, die sich durch die Erweiterung von NLP Systemen und aus Versuchen, diese zu verbessern, ergeben, sind allerdings unterschiedlich und erfordern noch viel Arbeit in der Forschung.

# Literaturverzeichnis

[1] Madeleine Bates. Models of natural language understanding. *Proceedings of the National Academy of Sciences of the United States of America*, 92(22):9977–9982, 1995.

[2] Wouter Beek and Sam Neaves. Amazon Alexa skill with SWI-Prolog, GitHub Repository. `https://github.com/PlayingWithProlog/alexa`, 2017. Commit: 1a947c49fbc2177ffa9de2091dabd24071626a2a. Accessed: 2018-08-05.

[3] Christian Bitter, David A Elizondo, and Yingjie Yang. Natural language processing: a prolog perspective. *Artificial Intelligence Review*, 33(1-2):151–173, 2010.

[4] Michael J Callaghan, Victor B Putinelu, Jeremy Ball, Jorge C Salillas, Thibault Vannier, Augusto G Eguíluz, and Niall McShane. Practical Use of Virtual Assistants and Voice User Interfaces in Engineering Laboratories. In *Online Engineering & Internet of Things*, pages 660–671. Springer, 2018.

[5] Peter Clark and Niranjan Balasubramanian. Interpreting and Reasoning with Simple Natural Language Sentences. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.4998&rep=rep1&type=pdf`, 2014.

[6] Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *History of programming languages—II*, pages 331–367. ACM, 1992.

[7] Atli F Einarsson, Patrekur Patreksson, Mohammad Hamdaqa, and Abdelwahab Hamou-Lhadj. SmartHomeML: Towards a Domain-Specific Modeling Language for Creating Smart Home Applications. In *2017 IEEE International Congress on Internet of Things (ICIOT)*, pages 82–88. IEEE, 2017.

[8] Norbert E Fuchs. First-Order Reasoning for Attempto Controlled English. In *International Workshop on Controlled Natural Language*, pages 73–94. Springer, 2010.

[9] Norbert E Fuchs. Reasoning in Attempto Controlled English: Non-Monotonicity. In *International Workshop on Controlled Natural Language*, pages 13–24. Springer, 2016.

[10] Norbert E Fuchs, Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider. Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In *Proceedings of the First international conference on Reasoning Web*, pages 213–250. Springer, 2005.

[11] Norbert E Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In *Reasoning Web*, pages 104–124. Springer, 2008.

[12] Norbert E Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In *FLAIRS Conference*, volume 12, pages 664–669, 2006.

[13] Norbert E Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English (ACE) Language Manual Version 3.0. `http://attempto.ifi.uzh.ch/site/pubs/papers/ace3manual.pdf`, 1999.

[14] Norbert E Fuchs and Rolf Schwitter. Specifying Logic Programs in Controlled Natural Language. *Workshop on Computational Logic for Natural Language Processing*, 1995.

[15] Norbert E Fuchs and Rolf Schwitter. Attempto Controlled English (ACE). *The First International Workshop on Controlled Language Applications*, 1996.

[16] Norbert E Fuchs and Rolf Schwitter. Attempto Controlled English. `http://attempto.ifi.uzh.ch/site/talks/files/Talk.Stanford.05.pdf`, 2005.

[17] Stefan Hoefler. The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0. `http://attempto.ifi.uzh.ch/site/pubs/papers/hoefler2004theSyntax.pdf`, 2004.

[18] Lucja Iwańska. Logical Reasoning in Natural Language: It Is All about Knowledge. *Minds and Machines*, 3(4):475–510, 1993.

[19] Kaarel Kaljurand. *Attempto controlled English as a Semantic Web language*. PhD thesis, University of Tartu, 2007.

[20] Robert A Kowalski. The early years of logic programming. *Communications of the ACM*, 31(1):38–43, 1988.

[21] Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, 2014.

[22] Adam Lally and Paul Fodor. Natural Language Processing With Prolog in the IBM Watson System. *The Association for Logic Programming (ALP) Newsletter*, 2011.

[23] Gustavo López, Luis Quesada, and Luis A Guerrero. Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, 2017.

