Karlsruhe Institute of Technology

Erkennung von Aktionen in gesprochener Sprache

Bachelorarbeit
von

Yue Ou

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf Reussner
Betreuender Mitarbeiter:	Dipl.-Inform. Sebastian Weigelt

Bearbeitungszeit: 01.06.2016 – 30.09.2016

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 30.09.2016

. . . . . .

(**Yue Ou**)

Kurzfassung

In der natürlichen Sprache sind Aktionen enthalten. Aktionen sind ein wesentlicher Bestandteil der Semantik. Im Rahmen des Projekts PARSE wird in dieser Bachelorarbeit einen Agent entworfen und implementiert, der Aktionen erkennt. Eine Aktion besteht aus Akteur, Prädikat und Parameter. Der Agent soll die initiale Handlungsrepräsentation entgegennehmen, die darin enthaltenen Aktionen extrahieren und diese in einem Graph darstellen.

Diese Bachelorarbeit hat während dem Entwurf zwei Hauptfragen gestellt und beantwortet: Wie sollen Aktionen dargestellt werden und wie sollen Aktionen erkannt werden, damit sie auf die Darstellung abgebildet werden können?

Token werden in Knoten dargestellt und Rolle zugeordnet. Eine Rolle kann die Werte *Akteur*, *Prädikat*, *Was*, *Wer*, *Wo*, *Wann*, *Wie*, *Warum* annehmen. Die Relationen zwischen einzelnen Aktionen, Akteure, Prädikate, Parameter und Token innerhalb einer Phrase werden mittels Kanten dargestellt. Die Rollenerkennung erfolgt über semantische Rollen, Eigennamen, Phrasen und Wortarten.

Für die Transkriptionen, die im Rahmen der Arbeit von Günes[Gün15] entstanden sind, wird eine Genauigkeit bei Rollenidentifizierung von 78.25%, eine Präzision bei Kanten von 73.17% und eine Ausbeute bei Kanten von 77.87% erreicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Es werden immer mehr Maschinen in unser Leben eingesetzt. Sie können z.B. im Haushalt, in der Industrie und in der Bildung Aufgaben für Menschen erledigen. Die einfachste Weise, eine Maschine zu steuern, ist durch gesprochene Sprache. Es ist daher wichtig, dass die Maschinen die Sätze verstehen, die Aktionen darin erkennen, damit sie diese ausführen können. Da die Maschinen nur künstliche Intelligenz haben und nur von Programmen gesteuert werden können, ist es unsere Aufgabe, eine semantische Darstellung für Aktionen zu bauen und die natürliche Sprache darauf abzubilden.

1.1. Aktionen in gesprochener Sprache

Eine Aktion besteht aus einer Tätigkeit, die von einem Akteur ausgeübt wird. Dabei können zusätzlichen Informationen enthalten sein, zum Beispiel das Objekt, der Zeitpunkt und der Ort. Damit die Maschinen die Aktionen verstehen, ist es sinnvoll, diese Informationen aus unterschiedlich konstruierten Sätzen zu extrahieren und einheitlich darzustellen.

Es gibt vier Konstruktionen für Sätze in Englisch: deklarative Sätze, imperative Sätze, Ja/Nein-Fragen und W-Fragen[Jur09]. Diese Arbeit wird sich mit Aktionen beschäftigen und daher nur die imperative Sätze und deklarative Sätze betrachten.

Es handelt sich bei einem imperativen Satz um einen Befehl. Ein Beispiel dafür ist:

Beispiel: Ein Befehl

„Get me an orange juice.“

Um diesen Satz semantisch zu erfassen ist es wichtig zu erkennen, dass es um die Tätigkeit „get“ geht und dass „me“ und „orange juice“ zusätzliche Informationen sind. Ein Akteur ist hier nicht explizit angegeben.

In einem deklarativen Satz ist meistens eine Beschreibung enthalten. Allerdings kann die Beschreibung auch eine Aktion enthalten. Folgende Beispiele enthalten die gleiche Aktion:

Beispiel: Beschreibungen

„Armar brought the popcorn to me.“
„The popcorn was brought to me by Armar.“

In beiden Sätzen handelt es sich um die gleiche Aktion: „bring“, den gleichen Akteur: „Armar“ und das gleiche Objekt: „me“. Der erste Satz ist im Aktiv formuliert, der zweite hingegen im Passiv. Trotz der unterschiedlichen syntaktischen Strukturen, soll für solche Sätzen eine einheitliche semantische Darstellung konstruiert werden, da die Semantik gleich ist.

1.2. Zielsetzung der Bachelorarbeit

Das Ziel dieser Bachelorarbeit ist, einen Agent für das Projekt PARSE (vorgestellt im Kapitel 3) zu entwickeln. Der Agent soll die von SNLP (vorgestellt im Abschnitt 3.4) erzeugte initiale Handlungsrepräsentation entgegennehmen und einzelne Aktionen daraus extrahieren. Eine Aktion enthält die folgenden Informationen: der Akteur, das Prädikat und die Parameter. Der Akteur führt die Aktion aus. Das Prädikat bestimmt die Tätigkeit und besteht immer aus Verben. Die Parameter sind die Objekte des Satzes und beantworten die Fragen „was“, „wen/ wem“, „wo“, „wann“, „wie“ und „warum“ (die W-Fragen [Jur09]). Der Agent modifiziert die initiale Handlungsrepräsentation um diese Informationen wieder im Graph darzustellen. Nach der Bearbeitung soll anhand des Graphs erkennbar sein, welche Aktionen in der Eingabe enthalten sind.

1.3. Übersicht und Struktur der Bachelorarbeit

In dieser Ausarbeitung werden zuerst die Grundlagen (Kapitel 2) vorgestellt, die für die Erkennung und die Darstellung der Aktionen in gesprochener Sprache wichtig sind. Dann wird das Projekt PARSE (Kapitel 3) vorgestellt. Anschließend werden die verwandten Arbeiten (Kapitel 4) im Bereich Sprachverstehen vorgestellt und diskutiert. Danach wird eine Analyse (Kapitel 5) durchgeführt und ein Entwurf einer konkreten Lösung aufgestellt. Auf dieser Basis wird das Werkzeug implementiert (Kapitel 6), und mit einer Evaluation (Kapitel 7) bewertet.

2. Grundlagen

In diesem Kapitel werden die Grundlagen, die für die Erkennung und die Darstellung von Aktionen in gesprochener Sprache von Bedeutung sind, präsentiert.

Zuerst wird der Begriff des Graphen (Abschnitt 2.1) eingeführt. Graphen werden benutzt, um natürliche Sätze formal darzustellen. Dann werden zwei Teilgebiete der Linguistik, Syntax und Semantik (Abschnitt 2.2), vorgestellt. Danach wird auf die Informationen eingegangen, die dabei helfen, Wörter semantisch zu erfassen. Darunter sind Wortart (Abschnitt 2.4), Phrase (Abschnitt 2.5), Eigenname (Abschnitt 2.6) und semantische Rolle (Abschnitt 2.7). Dabei werden zwei Formate der Phrasenmarkierung - IOB-Format (Abschnitt 2.5.1) und IOBES-Format (Abschnitt 2.5.2) vorgestellt. Am Ende werden Korpora vorgestellt, die interessant sind.

2.1. Graph

Ein **Graph** ist eine Struktur in der Graphentheorie. Ein Graph besteht aus einer endlichen, nicht leeren Menge von **Knoten** (engl. vertex) und einer endlichen Menge von **Kanten** (engl. edge). Knoten sind die Objekte im Graphen. Kanten sind die Verbindungen zwischen den Knoten. Ein Graph kann gerichtet oder ungerichtet sein. Ein gerichteter Graph besteht aus gerichteten Kanten. Eine gerichtete Kante ist ein geordnetes Paar aus zwei Knoten. Ein ungerichteter Graph besteht aus ungerichteten Kanten. Eine ungerichtete Kante ist eine zweielementige Menge von Knoten.

In dem Beispiel in Abbildung 2.1 ist ein gerichteter Graph dargestellt. Dieser Graph hat eine Knotenmenge

$$V = \{0, 1, 2, 3, 4, 5\}$$

und eine Kantenmenge

$$E = \{(0, 1), (0, 3), (1, 2), (1, 3), (4, 5), (5, 4)\}.$$

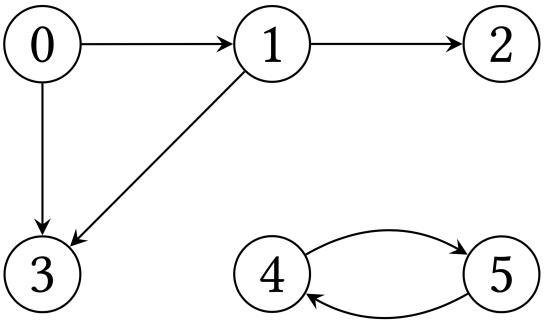

Abbildung 2.1.: Ein gerichteter Graph

In dem Beispiel in Abbildung 2.2 ist ein ungerichteter Graph dargestellt. Dieser Graph hat eine Knotenmenge

$$V = \{0, 1, 2, 3, 4, 5\}$$

und eine Kantenmenge

$$E = \{\{0, 1\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{4, 5\}\}$$

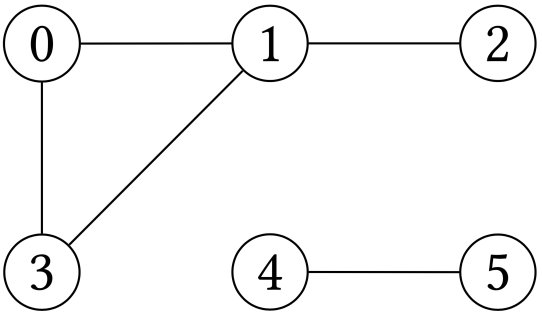

Abbildung 2.2.: Ein ungerichteter Graph

2.2. Syntax und Semantik

Das Wort **Syntax** stammt von dem griechischen Wort „syntaxis“, das „Koordination und Ordnung“ bedeutet. Syntax beschreibt die Regeln, Wörter in Sätze anzuordnen, ohne auf die Bedeutungen einzugehen. Syntax ist ein Teilgebiet der Grammatik. Syntax bezieht sich auf formale und natürliche Sprache.

Die **Semantik** beschreibt die Bedeutungen von Wörtern, Phrasen und Sätzen. Sie beschäftigt sich auch mit den Beziehungen zwischen Wörtern und Phrasen. Semantik bezieht sich wie Syntax auf formale und natürliche Sprache.

Diese zwei Begriffe werden durch ein Beispiel verständlicher:

Beispiel: ein syntaktisch richtiger aber semantisch falscher Satz
Colorless green ideas sleep furiously [Cho02]

Dieser Satz ist syntaktisch richtig, aber semantisch falsch, weil er grammatisch gesehen richtig ist, aber keine sinnvolle Bedeutung beinhaltet.

2.3. Token

Tokenisierung ist ein Vorgang, um Texte in Wörter zu segmentieren. Die Wörter werden auch als **Tokens** bezeichnet. Die Tokenisierung eines Eingabetextes ist die Voraussetzung der weiteren Verarbeitung. Das einfachste Tokenisierungsverfahren ist die Leerzeichen-Tokenisierung. Der Eingabetext wird an den Leerzeichen und Satzzeichen aufgetrennt.

2.4. Wortart

Wortarten (engl. Part of Speech - POS) sind Kategorien von Wörtern, die ähnliche grammatische Eigenschaften haben [Jur09]. Wörtern von der gleichen Wortart spielen meistens eine ähnliche syntaktische Rolle in Sätzen. Dionysius Thrax hat circa im Jahr 100 v. Chr das linguistische Wissen von Griechisch zusammengefasst und acht Wortarten vorgeschlagen: Nomen, Verb, Pronomen, Präposition, Adverb, Konjunktion, Partizip und Artikel. Diese Arbeit ist ein Ausgangspunkt von vielen heutigen Arbeiten im Bereich Linguistik. Die heutigen Listen von Wortarten sind im Vergleich länger.

Wortarterkennnug (engl. Part-of-Speech Tagging - POS Tagging) ist ein Prozess, der jedem Wort eine Wortart zuordnet. Systeme, die Wortarterkennung leisten, werden **Wortarterkenner** (engl. Part-of-Speech Tagger - POS Tagger) genannt. Jeder Wortarterkenner hat ein **Tagset**, nämlich die Auswahl von Wortarten. Die Penn Treebank[MKM+94] hat ein Tagset von 45 Wortarten, der Brown Corpus[Kje94] hat ein Tagset von 87 Wortarten und das C7 Tagset[RWL01] hat ein Tagset von 146 Wortarten.

In dieser Bachelorarbeit wird der Penn-Treebank-Standard benutzt. Im Anhang Abschnitt A.1 befindet sich das vollständige Part-of-Speech-Tagset von Penn Treebank.

Im Folgenden wird ein Beispiel zur Wortarterkennug gezeigt.

Beispiel: Wortarterkennug

Get/VB the/DT orange/NN juice/NN ./.

In dem Satz wird jedem Wort eine Wortart zugeordnet. Die zugehörige Wortart wird an dem Wort gehängt und sie werden durch ein Schrägstrich getrennt. Hier wird ausgezeichnet, dass „get“ ein Verb, „the“ ein Artikel, „orange“ ein Nomen und „ juice“ ebenfalls ein Nomen ist.

Ein Ausschnitt des Part-of-Speech-Tagsets der Penn Treebank, der für das obige Beispiel relevant ist, ist in der Tabelle Abschnitt 2.4 zu sehen.

Tabelle 2.1.: Penn Treebank Part-of-Speech Tagset Abschnitt

Number	Tag	Description	Examples
3	DT	Determiner	all an another any both
12	NN	Noun, singular or mass	common-carrier cabbage knuckle-duster
27	VB	Verb, base form	ask assemble assess assign
44	.	sentence terminator	. ! ?

2.5. Phrase

Phrasen[Jur09] (engl. Chunks) sind syntaktische Einheiten in der Linguistik und haben unterschiedliche grammatische Bedeutungen. Es gibt hauptsächlich Nominalphrase, Verbalphrasen, Adjektivphrase und Präpositionalphrase. Verschiedene Phrasen in einem Satz

überlappen sich nicht. In einem Satz können die Phrasen verschoben, aber nicht aufgetrennt werden. Der Unterschied zwischen Phrase und Wortart ist: Wortart wird einem einzelnen Wort zugewiesen gemäß seiner Rolle im Satz. Phrase wird einer Gruppe von Wörtern zugewiesen gemäß ihrer gemeinsamer Rolle im Satz.

Der Vorgang, Phrasen in Sätzen zu identifizieren, wird **Phrasenerkennung** (engl. Chunking) genannt. Wörten werden in Phrasen eingeteilt und Phrasen werden klassifiziert. In der Phrasenerkennung wird vorausgesetzt, dass der Text bereits tokenisiert wird und dass die Wortarterkennug bereits durchgeführt wird.

Für Phrasen wird auch der Penn-Treebank-Standard benutzen. Im Anhang Abschnitt A.2 befindet sich das vollständige Chunk-Tagset der Penn-Treebank.

Da Sätze in nicht überlappende Phrasen segmentiert werden, ist eine Klammernotation ausreichend, um Phrasen anzugeben. Das folgende Beispiel illustriert eine typische Klammernotation.

Beispiel: Phrasenerkennung [Jur09]

[NP The morning flight] [PP from] [NP Denver] [VP has arrived.]

Ein Ausschnitt des Chunk-Tagsets der Penn-Treebank, der für das obige Beispiel relevant ist, ist in der Tabelle Abschnitt 2.5 zu sehen. In der ersten Spalte steht das jeweilige Kennzeichen. Die Beschreibung ist in der zweiten Spalte. Die dritte Spalte gibt die Wortarten von Wörtern an, aus den die Phrase bestehen kann. Mögliche Beispielsphrasen sind in der letzten Spalte zu sehen.

Tabelle 2.2.: Penn Treebank Chunk Tagset Abschnitt

Tag	Description	Words	Example
NP	noun phrase	DT+RB+JJ+NN + PR	the strange bird
PP	prepositional phrase	TO+IN	in between
VP	verb phrase	RB+MD+VB	was looking

In jeder eckigen Klammer ist eine Phrase zu sehen und jeder Phrase ist eine Kategorie zugeordnet. Zum Beispiel ist „the morning flight“ eine Norminalphrase und „from“ eine Präpositionalphrase. Hierbei soll beachtet werden, dass jedes Wort hier zu einer Phrase gehört. Dies ist aber nicht immer der Fall. In manchen Werkzeug, die Phrasen erkennen, wird nur nach bestimmten Phrasen gesucht. In dem folgenden Beispiel wird nur nach Norminalphrasen gesucht.

Beispiel: Phrasenerkennung - Norminalphrasen [Jur09]

[NP The morning flight] from [NP Denver] has arrived.

Sowie dass jedem Wort in der Wortarterkennung eine Wortart zugeordnet wird, kann jedem Wort in der Phrasenerkennung ein Phrasekennzeichen zugeordnet werden. Die Phrasenmarkierung beantwortet folgende Fragen: Gehört das Wort zu einer Phrase? Zu welcher Phrase gehört es wenn es zu einer Phrase gehört? Wo beginnt und wo endet die Phrase? Somit kann auf die Klammernotation verzichtet werden.

Im Folgenden werden zwei Formate für die Phrasenmarkierung vorgestellt - IOB und IOBES.

2.5.1. IOB

Markierung im **IOB-Format** (engl. Inside-Outside-Beginning-Format) können den Beginn und das Innere der Phrasen darstellen. Die Elemente, die zu keiner Phrase gehören, können auch erkannt werden. Unter diesem Schema ist die Größe des IOB-Tagset $(2n + 1)$, wobei n die Anzahl der Kategorien im Phrase-Tagset ist.

Der Satz aus dem obigen Beispiel in Klammernotation wird jetzt im Abschnitt 2.5.1 IOB-Format bezeichnet.

Beispiel: Phrasenerkennung im IOB-Format [Jur09]

The	morning	flight	from	Devner	has	arrived
B_NP	I_NP	I_NP	B_PP	B_NP	B_VP	I_VP

„B“ kennzeichnet den Beginn einer Phrase und „I“ kennzeichnet das Innere einer Phrase. „O“ kennzeichnet Tokens, die zu keiner Phrase gehören. Das Ende einer Phrase ist implizit gegeben durch alle Übergänge von einem „I“ oder „B“ zu einem „B“ oder „O“.

Mit dem IOB-Format können auch Sätze ausgezeichnet werden, in denen nur nach bestimmten Phrasen gesucht wird. Im Folgenden ist der gleiche Satz mit Norminalphrasenkennzeichnung dargestellt.

Beispiel: Phrasenerkennung im IOB-Format - Norminalphrasen [Jur09]

The	morning	flight	from	Devner	has	arrived
B_NP	I_NP	I_NP	O	B_NP	O	O

2.5.2. IOBES

Das **IOBES-Format** (engl. Inside-Outside-Begin-End-Single-Format) unterscheidet sich vom IOB-Format darin, dass das Ende der Phrasen und Phrasen bestehend aus einem einzelnen Token zusätzlich gekennzeichnet werden. Die Größe des IOBES-Tagset ist $(4n + 1)$, wobei n die Anzahl der Kategorien im Phrase-Tagset ist.

Der Beispielssatz vom obigen Abschnitt wird im Folgenden mit IOBES-Format bezeichnet.

Beispiel: Phrasenerkennung im IOBES-Format [Jur09]

The	morning	flight	from	Devner	has	arrived
B_NP	I_NP	E_NP	S_PP	S_NP	B_VP	E_VP

Zwei zusätzliche Markierung werden benutzt: „E“ und „S“. „E“ kennzeichnet das Ende einer Phrase und „S“ kennzeichnet die Phrasen, die nur aus einem einzelnen Token bestehen.

Für Sätze, in denen nur nach bestimmten Phrasen gesucht wird, sieht die Bezeichnung folgendermaßen aus:

Beispiel: Phrasenerkennung im IOBES-Format - Norminalphrasen [Jur09]

The	morning	flight	from	Devner	has	arrived
B_NP	I_NP	E_NP	O	S_NP	O	O

2.6. Eigenname

Ein **Eigenname** (engl. Named Entity) ist ein Objekt in der realen Welt, das mit einem geeigneten Namen genannt werden kann [Jur09]. Beispiele dafür sind Personen, Orte, Organisationen, Produkte und so weiter.

Der Prozess, Eigennamen in Sätzen zu erkennen, wird **Eigennamenerkennung** (engl. Named-Entity Recognition - NER) genannt. Eigennamenerkennung ist der Startpunkt von vielen Anwendungen im Bereich Informationsextraktion. Eigennamenerkennung beinhaltet zwei Aufgaben: Zuerst müssen die Wörter, die Eigennamen bilden, festgelegt werden. Dann müssen die Eigennamen konkrete Typen (Person, Ort, ...) zugeordnet werden.

Im Anhang Abschnitt A.3 befindet sich eine Liste von Eigennamen mit den zugehörigen Kategorien und Beispielen.

Im Folgenden ist ein Beispiel zur Eigennamenerkennung.

Beispiel: Eigennamenerkennung [Jur09]

[ORG UAL Corp.], said the increase took effect Thursday and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

Hierbei wird die gleiche Klammernotation wie bei Phrasenerkennung verwendet. „UAL Corp.“ wird als Organisation, „Chicago“, „Dallas“, „Denver“, und „San Francisco“ werden alle als Ort gekennzeichnet.

Ein Ausschnitt der Liste von Eigennamen, der für das obige Beispiel relevant ist, ist in der Tabelle Abschnitt 2.6 zu sehen.

Tabelle 2.3.: Eigennamenliste Abschnitt

Type	Tag	Sample Categories	Example
Organization	ORG	Companies, agencies, political parties	IPCC
Location	LOC	Physical extents, mountains,lakes	Mt. Sanitas

2.7. Semantische Rolle

Eine **semantische Rolle** (engl. semantic role) bezeichnet die semantische Beziehung zwischen einer Phrase und dem Prädikat in einem Satz [PGX10]. Die Phrasen dienen als Argumente des Prädikats. Typische semantische Argumente sind Agent, Patient, Instrument, Zeit, Ort und Anlass.

Die **Semantische Rollenzuteilung** (engl. Semantic Role Labeling) ist eine wichtige Aufgabe in natürlicher Sprachverarbeitung. Die Zuweisung von semantischen Rollen dient dazu, Semantik von Sätzen zu interpretieren.

Um Semantic Role Labeling zu illustrieren, wird das folgende Beispiel aus dem PropBank-Korpus (siehe Abschnitt 2.8.2) verwendet:

Beispiel: Semantische Rollenzuteilung

[A0 He] [AM-MOD would] [AM-NEG n't] [V **accept**] [A1 anything of value] from [A2 those he was writing about].[1]

Die Rollen von dem Prädikat „accept“ ist in PropBank (sehe Abschnitt 2.8.2) so definiert:

1. V: verb
2. A0: acceptor
3. A1: thing accepted
4. A2: accepted-form
5. A3: attribute
6. AM-MOD: modal
7. AM-NEG: negation

Die Annotation für die semantischen Rollen des Satzes ist:

2.8. Korpora

Im Folgenden werden Korpora vorgestellt, die interessant sind.

2.8.1. Penn Treebank

Das Penn Treebank Projekt [MKM+94] annotiert natürliche Texte mit linguistische Strukturen und erzeugt eine Bank von Syntaxbäume, die grobe syntaktische und semantische Informationen enthalten. Treebank annotiert Texte mit Wortart (engl. Part-Of-Speech Tag - POS Tag).

2.8.2. PropBank

PropBank (engl. proposition bank) ist ein Sprachkorpus [KGP05]. PropBank fügt der syntaktischen Strukturen von Penn Treebank (2.8.1) semantische Rollen (semantic role labels) hinzu.

