Universität Trier
Computerlinguistik & Digital Humanities

Hausarbeit

zu dem Thema

Vorlage zur Erstellung einer Hausarbeit im Fach CL

23. Juni 2019

Vorname Name
Email: mein.name@uni-trier.de

Inhaltsverzeichnis

1 Einleitung

Chatbot Äugustaïst ein Chatbot, dessen Aufgabe es ist, seinem Nutzer Produkte aus dem Uni-Shop zu empfehlen. Dafür muss der Nutzer dem Bot Äugusta"via Texteingabe angeben, was gesucht wird. Diese Nutzereingaben werden mit Hilfe von Pattern-Matching analysiert und via PostgreSQL-Schnittstelle werden Antworten basierend auf Queries generiert, die dem geäuSSerten Wunsch des Nutzers entsprechen.

1.1 Ziele des Projekts

Vorrangiges Ziel des Projekts war der Versuch der Entwicklung eines computerlinguistischen Programms. Das Umsetzen im Studium angeeigneter Fähigkeiten theoretischer und praktischer Natur, wie z.B. Automatentheorie, Softwareentwurf und Datenbankentechnologie hat eine essentielle Rolle in der Konzeption und Realisierung des Programms gespielt. Weiteres Ziel war das Einarbeiten in ein fremdes Framework mit eigenem Interpreter, hier ChatScript (Wilcox 2019) von Bruce Wilcox, um ein entsprechendes Programm zu entwickeln.

1.2 Struktur der Arbeit

2 ChatScript: Grundlagen

Bei ChatScript handelt es sich um ein Framework, der es Entwicklern erlaubt, regelbasierte Chatbots zu entwickeln. Es handelt sich hierbei um "Natural Language tool"mit eigenem Interpreter, welcher Code in C++ übersezt.

2.1 ChatScript: Regeln

Regeln sind elementarer Bestandteil in ChatScript. In Regeln wird u.A. die Analyse von Nutzereingaben analysiert, Antworten des Bots festgelegt, Datenbankabfragen ausgeführt und der Gesprächsablauf gesteuert.

In Augusta finden sich Regeln in verschiedenen Formen wieder. Diese sind:

- t: für Ausgaben des Chatbots, ohne folgende Antwort
- u: für Ausgaben in Form von Fragen, das beduetet, dass auf Regeln mit u: eine Antwort des Nutzers folgen muss
- a:, b:, c: etc. für Antworten des Nutzers. Auf diese Regeln können Pattern-Matching-Ansätze angewandt werden, um Teile der Eingabe ggf. zu extrahieren.

Topics haben, auch wenn optional, meist einen Bezeichner, eine Bedingung zur Ausführung und es lassen sich dabei Variablen zuweisen. Ein abstraktes Beispiel:

Listing 2.1: Syntax für Regeln

```
t: BEZEICHNER (KONDITION) $meineVariable=Wert Ausgabe des Bots
```

Im folgenden einige Beispiele für Regeln:

Listing 2.2: Beispiel für t:

```
t:  ( %input<%userfirstline )
    ^keep()
    [Hallo] [Hi] [Guten Tag], ich bin Augusta und kann dich beraten, wenn du etwas aus
```

In dieser Regel wird der Nutzer begrüSSt und im folgenden der Gesprächsverlauf zur Regel 'GREET' weitergeleitet.

Listing 2.3: Beispiel für Gesprächsablauf mit Antworten u:, a: und b:

```
u: INTRO ($enter211) [Das ist gut] [Das ist toll] [Das freut mich], [dabei kann ich d

            a: ( ~positiv ) $introyes = 1a
    # DATENBANK
```

```
    if (^dbinit(dbname = postgres port = 5432 user = postgres password = 1234)
        {[Lass uns anfangen] [Super, auf geht's]!  ^reuse( FIRSTQ )
    } else {dbinit failed - $$db_error  ^reuse( FIRSTQ ) }
a: ( ~negativ ) Tut mir Leid, ich kann eigentlich nur beraten. Bist du dir sic
    b: ( ~negativ )  $introyes = 1a if (^dbinit(dbname = postgres port = 5
                                else {dbinit failed - $$db_error ^reuse(FIRSTQ)}
b: ( ~positiv ) $enterEnd2 = 1b Das ist schade. ^reuse( ~ende.ASKIFHAPPY )
```

In diesem Abschnitt fragt der Bot mit zufällig ausgewählter Frage, ob der Kunde etwas kaufen möchte. Falls die Eingabe des Kunden ein Wort aus dem Konzept positiv enthält, wird entsprechend die Datenbank geladen und in die Regel FIRSTQ im selben Topic aktiviert. Falls die Eingabe ein Wort aus dem Konzept negativ enthält, so wird gefragt, ob man nicht wirklich was sucht. Je nach Antwort darauf wird eine entsprechende Regel, d.h. eines der beiden b: Regeln ausgeführt.