[24] Sam Neaves and Anne Ogborn. Playing with Prolog – Natural Language Reasoning with Alexa and SWI-Prolog. `https://www.youtube.com/watch?v=ScrEe1vsPug`, 2017. Accessed: 2018-08-05.

[25] Falco Nogatz. library(race), GitHub Repository. `https://github.com/fnogatz/race`, 2018. Commit: 22053a951b20875ad885b7a3323cc02a5706d494. Accessed: 2018-08-05.

[26] Pierre M Nugues. An Introduction to Prolog. In *An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German*, pages 433–486. Springer, 2006.

[27] Gianfranco Rossi. Uses of Prolog in Implementation of Expert Systems. *New generation computing*, 4(3):321–329, 1986.

[28] Rolf Schwitter. Controlled Natural Languages for Knowledge Representation. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1113–1121. Association for Computational Linguistics, 2010.

[29] Rolf Schwitter and Norbert E Fuchs. Attempto - From Specifications in Controlled Natural Language towards Executable Specifications. In *EMISA Workshop "Natuerlichsprachlicher Entwurf von Informationssystemen"*, 1996.

[30] Olga Štěpánková and Petr Štěpánek. Prolog: A Step towards the Future of Programming. In *Advanced Topics in Artificial Intelligence*, pages 50–81. Springer, 1992.

[31] Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljanovic, Brian Davis, Norbert E Fuchs, Stefan Hoefler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, et al. On Controlled Natural Languages: Properties and Prospects. In *International Workshop on Controlled Natural Language*, pages 281–289. Springer, 2009.

# Erklärung

Ich, Julia Kübert, Matrikel-Nr. 2052525, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

*Attempto Controlled English für Amazon Alexa*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Bachelorarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der Universität abzugeben oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Würzburg, den 13. August 2018

---

Julia Kübert

# A. Appendix

## A.1. Bildquellen aus Abbildung 3.1

- Amazon Logo: https://images-eu.ssl-images-amazon.com/images/G/03/misc/xsite/logos/a.de_logo_RGB_online_weiss.jpg
- SWI-Prolog Logo: http://www.swi-prolog.org/
- Abbildung des Amazon Echo: https://amazon-presse.de/Kindle---Fire/Echo-und-Alexa/Presskit?path=53ab8422-2681-43f4-8d9c-81c27eb9c894&aid=7f595fa5-7d3a-4d57-8e53-c80ac21c769e
- Abbildung der Server: https://pixabay.com/de/pc-schale-computer-server-gro%C3%9Fe-307363/
- Abbildung des Nutzers: https://pixabay.com/de/m%C3%A4nnchen-3d-model-freigestellt-3d-2364349/

## A.2. Wichtige Teile des Programmcodes der entwickelten Anwendung *AlexaACE*

### A.2.1. Extrahieren der Informationen des JSON-Inputs

In diesem Programmcode werden die wichtigsten Informationen des erhaltenen JSON-Inputs extrahiert und in entsprechenden Variablen abgelegt.

Listing A.1: Extrahieren der Informationen des JSON-Inputs in alexa_mod.pl

```
1 % Generate the JSON output in the variable DictOut
2 alexa(Request) :-
3   % Extract the information of the received JSON input and store it in the
      variable DictIn
4   http_read_json_dict(Request, DictIn),
5   % Compute DictOut depending on the given DictIn
6   handle_dict(DictIn, DictOut),
7   % Generate the JSON reply
```

```
8    reply_json(DictOut).
9
10 % Compute DictOut depending on the given DictIn
11 handle_dict(DictIn, DictOut) :-
12    % Extract the IntentName for the following execution
13    get_intent(DictIn, IntentName),
14    % Print the IntentName on the Prolog console
15    writeln(user_output, IntentName),
16    % Compute the DictOut for the inserted DictIn and the corresponding
          IntentName
17    intent_dictOut(IntentName,DictIn,DictOut).
18
19 % Extract the IntentName by using get_dict/3, which returns the information
        that is associated with the first argument
20 get_intent(DictIn, IntentName) :-
21    get_dict(request, DictIn, RequestObject),
22    get_dict(intent, RequestObject, IntentObject),
23    get_dict(name, IntentObject, IntentName).
```

### A.2.2. Bearbeitung der einzelnen Intents

In diesen Codefragmenten werden die Prädikate vorgestellt, welche die Ausgaben des jeweiligen Intents berechnen, wobei „prove“ bereits unter Listing 3.5 im Abschnitt 3.2.2 gezeigt wurde.