Es ist schwer, für alle Prädikate einen universellen Satz von semantische Rollen zu definieren, weil Verben unterschiedliche Parameter benötigen. Deswegen definiert PropBank einen Satz von semantische Parameter für jedes Verb. Die semantischen Parameter von jedem Verb werden von 0 an durchnummeriert. Arg0 ist meistens der Agent. Arg1 ist meistens der Patient. Für die höhere nummerierte Argumente können keine generellen Aussagen getroffen werden. PropBank definiert allerdings einen Satz von generelleren Hilfsargumenten. Dieses gilt für allen Verben. Dieser Satz ist in der Tabelle 2.4 zu sehen.

Tabelle 2.4.: Subtypes of the ArgM modifier tag

LOC	location
EXT	extent
DIS	discourse connectives
ADV	general-purpose
NEG	negation marker
MOD	modal verb
CAU	cause
TMP	time
PNC	purpose
MNR	manner
DIR	direction

[1] http://www.cs.upc.edu/ srlconll/

2.8.3. SENNA

SENNA [CWB+11] ist ein Werkzeug, das Wortarten, Phrasen, Eigennamen, und semantische Rollen erkennen kann. SENNA ist von keinen anderen natürlichen Sprachbearbeitungssystemen abhängig und analysiert die Eingabe eingenständig. SENNA wurde in C geschrieben und hat eine einfache Architektur. Für die Eingabe ist nur Text nötig. Nimmt man

„Ben has killed Han with a light saber and Chewie cried."

als Eingabetext, dann sieht die Ausgabe von SENNA folgendermaßen aus:

1	2	3	4	5	6	7	8
Ben	NNP	S-NP	S-PER	-	S-A0	O	(S1(S(S(NP*)
has	VBZ	B-VP	O	-	O	O	(VP*
killed	VBN	E-VP	O	killed	S-V	O	(VP*
Han	NNP	S-NP	S-PER	-	S-A1	O	(NP*)
with	IN	S-PP	O	-	B-AM-MNR	O	(PP*
a	DT	B-NP	O	-	I-AM-MNR	O	(NP*
light	JJ	I-NP	O	-	I-AM-MNR	O	*
saber	NN	I-NP	O	-	E-AM-MNR	O	*)))))
and	CC	I-NP	O	-	O	O	*
Chewie	NNP	E-NP	S-PER	-	O	S-A0	(S(NP*)
cried	VBD	S-VP	O	cried	O	S-V	(VP*))
.	.	O	O	-	O	O	*))

Jede Zeile repräsentiert ein Token. In den Spalten sind verschiedene Informationen zu dem Token. Alle Informationen werden in IOBES-Format (siehe Abschnitt 2.5.2) dargestellt. Die Nummerierung dienen nur zur Vereinfachung der Erklärung.

1. In der ersten Spalte steht das tokenisierte Wort.
2. In der zweiten Spalte wird jedes Token mit der zugehörigen Wortart (siehe Abschnitt 2.4) gekennzeichnet. „Ben" hat z.B. die Wortart „Substantiv-Eigenname" (NNP) und „killed" hat z.B. die Wortart „Partizip-Verb" (VBN).
3. In der dritten Spalte wird die Phrase (vorgestellt in Abschnitt 2.5) markiert. „Han" gehört z.B. zu einer „Norminalphrase" (NP) Phrase und diese Phrase besteht nur aus dem Token „Han".
4. In der vierten Spalte ist die Eigennamenerkennung (siehe Abschnitt 2.6) dargestellt. „Ben", „Han" und „Chewie" sind die erkannten Eigennamen, die als Personen klassifiziert wurden
5. Die fünfte Spalte gibt an, ob die semantische Rolle (siehe Abschnitt 2.7) des Tokens von SENNA als ein Verb identifiziert wird. „Killed" und „cried" werden als Verben erkannt.
6. Es gibt für jedes von SENNA erkanntes Verb eine zusätzliche Spalte, die semantische Rollen angibt. In diesem Beispiel steht in der sechsten Spalte die semantischen Rollen aller Token bezüglich des Verbs „killed", z.B. ist „Ben" der Akteur und „with a light saber" ein Argument des Verbs. In der siebten Zeile sind die semantischen Rollen bezüglich des Verbs „cried". Alle Tokens bevor „and" haben keine Relation zu „cried" und werden daher mit „O" gekennzeichnet.
7. In der letzten Zeile wird der Syntaxbaum in einer Klammerdarstellung dargestellt.

3. PARSE

In diesem Kapitel wird das Projekt PARSE[WT15] vorgestellt. Das Projekt PARSE wurde am Karlsruhe Institut für Technologie entwickelt. PARSE ist ein Werkzeug, das gesprochene natürliche Sprache analysieren und bearbeiten kann und Programmieren in natürlichen Sprachen ermöglicht.

3.1. Ziel von PARSE

Das Ziel von PARSE ist, Programmieren mit gesprochenen natürlichen Sprachen in verschiedenen Bereichen zu verwirklichen. Ein Zielsystem ist ARMARIII[ARA$^+$06]. Hier sollen einem Haushaltsroboter neue Anweisungen mit gesprochener Sprache beigebracht werden. Ein anderes Zielsystem ist Alice[Coo00]. Dabei sollen Schüler Programmierung in 3D-Umgebung beigebracht werden.

3.2. Konstruktionsprinzipien

PARSE verfolgt folgenden Prinzipien:

- Agentbasiert: PARSE basiert auf Agenten, die unabhängig voneinander arbeiten. Jeder Agent integriert eine oder mehrere Funktionalitäten in das ganze Projekt. Der Vorteil daran ist die Einfachkeit der Integration neuer Funktionalitäten. Außerdem können Agenten separat voneinander evaluiert werden.
- Wissenbasiert: PARSE benutzt externe Datenbanken um ein tieferes Sprachverständnis zu erreichen. Es ermöglicht Disambiguierung und Eigennamenerkennug (siehe Abschnitt 2.6). Zwei Datenbanken, die PARSE benutzt, sind Cyc[Cyc08] und WordNet[Miler].
- Evaluationgetrieben: PARSE evaluiert das System durchgehend um Qualität und Fortschritte zu gewährleisten. In der Evaluation werden realistische Beispiele benutzt, zum Beispiel die Befehle an Haushaltsroboter.
- Domänenunabhängiges Sprachverständnis: In PARSE ist das Sprachverständnis domänenunabhängig. Auf Domänenwissen kann können nur über eine definierte Schnittstelle zugegriffen werden. Dadurch ist es einfach, neue Domänen hinzuzufügen und Veränderungen in einer Domäne sind für den Rest des Systems transparent.

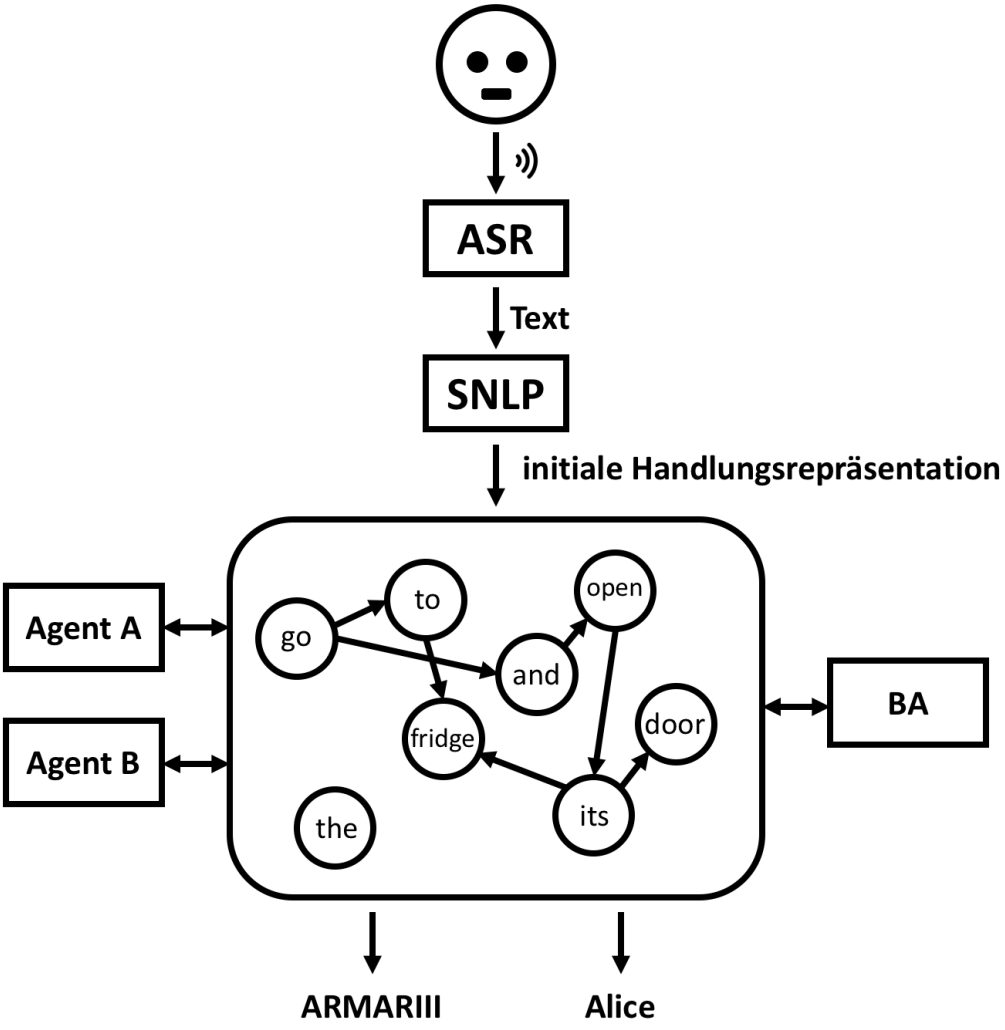

Abbildung 3.1.: Die Architektur von PARSE

3.3. Architektur von PARSE

In der Abbildung 3.1 ist die Architektur von PARSE zu sehen. Ein automatischer Spracherkenner (engl. Automatic Speech Recognizer - ASR) transformiert das akustische Signal in Text. Die seichte Sprachverarbeitung (engl. Shallow Natural Language Processing - SNLP) erzeugt aus dem Text einen gerichteten Graph (Abschnitt 2.1), der als initiale Handlungsrepräsentation dient. Der erzeugte Graph wird auch als die interne Darstellung der Daten verwendet. PARSE basiert auf mehrere Agenten, die verschiedene Aufgaben erfüllen. Die Agenten extrahieren Informationen aus dem Graph und modifizieren den Graph, um die Ergebnisse wieder darzustellen. Die aufgearbeitete Repräsentation kann dann auf verschieden Zielsysteme abgebildet werden.

3.4. Seichte Sprachverarbeitung (SNLP)

Die seichte Sprachverarbeitung (engl. Shallow Natural Language Processing - SNLP)[Koc15] ist ein Werkzeug, das Texte auf initiale Handlungsrepräsentationen abbildet. Das Ziel von

SNLP ist es, einen Graph bereitzustellen, der als initiale Handlungsrepräsentation dient und von den Agenten benutzt werden kann. Die Eingabe ist ein Text, der mittels eines Spracherkenners aus einem akustischen Signal erzeugt wurde. SNLP weist jedem Wort in der Eingabe folgende Informationen zu: die Wortart (engl. POS Tag), die phrasale Markierung (engl. Chunking) und die Befehlsnummer. Diese Informationen wurden bereits in Kapitel 2 detailliert vorgestellt.

Nachstehend ist eine beispielhafte Eingabe und die zugehörige Ausgabe von SNLP dargestellt.

Beispiel:

„Armar get me orange juice and popcorn.“

Ausgabe:

[Armar(NNP/0/B-NP/NP/0/0), get(VB/0/B-VP/VP/0/0),
me(PRP/0/B-NP/NP/0/0), orange(JJ/0/B-NP/NP/0/3),
juice(NN/0/I-NP/NP/1/2), and(CC/1/I-NP/NP/2/1),
popcorn.(NN/1/I-NP/NP/3/0)]

In der Ausgabe steht hinter jedem Wort eine Liste von sechs Attributen: die Wortart (siehe Abschnitt 2.4), die Befehlsnummer, die Phrase mit IOB-Format (siehe Abschnitt 2.5), die Phrase ohne IOB-Format, die Anzahl von Wörtern vor dem Wort, die zu derselben Phrase gehören und die Anzahl von Wörtern nach dem Wort, die zu derselben Phrase gehören.

3.5. Einordnung der Bachelorarbeit

In dieser Bachelorarbeit soll einen Agent gebaut werden, der die von SNLP erzeugte initiale Handlungsrepräsentation entgegennimmt und einzelne Aktionen daraus extrahiert. Der Agent soll die initiale Handlungsrepräsentation modifizieren und die gewonnen Informationen wieder im Graph darstellen. Nach der Bearbeitung soll anhand vom Graph erkennbar sein, welche Aktionen in der Eingabe enthalten sind.

4. Verwandte Arbeiten

Aktionen sind ein wesentlicher Bestandteil der Semantik in der natürlichen Sprache. Viele Arbeiten im Bereich natürliches Sprachverstehen (engl. Natural language understanding - NLU) haben sich bereits damit beschäftigt, die Aktionen zu erkennen. In diesem Kapitel werden die verwandten Arbeiten vorgestellt, diskutiert und bewertet.

4.1. „NLC“ as a prototype

NLC [BB79] ist eine der ersten Systeme, das dem Benutzer ermöglicht, englische Befehle in Rechner einzugeben und sie ausführen lassen. Das System bearbeitet Dateien in Form von Matrizen und Tabellen. Es wird mit Zeilen, Spalten, Einträge und einfache Operationen zwischen diesen gearbeitet. Ein Beispielprogramm, das von NLC bearbeitet werden kann, ist:

„ Choose a row in the matrix.

Put the average of the first four entries in that row into its last entry.

Double its fifth entry and add that to the last entry of that row.

Devide its last entry by 3.

Repeat for the other rows.“

Nachdem die Befehle eingegeben werden, werden die Rechnungen erst auf eine Beispielreihe ausgeführt. Der Benutzer kann diese beobachten, und gegebenenfalls Fehler entdecken. Wenn die Rechnungen auf der Beispielreihe korrekt sind, werden sie auf die restlichen Reihen angewendet. Somit bietet das System eine größere Wahrscheinlichkeit, dass die Eingaben richtig interpretiert werden. Es gibt allerdings auch Einschränkungen, so wird eingeschränkt, dass alle Eingaben im Imperativ sind. Diese Einschränkung kann die Systemarchitektur für die Bearbeitung verbessern und vereinfachen. Aber die Ausdrucksfähigkeit vom Benutzer wird reduziert. „Add row 1 to row 3“ kann beispielsweise von dem System verstanden werden, wobei „Row 1 is to be added to row 3“, mit der gleichen Semantik, nicht erlaubt ist.

NLC hat vier Verarbeitungsstufen: morphologisch, syntaktisch, semantisch und rechnerisch.

Scanner Der Scanner extrahiert Token aus der Eingabe. „Add five to the 2nd positive entry in col 2.“ wird beispielsweise tokenisiert als „Add“, „five“, „te“, „the“, „2nd“, „positive“, „entry“, „in“, „col“, „2“, „.“ Der Scanner sucht auch nach möglichen Bedeutungen von der Token. „Add“ kann z.B. ein imperatives Verb sein und „five“ kann z.B. ein Integer sein.

Parser Der Parser bestimmt die Struktur der Befehlen. Er ordnet die Verben der Matrixdomäne in folgenden Kategorien:

1. Verben mit genau einem Operanden, z.B. „double“, „negate“
2. Verben mit zwei Operanden und einer Präposition, z.B. „add (to)“, „subtract (from)“
3. Verben mit einer Präposition und einem Operanden, z.B. „add up“, „round off“
4. Verben mit zwei Operanden, ohne Präposition, z.B. „call column 5 terry“
5. Verben mit keinen Operanden, z.B. „quit“, „back up“

Semantikprozessor Der Semantikprozessor ist dafür verantwortlich, die Matrixeinträge zu bestimmen, die der Benutzer bearbeiten möchte.

Matrixcomputer Der Matrixcomputer führt Rechnungen aus.

4.2. Metafor

Metafor [Liu05] ist eine intelligente Benutzerschnittstelle, die natürliche Sprache in Programmcode umwandelt. Der Benutzer beschreibt ein Programm mittels geschriebener natürlicher Sprache. Dann erzeugt Metafor ein entsprechendes Codefragment in Python. Dieses Quelltextfragment enthält die Programsturktur und dient zur Ideenfindung. Der Parser überdeckt nicht alle grammatikalisch korrekten Konstrukte, und erzeugt kein fertiges ausführbares Programm.

Metafor benutzt intern eine Verb-Subjekt-Objekt-Objekt-Struktur für die semantische Darstellung der natürlichen Sätze. Sie enthält die wichtigsten Informationen einer Aktion: den Akteur (Subjekt), das Prädikat (Verb) und die Parameter (Objekt).

Das natürliches Sprachverarbeitungswerkzeug MontyLingua [Liu04] wird von Metafor benutzt, um die einzelnen Eingaben zu bearbeiten. MontyLingua extrahiert Wortarten, Semantische Rolle, Phrasen und Lemmatisierung. MontyLingua nimmt ein Textdokument als Eingabe und gibt Verb-Subjekt-Objekt-Objekt-Struktur aus. MontyLingua integriert Weltwissen und kann daher auch Namen, Orte, Ereignisse und Zeiten extrahieren. Ein Beispiel ist:

Tiger Woods wrapped up the tournament at four under par.

==(MONTYLINGUA)==>

(Verb: „wrap up“,

Subj: „Tiger Woods“,

Obj1: „tournament“,

Obj2: „at four under par“)

4.3. Natural Language Processing for Natural Language Programming

Die Arbeit „Natural Language Processing for Natural Language Programming“ [MLL06] ist eine Weiterentwicklung von Metafor.

Im Vergleich zu Metafor wird in der Arbeit Fokus auf prozedurale Programmierung gelegt. Es werden Prozeduren aus Schritten und Schleifen konstruiert. Schritte und Schleife sind die schwierigsten Teile von prozeduraler Programmierung. Schritte, Schleife und Kommentare werden hier automatisch identifiziert und in ein Programmgerüst konvertiert.

Es gibt drei Hauptkomponente für die Sprachverarbeitung:

Der Schrittefinder Der Schrittfinder liest den Eingabetext und unterteilt ihn in Schritte, die in Programmtext umgewandelt werden können. Der Schrittfinder benutzt zuerst Brill's Tagger[Brier], um die Wortarten zu extrahieren. Dann sucht der Schrittfinder nach allen Verben, die zu Programmfunktionen umgewandelt werden können, z.B. „read“, „write“ und „count“. Danach werden die Grenzen der Schritten festgelegt. Ein neuer Schritt fängt an, wenn ein neuer Satz beginnt, oder wenn ein neues aktives Verb vorkommt (meistens in einem Nebensatz). Letztlich werden die Objekte jeder Aktion identifiziert.

Der Schleifefinder Der Schleiferfinder identifiziert die natürlichsprachlichen Strukturen, die auf Wiederholungen hinweisen. Die Schlüsselwörter von Wiederholungen sind z.B. „each X“, „every X“, „all X“. Die Nomen im Plural sind auch potentielle Indikatoren von Wiederholungen.

Der Kommentare-Identifikator Der Kommentare-Identifikator identifiziert beschreibende Statements identifiziert, die Kommentare sein können.

Zur Evaluation wurde ein Korpus von Programmieraufgaben aus dem Internet gesammelt. Es wurden auch Musterlösungen manuell erstellt und damit verglichen. Die Präzision der Identifikation der Schritte ist 86,0% und die Präzision der Identifikation der Schleife ist 80,6%.

4.4. Erkennung und semantische Assoziation von Entitäten in natürlichsprachlichen Texten

In der Arbeit „Erkennung und semantische Assoziation von Entitäten in natürlichsprachlichen Texten“ [Wei14] wird versucht, Entitäten in natürlichsprachlichen Texten zu identifizieren und mit Individuen von einer Ontologie zu assoziieren. Somit wird die natürliche Sprache mit einem wissenschaftlichen Modell assoziiert. Das Endziel ist dabei, Programmierung in natürlicher Sprache zu ermöglichen.

Um natürlichsprachliche Sätze darzustellen, werden Atomarer-Satz-Objekte benutzt. Die Atomarer-Satz-Objekte bestehen aus Subjekt, Prädikat und Objekte, die jeweils als Wortlisten angegeben sind. Das Klassendiagramm ist in der Abbildung 4.1 dargestellt.

Jeder atomare Satz ist entweder eine Aktion (engl. action) oder eine Beschreibung (engl. description). Dieser Typ muss festgelegt werden, damit die semantische Informationen verarbeitet werden können.

Ein Beispiel für das Atomarer-Satz-Objekt ist: „The frog and the bunny turn to face Alice.“ In dem Satz sind zwei Teilsätze, nämlich „The frog turn to face Alice.“ und „The bunny turn to face Alice.“ Sie sind beide vom Typ Aktion. Zwei Atomarer-Satz-Objekte werden somit instanziiert, mit dem gleichen Prädikat und Objekt aber verschiedenen Subjekte. Die Darstellung der Atomarer-Satz-Objekte steht in der Abbildung4.2.

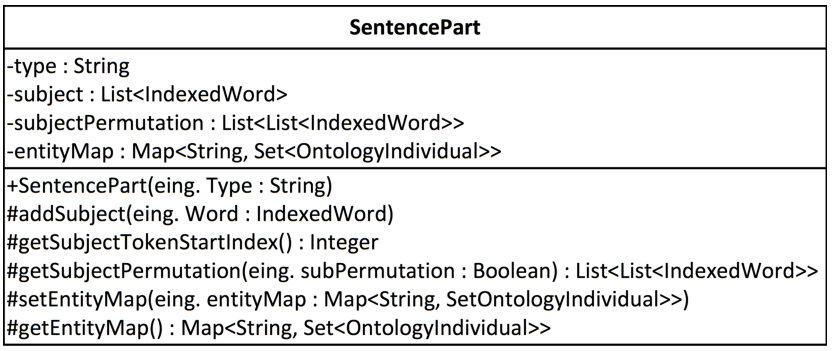

Abbildung 4.1.: Atomarer-Satz-Objekt

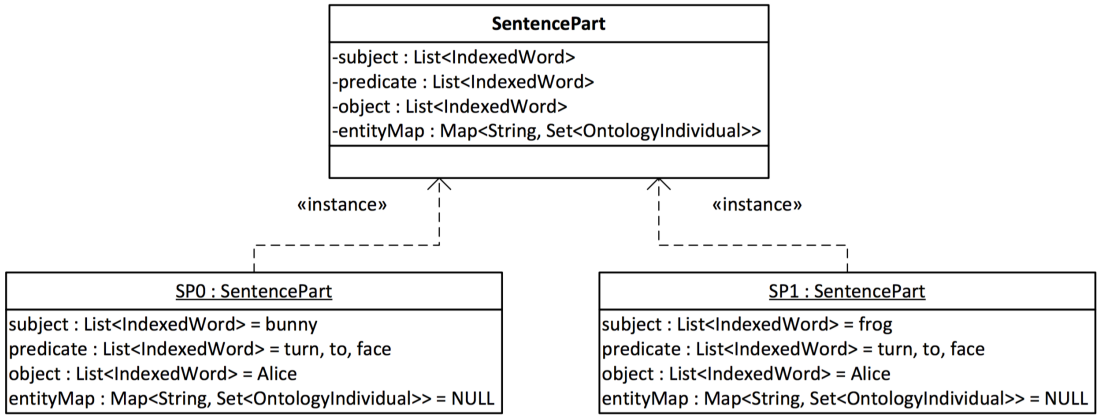

Abbildung 4.2.: „The frog and bunny turn to face Alice.“

4.5. FrameNet

Das Berkeley FrameNet Projekt [BFL98] hat als erstes ein Lexikon produziert, das systematisch Prädikaten und zugehörenden Rollen beschrieben hat. Das Lexikon kann von sowohl von Menschen, als auch von Maschinen verstanden werden und beinhaltet mehr als 10,000 Wörter und Phrasen, die mit Bedeutungen und Anwendung annotiert sind. Die Hauptmerkmale des Projektes sind: der Einsatz in semantische und syntaktische Generalisierung und die Repräsentation der Valenzen von den Zielwörten. Das Ergebnis von FrameNet ist die FrameNet Datenbank und die relevanten Softwarewerkzeuge. Die Datenbank enthält die Beschreibung der semantische Rahmen (engl. semantic frames) und die semantische und syntaktische Repräsentation der Valenzen von tausenden Wörter und Phrasen. Der Fokus von FrameNet ist die Kodierung von semantische Informationen zu Beschreibungen, die von Maschinen verstanden werden können.