2.2 ChatScript: Topics

Die Quelldateien in einem ChatScript-Programm werden auch als "Topics"bezeichnet und haben die Dateiendung .top. Topics können als Zusammenfassung mehrerer Regeln betrachtet werden. Es bietet sich an, Topics als Module eines Chatbots zu handhaben. So wird z.B. in diesem Projekt Äugusta"die Datenbankabfrage als eignenes Topic gehandhabt. Topics beginnen immer mit einer Tilde -äls Präfix gefolgt vom Topicnamen, meist als erste Zeile in einer .top-Datei oder vor Auflisting von Regeln:

Listing 2.4: Topicbezeichner in dbsearch.top

```
topic: ~dbsearch   [] ($gosearch)
```

2.3 ChatScript: Konzepte

Syntaktisch werden Konzepte wie topics gehandhabt, das heiSSt, dass Konzepte mit Konzeptname referenziert werden. Ein Konzept kann als eine Menge von Synonymen verstanden werden, ähnlich zu Einträgen in einem Thesaurus. Obwohl es sich hier um Mengen handelt, stehen diese aus syntaktischer Sicht in Klammern () oder []. Ein Beispiel anhand des Konzepts "ciao":

Listing 2.5: Konzept 'ciao' aus konzepte.top

```
concept: ~ciao [ciao tschüss "good bye" bye "daje" "eddi un merci" "äddi a merci"]
```

Anzumerken ist, dass multiword expressions in Anführungszeichen stehen müssen, da einzelne Bestandteile dieser sonst als einzelne Elemente betrachtet werden.

Konzepte sind besonders wichtig in Äugusta", da es sich anbietet, das Pattern-Matching von Konzepten abhängig zu machen. So wird mit Hilfe von Konzepten das Arbeiten mit der PostgreSQL-Schnittstelle gewährleistet: Für jeden möglichen Eintrag pro Spalte in der PostgreSQL Datenbank existiert eine Konzept. Als Beispiel alle Einträge für die Spalte Änwendungszweck":

Listing 2.6: Konzept 'anwendungszweck' aus konzepte.top

```
concept: ~anwendungszweck [Unterhaltung Schreibwaren EssenTrinken Essentrinken Alltag
```

Der Nutzer beschreibt beispielsweise in einem Satz, welchen Anwendungszweck das gesuchte Produkt erfüllen soll. Wird ein Wort aus dem Konzept gefunden, so wird dieser als solches Erkannt, sofern vom Skript vorgesehen. Damit wird sichergestellt, dass in den Queries nur Eigenschaften vorkommen, die in Datenbankeinträgen existieren. Sollte der Anwender nach Eigenschaften suchen, die nicht existieren, so werden diese ignoriert (Siehe: Pattern-Matching, Query).

Des Weiteren erlaubt es ChatScript, mehrere Konzepte in einem Konzept zu vereinen. Ein solches Konzept ist als Vereinigung mehrerer Konzepte zu verstehen:

Listing 2.7: Konzept 'positivkaufen' aus konzepte.top

```
concept: ~positivkaufen [ ~yes ~zustimmung ~kaufen]
```

'Positivkaufen' enthält somit alle Wörter, die in yes, zustimmung und kaufen vorkommen.

2.4 ChatScript: Bedingungen zur Steuerung von Programmablauf

2.5 ChatScript: Befehle zur Steuerung von Programmablauf

2.6 ChatScript: Pattern-Matching

2.7 ChatScript: Zufällige Ausgabe

3 Augusta: Programmlogik

Hier, stelle in z.B. einem Automaten dar, wie der Programmablauf normalerweise ablaufen soll

3.1 Befehle und Programmablauf

3.2 Variablen und Programmablauf

3.3 Konzepte und Programmablauf

3.4 Programmablauf im Detail

4 Kapitel1j

4.1 UnterKapitel2j

4.2 UnterKapitel3j

5 Fazit

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

WILCOX, BRUCE (2019) *ChatScript*. https://github.com/ChatScript/ChatScript.