- **„remember“:**

Listing A.2: Umsetzung des Intents „remember“ in alexa_mod.pl

```
1 % Execute the remember Intent of the skill
2 intent_dictOut("remember", DictIn, DictOut) :-
3    % Extract important infomation of the DictIn by using the predefined
          predicate get_dict/3
4    get_dict(session, DictIn, SessionObject),
5    get_dict(sessionId, SessionObject, SessionId),
6    get_dict(request, DictIn, RequestObject),
7    get_dict(intent, RequestObject, IntentObject),
8    get_dict(slots, IntentObject, SlotsObject),
9    get_dict(rememberslot, SlotsObject, MySlotObject),
10   get_dict(value, MySlotObject, ValueMaybeDot),
11   % Add a dot to a sentence, if there is no dot available in the inserted
          sentence; this is required for RACE
12   maybe_dot(ValueMaybeDot, Value),
13   % debug message
```

```
14      debug(alexa, 'remember intent with value: ~w~n', [Value]),
15      % Convert the Value of the slot to a String
16      atom_string(ValueAtom, Value),
17      % Check consistency of the inserted sentence by using the predicate
            check_consistency of the library(race)
18      check_consistency(ValueAtom, Inconsistencies, Variant),
19      % check, if RACE returned insonsistencies
20      ( Inconsistencies \= [] ->
21         % if there are inconsistencies, store them
22         maplist(remove_fact, Inconsistencies, InconsistenciesFact),
23         % Convert them into a String
24         atomics_to_string(InconsistenciesFact, ' ',
               InconsistenciesFactReason),
25         debug(alexa, 'Inconsistent fact because: ~w~n',
               InconsistenciesFactReason),
26         % Concat the answer
27         atom_concat('Inconsistent fact because: ', InconsistenciesFactReason
               , InconsistenciesFactOutput),
28         % Generate DictOut
29         my_json_answer(InconsistenciesFactOutput, DictOut)
30         ;
31         % else (if there are no inconsistencies)
32         % print 'Consistent fact' on the Prolog console
33         writeln(user_output, 'Consistent fact'),
34         % Get facts from the database and combine it with the inserted
               sentence
35         combine_sentences(Variant, CombinedSentences),
36         % Print the content of the database on the Prolog console
37         format(user_output, 'Content Database: ~w~n', [CombinedSentences]),
38         !,
39         % Check consistency of the database together with the inserted
               sentence by using the predicate check_consistency of the library
               (race)
40         check_consistency(CombinedSentences, InconsistenciesDatabase),
41         % Print informationen of execution
42         writeln(user_output, 'Check with database...'),
43         % check, if RACE returned insonsistencies
44         (InconsistenciesDatabase \= [] ->
45            % if there are Inconsistencies, store them
46            maplist(remove_fact, InconsistenciesDatabase,
                  InconsistenciesDatabaseWithoutFact),
47            % Convert them into a String
48            atomics_to_string(InconsistenciesDatabaseWithoutFact, ' ',
                  InconsistenciesDatabaseReason),
```

```
49          debug(alexa, 'Inconsistent with database because: ~w~n',
                InconsistenciesDatabaseReason),
50          % Concat the answer
51          atom_concat('Inconsistent database because: ',
                InconsistenciesDatabaseReason, InconsistenciesDatabaseOutput)
                ,
52          % Generate DictOut
53          my_json_answer(InconsistenciesDatabaseOutput, DictOut)
54          ;
55          % else (if there are no insonsistencies)
56          % Add the new fact to the database
57          assert(knowledge(Variant)),
58          % Concat the answer
59          atom_concat('This fact was saved in the database: ', Value,
                SuccessfulAnswer),
60          debug(alexa, 'This fact was saved in the database: ~w~n', Value),
61          % Generate DictOut
62          my_json_answer(SuccessfulAnswer, DictOut)
63      )
64  ).
```