FrameNet basiert auf die Theorie Rahmensemantik (engl. **frame semantics**). Die Idee ist, die Bedeutung von den meisten Wörtern basiert auf einen semantischen Rahmen: der Typ von dem Event, Relationen und Teilnehmer. Als Beispiel enthält Kochen meistens eine Person - den Koch, das zubereitete Essen, Behälter für das Essen und den Herd. In FrameNet werden diese Informationen repräsentiert als einen Rahmen (engl. **frame**) „Apply_heat“ und Rahmenelemente (engl. **frame elements - FEs**) „Cook“, „Essen“, „Container“ und „Herd“. Wörter, die einen Rahmen hervorrufen, heißen lexikalische Einheiten (engl. **lexical units - LUs**). „fry“, „bake“ und „boil“ sind LUs von „Apply_heat“. FrameNet definiert Rahmen und annotiert die mit FEs. Ein Beispiel ist:

$$[_{\text{Cook}} \text{ The boys}] \text{ GRILL } [_{\text{Food}} \text{ their catches}] \ [_{\text{Heating\_instrument}} \text{ on an open fire}].$$

5. Analyse und Entwurf

Die Aufgabe dieser Bachelorarbeit ist es, Aktionen in gesprochener Sprache zu erkennen. Das Projekt PARSE (siehe Kapitel 3) erfordert es, einen Agenten zu implementieren. Der Agent soll die von SNLP und SRL(siehe Abschnitt 3.4) erzeugte initiale Handlungsrepräsentation entgegennehmen und einzelne Aktionen daraus extrahieren. Der Agent soll die initiale Handlungsrepräsentation modifizieren um diese Informationen wieder im Graph darzustellen. Nach der Bearbeitung soll anhand vom Graph erkennbar sein, welche Aktionen in der Eingabe enthalten sind.

Daraus ergeben sich zwei Fragen, die in diesem Kapitel beantwortet werden sollen:

1. Wie sollen Aktionen dargestellt werden?
2. Wie sollen Aktionen erkannt werden, damit sie auf die Darstellung abgebildet werden können?

Im Abschnitt 5.1 wird zuerst darauf eingegangen, was Aktionen sind und was die Bestandteile von Aktionen sind. Dann wird vorgestellt, welche Möglichkeiten es gibt, Aktionen darzustellen. Dabei wird diskutiert, welche Informationen zur Darstellung einer Aktion nötig sind. Die Vorteile und Nachteile der jeweiligen Darstellungsmsmöglichkeiten werden vorgestellt und und schlussendlich die als beste aus der Analyse hervorgegangene ausgewählt.

Im Abschnitt 5.1.5 werden die Herausforderungen in der Aktiondarstellung präsentiert, diskutiert und teilweise gelöst. Weiterhin werden zusätzliche Wunschkriterien für die Erkennung von Definitionen definiert.

Im Abschnitt 5.2 wird diskutiert, wie die in einer Aktion enthaltene Informationen extrahiert werden sollen, damit sie auf die im Abschnitt 5.1 vorgestellten Darstellung abgebildet werden können.

5.1. Darstellung der Aktionen

Gesprochene Sätze können Aktionen enthalten [DSBS09]. Aktionen sind Tätigkeiten, die von Menschen oder Objekte wie ARMARIII ausgeübt werden können. Eine Aktion besteht dabei üblicherweise aus mehreren Bestandteile: Diese sind ein handelnder Akteur, einem Prädikat, welches die eigentliche Handlung beschreibt, und Parametern, welche die Aktion modifizieren bzw. zusätzliche Informationen hinzufügen. Nachfolgend sind die einzelnen Bestandteile näher beschrieben:

- Akteur: Akteure werden von Nominalphrasen gebildet. Im Satz nehmen sie meist die Rolle des Subjektes ein. Man findet den Akteur oder das Subjekt vom Satz mit der Frage „wer?“ oder „was?“.
- Prädikat: Das Prädikat der Aktion wird von einer Verbalphrase gebildet. Das Prädikat beschreibt die eigentliche Handlung der Aktion. Man findet das Prädikat vom Satz mit der Frage „was macht der Akteur?“.
- Parameter: Parameter der Aktion werden von Nominalphrasen, Präpositionalphrasen und Adverbialphrasen gebildet. Im Satz nehmen sie häufig die Rolle des direkten oder indirekten Objektes ein. Die Parameter bieten zusätzliche Informationen an, zum Beispiel das Objekt, den Zeitpunkt und den Ort. Man findet die Parameter mit den Fragen „wen?“, „wem?“, „was?“, „wann?“, „wo?“, „wie?“ und „warum?“.

„Akteur“, „Prädikat“ und „Parameter“ bieten somit alle Informationen an, die in einer Aktion enthalten sind. Eine Aktion kann in diese Bestandteile aufgeteilt werden.

Anhand dem folgenden Beispiel werden die Begriffe *Aktion*, *Akteur*, *Prädikat* und „Parameter“ illustriert.

Beispiel: Eine Aktion

„Armar brought me the popcorn.“

In diesem Beispiel führt „Armar“ die Aktion aus und ist somit der Akteur. „Brought“ beschreibt die genaue Tätigkeit und ist somit das Prädikat. „Me“ beschreibt ein Objekt beziehungsweise beantwortet die Frage „wem?“. Somit ist es ein Parameter. „The popcorn“ beschreibt auch ein Objekt und antwortet die Frage „was?“. Somit ist „the popcorn“ ein weiterer Parameter.

Aktionen müssen keinen Akteur enthalten. Ein Beispiel für eine Aktion ohne Akteur ist:

Beispiel: Eine Aktion ohne Akteur

„Get me an orange juice.“

Dabei handelt es sich um einen imperativen Satz. In imperativen Sätzen sind die Akteure nicht explizit angegeben.

Aktionen müssen auch keinen Parameter enthalten. Ein Beispiel für eine Aktion ohne Parameter ist:

Beispiel: Eine Aktion ohne Parameter

„Armar is listening.“

In einer Aktion können auch gleichzeitig der Akteur und die Parameter fehlen. Ein Beispiel für eine Aktion ohne Akteur und Paramter ist:

Beispiel: Eine Aktion ohne Akteur und Parameter

„Go.“

Anhand der obigen Beispielen kann man schlussfolgern, dass eine Aktion keinen Akteur und Parameter, aber immer ein Prädikat enhalten muss. Somit ist das Prädikat der wichtigste Teil einer Aktion.

Eine Eingabe kann mehrere Aktionen unterhalten. Diese soll unterschieden werden.

Um Aktionen darzustellen, gibt es zwei Lösungsmöglichkeiten: Token-basierte Ansätze und graph-basierte Ansätze.

5.1.1. VSOO-Struktur

Die Verb-Subjekt-Objekt-Objekt-Struktur ist ein typischer Vertreter der Token-basierten Ansätze. Sie wird von Metafor (siehe Abschnitt 4.2) verwendet, um natürlichsprachliche Sätze darzustellen. Wie der Name aussagt, kann die VSOO-Struktur ein Verb, ein Subjekt und zwei Objekte präsentieren. Diese Struktur kann auch benutzt werden, um Aktionen darzustellen. In diesem Fall bezeichnet „Verb“ das Prädikat, „Subjekt“ den Akteur und „Objekt“ den Parameter. Somit kann die VSOO-Struktur eine Aktion aus einem Prädikat, einem Akteur und zwei Parameter präsentieren. Mittels dieser Struktur können alle Informationen in einer Aktion, die in einer VSOO-Struktur dargestellt werden kann, direkt gefunden werden.

Die VSOO-Struktur hat folgenden Probleme: Zum einem können nur einen Akteur, ein Prädikat und zwei Parameter präsentiert werden. Diese ist problematisch, weil die Aktionen, die nicht genau diese Anzahl von Akteur, Prädikat und Paramter haben, nicht dargestellt werden können. Zum anderen kann in einer VSOO-Struktur nur eine Aktion dargestellt werden. Diese ist nicht ausreichend für die Eingabe von Texten, wo mehrere Aktionen vorhanden sind. Das erste Problem kann gelöst werden, indem man mehr Objekte in der Struktur zulässt. Das zweite Problem kann gelöst werden, indem man für jede Aktion eine VSOO-Struktur erstellt und sie hintereinanderstellt.

Die VSOO-Struktur hat noch weitere Probleme: Die Objekte in der Struktur werden nicht voneinander unterschieden. Es kann daher nicht zwischen den W-Fragen unterschieden werden. Außerdem ist nur sequentieller Zugriff auf die Elemente erlaubt. Diese verursacht Ineffizienz. Aus diesen Gründen wird auf die VSOO-Struktur verzichtet.

5.1.2. Eine graph-basierte Darstellung

Die Ausgabe von SNLP (siehe Abschnitt 3.4) wird in einem gerichteten Graph präsentiert. Sie dient als eine initiale Handlungsrepräsentation und ist die Eingabe dieser Bachelorarbeit. Daher kommt die Idee, Aktionen in Graphen darzustellen.

In der initialen Handlungsrepräsentation wird ein Token als ein Knoten dargestellt. Jeder Knoten enthält Informationen zu dem zugehörigen Token als Attribute. Die Wortart, die Phrase und die Befehlsnummer sind zum Beispiel als Informationen enthalten. Die Tokens in der initialen Handlungsrepräsentation werden mit Kanten verbunden. Diese Kanten dienen lediglich dazu, die Reihenfolge der Tokens in der Benutzereingabe darzustellen.

Um die Aktionen darzustellen, kann ein graph-basierter Ansatz verwendet werden. In diesem Ansatz wird der Graph von SNLP übernommen und modifiziert. Die Knoten in der initialen Handlungsrepräsentation werden modifiziert und neue Kanten werden hinzugefügt.

5.1.2.1. Rolle

Wie oben genannt, kann eine Aktion in Akteure, Prädikate und Parameter aufgeteilt werden. Jedes Token gehört zu einem dieser drei Bestandteile. Um diese Information darzustellen kann jedem Token eine Attribute „Rolle“ hinzugefügt werden, die die entsprechende Rolle des Tokens enthält. Akteur und Prädikat sind zwei Arten von Rollen. Es gibt verschiedene Parameter, die unterschiedliche W-Fragen beantworten: „wen“, „wem“, „was“, „wann“, „wo“, „wie“, „warum“. Es ist sinnvoll, die sechs W-Fragen statt der Parameter zu Rolle hinzuzufügen, weil die Informationen in einer Aktion somit nach der Semantik gruppiert sind und schnell gefunden werden können.

5.1.2.2. Befehlsnummer

In der initialen Handlungsrepräsentation hat jedes Token bereits eine Befehlsnummer als Attribut. Ab jedem Verb und das Wort „and“ werden die Wörter eine neue Befehlsnummer zugewiesen. So wird die Eingabe in verschieden Befehle geteilt.

Die Befehlsnummer kann in der graph-basierten Darstellung der Aktionen dazu dienen, den Graph in mehrere Aktionen aufzuteilen und den Tokens einzelne Aktionen zuzuweisen.

5.1.2.3. Beispiel

Eine mögliche graph-basierte Darstellung der Aktionen wird anhand dem Beispiel „Armar came and he brought me an orange juice“ illustriert. Die Darstellung ist in der Abbildung 5.4 zu sehen.

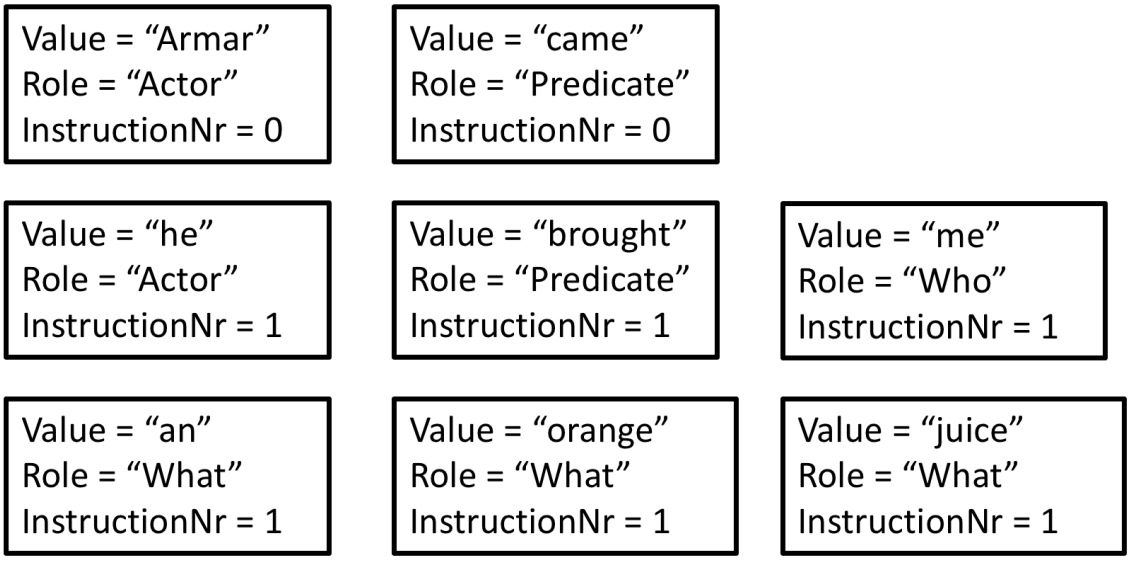

Abbildung 5.1.: Eine mögliche graph-basierte Darstellung der Aktion „Armar came and he brought me an orange juice.“

Dieses Beispiel enthält zwei Aktionen mit mehreren Parametern und einen Akteur. Der ersten Aktion „Armar came“ wird die Befehlsnummer „0“ zugewiesen. Diese kann anhand der Attribute `InstructionNr` im Knoten erkannt werden. In dieser Aktion spielt „Armar“ die Rolle des Akteurs und „came“ die Rolle des Prädikats. Dies kann anhand der Attribute `Role` in den Knoten erkannt werden. Die Attribute `Value` beinhaltet das Wort selbst.

Durch die Konjunktion „and“ wird eine zweite Aktion verknüpft. Das Token „and“ selbst wird aber nicht im Graph dargestellt, weil es nur zur Verknüpfung dient und keine Rolle in der Aktion einnimmt. In SNLP wird das Wort „and“ in einem Token repräsentiert. In der Aktiondarstellung spielt es allerdings keine Rolle.

Die zweite Aktion in diesem Beispiel ist „he brought me an orange juice“. Diese Aktion hat die Befehlsnummer „1“. In dieser Aktion spielt „brought“ die Rolle des Prädikats und „he“ die Rolle von Akteur. „Me“ ist ein Parameter und beantwortet die Frage „wer“. Deswegen hat es als Rolle `Who`. „An orange juice“ spielt die Rolle von „what“. Somit hat jedes Wort in der Phrase die Rolle `What`.

5.1.2.4. Nachteile

Ein Nachteil von der obigen Darstellung ist es, dass nur sequentieller Zugriff erlaubt ist.

Ein weiteres Problem ist es, dass ein Token zu mehreren Aktionen gehören kann. In dem Beispiel

Beispiel:

„Armar came and worked.“

gibt es zwei Aktionen: „Armar came“ und „Armar worked“. Das Token „Armar“ ist der Akteur sowohl der ersten als auch der zweiten Aktion. Ähnliche Fälle werden im Abschnitt 5.1.5 detaillierter vorgestellt. Mit der oben vorgestellten Darstellung ist der Graph in Abbildung 5.2 zu sehen.

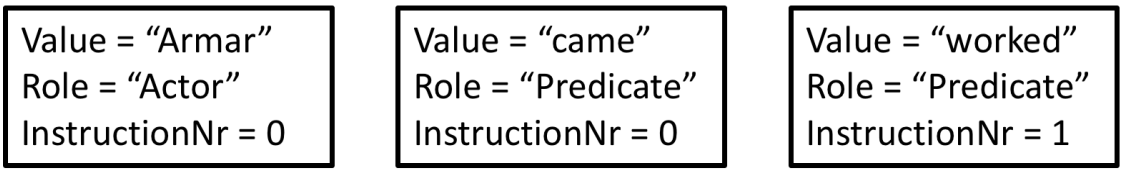

Abbildung 5.2.: Kanten in „Armar came and worked.“

Mit dieser Darstellung kann es nicht sichtbar sein, dass „Armar“ auch der zweiten Aktion gehört.

Dieses Problem kann durch Kanten gelöst werden. Im nächsten Abschnitt wird die verbesserte Darstellung vorgestellt.

5.1.3. Eine graph-basierte Darstellung mit Kanten

5.1.3.1. Kanten

Um die Zusammenhänge zwischen Tokens darzustellen und damit die Aktionen und die zugehörigen Informationen besser erkannt werden können, werden neue Kanten benötigt. Diese sollen die folgenden Funktionen erfüllen:

- Die Kanten sollen die Prädikate mit ihren Akteuren und Parametern verbinden.
- Die Kanten sollen die Token innerhalb einer Phrase, die gemeinsam eine Rolle spielen, verbinden.
- Wenn mehrere Aktionen im Graph vorhanden sind, sollen die Kanten die einzelnen Aktionen miteinander verbinden, die der Reihenfolge der Aktionen entsprechen.

Um zwei Aktionen zu verbinden werden lediglich die Prädikate der Aktionen verbunden. Diese ist die beste Möglichkeit, weil eine Aktion immer ein Prädikat enthält (Abschnitt 5.1).

Jede Kante kann nur zwei Knoten verbinden. Ein Prädikat, Akteur oder Parameter kann allerdings aus mehreren Knoten bestehen. Um zwei Phrasen zu verbinden können das erste Token einer Phrase mit dem ersten Token der nächsten Phrase verbunden werden. Dies wird mit dem folgenden Beispiel illusriert.

Beispiel:

„Bring the professor a book.“

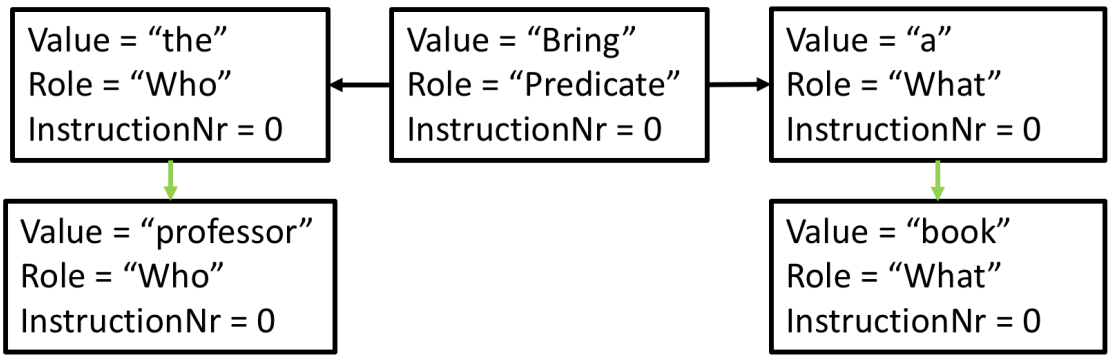

Abbildung 5.3.: Kanten in „Bring the professor a book.“

Die Abbildung Abbildung 5.3 zeigt die Kanten zu dem Beispiel. „The professor“ und „a book“ sind Parameter in der Aktion. Die grünen Kanten verbinden Token innerhalb einer Phrase. Um das Prädikat „bring“ mit dem Parameter „the professor“ zu verbinden, wird das erste Token des Prädikats („bring“) und das erste Token des Parameters „the“ mit einer schwarzen Kante verbunden. Analog werden „bring“ und „a“ verbunden.

Um zwischen den Kanten mit verschiedenen Funktionalitäten zu unterscheiden ist es sinnvoll, den Kanten verschiedenen Typen zuzuweisen. Kanten, die Token innerhalb einer Phrase verbinden und Kanten, die einzelnen Aktionen miteinander verbinden können jeweils zwei Typen von Kanten repräsentieren. Kanten, die Prädikate mit Akteuren und Parametern verbinden, können unterschieden werden:

1. Kanten, die Prädikate mit Akteuren verbinden
2. Kanten, die Prädikate mit Parametern der Art „what“ verbinden
3. Kanten, die Prädikate mit Parametern der Art „who“ verbinden
4. Kanten, die Prädikate mit Parametern der Art „when“ verbinden
5. Kanten, die Prädikate mit Parametern der Art „where“ verbinden
6. Kanten, die Prädikate mit Parametern der Art „why“ verbinden
7. Kanten, die Prädikate mit Parametern der Art „how“ verbinden

Diese kann allerdings auch vereinfacht werden. Dies wird im Abschnitt 5.1.4 näher diskutiert.

Der modifizierte Graph ist grundsätzlich ein Baum wenn man die neu eingefügten Kanten verfolgt. Das erste Prädikat ist die Wurzel des Baums.

5.1.3.2. Beispiel

Eine mögliche graph-basierte Darstellung der Aktionen wird anhand dem Beispiel „Armar came and he brought me an orange juice“ illustriert. Die Darstellung ist in der Abbildung 5.4 zu sehen. Zur Veranschaulichung werden die Prädikate rot, Akteure blau und Parameter grün dargestellt. Kanten, die Prädikate mit ihren Akteuren verbinden, werden pink dargestellt. Kanten, die Prädikate mit Parametern der Art „who“ verbinden, werden orange dargestellt. Kanten, die Prädikate mit Prametern der Art „what“ verbinden, werden lila dargestellt. Kanten, die die Tokens innerhalb einer Phrase verbinden, werden grün dargestellt. Kanten, die Aktionen miteinander verbinden, werden rot dargestellt.

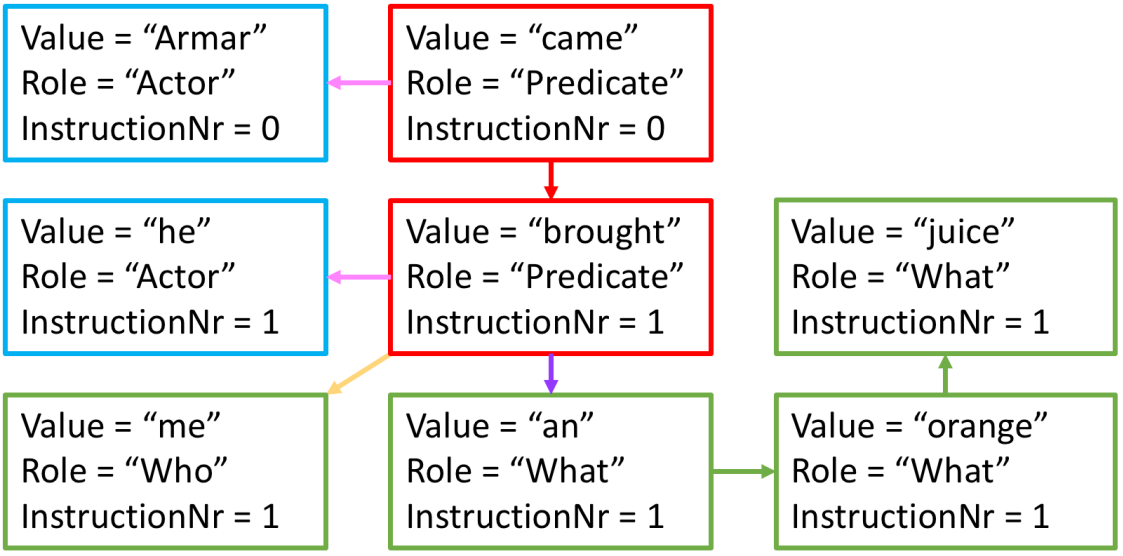

Abbildung 5.4.: Eine mögliche graph-basierte Darstellung der Aktion „Armar came and he brought me an orange juice.“

Es wird mit einer pinken Kante von „came“ auf „Armar“ gezeigt, weil die pinken Kanten Prädikate mit ihren Akteuren verbinden.

Es wird mit grünen Kanten von „an“ auf „orange“ und von „orange“ auf „ juice“ gezeigt, um sie zu verbinden. Es wird jeweils mit pinken, orangen und lila-farbigen Kanten von „brought“ auf „he“, „me“ und „an“ gezeigt. Es ist besonders zu beachten, dass es von „brought“ auf „an“ gezeigt wird, weil „an“ das erste Token der Phrase „an orange juice“ ist.

Rote Kanten verbinden Prädikaten von verschiedenen Aktionen. Deswegen wird mit einer roten Kanten von „came“ auf „brought“ gezeigt.

5.1.3.3. Verbesserungsmöglichkeiten

Wie im Abschnitt 5.1.2.4 diskutiert, gibt es die Fälle wo ein Token zu mehrere Aktionen gehört. Daher ist die Befehlsnummer nicht sinnvoll, um Aktionen in der Darstellung auseinanderzutrennen. Die Befehlsnummer wird lediglich gebraucht, um am Anfang der Bearbeitung die Aktionen auseinanderzutrennen und diese für die weitere Bearbeitung bereitzustellen.

Es wird daher in der Darstellung der Aktionen auf die Befehlsnummer verzichtet werden. Die verbesserte Darstellung wird im nächsten Abschnitt vorgestellt.

5.1.4. Eine graph-basierte Darstellung ohne Befehlsnummer

Die im Abschnitt 5.1.2 vorgestellte graph-basierte Darstellung wird im Laufe der Bachelorarbeit weiter verbessert. Es wird auf die Befehlsnummer verzichtet und die Typisierung der Kanten wird vereinfacht.

5.1.4.1. Befehlsnummer

Im Abschnitt 5.1.2 wird eine graph-basierte Darstellung für Aktionen vorgeschlagen, bei der jedem Token eine Befehlsnummer zugewiesen wird. Die Befehlsnummern dienen dazu, den Token einzelne Aktionen zuzuweisen. Die Befehlsnummern sind allerdings auch

implizit durch die Kanten gegeben. Wenn der Graph entlang der Kanten, die Aktionen miteinander verbinden, verfolgt wird, kann die Reihenfolge der Prädikate bestimmt werden. Jedes Prädikat gehört genau zu einer Aktion. Die Akteure und Parameter werden mit ihren Prädikaten auch durch Kanten verbunden. Somit kann auch herausgefunden werden, zu welcher Aktion beziehungsweise welchen Aktionen ein Akteur oder Parameter gehört.

Außerdem ist es nicht immer möglich, jedem Akteur oder Parameter eine einzelne Aktion zuzuweisen. Es existieren die Fälle, dass ein Token zu mehreren Aktionen gehört. Dieses Problem wird in Abschnitt 5.1.5 diskutiert.

Deswegen wird auf Befehlsnummern zur Identifizierung von Aktionen verzichtet.

5.1.4.2. Kanten

Im Abschnitt 5.1.2 gibt es neun Kantentypen, diese sind:

1. Kanten, die Token innerhalb einer Phrase, die gemeinsam eine Rolle spielen verbinden
2. Kanten, die einzelnen Aktionen miteinander verbinden
3. Kanten, die Prädikate mit Akteuren verbinden
4. Kanten, die Prädikate mit Parametern der Art „what“ verbinden
5. Kanten, die Prädikate mit Parametern der Art „who“ verbinden
6. Kanten, die Prädikate mit Parametern der Art „when“ verbinden
7. Kanten, die Prädikate mit Parametern der Art „where“ verbinden
8. Kanten, die Prädikate mit Parametern der Art „why“ verbinden
9. Kanten, die Prädikate mit Parametern der Art „how“ verbinden

Es ist allerdings überflüssig, zwischen den dritten bis neunten Kantentypen zu unterscheiden. Die Kanten dienen alle dazu, ein Prädikat mit seinem Akteuer oder Parameter zu verbinden. Der Typ der Endknoten kann anhand des Attributs `Role` erkannt werden.

Deswegen werden die dritten bis neunten Kantentypen zu einem Kantentyp zusammengefasst: Kanten, die Prädikaten mit Akteuren oder Parametern verbinden.

5.1.4.3. Beispiel

Eine verbesserte graph-basierte Darstellung der Aktionen wird anhand dem Besipiel „Armar came and he brought me an orange juice“ illustriert. Die Darstellung ist in der Abbildung 5.5 zu sehen. Zur Veranschaulichung werden die Prädikate rot, Akteure blau und Parameter grün dargestellt. Kanten, die Prädikaten mit ihren Akteuren oder Parametern verbinden, werden schwarz dargestellt. Kanten, die die Tokens innerhalb einer Phrase verbinden, werden grün dargestellt. Kanten, die Aktionen miteinander verbinden, werden rot dargestellt.

Das Token „came“ ist die Wurzel des Baums. Somit ist es das Prädikat der ersten Aktion. Das Token „Armar“ ist mit einer schwarzen Kante mit „came“ verbunden und hat die Rolle „Actor“. Somit ist es der Akteuer der ersten Aktion.

Es wird mit einer roten Kante von „came“ auf „brought“ gezeigt. Somit ist „brought“ das Prädikat in der zweiten Aktion. Es werden mit schwarzen Kanten auf „he“, „me“ und „an“ gezeigt, die jeweils die Rolle „Actor“, „Who“ und „What“ spielen. Somit ist „he“ der Akteur und „me“ und „an“ die Parameter der zweiten Aktion. „An“, „orange“ und „ juice“ werden mit grünen Kanten verbunden. Somit sind sie gemeinsam als eine Phrase ein Parameter von Typ „What“ der zweiten Aktion.

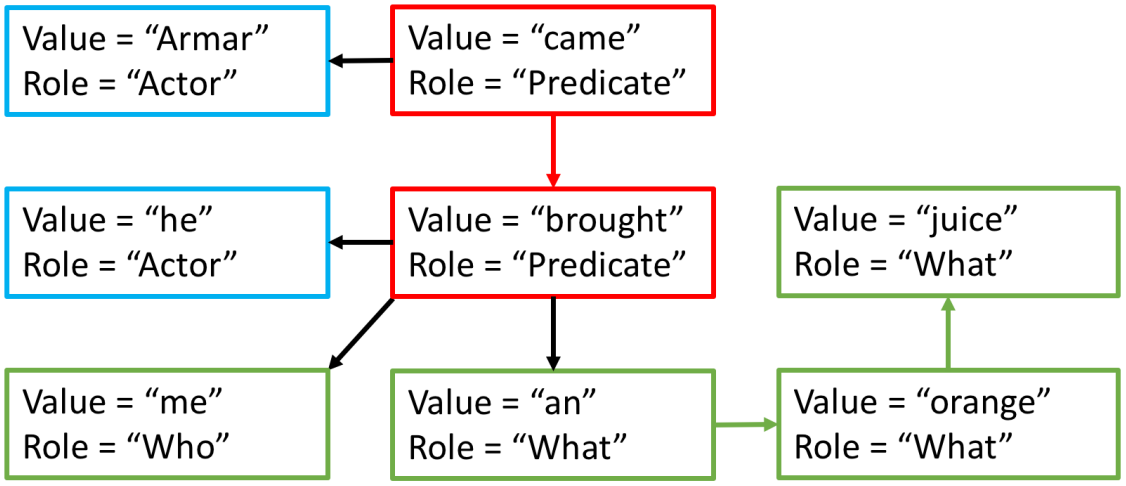

Abbildung 5.5.: Verbesserte graph-basierte Darstellung von „Armar came and he brought me an orange juice.“

5.1.4.4. Passive Sätze

Sätze, die gleiche Semantik haben, sollen eine gleiche Darstellung haben, egal ob sie aktiv oder passiv sind. Dies ist möglich, weil die Akteure und Parameter in verschiedenen Formulierungen die gleichen semantische Rollen haben.

Die folgenden zwei Sätze haben die gleiche Semantik.

Beispiele mit gleicher Semantik:

„Armar brought me an orange juice.“
„An orange juice was brought to me by Armar.“

Die graph-basierte Darstellung von den Sätze sind in der Abbildung 5.6 zu sehen.

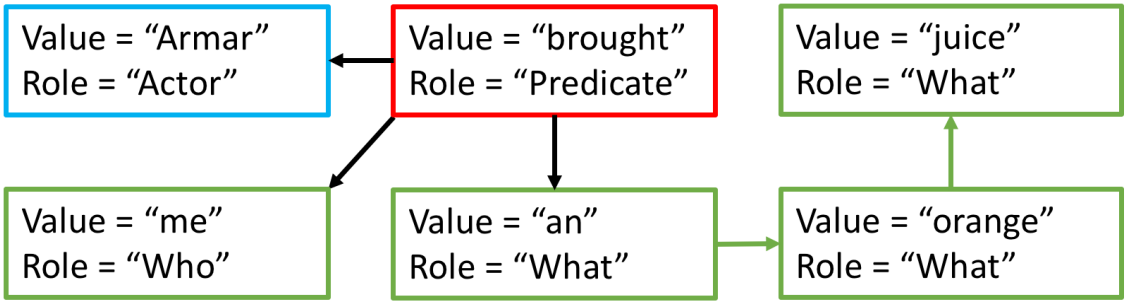

Abbildung 5.6.: Die Aktionen „Armar brought me an orange juice.“ und „An orange juice was brought to me by Armar“ haben die gleiche Darstellung

5.1.5. Herausforderung in der Aktiondarstellung

Eine Herausforderung in der Aktiondarstellung stellt die Konjunktion „and“ dar. Das Wort „and“ hat mehrere Funktionen. [Dic16]

1. „And“ verbindet nebengeordnete Wörter, Phrase und vollständige Sätze, zum Beispiel „pens and pencils“

2. „And“ verbindet zwei Hauptsätze und kennzeichnet eine zeitliche Reihnenfolge, zum Beispiel „He read for an hour and went to bed“
3. „And“ verbindet zwei gleichzeitige Ereignisse, zum Beispiel „to sleep and dream.“

„And“ hat auch unterschiedlichen Funktionen in Aktionen.

Von der ersten oben genannten Kategorie können zwei weitere Kategorien für Aktionen extrahiert werden: „and“ verbindet vollständige Aktionen und „and“ verbindet Parameter. Wenn ein „and“ vollständige Sätze verbindet, verbindet es auch vollständige Aktionen. Wenn ein „and“ Wörter oder Phrasen verbindet, verbindet es einzelnen Parameter.

Die zweite und dritte Kategorie haben für Aktionen die gleiche Bedeutung: das „and“ verbindet zwei Aktionen, egal ob sie gleichzeitig oder nacheinander passieren.

Diese drei Kategorien werden im Folgenden genauer vorgestellt.

„And“ verbindet vollständige Aktionen

In dieser Kategorie verbindet das Wort „and“ separate Sätze, die jeweils eine Aktion enthalten.

Ein Beispiel aus dieser Kategorie ist:

Beispiel:

„Armar came and he brought me an orange juice.“

In Abschnitt 5.1.2 und Abschnitt 5.1.4 wird vorgestellt, wie dieser Satz graphisch dargestellt werden soll. Die zugehörige Darstellung ist in der Abbildung 5.5 zu sehen.

„And“ verbindet unvollständige Aktionen mit gemeinsamen Akteur und unterschiedlichen Prädikaten

In dieser Kategorie sind auch mehrere Aktionen enthalten. Der Akteur kommt einmal vor und führt mehrere Aktionen aus.

Ein Beispiel aus dieser Kategorie ist:

Beispiel:

„Armar came and brought me an orange juice.“

Die eine Aktion in diesem Beispiel ist: „Armar came.“ Die andere ist: „brought me an orange juice.“ Der Akteur der zweiten Aktion ist implizit gegeben und der gleiche wie der Akteur der ersten Aktion. Vollständig ausgeschrieben wäre die zweite Aktion: „Armar brought me an orange juice.“ Das heißt, das Token „Armar“ gehört sowohl zu der ersten Aktion, als auch zu der zweiten Aktion. In diesem Fall ist die Zugehörigkeit zu einer Aktion nicht eindeutig. Deswegen ist eine Zuweisung von Befehlsnummer unmöglich.

Die graph-basierte Darstellung dieses Beispiels ist in der Abbildung 5.7 zu sehen.

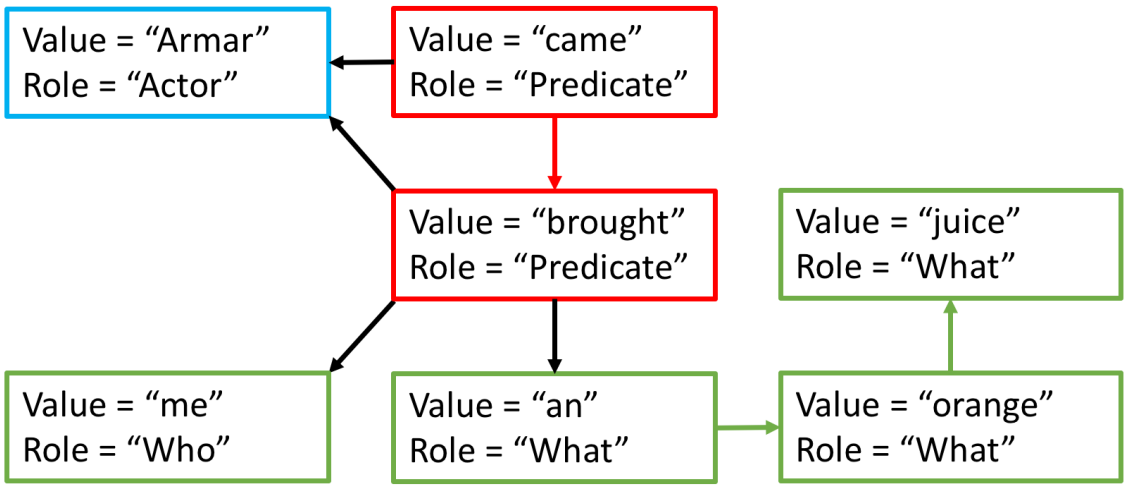

Abbildung 5.7.: Eine graph-basierte Darstellung der Aktion „Armar came and brought me an orange juice.“

In imperativen Sätzen kann es auch vorkommen, dass nur ein Akteur einmal auftritt und darauffolgend sind mehrere Aktionen. Ein Beispiel ist:

Beispiel:

„Armar bring me an orange juice. Then get me an apple juice. Also bring me a grape juice please.“

In solchen Fällen wird der Akteur der ersten Aktion den nachfolgenden Aktionen zugewiesen.

„And“ verbindet Parameter

In dieser Kategorie verbindet die Konjunktion „and“ mehrere Satzteile.

Ein Beispiel aus dieser Kategorie ist:

Beispiel:

„Armar brought me an orange juice and an apple juice.“

In diesem Beispiel sind dies die Phrasen „an orange juice“ und „an apple juice“. Semantisch gesehen enthält der Satz zwei Aktionen, nämlich „Armar brought me an orange juice“ und „Armar brought me an apple juice“. Das Prädikat „brought“ wird in beiden Aktionen benutzt. Wenn der Knoten „brought“ auch in beiden Aktionen benutzt würde, würde die Kanten, die Aktionen verbinden, eine Schlinge an dem Knoten „brought“ bilden. Diese ist problematisch, weil es unklar wird, welche Aktion zuerst ausgeführt wird und wie oft die Schlinge durchgelaufen werden soll. Deswegen ist diese Möglichkeit undurchführbar. Ein anderer Ansatz ist, das Prädikat zu duplizieren. Der zweite Aktion wird das „neue“ Prädikat „brought“ zugewiesen. Der Akteur „Armar“ und der Parameter „me“ gehören sowohl zu der ersten Aktion als auch zu der zweiten Aktion.

Die graph-basierte Darstellung dieses Beispiels ist in der Abbildung 5.8 zu sehen. Das neu erzeugte Prädikat wird lila markiert.

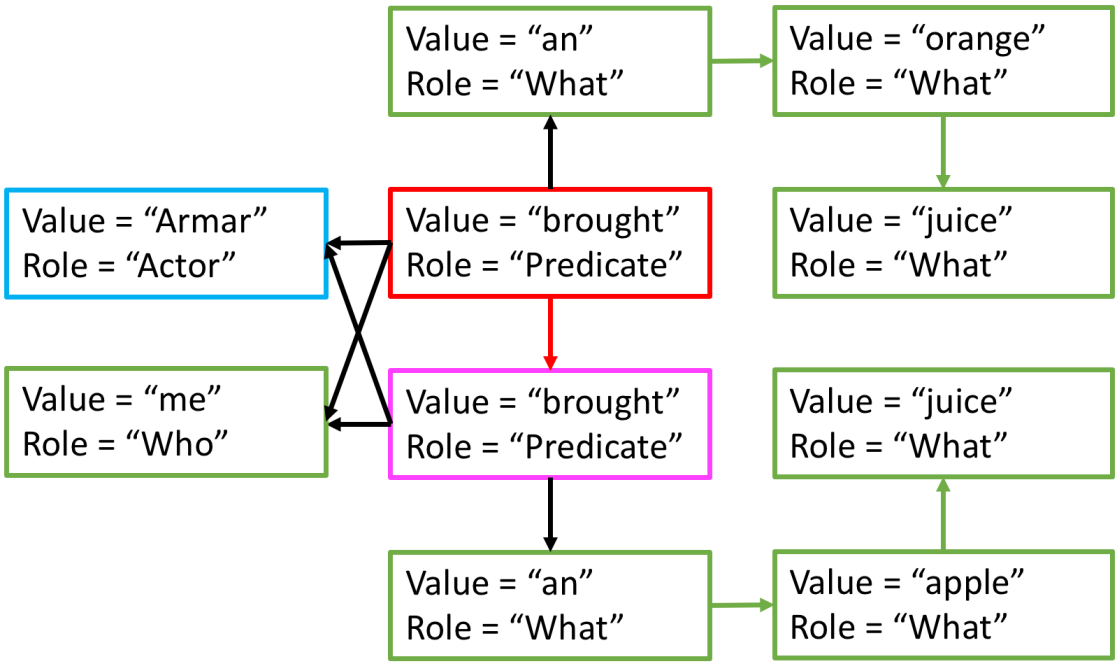

Abbildung 5.8.: Eine graph-basierte Darstellung der Aktion „Armar brought me an orange juice and an apple juice.“

Zwei ähnliche Beipiele sind:

Beispiel:

„Armar and Asfour came.“

Beispiel:

„Armar works on Monday and Tuesday.“

In den beiden Beispielen muss wie im oberen Beispiel herangegangen werden. Die Prädikate müssen dupliziert werden.

Es gibt einige Ausnahmen, die ausgeschlossen werden sollen. Zum Beispiel enthält der Satz „Armar brought me a mac and cheese.“ nur eine Aktion. „mac and cheese“ ist ein Gericht, obwohl es das Wort „and“ enthält und darf nicht getrennt werden.

5.2. Erkennung der Aktionen

Informationen in den gesprochenen Sätzen müssen extrahiert und auf die Bestandteile der Aktionen (Akteur, Prädikat, und Parameter) abgebildet werden. Somit kann es für die Aktionen, die in den Sätzen enthalten sind, eine graph-basierte Darstellung erstellt werden. Es gibt zwei Schritte um dieses Ziel zu realisieren:

1. Für Sätze oder Teilsätze, die eine Aktion enthalten, werden den Tokens entsprechenden Rollen zugeordnet.
2. Paramter werden auch Aktionen zugeordnet, die durch ein „and“ verbunden sind.

5.2.1. Identifizierung der Rollen in einzelnen Aktion

Zuerst werden für Sätze oder Teilsätze, die eine einzelne Aktion enthalten, Rollen zugeordnet. Für das Beispiel

Beispiel:

„Armar came and brought me an orange juice.“

werden „Armar came“ und „brought me an orange juice“ separat bearbeitet.

In „Armar came“ wird „Armar“ die Rolle *actor* und „came“ die Rolle *predicate* zugeteilt. In „brought me an orange juice“ wird „brought“ die Rolle *predicate*, „me“ die Rolle *who* und „an orange juice“ die Rolle *what* zugeteilt.

Für das Beispiel

Beispiel:

„Armar brought me an orange juice and an apple juice.“

werden „Armar brought me an orange juice“ und „an apple juice“ separat bearbeitet.

In „Armar brought me an orange juice“ wird „Armar“ die Rolle *actor*, „brought“ die Rolle *predicate*, „me“ die Rolle *who* und „an orange juice“ die Rolle *what* zugeteilt. In „an apple juice“ wird „an apple juice“ die Rolle *what* zugeteilt.

Es ist offensichtlich, dass in der zweiten Aktion der Akteur und der Parameter *who* noch fehlen. Diese wird im nächsten Schritt bearbeitet.

Die Erkennung der Rollen von einzelnen Tokens basiert auf die Informationen von SRLabeler, SENNA und SNLP. Der SRLabeler ordnet Token semantischen Rollen zu. Akteure werden mit „A0“ gekennzeichnet und Prädikaten werden mit „V“ gekennzeichnet. Anhand semantischen Rollen werden Akteure und Prädikate erkannt. SENNA führt eine Eigennamenerkennung durch. Personen und Orte werden mit Eigennamen gekennzeichet. Anhand der Eigennamen werden Parameter *Who* und *Where* erkannt. SNLP führt Phrasenerkennung und Wortarterkennnug durch. Adverbialphrasen und Pronomen werden dadurch gekennzeichnet. Anhand Phrasen und Wortarten werden Parameter *Who* und *How* erkannt.

Dieses Verfahren wird im Abschnitt 6.2.2 auf der Implementierungsebene erklärt.

5.2.2. Zuordnung der Parameter in Aktionen

Teilsätze, die mit anderen durch die Konjunktion „and“ verbunden sind, werden weiter bearbeitet. Es wird zuerst geprüft, welche von den drei Fällen, die im Abschnitt 5.1.5 vorgestellt sind, auftritt.

Im Folgenden wird erklärt, wie die drei Fällen behandelt werden.

„And“ verbindet vollständige Aktionen

Es handelt sich um zwei separaten Aktionen. Kein Paramter gehört zu mehreren Aktionen. Die Rollen der Token werden im ersten Schritt erkannt. Im diesen Schritt müssen sie daher nicht weiter bearbeitet werden.

Ein typisches Beispiel in dieser Kategorie ist:

Beispiel:

„Armar came and he brouht me an orange juice“

„And“ verbindet unvollständige Aktionen mit gemeinsamen Akteur und unterschiedlichen Prädikaten

In dieser Kategorie ist das Prädikat in allen Aktionen vorhanden. Der Akteur kommt allerdings nur einmalig vor. In diesem Fall wird der Akteur von der ersten Aktion allen darauf folgenden Aktionen als Akteur zugewiesen, bis eine Aktion vorkommt, wo der Akteur vorhanden ist.

Das oben genannte Beispiel

Beispiel:

„Armar came and brought me an orange juice“

gehört zu dieser Kategorie.

In der zweiten Aktion „brought me an orange juice“ ist kein Akteur vorhanden. Daher wird der Akteur der ersten Aktion ihr zugewiesen.

„And“ verbindet Parameter

Wenn ein Parameter durch ein „and“ mit einer vollständigen Aktion verbunden ist, wird ihr alle Akteur und Parameter zugewiesen, die in der vollständigen Aktion enthalten ist, bis auf diesem Parameter selbst.