- **„question“:**

Listing A.3: Umsetzung des Intents „question“ in alexa_mod.pl

```
1  % Execute the question Intent of the skill
2  intent_dictOut("question", DictIn, DictOut) :-
3    % Extract important infomation of the DictIn by using the predefined
         predicate get_dict/3
4    get_dict(session, DictIn, SessionObject),
5    get_dict(sessionId, SessionObject, SessionId),
6    get_dict(request, DictIn, RequestObject),
7    get_dict(intent, RequestObject, IntentObject),
8    get_dict(slots, IntentObject, SlotsObject),
9    get_dict(questionslot, SlotsObject, MySlotObject),
10   get_dict(value, MySlotObject, ValueMaybeQuestionmark),
11   % Add a questionmark to a sentence, if there is no questionmark
         available in the inserted sentence; this is required for RACE
12   maybe_questionmark(ValueMaybeQuestionmark, Value),
13   % debug message
14   debug(alexa, 'question intent with value: ~w~n', [Value]),
15   % Get the whole knowledge which is stored in the database
16   get_whole_database(WholeDatabase),
17   % Print the knowledge on the Prolog console
18   writeln(user_output, 'The Database already knows: '),
19   writeln(user_output, WholeDatabase),
```

```
20    % Solve if the inserted question can be answered with the available
         knowledge by using ask_with_answers/3 of the library(race)
21    ask_with_answers(WholeDatabase, Value, Result),
22    % Print the received answer on the Prolog console
23    writeln(user_output, 'The server gave me this answer: '),
24    writeln(user_output, Result),
25    % Check if the received result contains the term 'results'
26    (Result = results([ResultsString|_]) ->
27       % if there are results print the first derivation
28       debug(alexa, 'Yes, the question could be answered with: ~w~n', [
            ResultsString]),
29       % Concat the answer
30       string_concat('Yes, the question could be answered with: ',
            ResultsString, AnswerQuery)
31       ;
32       % else (if the question could not be answered)
33       writeln(user_output, "The question could not be answered. "),
34       % Concat the answer
35       atom_concat('The question could not be answered because the
            requested facts are not available in the database. ', 'Try again
            ', AnswerQuery)
36       ),
37    % Generate DictOut
38    my_json_answer(AnswerQuery, DictOut).
```

- **kein definierter Intent:**

Listing A.4: Umsetzung nicht zuordenbarer Intents in alexa_mod.pl

```
1 % If the IntentName does not match with one of the defined below,
     generate a error message for the reply
2 intent_dictOut(_, _, DictOut) :-
3   my_json_answer('Error parsing', DictOut).
```

### A.2.3. Wichtige Hilfsprädikate

Hier werden wichtige eigens definierte Hilfsprädikate vorgestellt, die beispielsweise zur Bearbeitung der einzelnen Intents verwendet werden.

Listing A.5: Wichtige Hilfsprädikate in alexa_mod.pl

```
1 % Generates a JSON Answer with the required structure
2 my_json_answer(Message, X) :-
3   X = _{
4     response: _{
```

```
5         shouldEndSession: false,
6         outputSpeech:_{type: "PlainText", text: Message}
7       },
8       version:"1.0"
9     }.
10
11 % Checks if MaybeDot has a dot and add one, if there is no dot. Return a
       sentence with dot
12 maybe_dot(MaybeDot, SureDot) :-
13    (atom_concat(_, '.', MaybeDot) ->
14       MaybeDot=SureDot
15       ;
16       atom_concat(MaybeDot, '.', SureDot)
17    ).
18
19 % Combine the facts from the database with new knowledge stored in the
       variable 'Value'
20 combine_sentences(Value, CombinedSentences) :-
21    findall(Z, knowledge(Z), Sentences),
22    flatten(Sentences, SentenceList),
23    atomics_to_string(SentenceList, ' ', SentencesString),
24    atom_concat(SentencesString, Value, CombinedSentences).
25
26 % Extract the whole database
27 get_whole_database(WholeDatabase) :-
28    findall(Z, knowledge(Z), Sentences),
29    flatten(Sentences, SentenceList),
30    atomics_to_string(SentenceList, ' ', WholeDatabase).
```