Ein typisches Beispiel in dieser Kategorie ist

Beispiel:

„Armar brought me an orange juice and an apple juice“

Hier verbindet das „and“ zwei Parameter. Nach dem ersten Schritte wird in der zweiten Aktion nur „an orange juice“ als „What“ erkannt. Die erste Aktion ist „Armar brought me an orange juice“. Die zweite Aktion soll genau die Gleiche wie die erste Aktion sein, wenn „an orange juice“ durch „an apple juice“ ersetzt wird: „Armar brought me an apple juice“. Das heißt, der Akteur:„Armar“ und der Parameter *Who*: „me“ werden der zweiten Aktion zugewiesen. Der Parameter *What* wird nicht zugewiesen, weil er in der zweiten Aktion bereits vorhanden ist.

5.3. Ablauf

Um die Schritte, die in Abschnitt 5.2 vorgestellt sind, zusammenzufassen, wird ein grobes Ablaufdiagramm des Agenten in der Abbildung 5.9 gezeichnet.

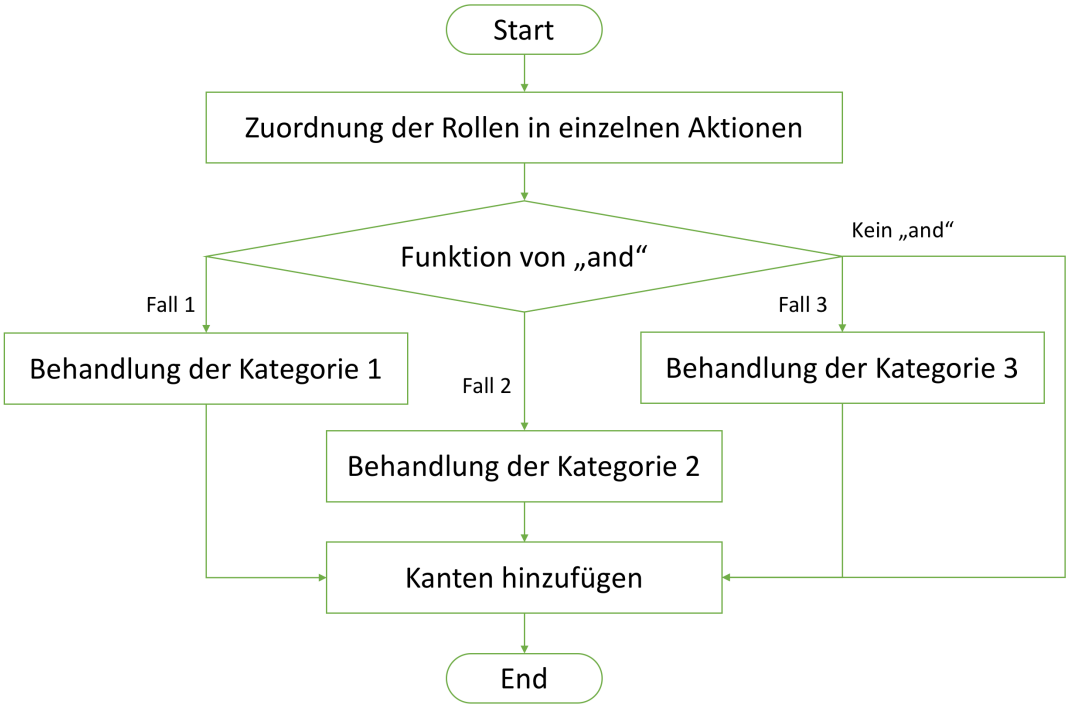

Abbildung 5.9.: Grober Ablauf des Agenten

Zuerst werden in Sätzen oder Teilsätzen, die eine Aktion enthalten, den Tokens entsprechenden Rollen zugeordnet. Dann werden Paramter Aktionen zugeordnet, die durch ein „and“ verbunden sind. Zum Schluss werden Kanten anhand den Parametern hinzugefügt.

6. Implementierung

In diesem Kapitel werden zuerst die Eingabe des Agenten (Abschnitt 6.1) vorgestellt, weil die Implementierung auf der Eingabe basiert. Dann werden die Aufgaben, die im Kapitel 5 vorgestellt wurden wiederholt und auf der Implementierungsebene illustriert. Diese findet man im Abschnitt 6.2. Im Abschnitt 6.3 werden die Implementierungsdetails beschrieben. Im Abschnitt 6.4 wird vorgestellt, wie das erstellte Werkzeug verwendet werden soll und wie das Ergebnis ausgegeben werden soll.

6.1. Eingabe des Agenten

Da das erstellte Werkzeug innerhalb des Projekts PARSE (Kapitel 3) als ein Agent arbeitet, beeinflusst die Eingabe die Implementierung. Die Datenstrukturen in der Implementierung basieren auf die Datenstrukturen in der Eingabe. Die natürlichen Sätze werden zuerst von SNLP und SRLabeler bearbeitet, und dann an dem erstellten Agent weitergegeben (siehe Abbildung 3.1).

6.1.1. SNLP

Die Ausgabe von SNLP (Abschnitt 3.4) dient als Eingabe des erstellten Agenten. Deswegen muss SNLP genauer betrachtet werden.

Die Eingabe von SNLP ist ein `PrePipelineData-Objekt`, das den Eingabesatz enthält. Ein Tokenizer tokenisiert die Eingabe, bevor sie an SNLP weitergegeben wird. SNLP wandelt den Satz in einen Graph, der den Satz enthält um, und fügt Informationen hinzu. SNLP weist jedem Token im Satz folgende Informationen zu: die Wortart (engl. POS Tag), die phrasale Markierung (engl. Chunking) und die Befehlsnummer. Der von SNLP erzeugte Graph dient als eine initiale Handlungsrepräsentation im Projekt PARSE.

Der Graph (Abschnitt 2.1) ist gerichtet und enthält Knoten und Kanten. Die enthaltenen Informationen werden in den nachfolgenden Abschnitten vorgestellt.

6.1.1.1. Knoten

Jeder Knoten hat einen Typ und Attribute. SNLP stellt jedes Wort in einem Knoten vom Typ „token“ dar. SNLP fügt dem Knoten sechs Attribute hinzu. Sie sind:

- value: das Token als ein String

- pos: die Wortart (Abschnitt 2.4) des Tokens
- chunkIOB: die Phrase (Abschnitt 2.5) des Tokens im IOB-Format (Abschnitt 2.5.1)
- chunkName: der Phrasename des Tokens
- predecessors: die Anzahl von Tokens, die zu derselben Phrase gehören vor diesem Token,
- successors: die Anzahl von Tokens, die zu derselben Phrase gehören nach diesem Token
- instructionNumber: die Nummer des Befehls, zu dem das Token gehört
- position: die Position des Tokens in der Eingabe. Diese wird von 0 an durchnummeriert.

Eine Aufzählung von allen Knoten, die zugehörigen Attributen und Werten bei der Beispieleingabe „Armar brought me an orange juice and an apple juice.“ ist in der Abbildung 6.1 zu sehen.

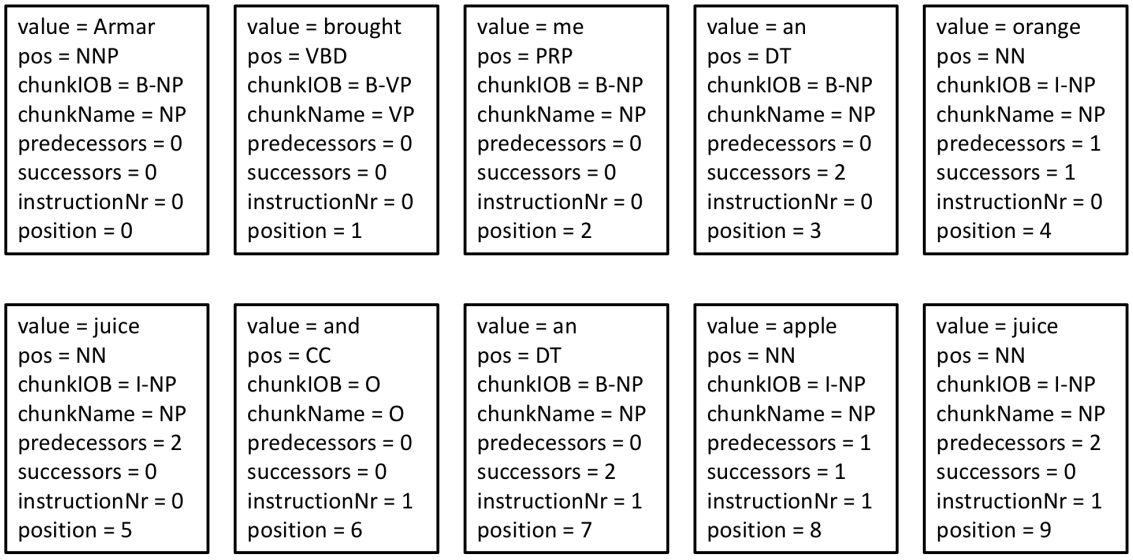

Abbildung 6.1.: Knoten von „Armar brought me an orange juice and an apple juice“

6.1.1.2. Kanten

Jede Kante hat einen Typ und Attribute. SNLP fügt dem Graph Kanten von Typ „relation“ hinzu. Die „relation“ Kanten verbinden die Tokens in der Eingabe, sodass die Reihenfolge von den Tokens wiedererkannt werden können. Jede Kante zeigt von einem Token auf das nachfolgende Token. Die Kanten enthalten eine Attribute „value“. Diese hat den Wert „NEXT“.

Eine Liste von allen Kanten bei der Beispieleingabe „Armar brought me an orange juice and an apple juice.“ ist in der Abbildung 6.2 zu sehen.

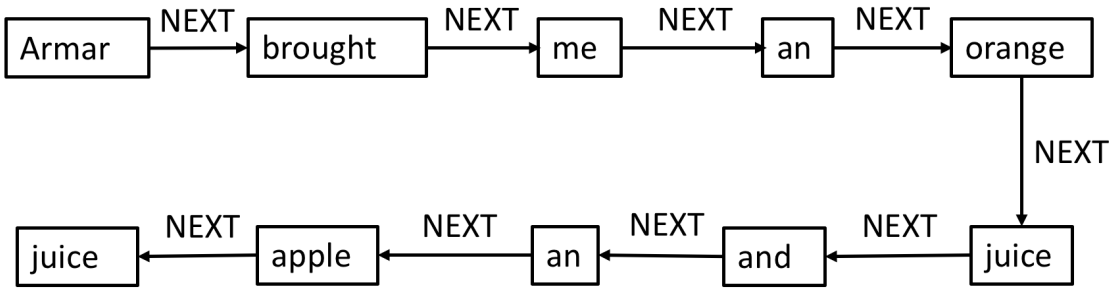

Abbildung 6.2.: Kanten von „Armar brought me an orange juice and an apple juice“

6.1.2. SRLabeler

Der SRLabeler (engl. Semantic Role Labeler) ist ein Vorverarbeitungsschritt im Projekt PARSE. SRLabeler fügt der initialen Handlungsrepräsentation Informationen über semantischen Rollenzuteilung (siehe Abschnitt 2.7) zu. SRLabeler wird aufgerufen, bevor der in dieser Bachelorarbeit erstellte Agent aufgerufen wird. Daher bekommt der erstellte Agent auch die Ausgabe von SRLabeler als Eingabe.

SRLabeler fügt der initialen Handlungsrepräsentation einen neuen Typ von Kanten hinzu: *srl*. Eine *srl* Kante hat die folgenden Attribute:

1. IOBES
2. ROLE_VALUE_NAME
3. PROPBANK_ROLE_DESCRIPTION
4. VN_ROLE_NAME
5. ROLE_CONFIDENCE_NAME

Darunter sind *IOBES* und *ROLE_VALUE_NAME* interessant für diese Bachelorarbeit. *IOBES* kennzeichnet den Beginn, das Innere und das Ende von einer Phrase. Wenn die Phrase aus einem einzelnen Token besteht, wird diese auch durch *IOBES* gekennzeichnet. *ROLE_VALUE_NAME* kennzeichnet die semantische Beziehung zwischen dem Endknoten und dem Startknoten der Kante.

Unter der Eingabe „Armar brought me an orange juice and an apple juice“ wird SRLabeler dem Graph folgenden Kanten hinzugefügt:

***srl* Kanten von „Armar brought me an orange juice and an apple juice.“**

brought —S-A0—>Armar
brought —S-V—>brought
brought —S-A2—>me
brought —B-A1—>an
an —I-A1—>orange
orange —E-A1—>juice

„Armar“ dient als das Argument *A0* (Subjekt) des Prädikats „brought“. Durch die IOBES-Kennzeichen *S* wird gekennzeichnet, dass Armar das einzige Token im Subjekt ist. „Brought“ dient als das Argument *V* (Verb) in der Aktion. Es ist das einzige Token im Verb. „Me“ dient als das Argument *A2* des Prädikats „brought“. Es ist das einzige Token im Verb.

„An“ dient als das Argument *A1* des Prädikats „brought“. Es ist der Beginn des Arguments. „Orange“ und „ juice“ dienen auch als Argument *A1* und sind jeweils das Innere und das Ende des Arguments.

6.2. Erkennung und Darstellung von Aktionen

Im Kapitel 5 wurde vorgestellt,

1. wie Aktionen dargestellt werden sollen (Abschnitt 5.1)
2. und wie Aktionen erkannt werden sollen, damit sie auf die Darstellung abgebildet werden können. (Abschnitt 5.2)

Zur Erkennung von Aktionen wurden in Kapitel 5 zwei Schritte definiert: Zuerst werden Rollen in einzelnen Aktionen zugeordnet. Dann werden Parameter Aktionen zugeordnet, die durch „and“ verbunden sind

In den nachfolgenden Abschnitten werden die Fragestellungen aus Kapitel 5 verfeinert und anschließend auf Implementierungsebene beantwortet.

1. wie werden Aktionen dargestellt (Abschnitt 6.2.1)?
2. wie werden Rollen in einzelnen Aktionen zugewiesen (Abschnitt 6.2.2)?
3. wie werden Parameter Aktionen zugewiesen, die durch „and“ verbunden sind. (Abschnitt 6.2.3)?
4. wie werden Kanten hinzugefügt (Abschnitt 6.2.4)?

6.2.1. Darstellung der Aktionen

Die interne Repräsentation von Sätzen in PARSE ist ein Graph. Die Information über die Aktionen, die aus den Sätzen extrahiert werden, werden dem Graph hinzugefügt.

6.2.1.1. Knoten

Aktionen haben die Bestandteile Akteur, Prädikat und Parameter - die W-Fragen. Jeder Bestandteil besteht aus Token. Jedes Token spielt somit eine Rolle von Akteur, Prädikat oder Parameter.

Die Tokens werden als Knoten repräsentiert. Die Knoten, die Tokens in der Eingabe enthalten, haben den Typ von „token“. Die Attribute „value“ in „token“ Knoten enthält den Inhalt von dem jeweiligen Token. Den Knoten wird für die Aktiondarstelung eine Attribute „role“ hinzugefügt. Diese kann einen Wert von (*ACTOR, PREDICATE, WHAT, WHEN, WHO, WHERE, WHY, HOW*) annehmen. Token, denen keine Rolle zugeordnet wird, zum Beispiel „and“ und „but“, haben role = null.

Wenn ein Prädikat dupliziert wird, entsteht ein neuer Knoten. Dieser Knoten soll einen anderen Typ als „token“ haben, damit unterschieden werden kann, was in der Eingabe vorkommt und was hinzugefügt wird. Es wird ein neuer Knotentyp „newPredicate“ für solche Knoten erstellt. „newPredicate“ Knoten müssen keine Attribute „role“ haben, weil sie ausschließlich Prädikat sein können. „newPredicate“ Knoten haben nur ein Attribut: „value“. „value“ enthält den gleichen Wert wie das alte Prädikat.

6.2.1.2. Kanten

Im Graph sind bereits Kanten vorhanden, die von einem Knoten auf den nächsten Knoten zeigen, nach der Reihenfolge des Vorkommens in der Eingabe. Sie haben den Typ von „relation“. Es wird für die Aktiondarstellung ein neuer Typ von Kanten erstellt: „relationInAction“. Sie dienen alle dazu, Relationen zwischen Tokens in Aktionen darzustellen. Da sie ausschließlich von der Aktionerkennung benutzt werden, werden sie in einem Typ von Kanten mit verschiedenen Attribute zusammengefasst, statt in verschieden Typen von Kanten. Diese Kanten haben eine Attribute „type“. Es gibt drei möglichen Werte:

1. INSIDE_CHUNK: Kanten, die Tokens innerhalb einem Phrase, die gemeinsam eine Rolle spielen, verbinden
2. NEXT_ACTION: Kanten, die einzelnen Aktionen miteinander verbinden
3. PREDICATE_TO_PARA: Kanten, die Prädikaten mit Akteuren oder Parametern verbinden.

6.2.2. Rollen identifizieren in einzelnen Aktionen

Um Rollen von Token in einer einzelnen Aktionen zu identifizieren werden Informationen von SRL, SENNA und SNLP benötigt.

Der Algorithmus sieht wie im Folgenden aus:

Zuerst werden Rollen anhand SRL identifiziert. Wenn ein Token „A0“ zugewiesen ist, ist es ein „Actor“. Wenn ein Token „V“ zugewiesen ist, ist es ein „Predicate“. Wenn ein Token „AM-LOC“ zugewiesen ist, ist es ein „Where“. Wenn ein Token „AM-TMP“ zugewiesen ist, ist es ein „When“.

Für Tokens, die nach SRL keine Rolle zugeordnet bekommen haben, wird weiter anhand der Eigennamenerkennung von SENNA bearbeitet. Wenn ein Token „S-PER“ zugewiesen ist, ist es ein „Who“. Wenn ein Token „S-LOC“ zugewiesen ist, ist es ein „Where“.

Für Tokens, die nach SENNA auch keine Rolle zugeordnet bekommen haben, wird anhand der Phrasenerkennung und Wortarterkennung von SNLP bearbeitet. Wenn ein Token die Wortart „PRP“ hat, ist es ein „Who“. Wenn ein Token die Phrase „ADVP“ hat, ist es ein „How“. Sonst, wenn ein Token die Phrase „NP“ hat, ist es ein „What“.

Der Pseudocode zu dem obigen Algorithmus ist in Quelltextausschnitt 6.1 zu sehen.

Quelltextausschnitt 6.1: Algorithmus zu Rollenidentifizierung in einzelnen Aktionen

```
//SRLabeler
if SRL = A0 then role = Actor
else if SRL = V then role = Predicate
else if SRL = AM_LOC then role = Where
else if SRL = AM_TMP then role = When
//SENNA
else if Name_Entity = S_PER then role = Who
else if Name_Entity = S_LOC then role = Where
//SNLP
else if POS = PRP then role = Who
else if Chunk = ADVP then role = How
else if Chunk = NP then role = What
```

Dier Erkennung der Aktionen wird mittels einem `RoleIdentifier` (siehe Abschnitt 6.3.1.7) durchgeführt.

6.2.3. Bearbeitung der Aktionen mit „and“

Es wurde im Abschnitt 5.1.5 drei Kategorien der Nutzung von „and“ vorgestellt. Sie sind:

- „And“ verbindet vollständige Aktionen
- „And“ verbindet unvollständige Aktionen mit gemeinsamen Akteur und unterschiedlichen Prädikaten
- „And“ verbindet Parameter

Im ersten Fall verursacht das „and“ keinen zusätzlichen Aufwand. Die Bearbeitung des zweiten und dritten Falls wird im Folgenden detaillierter vorgestellt.

6.2.3.1. „And“ verbindet unvollständige Aktionen mit gemeinsamen Akteur und unterschiedlichen Prädikaten

Wenn in einer Aktion mit Befehlsnummer n kein „Actor“ vorhanden ist, wird der „Actor“ von der Aktion mit Befehlsnummer $n-1$ zugewiesen. Wenn in der Aktion mit Befehlsnummer $n-1$ auch kein „Actor“ vorhanden ist, wird der „Actor“ von der Aktion mit Befehlsnummer $n-2$ den Aktionen mit Befehlsnummern $n-1$ und n zugewiesen, und so weiter. Falls auch in der Aktion mit Befehlsnummer 0 auch kein „Actor“ vorhanden ist, bleiben die Aktionen mit Befehlsnummern 1 bis n ohne „Actor“.

Der Quelltext zu diesem Algorithmus ist in Quelltextausschnitt 6.2 zu sehen.

Quelltextausschnitt 6.2: Zuweisung der Akteur

```
if (this.action.getParameterMap().get(Role.PREDICATE).size() != 0
&& this.action.getParameterMap().get(Role.ACTOR).size() == 0 && this.
    priorAction != null
&& this.priorAction.getParameterMap().get(Role.ACTOR).size() != 0) {
        this.action.getParameterMap().put(Role.ACTOR, this.priorAction.
            getParameterMap().get(Role.ACTOR));
}
```

6.2.3.2. „And“ verbindet Parameter

Wenn in einer Aktion x kein Prädikat vorhanden ist, besteht die Aktion nur aus einem Parameter und benötigt ein dupliziertes Prädikat.

Es wird zuerst geprüft, wo das „and“ auftritt. Der erste Fall ist, „and“ tritt am Anfang dieser Aktion auf. Ein Beispiel ist „Armar brought me an orange juice and an apple juice.“ In diesem Fall wird der Aktion „an apple juice“ (x) mit der vorherigen Aktion „Armar brought me an orange juice“ verknüpft. Der zweite Fall ist, „and“ tritt am Anfang nächsten Aktion auf. Ein Beispiel ist „Armar and Asfour came.“ In diesem Fall wird die Aktion „Armar“ (x) mit der nachkommenden Aktion „Asfour came“ verknüpft. Nun wird die verknüpfte Aktion („Armar brought me an orange juice“ und „Asfour came“) y genannt.

Neue Knoten vom Typ `newPredicate` werden erzeugt, die die gleiche `Value` haben wie die Knoten, die das `Predicate` in Aktion y repräsentieren. Diese Knoten werden als `Predicate` der Aktion x zugewiesen.

Alle anderen Parameter, die in der Aktion y vorhanden aber in x nicht vorhanden sind, werden der Aktion x zugewiesen. „Armar“ und „me“ im Beispiel

Diese Aufgaben werden mittels einem `AndAnalyser` (siehe Abschnitt 6.3.1.8) erfüllt.

6.2.4. Kanten hinzufügen

Kanten werden anhand der Parametern in Aktionen hinzugefügt.

Zuerst werden zwischen Tokens innerhalb einer Phrase, die gemeinsam eine Rolle spielen, Kanten hinzugefügt. Diese Kanten haben „INSIDE_CHUNK“ als `type`. Seien die Tokens $t_1, t_2, t_3, ..., t_n$ in der Phrase. Die Kanten verbinden t_1 (Startknoten) und t_2 (Endknoten), t_2 (Startknoten) mit t_3 (Endknoten), ... , $t_(n-1)$ (Startknoten) mit t_n (Endknoten).

Anschließend werden die Prädikate mit allen ihren Akteuren und Parametern verbunden. Diese Kanten haben „PREDICATE_TO_PARA“ als `type`. Sie zeigen alle von dem ersten Token des Prädikats auf dem ersten Token des Akteurs oder Parameters.

Zum Schluss werden alle Prädikate miteinander verbunden. Diese entspricht die Reihenfolge des Auftritts in der Eingabe. Die Kanten haben „NEXT_ACTION“ als `type`. Sie zeigen vom ersten Token eines Prädikats auf dem ersten Token des nachkommenden Prädikats.

Diese Aufgabe wird mittels einem `ArcBuilder` (siehe Abschnitt 6.3.1.9) erfüllt.

6.3. Implementierungsdetails und UML Diagramme

Der Agent wird in `ActionRecognizer` implementiert und durch `RoleIdentifier`, `AndAnalyser` und `ArcBuilder` unterstützt. Der `ActionRecognizer` enthält einen `ActionGraph`. Der `ActionGraph` speichert alle Informationen zur Eingabe und Aktionen.

In den folgenden Abschnitten werden die Implementierungsdetails vorgestellt.

6.3.1. Komponenten

Die wichtigsten Klassen und Komponenten, die in der Implementierung entstanden sind, werden im Folgenden beschrieben. Die wichtigsten Attributen und Methoden werden auch vorgestellt.

6.3.1.1. ActionRecognizer

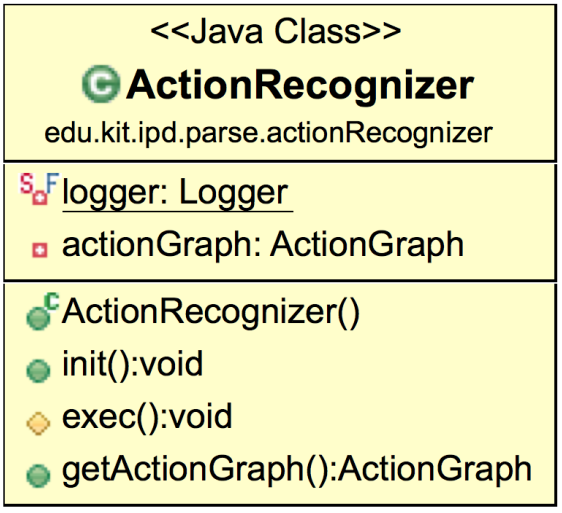

Abbildung 6.3.: ActionRecognizer

`ActionRecognizer` ist der in dieser Bachelorarbeit implementierte Agent. ActionRecognizer bekommt eine initiale Handlungsrepräsentation von natürlichsprachlichen Sätzen als Eingabe, extrahiert die Aktionen daraus, und fügt die Informationen im Graph hinzu.

Attribute

- `actionGraph` Die interne graphbasierte Repräsentation von der Eingabe.

Methoden

+ `init()` Setzt dem Agent ein Id.
+ `exec()` Analysiert die Eingabe, extrahiert die Aktionen und fügt diese in dem Graph hinzu. Wird nur aufgerufen wenn der Graph als Eingabe gesetzt wird.

6.3.1.2. ActionGraph

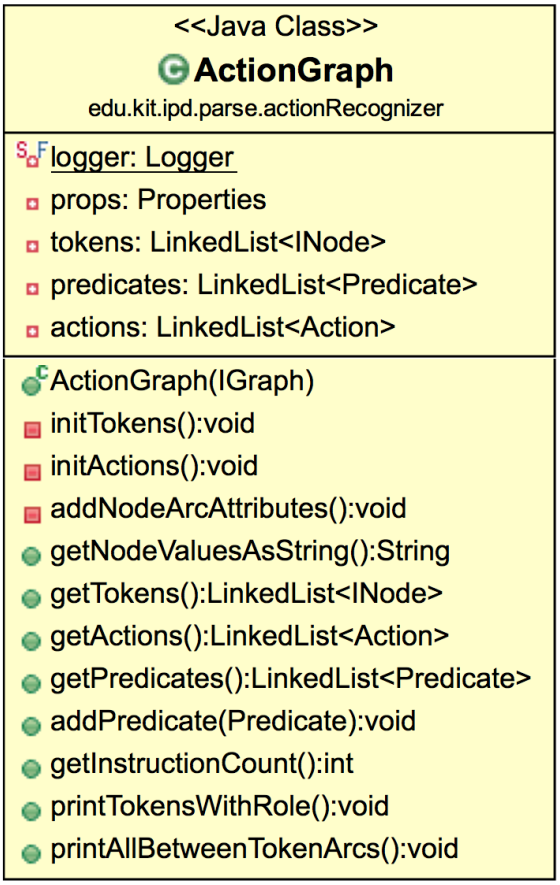

Abbildung 6.4.: ActionGraph

`ActionGraph` wird von ActionRecognizer benutzt. `ActionGraph` ist eine interne graphbasierte Darstellung der natürlichsprachlichen Sätze. Die Informationen zu einem Token werden mit dem Token selbst in einem Knoten gespeichert. Die aus den Sätzen extrahierten Aktionen werden auch im ActionGraph gespeichert. `ActionGraph` implementiert `IGraph`. Es wird erstellt um Informationen zu speichern, die für Aktionen wichtig sind. Eigene Funktionen werden auch im `ActionGraph` implementiert.

Attribute

- `tokens`	Die Liste aller Knoten im Graph. Die Knoten enthalten die Token in der Eingabe. Die Knoten sind sortiert nach ihrem Auftreten in der Eingabe.
- `predicates`	Die Liste aller Prädikate in der Eingabe. Die duplizierte Prädikate sind auch enthalten. Die Prädikaten sind sortiert nach dem Auftreten der zugehörigen Aktionen.
- `actions`	Die Liste aller Aktionen in der Eingabe. Die Aktionen sind sortiert nach ihrem Auftreten in der Eingabe.

Methoden

- `initTokens()`	Initialisiert Token anhand aller unsortierten Knoten und „NEXT“ Kanten (Kanten vom Typ „relation“) im Graph.
- `initActions()`	Gruppiert die Knoten nach der Befehlsnummer und fügt sie in `actions` hinzu.
- `addNodeArcAttributes()`	Fügt den Knoten vom Typ „token“ ein Attribut hinzu. Dieses Attribut hat den Namen „role“ und den Typ „Role“. Die Methode fügt dem Graph einen neuen Typ von Knoten hinzu. Dieser Typ wird „newPredicate“ genannt. Dann wird ein Attribut zu den Knoten vom Typ „newPredicate“ hinzugefügt. Das Attribut hat den Namen „value“ und den Typ String. Zum Schluss wird ein neuer Typ von Kanten hinzugefügt. Dieser Typ wird „relationInAction“ genannt. Ein Attribut wird zu den Kanten von diesem Typ hinzugefügt. Dieses Attribut hat den Namen „type“ und den Typ „BetweenRole“.

6.3.1.3. Action

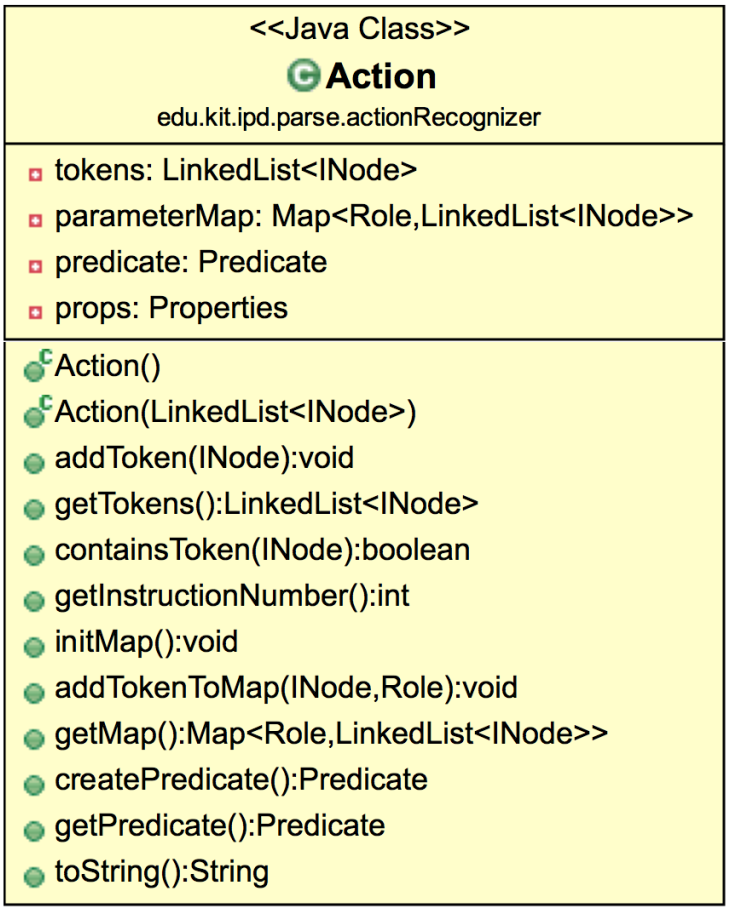

Abbildung 6.5.: Action

Die Klasse repräsentiert Aktionen. Eine Aktion enthält Token. Eine Aktion hat genau ein Prädikat. Eine Aktion kann auch einen Akteur und Parameter enthalten. Die Parameter müssen nicht als Token in der Aktion enthalten sein. Sie können von `AndAnalyser` zu der Aktion zugewiesen sein.

Attribute

- `tokens`	Die Liste aller Knoten in der Aktion.
- `parameterMap`	Bildet Akteur, Prädikat und andere Parameter auf Listen von Knoten ab.

Methoden

+ `createPredicate()`	Erzeugt ein predicate mittels Informationen in parameterMap. Kann nur dann aufgerufen werden, wenn die parameterMap initialisiert ist.

6.3.1.4. Predicate

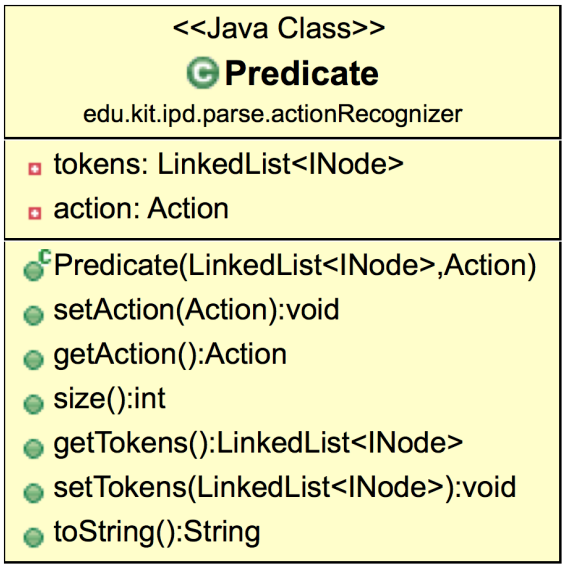

Abbildung 6.6.: Predicate

Die Klasse `Predicate` repräsentiert Prädikate. Ein Prädikat enthält Tokens. Ein Prädikat gehört zu genau einer Aktion.

Attribute

- `tokens` Die Liste aller Knoten in dem Prädikat.
- `action` Die zu dem Prädikat gehörige Aktion.

6.3.1.5. Role

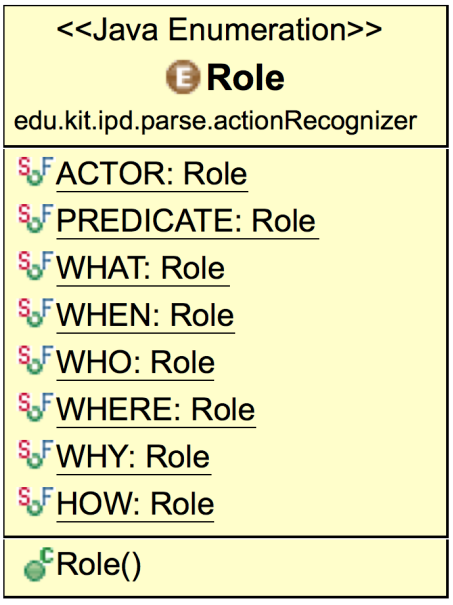

Abbildung 6.7.: Role

`Role` ist ein Enum. Role definiert die mögliche Rollen, die ein Token spielen kann. Sie sind: ACTOR, PREDICATE, WHAT, WHEN, WHO, WHERE, WHY und HOW.

6.3.1.6. BetweenRole

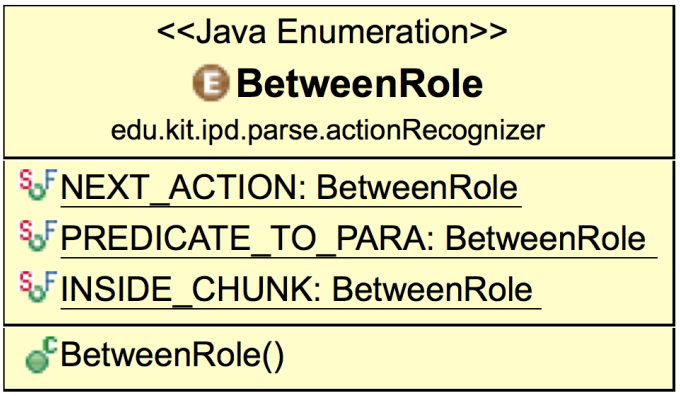

Abbildung 6.8.: BetweenRole

`BetweenRole` ist ein Enum. BetweenRole definiert die mögliche Typen, die eine Kante vom Typ „relationInAction“ haben kann. Sie sind: NEXT_ACTION, PREDICATE_TO_PARA und INSIDE_CHUNK

6.3.1.7. RoleIdentifier

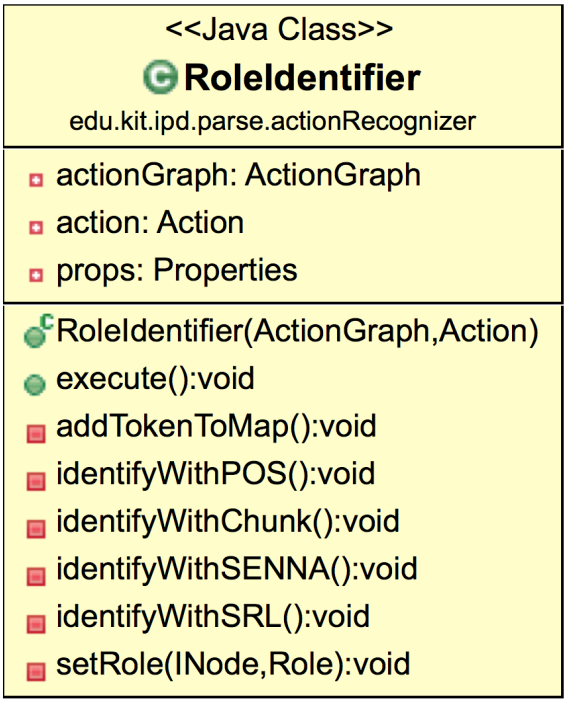

Abbildung 6.9.: RoleIdentifier

`RoleIdentifier` bestimmt die Rollen der Tokens in einer Aktion anhand dem Algorithmus im Abschnitt 6.2.2. Die Rollen werden als Attribute zu den Knoten hinzugefügt und die `parameterMap` der Aktion wird initialisiert.

Attribute

- `action`	Die Aktion, in der die Tokens analysiert werden.

Methoden

+ `execute()`	Analysiert die Tokens und bestimmt die Rollen. Fügt die Rollen als Attribute zu den Knoten hinzu und initialisiert die `parameterMap` der Aktion.
- `addTokenToMap()`	Nachdem die Rollen aller Tokens in der Aktion bestimmt werden, werden die Knoten zu der `parameterMap` von der Aktion hinzugefügt.
- `identifyWithPOS()`	Bestimmt die Rollen anhand der Wortarterkennung.
- `identifyWithChunk()`	Bestimmt die Rollen anhand der phrasalen Markierung.
- `identifyWithSENNA()`	Bestimmt die Rollen anhand Eigennameerkennung von SENNA.
- `identifyWithSRL()`	Bestimmt die Rollen anhand des SRLabelers.
- `setRole()`	Setzt die Rolle von einem Token wenn diese null ist. So wird vermieden, dass zwei Methoden einem Token unterschiedlichen Rollen zuordnet.

6.3.1.8. AndAnalyser

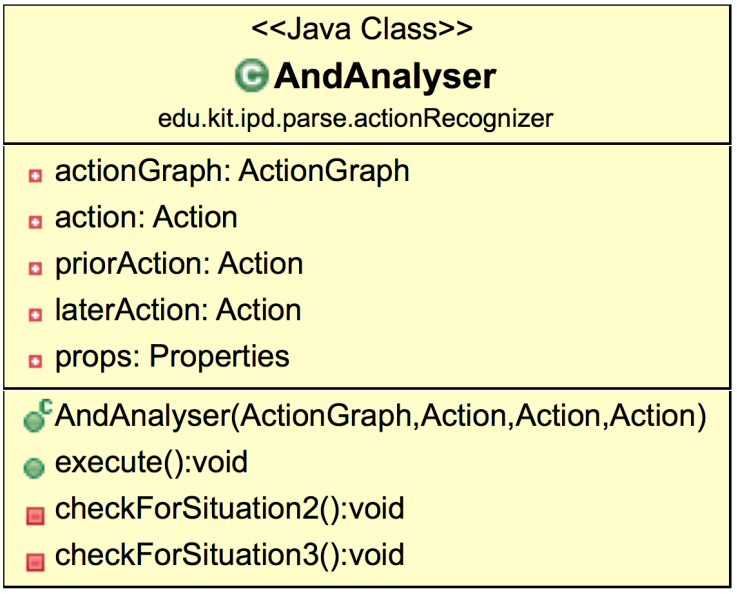

Abbildung 6.10.: AndAnalyser

`AndAnalyser` untersucht eine Aktion dahingehend, ob sie durch ein „and“ mit anderen Aktionen verbunden ist, fügt Parameter zu der Aktion hinzu wenn nötig.

Attribute

- `action`	Die aktuelle Aktion.
- `priorAction`	Die vorherige Aktion.
- `laterAction`	Die nachfolgende Aktion.

Methoden

+ `execute()`	Überprüft, ob Fälle wie in Abschnitt 5.2.2 und Abschnitt 5.2.2 vorkommen. Wenn dann werden Pramater zu der Aktion hinzugefügt.
- `checkForSituation2()`	Wenn der Fall aus Abschnitt 5.2.2 auftritt, wird der Aktion der Akteur der vorherigen Aktion zugewiesen.
- `checkForSituation3()`	Wenn der Fall aus Abschnitt 5.2.2 auftritt, wird ein Prädikat dupliziert und der Aktion die Parameter der verbundenen Aktion zugewiesen.

6.3.1.9. ArcBuilder

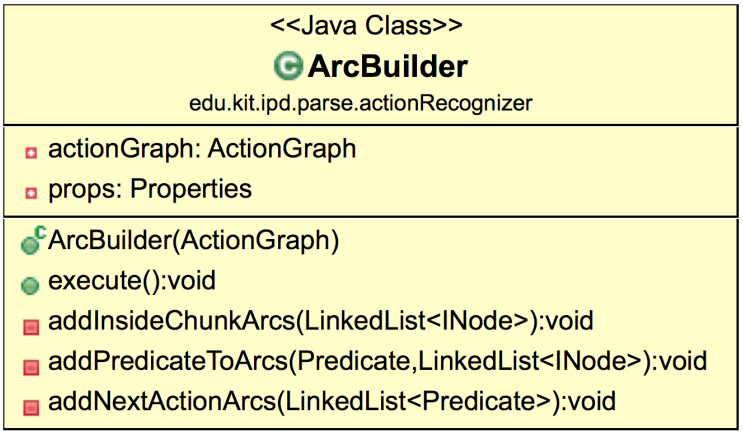

Abbildung 6.11.: ArcBuilder

`ArcBuilder` fügt dem Graph Kanten hinzu nachdem die Rollen von Tokens erkannt und die Parameter von Aktionen bestimmt sind.

Methoden

+ `execute()`	Fügt dem Graph Kanten hinzu.
- `addInsideChunkArcs()`	Fügt dem Graph INSIDE_CHUNK Kanten hinzu.
- `addPredicateToArcs()`	Fügt dem Graph PREDICATE_TO_PARA Kanten hinzu.
- `addNextActionArcs()`	Fügt dem Graph NEXT_ACTION Kanten hinzu.

6.3.2. Ablauf

Der Ablauf des Agenten ist in der Abbildung 6.12 zu sehen.

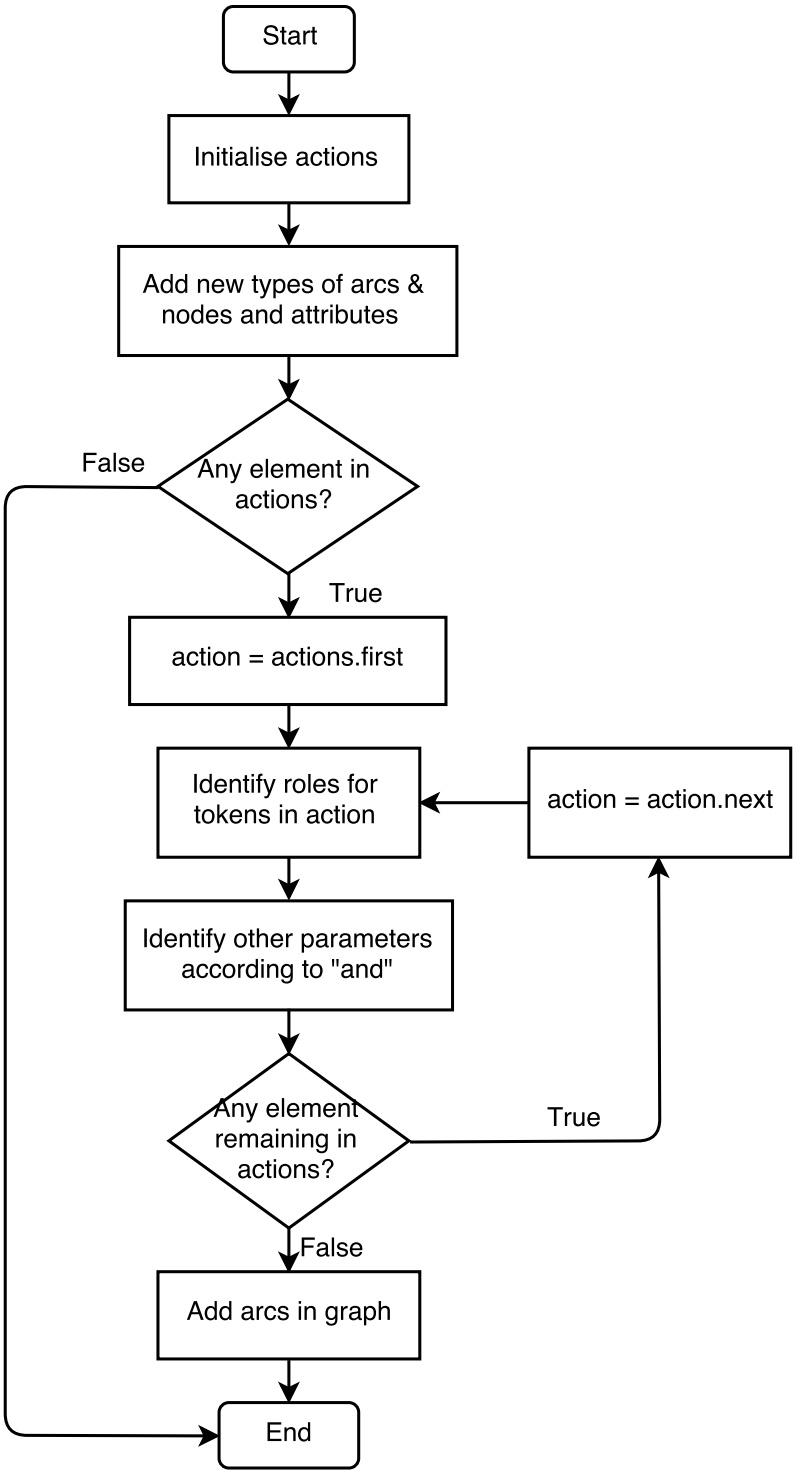

Abbildung 6.12.: Ablauf

Zuerst gruppiert der ActionGraph die Tokens in separate Aktionen. Die Aktionen werden in einer Instanz der Klasse `ActionGraph` gespeichert. Anschließend werden die benötigten neuen Knoten- und Kantentypen und neue Attribute zu bestehenden Typen zum Graph hinzugefügt.

Dann wird für jede Aktion Folgendes ausgeführt: Der `RoleIdentifier` bestimmt die Rollen für allen Token in der Aktion. Die identifizierte Rollen der Token werden in der `parame-`

terMap der jeweiligen Aktion gespeichert. Dann analysiert der AndAnalyser, ob weitere Knoten als Parameter der Aktion hinzugefügt werden sollen. Nachdem alle Parameter in der Aktion bestimmt sind, wird das Prädikat von der Aktion im ActionGraph gespeichert.

Zum Schluss fügt ArcBuilder dem Graph Kanten anhand der parameterMap von jeder Aktion hinzu.

6.4. Verwendung des Agenten

Im Folgenden wird vorgestellt, wie der Agent unter Eingabe natürlichsprachlichen Sätzen benutzt werden soll. In dieser Weise kann der Agent ausprobiert und getestet werden.

6.4.1. Eingabe

Um in den Agent natürlichsprachliche Sätze einzugeben, wird ein PrePipelineData-Objekt erstellt, das die Sätze als Transkription enthält. Dann werden Tokenizer, ShallowNLP und SRLabeler aufgerufen. Sie fügen Informationen zur PrePipelineData hinzu. Zum Schluss wird das PrePipelineData-Objekt an dem ActionRecognizer weitergegeben. ActionRecognizer kann nun die Aktionen extrahieren.

Der Quelltext zur Eingabe wird im Quelltextausschnitt 6.3 aufgelistet.

Quelltextausschnitt 6.3: Verwendung des Agenten - Eingabe

```
IGraph graph;
PrePipelineData ppd = new PrePipelineData();
String input = "Armar brought me an orange juice and an apple juice.";
ppd.setTranscription(input);

Tokenizer tokenizer = new Tokenizer();
tokenizer.init();
ShallowNLP snlp = new ShallowNLP();
snlp.init();
SRLabeler srLabeler = new SRLabeler();
srLabeler.init();

try {
        tokenizer.exec(ppd);
        snlp.exec(ppd);
        srLabeler.exec(ppd);
} catch (PipelineStageException e1) {
        e1.printStackTrace();
}

try {
        graph = ppd.getGraph();
} catch (MissingDataException e) {
        e.printStackTrace();
}

ActionRecognizer recognizer = new ActionRecognizer();
recognizer.setGraph(graph.clone());
recognizer.init();
recognizer.exec();
```

6.4.2. Ausgabe

Um das Ergebnis des Agenten anzuzeigen gibt es zwei Methoden.

6.4.2.1. ActionGraph

In `ActionGraph` gibt es eine `toString()`-Methode, die eine Repräsentation des Graphen erzeugt.

Die Ausgabe für das Beispiel „Armar brought me an orange juice and an apple juice.“ ist in der Abbildung 6.13 zu sehen:

```
Node:
Armar(ACTOR) brought(PREDICATE) me(WHO) an(WHAT) orange(WHAT) juice(WHAT)
and(null) an(WHAT) apple(WHAT) juice(WHAT) brought'(null)

Arc:
brought--PREDICATE_TO_PARA-->Armar
an--INSIDE_CHUNK-->orange
orange--INSIDE_CHUNK-->juice
brought--PREDICATE_TO_PARA-->an
brought--PREDICATE_TO_PARA-->me
brought'--PREDICATE_TO_PARA-->Armar
an--INSIDE_CHUNK-->apple
apple--INSIDE_CHUNK-->juice
brought'--PREDICATE_TO_PARA-->an
brought'--PREDICATE_TO_PARA-->me
brought--NEXT_ACTION-->brought'
```

Abbildung 6.13.: Ausgabe mit ActionGraph

6.4.2.2. SimpleActionGraph

Die Klasse `SimpleActionGraph` enthält nur Informationen, die für Aktionen wichtig sind. Diese sind: Token, die zugehörigen Rollen, Kanten und die zugehörigen Typen. Der Graph wird vollständig im Kapitel 7 beschrieben.

Der `ActionGraph` kann in einem `SimpleActionGraph` umgewandelt werden. Dann kann der `SimpleActionGraph` wie im Quelltextausschnitt 6.4 ausgegeben werden:

Quelltextausschnitt 6.4: Ausgabe mit `SimpleActionGraph`

```
SimpleActionGraph simpleActionGraph = new SimpleActionGraph(recognizer.
    getActionGraph());
System.out.println(simpleActionGraph);
```

Die Ausgabe für das Beispiel „Armar brought me an orange juice and an apple juice.“ ist in der Abbildung 6.14 zu sehen:

```
Nodes:
0.Armar(ACTOR) 1.brought(PREDICATE) 2.me(WHO) 3.an(WHAT) 4.orange(WHAT) 5.juice(
WHAT) 6.and(null) 7.an(WHAT) 8.apple(WHAT) 9.juice(WHAT) 10.brought'(null)

Arcs:
1.brought--PREDICATE_TO_PARA-->0.Armar
3.an--INSIDE_CHUNK-->4.orange
4.orange--INSIDE_CHUNK-->5.juice
1.brought--PREDICATE_TO_PARA-->3.an
1.brought--PREDICATE_TO_PARA-->2.me
10.brought'--PREDICATE_TO_PARA-->0.Armar
7.an--INSIDE_CHUNK-->8.apple
8.apple--INSIDE_CHUNK-->9.juice
10.brought'--PREDICATE_TO_PARA-->7.an
10.brought'--PREDICATE_TO_PARA-->2.me
1.brought--NEXT_ACTION-->10.brought'
```

Abbildung 6.14.: Ausgabe mit SimpleActionGraph

In der Ausgabe werden die Knoten durchnummeriert. Somit wird disambiguiert, auf welchen Knoten eine Kante zeigt, wenn zwei Knoten mit gleichem Wert existieren.

7. Evaluation

In diesem Kapitel wird der erstellte Agent evaluiert. Dabei wird ein Evaluationskorpus erstellt, ein Vergleichsvorgang entworfen und ein Evaluation-Werkzeug gebaut. Schließlich werden die Evaluationsergebnisse gezeigt und diskutiert.

7.1. Evaluationskorpus

Ein Korpus von Beispielen wird benötigt, um den erstellten Agent zu evaluieren. Die Eingabetexte werden zuerst vom Tokenizer, SNLP und SRLabeler bearbeitet, und dann an den Agenten weitergegeben.

Es werden zwei Gruppen von Eingaben betrachtet: Die erste Gruppe sind zufällige Beispiele, die extra für den Agent erstellt wurden. Die zweite Gruppe sind Transkriptionen, die im Rahmen der Arbeit von Günes [Gün15] entstanden sind.

7.1.1. Eigene Beispiele

Die eigene Beispiele sind im Laufe der Erstellung des Agenten entstanden, um die Implementierung zu testen. Sie werden danach gesammelt. Dabei wird hauptsächlich getestet, ob Eingaben, die das Wort „and“ enthalten, richtig bearbeitet werden.

Ein Ausschnitt der Liste ist unten zu sehen:

Eigene Beispiele

1. Get me an orange juice
2. Armar brought me an orange juice and brought Sebastian an apple juice
3. Armar likes Japan and Canada

7.1.2. Transkriptionen

226 Transkriptionen sind im Rahmen der Arbeit von Günes entstanden. 113 Transkriptionen davon sind die Ausgaben eines Spracherkenners (siehe Kapitel 3) und fehlerhaft.

Einige Beispiele sind in der Liste unten zu sehen:

Transkriptionen 1

- okay go to the table crap popcorn come to me give me the popcorn witches in your hand
- mama can you please bring medium popcorn bag
- Im at please put the green cap in the dishwasher and I forgot to close it afterwards

Die anderen 113 davon sind von Hand angefertigten Transkriptionen.

Einige Beispiele sind in der Liste unten zu sehen:

Transkriptionen 2

- okay go to the table grab popcorn come to me give me the popcorn which is in your hand
- Armar can you please bring me the popcorn bag
- Armar please put the green cup into the dishwasher and dont forget to close it afterwards

7.2. Vergleichsvorgang

Jedes Beispiel im Evaluationskorpus wird vom Agent bearbeitet. Das Ergebnis wird mit der zugehörigen Musterlösung (gewünschten Ergebnis) verglichen. Es wird geprüft, ob jedem Token die richtige Rolle zugeordnet wird und ob alle Kanten richtig hinzugefügt werden.

7.2.1. Genauigkeit - Rollen

Für jedes Token wird geprüft, ob die Rolle richtig zugeordnet wird. Die Genauigkeit (engl. accuracy) wird berechnet durch die Anzahl der richtig zugeordneten Tokens geteilt durch die Anzahl der Tokens.

$$Genauigkeit = Anzahl\ der\ richtig\ identifizierten\ Token \div AnzahlderToken \quad (7.1)$$

Die Berechnung wird anhand dem folgenden Beispiel genauer erklärt.

Beispiel:

Eingabe: Armar brought me an orange juice.

Musterlösung - Rollen

Armar(Actor) brought(Predicate) me(Who) an(What) orange(What) juice(What)

Ergebnis vom Agent - Rollen

Armar(Actor) brought(Predicate) me(Who) an(What) orange(What) juice(How)

In dem obigen Beispiel sind sechs Token enthalten, fünf davon wurden die richtigen Rollen zugeordnet. Dem Token „juice“ wurde die falsche Rolle *How* statt der gewünschte Rolle *What* zugeordnet. Daher ist die Genauigkeit der Rollenzuteilung für das obige Beispiel

$$Genauigkeit = 5 \div 6 = 0.8333$$

.

7.2.2. Präzision und Ausbeute - Kanten

Für alle Kanten wird überprüft, ob die gewünschten Kanten hinzugefügt werden und ob keine überflüssigen Kanten hinzugefügt wurden. Dafür werden die Maße Präzision und Ausbeute (engl. precision and recall) benutzt.

Vier Begriffe sind für Präzision und Ausbeute wichtig:

- richtig-Positive: Richtig-Positive gibt den Anteil der korrekt als positiv klassifizierten Elemente. Eine Kante ist richtig positiv, wenn sie gewünscht ist und hinzugefügt wird.
- richtig-Negative: Richtig-Negative gibt den Anteil der korrekt als negativ klassifizierten Elemente. Eine Kante ist richtig negativ, wenn sie ungewünsch ist und nicht hinzugefügt wird.
- falsch-Positive: Falsch-Positive gibt den Anteil der als positiv klassifizierten Elemente, die eigentlich negativ sind. Farsch-Positive wird auch Type I Fehler genannt. Eine Kante ist falsch positiv, wenn sie nicht gewünscht ist, aber hinzugefügt wird.
- falsch-Negative: Falsch-Negative gibt den Anteil der als negativ klassifizierten Elemente, die eigentlich positiv sind. Falsch-Negative wird auch Type II Fehler genannt. Eine Kante ist falsch negativ, wenn sie gewünscht ist, aber nicht hinzugefügt wird.

7.2.2.1. Präzision

Präzision (engl. precision) ist ein Maß dafür, wie präzise ein Klassifikator klassifiziert. Für die Kanten wird die Genauigkeit berechnet durch die Anzahl der richtig hinzugefügten Kanten geteilt durch die Anzahl der hinzugefügten Kanten:

$$Praezision = richtig\ Positive \div (richtig\ Positive + falsch\ Positive) \tag{7.2}$$

7.2.2.2. Ausbeute

Ausbeute (engl. recall) ist ein Maß dafür, wie umfangreich ein Klassifikator klassifiziert. Für die Kanten wird die Ausbeute berechnet durch die Anzahl der richtig hinzugefügten Kanten geteilt durch die Anzahl der gewünschten Kanten.

$$Ausbeute = richtig\ Positive \div (richtig\ Positive + falsch\ Negative) \tag{7.3}$$

Eine hinzugefügte Kante wird als richtig positiv identifiziert, wenn sie mit einer Kante aus der Musterlösung übereinstimmt hinsichtlich dem Startknoten, dem Endknoten und dem Kantentyp.

Die Berechnung wird anhand des folgenden Beispiels genauer erklärt.

Beispiel:
Eingabe: Armar helped me yesterday.
Musterlösung - Kanten

helped –(PREDICATE_TO_PARA)–> Armar

helped –(PREDICATE_TO_PARA)–> me

helped –(PREDICATE_TO_PARA)–> yesterday

Ergebnis des Agenten - Kanten

helped –(PREDICATE_TO_PARA)–> Armar

helped –(PREDICATE_TO_PARA)–> me

Armar –(INSIDE_CHUNK)–> me

me –(INSIDE_CHUNK)–> yesterday

Im obigen Beispiel hat der Agent zwei von den drei gewünschten Kanten hinzugefügt. Der Agent hat auch zusätzlich zwei falsche Kanten hinzugefügt. Daher ist die Präzision

$$Praezision = 2 \div 4 = 0.5$$

Die Ausbeute ist

$$Ausbeute = 2 \div 3 = 0.6667$$

7.3. Evaluation-Werkzeug

Ein Evaluation-Werkzeug wurde entwickelt um den Vergleich zu automatisieren. Das Evaluation-Werkzeug nimmt eine Liste von Beispielen mit Musterlösungen entgegen, gibt die Beispiele an dem Agent weiter und vergleicht die Ergebnisse des Agenten mit den Musterlösungen. Nach dem Vergleich berechnet das Evaluation-Werkzeug die durchschnittliche Genauigkeit der Rollen und Präzision und Ausbeute der Kanten.

7.3.1. Eingabe - Beispiele und Musterlösungen

Beispiele und die Musterlösungen werden in XML-Dateien abgelegt (siehe Quelltextausschnitt Quelltextausschnitt 7.1).

Quelltextausschnitt 7.1: Beispiele und Musterlösungen

```
<example name="1">Get me an orange juice</example>
<role name="1">0.Get(Predicate) 1.me(Who) 2.an(What) 3.orange(What) 4.
    juice(What)</role>
<arc name="1">0-1(pp) 0-2(pp) 2-3(ic) 3-4(ic)</arc>>

<example name="2">Armar bring me an orange juice</example>
<role name="2">0.Armar(Actor) 1.bring(Predicate) 2.me(Who) 3.an(What)
    4.orange(What) 5.juice(What)</role>
<arc name="2">1-0(pp) 1-2(pp) 1-3(pp) 3-4(ic) 4-5(ic)</arc>
```

Zwischen den „example“ Tags sind die Beispiele zu sehen. Jedem Beispiel wird ein Name zugeordnet.

Zwischen den „role“ Tags sind die Musterlösungen von Rollen zu sehen. Die Tokens werden mit Index durchnummeriert. Nach jedem Token steht die zugeordnete Rolle in Klammer. Die Tokens werden voneinander mit Leerstellen getrennt.

Zwischen den „arc“ Tags sind die Musterlösungen von Kanten zu sehen. Jede Kante wird durch ein Startknoten, ein Endknoten und ein Kantentyp dargestellt. Der Startknoten und der Endknoten werden mit Index präsentiert und mit einem Strich verbunden. Der Kantentyp steht in Klammer. „pp“ steht für *PREDICATE_TO_PARA*, „ic“ steht für *INSIDE_CHUNK* und „na“ steht für *NEXT_ACTION*. Die Kanten werden voneinander mit Leerstellen getrennt.

7.3.2. Komponenten

Für das Evaluation-Werkzeug wurden SimpleActionGraph , SimpleActionNode und SimpleAcitonArc erstellt. Sie enthalten nur die notwendigen Informationen zur Repräsentation von Aktionen. Ein SimpleGraphComparer vergleicht zwei SimpleActionGraph, die jeweils eine Musterlösung und ein Ergebnis vom Agent repräsentieren.

7.4. Evaluationsergebnisse

Die durchschnittliche Evaluationsergebnisse sind in der Tabelle 7.1 zu sehen.

	Rollen-Genauigkeit	Kanten-Präzision	Kanten-Ausbeute
Eigene Beispiele	100%	100%	100%
Transkriptionen Sprache	57.83%	48.25%	50.26%
Transkriptionen händisch	78.25%	73.17%	77.87%

Tabelle 7.1.: Evaluationsergebnisse

Die eigene Beispiele können 100% richtig bearbeitet werden. Diese ist auch zu erwarten, weil die Beispiele dazu dienen, die Implementierung zu testen.

Die Ergebnisse für die erste Gruppe von Transkriptionen sind nicht optimal. Es liegt daran, dass die Transkriptionen Fehler enthalten. SNLP und SRLabeler können diese nicht richtig bearbeiten und fügen falsche Informationen hinzu. Da die Aktionerkennung auf die Informationen von SNLP und SRLabeler basiert, ist das Ergebnis von dem Agent auch fehlerhaft.

Die Ergebnisse für die zweite Gruppe von Transkriptionen sind besser im Vergleich zu der ersten Gruppe. Es liegt daran, dass die Transkriptionen von Hand gefertigt wurden und somit richtiger sind. Daher sind die Informationen von SNLP und SRLabeler richtiger. Die Ergebnisse des Agenten ist deswegen besser.

7.4.1. Fehlerquelle - SNLP und SRLabeler

Folgende Fälle können nicht von SNLP und SRLabeler richtig bearbeitet werden, daher werden Fehler in der Aktionerkennung verursacht :

- „Hello“, „Hallo“ und „Hi“ sollen keine Rolle zugewiesen sein, weil sie keine Information für Aktionen anbieten. Sie werden allerdings von SNLP das falsche POS-Tag Nomen (NNP) statt Interjektion (UH) zugewiesen. Daher werden sie vom Agent die falsche Rolle „What“ zugewiesen.
- Der Satz „Bring me the popcorn He stands in the kitchen“ soll in zwei Aktionen aufgeteil werden: „Bring me the popcorn“ und „He stands in the kitchen“. SNLP ordnet allerdings das Token „He“ zu der ersten Aktion und weist ihm die semantische Rolle A0. Daher wird He vom Agent die richtige Rolle „Actor“ zugewiesen, jedoch in der falschen Aktion.

7.4.2. Fehlerquelle - Agent

Folgende Fälle können nicht von dem Agent richtig bearbeitet werden:

- In Fällen wie „Bring me an orange juice from the fridge“ oder „Put the popcorn in the kitchen“ werden „the fridge“ und „the kitchen“ die falsche Rolle „What“ statt der richtigen Rolle „Where“ zugeordnet. Es liegt daran, dass `SRLabeler` solche Parameter nicht als ein Ort identifiziert und es auch kein Eigename ist. Da es eine Nominalphrase ist, wird es von dem Algorithmus, der in die Quelltextausschnitt 6.1 vorgestellt wurde, die Rolle „What“ zugewiesen.

  Dieser Fehler ist nicht unerwartet, weil der Algorithmus keinen Ort erkennen kann, wenn der `SRLabeler` ihn nicht als Ort kennzeichnet. Um diese Fehler zu korrigieren, kann eine Datenbank aufgestellt werden, die unter einem Zielsystem bestimmten Nominalphrasen als Orte vordefinieren.

- Aktionen mit Nebensätze können nicht richtig bearbeitet. Die Nebensätze sollen vernachlässigt werden, weil darin keine Aktionen enthalten sind. Der Agent extrahiert trotzdem mögliche Informationen. Zum Beispiel wird in „Bring me the popcorn which is in the kitchen“ der Teil „Bring me the popcorn“ richtig bearbeitet, aber „which“, das zweite „the“ und „kitchen“ werden überflüssig die Rolle „What“ zugewiesen.

  Nebensätze wurden übersehen in dem Entwurf. Dieser Fehler kann behoben werden, indem der Agent ab bestimmten Konjunktionen wie „which“ und „what“ eine Aktion ignoriert.

- In einer Frage wie „Armar, could you bring me...“ wird Armar vom Agent die Rolle „Who“ zugewiesen. In diesem Satz ist entweder „Armar“ oder „you“ der Akteur. Sie zeigen beide auf die gleiche Person. Daher soll Armar entweder die Rolle „Actor“ zugewiesen sein, wenn „Armar“ als Akteur dient, oder keine Rolle zugewiesen sein, wenn „you“ als Akteur dient.

  Fragen zu bearbeiten gehört eigentlich zu den Wunschkriterien dieser Bachelorarbeit. Solchen Fragen wie „Could you ...“ oder „Would you please ...“ sind allerdings einfache Fragen, die leicht zu imperativen Sätzen umgewandelt werden können. Diese wurde im Entwurf übersehen.

- Sätze, die keine Aktionen enthalten, werden auch vom Agent als Aktionen behandelt. Zum Beispiel in „Thank you very much“ wird „Thank“ als „Predicate“, „you“ als „Who“ und „very much“ als „how“ erkannt.

  Es ist gewünscht, dass Sätze, die keine Aktionen enthalten, nicht bearbeitet werden und dass alle Tokens keine Rolle zugewiesen werden. Die Unterscheidung zwischen Aktionen und Beschreibungen ist allerdings kompliziert und muss nicht in dieser Bachelorarbeit gelöst werden. Im Abschnitt 8 wird es genauer diskutiert.

Falsche Rollenzuweisung verursacht auch falsche Kanten. Nimmt man „Bring me an orange juice from the fridge“ als Beispiel: Da „the fridge“ wie „an orange juice“ die Rolle „What“ zugewiesen ist, wird angenommen, dass sie zu derselben Phrase gehören und einen gemeinsamen Parameter bilden. Daher wird zwischen „ juice“ und „the“ eine falsche „INSIDE_CHUNK“ Kante hinzugefügt und die gewünschte Kante zwischen „bring“ und „the“ vom Typ „PREDICATE_TO_PARA“ fehlt.

Für die handgefertigten Transkriptionen wird eine Genauigkeit bei Rollenidentifizierung von 78.25%, eine Präzision bei Kanten von 73.17% und eine Ausbeute bei Kanten von 77.87% erreicht. Dieses Ergebnis ist zufriedenstellend, weil die meisten Fehler kommen wiederholt von dem Wort „Hello“ und den falschen Rollenzuweisung von Phrasen wie „the fridge“.

Um das Ergebnis in der Zukunft zu verbessern müssen die oben genannten Fehler, sowohl von SNLP und SRLabeler, als auch vom Agent, korrigiert werden. Um die möglichen Fehler, die in der Evaluation nicht entstanden sind, vorzubeugen, sollen die Herausforderung im Kapitel 8 auch erfüllt werden.

8. Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde einen Agent für das Projekt PARSE erstellt. Der Agent nimmt die von SNLP erzeugte initiale Handlungsrepräsentation entgegen und extrahieren einzelne Aktionen daraus. Eine Aktion besteht aus Akteur, Prädikat und Parameter. Der Akteur nimmt die Rolle des Subjektes ein. Das Prädikat beschreibt die eigentliche Handlung der Aktion. Der Parameter nimmt die Rolle des direkten oder indirekten Objektes ein und bietet zusätzliche Informationen an. Es gibt sechs Arten von Parametern: *Was*, *Wer*, *Wo*, *Wann*, *Wie*, *Warum*. Jeder Parameter beantwortet die jeweilige Frage. Der Agent modifiziert die initiale Handlungsrepräsentation um Aktionen im Graph darzustellen. Nach der Bearbeitung ist anhand des Graphs erkennbar, welche Aktionen in der Eingabe enthalten sind.

Die zwei wichtigsten Fragen, die in dieser Bachelorarbeit gestellt wurden, sind:

- Wie sollen Aktionen dargestellt werden?
- Wie sollen Aktionen erkannt werden, damit sie auf die Darstellung abgebildet werden können?

Um die erste Frage zu beantworten wurde eine graph-basierte Darstellung für Aktionen entworfen. Jedes Token wird in einem Knoten dargestellt und eine Rolle zugeordnet. Eine Rolle kann die Werte *Akteur*, *Prädikat*, *Was*, *Wer*, *Wo*, *Wann*, *Wie*, *Warum* annehmen. Die Relationen zwischen einzelnen Aktionen, Akteure, Prädikate, Parameter und Token innerhalb einer Phrase werden mittels Kanten dargestellt. Es gibt drei Arten von Kanten: Kanten, die Token innerhalb einer Phrase, die gemeinsam eine Rolle spielen verbinden, Kanten, die einzelnen Aktionen miteinander verbinden und Kanten, die Prädikate mit ihren Akteuren oder Parametern verbinden.

Eine Herausforderung in der Aktiondarstellung stellt die Konjunktion „and“ dar. „And“ kann unvollständige Aktionen oder Parameter mit einer vollständigen Aktion verbinden. In solchen Fällen werden Token, die nicht in einer Aktion enthalten sind, dieser Aktion als Parameter zugewiesen.

Um die zweite Frage zu beantworten werden zwei Schritte benötigt. Zuerst wird in einzelnen Aktionen den Token Rollen zugewiesen. Die Rollenerkennung erfolgt über semantische Rollen, Eigennamen, Phrasen und Wortarten. Danach werden Parametern Aktionen zugewiesen, die durch die Konjunktion „and“ verbunden sind.

Für die Transkriptionen, die im Rahmen der Arbeit von Günes entstanden sind, wird eine Genauigkeit bei Rollenidentifizierung von 78.25%, eine Präzision bei Kanten von 73.17%

und eine Ausbeute bei Kanten von 77.87% erreicht. Um dieses Ergebnis zu verbessern sollen in der Zukunft folgende Herausforderungen erfüllt werden.

Andere Nutzung von „and“

Die Konjunktion „and“ verbindet nicht immer zwei Aktionen. [Dic16] Beispiele dafür sind:

- Armar brought me a mac and cheese.
- Armar brought me a ham and cheese sandwich.

„Mac and cheese“ ist der Name eines Gerichts. „Ham and cheese“ bezeichnet als eine Phrase die Art des Sandwichs. In diesen Fällen sollen die Phrasen nicht getrennt und die Existenz des „and“ soll ignoriert werden.

Andere Fälle, die ausgeschlossen werden sollen, sind:

- „2 and 2 are 4.“ In diesem Fall bedeutet das „and“ addieren.
- „He coughed and coughed.“ In diesem Fall bezeichnet das „and“ die wiederholende Aktion.
- „There are bargains and bargains, so watch out.“ In diesem Fall impliziert das „and“ zwei verschiedene Begriffe, die den gleichen Namen haben.
- „And then it happened.“ In diesem Fall wird das „and“ benutzt, um einen neuen Satz einzuführen.
- „He accepted the job, no ands or buts about it.“ In diesem Fall wird das „and“ als Nomen benutzt und bedeutet Details oder zusätzliche Bedingungen.
- „And so on“ und „and so forth“ sind Idiome, und sollten daher ignoriert werden.

„Quantoren“

„All“ und „every“ stellt ein ähnliches Problem wie „and“ dar. Im Zielsystem ARMARIII werden Befehle hintereinander an dem Roboter gegeben. Aus dem Befehl „Get all the books from the table“ sollten mehrere Aktionen extrahiert werden. Es wird für alle Bücher (book 1, book 2, ...), die auf dem Tisch stehen, jeweils eine Aktion extrahiert: „Get book 1 from the table.“, „Get book 2 from the table.“, usw. Dabei sind allerdings die Kenntnisse über die vorherigen Eingaben erforderlich.

Unterscheidung von Aktionen und Beschreibungen

Ein imperativer Satz enthält immer eine Aktion. Ein deklarativer Satz dahingegen enthält entweder eine Aktion oder eine Zustandsbeschreibung. „Armar can talk“ ist zum Beispiel eine Beschreibung und „Armar is talking“ ist eine Aktion.

Es kann nicht lediglich anhand der Zeitform unterschieden werden, ob ein Satz eine Aktion oder Beschreibung enthält. In den meisten Fällen wird „simple present tense“ benutzt, um Gewohnheit, generelle Wahrheit, konstante Situation und Emotionen zu äußern, wie „I smoke“ (Gewohnheit), „I work in London“ (konstante Situation) und „London is a large city“ (generelle Wahrheit). Sie gehören alle zu Beschreibungen. „Simlp present tense“ wird allerdings auch benutzt, um eine fixe Aktion in der Zukunft zu äußern, wie „Your exam starts at 09.00“.

Es kann auch nicht anhand Art des Verbs kategorisiert werden. „He works there“ ist eine Beschreibung und „He is working“ ist eine Aktion. „The pilot flies a plane“ ist eine Beschreibung und „The pilot is flying a plane“ ist eine Aktion.

Fragen

In manchen Fragen sind Aktionen enthalten. Beispielsweise enthält „Could you bring me the popcorn from the kitchen table please?“ eine Aktion „Bring me the popcorn from the kitchen table“ Die für die Aktiondarstellung unnötigen Wörtern wie „could“, „you“, „please“ sollen vernachlässigt werden.

Literaturverzeichnis

[ARA+06] Asfour, T. ; Regenstein, K. ; Azad, P. ; Schroder, J. ; Bierbaum, A. ; Vahrenkamp, N. ; Dillmann, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, S. 169–175 (zitiert auf Seite 11).

[BB79] Ballard, Bruce W. ; Biermann, Alan W.: Programming in Natural language:"NLC" as a Prototype. In: *Proceedings of the 1979 Annual Conference*, ACM, 1979, S. 228–237 (zitiert auf den Seiten 15 und 74).

[BFL98] Baker, Collin F. ; Fillmore, Charles J. ; Lowe, John B.: The Berkeley FrameNet Project. In: *IN PROCEEDINGS OF THE COLING-ACL*, 1998, S. 86–90 (zitiert auf Seite 18).

[Brier] Brill, Eric: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. In: *Comput. Linguist.* 21 (1995/December//), Nr. 4, S. 543–565. – ISSN 0891–2017 (zitiert auf den Seiten 17 und 73).

[Cho02] Chomsky, Noam: *Syntactic Structures*. Walter de Gruyter, 2002 (zitiert auf Seite 4).

[Coo00] Cooper, Stephen: Alice: A 3-D Tool for Introductory Programming Concepts. In: *Mathematics & Computer Science* (2000), Januar (zitiert auf Seite 11).

[CWB+11] Collobert, Ronan ; Weston, Jason ; Bottou, Léon ; Karlen, Michael ; Kavukcuoglu, Koray ; Kuksa, Pavel: Natural Language Processing (Almost) from Scratch. In: *Journal of Machine Learning Research* 12 (2011), Nr. Aug, S. 2493–2537. – ISSN ISSN 1533–7928 (zitiert auf Seite 10).

[Cyc08] Cycorp, I: The Cyc Project Home Page (2008) Available Online at: http://www. Cyc. Com. In: *Retrieved on December 10th* (2008) (zitiert auf Seite 11).

[Dic16] Dictionary.com Unabridged. (2016), August (zitiert auf den Seiten 29 und 64).

[DSBS09] Dzifcak, J. ; Scheutz, M. ; Baral, C. ; Schermerhorn, P.: What to Do and How to Do It: Translating Natural Language Directives into Temporal and Dynamic Logic Representation for Goal Management and Action Execution. In: *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, 2009, S. 4163–4168 (zitiert auf Seite 21).

[Gün15] Günes, Zeynep: *Aufbau Eines Sprachkorpus Zur Programmierung Autonomer Roboter Mittels Natürlicher Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Mai 2015 (zitiert auf den Seiten v und 55).

[Jur09] Jurafsky, Dan: *Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition /.* 2nd ed. Upper Saddle River, N.J. : : Pearson Prentice Hall,, c2009.. – ISBN 978–0–13–187321–6 (zitiert auf den Seiten 1, 2, 5, 6, 7 und 8).

[KGP05] Kingsbury, P. ; Gildea, D. ; Palmer, M.: The Proposition Bank: An Annotated Corpus of Semantic Roles. In: *Computational linguistics* 31 (2005), Nr. 1, S. 71–106 (zitiert auf Seite 9).

[Kje94] Kjellmer, Göran: *A Dictionary of English Collocations: Based on the Brown Corpus.* Oxford University Press, USA, 1994 (zitiert auf Seite 5).

[Koc15] Kocybik, Markus: *Projektion von Gesprochener Sprache Auf Eine Handlungsrepräsentation*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Juli 2015 (zitiert auf Seite 12).

[Liu04] Liu, Hugo: {MontyLingua: An End-to-End Natural Language Processor with Common Sense}. (2004) (zitiert auf Seite 16).

[Liu05] Liu, Hugo: Metafor: Visualizing Stories as Code. In: *10th International Conference on Intelligent User Interfaces*, ACM Press, 2005, S. 305–307 (zitiert auf den Seiten 16 und 72).

[Miler] Miller, George: WordNet: A Lexical Database for English. In: *Commun. ACM* 38 (1995/November//), Nr. 11, S. 39–41. `http://dx.doi.org/10.1145/219717.219748`. – DOI 10.1145/219717.219748. – ISSN 0001–0782 (zitiert auf Seite 11).

[MKM+94] Marcus, Mitchell ; Kim, Grace ; Marcinkiewicz, Mary A. ; Macintyre, Robert ; Bies, Ann ; Ferguson, Mark ; Katz, Karen ; Schasberger, Britta: The Penn Treebank: Annotating Predicate Argument Structure. In: *In ARPA Human Language Technology Workshop*, 1994, S. 114–119 (zitiert auf den Seiten 5 und 9).

[MLL06] Mihalcea, Rada ; Liu, Hugo ; Lieberman, Henry: NLP (Natural Language Processing) for NLP (Natural Language Programming). In: Gelbukh, Alexander (Hrsg.): *Computational Linguistics and Intelligent Text Processing.* Springer Berlin Heidelberg, Februar 2006 (Lecture Notes in Computer Science 3878). – ISBN 978–3–540–32205–4 978–3–540–32206–1, S. 319–330 (zitiert auf den Seiten 17 und 72).

[PGX10] Palmer, Martha ; Gildea, Daniel ; Xue, Nianwen: Semantic Role Labeling. In: *Synthesis Lectures on Human Language Technologies* 3 (2010), Januar, Nr. 1, S. 1–103. `http://dx.doi.org/10.2200/S00239ED1V01Y200912HLT006`. – DOI 10.2200/S00239ED1V01Y200912HLT006. – ISSN 1947–4040 (zitiert auf Seite 8).

[RWL01] Rayson, Paul ; Wilson, Andrew ; Leech, Geoffrey: Grammatical Word Class Variation within the British National Corpus Sampler. In: *Language and Computers* 36 (2001), November, Nr. 1, S. 295–306 (zitiert auf Seite 5).

[Wei14] Weigelt, Sebastian: *Programmieren in Natürlicher Sprache: Erkennung Und Semantische Assoziation von Entitäten in Natürlichsprachlichen Texten*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, Februar 2014 (zitiert auf den Seiten 17 und 74).

[WT15] Weigelt, S. ; Tichy, W.F.: Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. In: *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on* Bd. 2, 2015, S. 819–820 (zitiert auf Seite 11).

Anhang

A. Tagset

A.1. Penn Treebank Part-of-Speech Tagset

Number	Tag	Description	Examples
1	CC	Coordinating conjunction	& 'n and both but either
2	CD	Cardinal number	mid-1890 nine-thirty
3	DT	Determiner	all an another any both
4	EX	Existential there	there
5	FW	Foreign word	gemeinschaft hund ich jeux habeas
6	IN	Preposition/ subordinating conjunction	astride among uppon whether
7	JJ	Adjective	third ill-mannered pre-war regrettable
8	JJR	Adjective, comparative	bleaker braver breezier briefer brighter
9	JJS	Adjective, superlative	calmest choicest classiest cleanest
10	LS	List item marker	A A. B B. C D First
11	MD	Modal	can cannot could couldn't dare
12	NN	Noun, singular or mass	common-carrier cabbage knuckle-duster
13	NNS	Noun, plural	undergraduates scotches bric-a-brac
14	NNP	Proper noun, singular	Motown Venneboerger Czestochwa
15	NNPS	Proper noun, plural	Americans Americas Amharas
16	PDT	Predeterminer	all both half many quite such
17	POS	Possessive ending	' 's
18	PRP	Personal pronoun	hers herself him himself
19	PRP$	Possessive pronoun	her his mine my our
20	RB	Adverb	occasionally adventurously swiftly
21	RBR	Adverb, comparative	urther gloomier grander graver
22	RBS	Adverb, superlative	best biggest bluntest earliest
23	RP	Particle	aboard about across along
24	SYM	Symbol	% & ' " ".)).
25	TO	to	to
26	UH	Interjection	Goody Gosh Wow Jeepers Jee-sus
27	VB	Verb, base form	ask assemble assess assign
28	VBD	Verb, past tense	dipped pleaded swiped regummed
29	VBG	Verb, gerund or present participle	telegraphing stirring focusing angering
30	VBN	Verb, past participle	multihulled dilapidated aerosolized
31	VBP	Verb, non-3rd person singular present	predominate wrap resort sue twist
32	VBZ	Verb, 3rd person singular present	bases reconstructs marks mixes
33	WDT	Wh-determiner	that what whatever which
34	WP	Wh-pronoun	that what whatever whatsoever
35	WP$	Possessive wh-pronoun	whose
36	WRB	Wh-adverb	how however whence whenever
37	$	dollar	$ HK$ M$ NZ$ US$
38	“	opening quotation mark	‘ “
39	”	closing quotation mark	’ ”
40	(	opening parenthesis	([{
41	)	closing parenthesis	)] }
42	,	comma	,
43	–	dash	–
44	.	sentence terminator	. ! ?
45	:	colon or ellipsis	: ; ...

A.2. Penn Treebank Chunk Tagset

Tag	Description	Words	Example
NP	noun phrase	DT+RB+JJ+NN + PR	the strange bird
PP	prepositional phrase	TO+IN	in between
VP	verb phrase	RB+MD+VB	was looking
ADVP	adverb phrase	RB	also
ADJP	adjective phrase	CC+RB+JJ	warm and cosy
SBAR	subordinating conjunction	IN	whether or not
PRT	particle	RP	up the stairs
INTJ	interjection	UH	hello

A.3. Name Entity Types

Type	Tag	Sample Categories	Example
People	PER	Individuals, fictional characters	Turing
Organization	ORG	Companies, agencies, political parties	IPCC
Location	LOC	Physical extents, mountains,lakes	Mt. Sanitas
Geo-Political Entity	GPE	Countries, states, provinces, counties	Palo Alto
Facility	FAC	Bridges, buildings, airports	Lincoln Tunnel
Vehicles	VEH	Planes, trains, automobiles	Mini Cooper

B. Literaturanalysen

B.1. Metafor

[Liu05] Liu, Hugo: Metafor: Visualizing Stories as Code. In: *10th International Conference on Intelligent User Interfaces*, ACM Press, 2005, S. 305–307

Inhalte

I 1 Geschichte in Programmcode umwandeln.
I 2 Ziel: die Programmieranfänger helfen, die Intuition von Programmieren zu entwickeln und dem Fortgeschrittener ein Entwurfswerkzeug anbieten.
I 3 Introspektionsfunktion: Programmcode zu natürlichen Ausdruck umwandeln.
I 4 Als ein Brainstorming-Tool dienen.

Defizite

D 1 U-Tel: erzeugt kein Code.

Prämissen

P 1 Nur eine aussagekräftige Untermenge von Englisch wird benutzt.
P 2 Der Benutzer soll bestimmte Programmierkenntnisse haben.
P 3 Der erzeugte Code ist nicht ausführbar.
P 4 Nicht jede grammatisch korrekte Konstruktion von Sätzen kann verstanden werden.
P 5 Nicht jedes Wissen kennen.

Lösungen

L 1 Ambiguität in der Eingabe benutzen um flexibel zu bleiben mittels Refaktorisieren.
L 2 Parser - MontyLingua NLU benutzen. VSOO Struktur verwenden.
L 3 Integration with Common Sense Knowledge.
L 4 Programmatic Intepreter.
L 5 MetaforLingua.
L 6 Code Renderer.

Nachweise

N 1 User-Studie von 13 Leuten (Programmieranfänger & Fortgeschrittener)
N 2 Programmieranfänger schätzen, dass Metafor 22% Zeit spart. Fortgeschrittener schätzen 11%.
N 3 Programmieranfänger werden wahrscheinlich Metafor benutzen für Brainstorming. Fortegschrittener werden 31% wahrscheinlicher Metafor benutzen als auf dem Papier zu entwerfen.

Notizen

NO 1 Beschreibung in natürlicher Sprache als Repräsentation für Programme benutzen.
NO 2 Die Nominalphrasen sind Programmobjekte, die Verbalphrasen sind Funktionen, die Adjektivphrasen sind Eigenschaften.
NO 3 Untersuchen, wie Schüler in der fünften Klasse Programme beschreiben.

B.2. Natural Language Processing for Natural Language Programming

[MLL06] Mihalcea, Rada ; Liu, Hugo ; Lieberman, Henry: NLP (Natural Language Processing) for NLP (Natural Language Programming). In: Gelbukh, Alexander (Hrsg.): *Computational Linguistics and Intelligent Text Processing.* Springer Berlin Heidelberg, Februar 2006 (Lecture Notes in Computer Science 3878). – ISBN 978–3–540–32205–4 978–3–540–32206–1, S. 319–330

Inhalte

I 1 Propose a system that attempts to convert natural language text into computer programs.
I 2 Focus on procedural programming. Build procedures out of steps and loops.
I 3 Steps and loops are the most difficult aspects of procedural programming.
I 4 Automatically identify steps, loops, and comments, and convert them into a program skeleton that can be used as a starting point for writing a computer program, expected to be particularly useful for those who begin learning how to program.
I 5 Programming resembles storytelling.
I 6 The object-oriented and agent-oriented formats most closely embody human storytelling intuition
I 7 Correspondences between natural language and descriptive structures: noun chunks are things, verbs are actions, adjectives are properties of things, adverbs are parameters of actions.
I 8 Almost all natural languages are built atop the basic construction called independent clause, which at its heart has a who-does-what structure, or subject-verb-directObject-indirectObject (SVO) construction.
I 9 Natural language processing for natural language programming or natural language pro- gramming for natural language processing. Goes both ways.

Defizite

D 1 Early work rather ambitious, targeting the generation of complete computer programs that would compile and run (NLC Prototype)

Lösungen

For procedural programming:

L 1 Three main components:

a) The step finder, which has the role of reading an input natural language text and break it down into steps that can be turned into programming statements.
   i. Pre-process: tokenize and part-of-speech tag using Brill's [Brier] tagger.
   ii. Next: identify all verbs that could be potentially turned into program functions, such as e.g. read, write, count. Then find the boundaries of these steps: a new step will start either at the beginning of a new sentence, or whenever a new verb in the active voice is found (typically in a subordinate clause).
   iii. Finally: identify the object of each action.

b) The loop finder, which identifies the natural language structures that indicate repetition. Seek markers of repetition, such as each X, every X, all X. Also look for plural nouns as potential indicators of repetition.

c) The comment identification components, which identifies the descriptive statements that can be turned into program comments.

L 2 First break text down into steps that will represent action statements. Next each step is run through the comment identification component. Finally checks if a step consists of a repetitive statement for those that are not comments.

Nachweise

N 1 Programming assignment corpus collected using a Web crawler that searches the Web for pages containing the keywords programming and examples. Manually label main

steps and repetitive structures and compare to automatically generated program skeletons. Precision = number of correct programmatic structures (steps or loops) out of the total number of structures automatically identified = 86% for the step identification and 80.6% for the loop identification.

B.3. Erkennung und semantische Assoziation von Entitäten in natürlichsprachlichen Texten

[Wei14] Weigelt, Sebastian: *Programmieren in Natürlicher Sprache: Erkennung Und Semantische Assoziation von Entitäten in Natürlichsprachlichen Texten*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Diplomarbeit, Februar 2014

Inhalte

I 1 Projektziel: natürlichsprachliche Programmierung zu ermöglichen.
I 2 Entitäten in natürlichsprachlichen Texten identifizieren und mit Individuen einer Ontologie assoziieren.

Lösungen

L 1 Atomarer Satz

a) Das Atomarer-Satz-Objekt benutzen, um einen natürlichsprachlichen Teilsatz intern darzustellen. Es besteht aus Subjekt, Prädikat und Objekt, die als Wortlisten angegeben sind.
b) Jeder atomare Satz hat einen Typ, entweder „action“ (Aktion) oder „description“ (Beschreibung). Um die semantische Informationen zu verarbeiten, muss dieser Typ festgelegt werden.
c) Abbildung von dem Satzteil auf Ontologie-Individuen.
d) Bsp: The frog and the bunny turn to face Alice. Zwei Teilsätzen: über „und“ verknüpft. -> Zwei Atomarer-Satz-Objekt instantiiert. Beide sind vom Typ Aktion. Gleiche Prädikat und Objekt. Unterschiedliches Subjekt.

B.4. „NLC“ AS A PROTOTYPE

[BB79] Ballard, Bruce W. ; Biermann, Alan W.: Programming in Natural language:“NLC” as a Prototype. In: *Proceedings of the 1979 Annual Conference*, ACM, 1979, S. 228–237

Inhalte

I 1 NLC system enables a computer user to type English commands into a display terminal and watch them executed on example data shown on the screen. The system is designed to process data stored in matrices or tables, and any problem which can be represented in such structures can be handled if the total storage requirements are not excessive.
I 2 While entering the above program, the user could observe the system performing the calculation on a sample row and had the opportunity to verify the correctness of each action. NLC clearly marks each operand when executing a computation so that its operation can be checked. Then the user asked that the series of operation which had been observed to be correctly executed on the example row be executed on the remaining rows. -> Provide user with continuous feedback concerning program correctness while the computation is proceeding and continues a repetitive computation only after one pass through the loop has been verified to be correct.
I 3 The major effort of parsing in NLC is to recognize the imperative verb and its associated operands. => Noun phrases play an extremely important role.

Prämissen

P 1 The problem domain for the system is the world of matrices, and all references are to related concepts: rows, columns, entries, labels, and simple operations on them.

P 2 All input sentences are imperatives. It was found that users tended to use this form and that the system design could be greatly improved if the main verb could be assumed to begin the sentence. Thus users are told to begin every input with an imperative verb. Alters the user's expressive power. e.g. „Add row 1 to row 3“ can be understood while „Row 1 is to be added to row 3“ can't be accepted by the system.

Lösungen

L 1 The system's four stages of processings:

a) morphological

b) syntactic

c) semantic

d) computational

L 2 **The „Scanner“** The Scanner identifies from the characters of the input the individual tokens to be used during the syntactic phase. Tokens are precisely the maximal substrings containing no spaces. „Add five te the 2nd positive entry in col 2.“ contains tokens „Add“, „five“, „te“, „the“, „2nd“, „positive“, „entry“, „in“, „col“, „2“, „.“. The Scanner also finds possible meanings for the tokens. For example, „Add“ could be an imperative verb and „five“ could be an integer.

The „Parser“ The Parser is a syntax analyzer. It determines the structure of the user's request, like „add x to y.“ Put imperative verbs from the domain of matrices into 6 categories:

a) verbs that take exactly one operand. e.g. „double“, „negate“

b) verbs that take two operands which are distinguished by a preposition. The preposition is called „verbicle“ e.g. „add (to)“, „subtract (from)“

c) particle-taking verbs that take one operand. e.g. „add up“, „round off“

d) two-operand imperative verbs without verbicle „call column 5 terry“

e) zero-operand form: „quit“

f) zero-operand form: „back up“

The „Semantics“ Processor Responsible for determing the specific matrix entities referred to. The order to perform resolutions: semantic analysis from right to left.

The „Matrix Computer“ Does computation. Knowledge of the various imperatives is represented by a table, so that only a small number of primitives are required. „double“ is treated as „multiply by 2“, „increment“ as „add 1 to“. Before actual computations can be carried out, it is necessary to decide how the operands are to be treated.Implicit actions upon entries resemble the way in which containment information is utilized during semantic processing. In the case of „add rows 1 and 2 to rows 3 and 4“, the computation involves performing individual additions after pairing the entries in the rows.

L 3 Conjunctions need not be confined to well-defined groups of words.

Notizen

NO 1 An example of a natural language program that can be processed by NLC:

a) „Choose a row in the matrix.“

b) „Put the average of the first four entries in that row into its last entry.“

c) „Double its fifth entry and add that to the last entry of that row.“

d) „Divide its last entry by 3.“

e) „Repeat for the other rows.“

NO 2 Try to identify general principles wherever possible, rather than to treat minor issues in isolation.

NO 3 Linguistic knowledge: Valuable for designers of natural language processors: restrict the domain of discours. This serves to limit the kinds of structures to be recognized as well as the actual words to be considered.