Multimediales Sprachlehrsystem

Alexis Pangalos

17. September 2002

Projektbericht zur Veranstaltung „Multimediale Lehr- und Lernsysteme“ von Kilian Foth und Prof. Dr. Wolfgang Menzel an der Universität Hamburg im WS 2001/2002 und SS 2002.

Inhaltsverzeichnis

1 Einleitung

In diesem Projektbericht wird ein interaktives, multimediales Sprachlehrsystem beschrieben, das mit Hinblick auf einen Einsatz im Fremdsprachenunterricht entwickelt wurde. Ziel des Projekts war die Entwicklung eines kommunikativ anspruchsvollen Übungsszenarios, in dem eine weitgehend freie Spracheingabe möglich ist. Die Übung kann über das Internet durchgeführt werden. Wesentlich ist, dass ein Feedback generiert wird, das auf grammatikalische Fehler aufmerksam macht und sie gegebenenfalls verbessert.

Der Hauptgegenstand des Berichts ist eine detaillierte Beschreibung der Funktionsweise der Dialogsteuerung. Bevor diese gegeben wird, werden die weiteren Komponenten des Systems kurz beschrieben. In Abschnitt 2 wird der Aufbau der Sprachübung dargestellt. Eine Beschreibung der Benutzeroberfläche findet sich in Abschnitt 3. Die Funktionsweisen des verwendeten CDG-Parsers, der Fehlersimulation, und der Erklärungsgenerierung werden in den Abschnitten 4 und 5 angedeutet. Die Kommunikation zwischen den Komponenten wird in Abschnitt 6 kurz beschrieben. In Abschnitt 7 erfolgt die Beschreibung der Dialogsteuerung.

2 Aufbau der Übung

Die Sprachübung *Der Meister Albrecht und seine Schüler* richtet sich an Deutschlerner mit Grundkenntnissen, sie bietet eine Möglichkeit neben Wortschatz und Satzbau den Gebrauch von Lokalpräpositionen zu üben. Die Übungsaufgabe ist in Form eines Spiels mit einem klaren Spielziel gestaltet, zu deren Lösung der Schüler eine Strategie entwickeln muss. Er führt einen Dialog, in dem er seine Sätze selbst formulieren muss. Nur mit Hilfe der Antworten, die er bekommt, kann er die Aufgabe lösen.

Die Übungssituation hat folgende Gestalt: Der Deutschlerner ist der Schüler des berühmten Künstlers D. Albrecht, der das Stillleben *Der gedeckte Tisch* gemalt hat. Von diesem Bild sind neun Fälschungen aufgetaucht, von denen keine dem Originalbild gleicht. Die Gegenstände auf dem Bild sind in den Fälschungen anders angeordnet oder verändert. Es ist sehr lange her, dass Meister Albrecht das Bild gemalt hat, inzwischen ist er sehr alt, arm und blind geworden, so dass er das Originalbild nicht ohne fremde Hilfe erkennen kann. Seine Erinnerungen an das Bild sind aber noch recht gut.

Der Schüler hat die Aufgabe, dem Meister dabei zu helfen, das Originalbild herauszufinden. Dazu soll er die Bilder beschreiben und mit Hilfe der Antworten des Meisters das Original ausfindig machen. Wenn der Schüler das Originalbild erkannt hat, kann er dessen Nummer in ein dafür vorgesehenes Feld eintragen. Hat er es richtig erkannt, so hat er das Spiel gewonnen.

Meister Albrecht hat einen Papagei der ihn immer begleitet. Der Papagei hat die Aufgabe den Deutschlerner auf grammatikalische Fehler aufmerksam zu machen und sie zu verbessern.

3 Benutzeroberfläche

Der Webserver den der Sprachschüler von einem der üblichen Browser (z.B. Netscape) aus aufrufen kann ist mit PHP erstellt. PHP ist eine in HTML eingebettete Skriptsprache zum Erstellen dynamischer Webseiten.

Auf einer Startseite befindet sich eine Spielanleitung und ein Button der angeklickt werden kann, um die Übung zu starten. Zunächst muss sich der Schüler anmelden und ein Cookie zulassen. Bei der Anmeldung wird dem Schüler eine Spielernummer zugeordnet und per Zufallsgenerator eines der zehn Bilder als Originalbild ausgewählt. Der Schüler gelangt dann auf die Hauptseite des Spiels (vgl. Abbildung 1). Auf der Hauptseite sind die zehn Bilder, das Original einschließlich

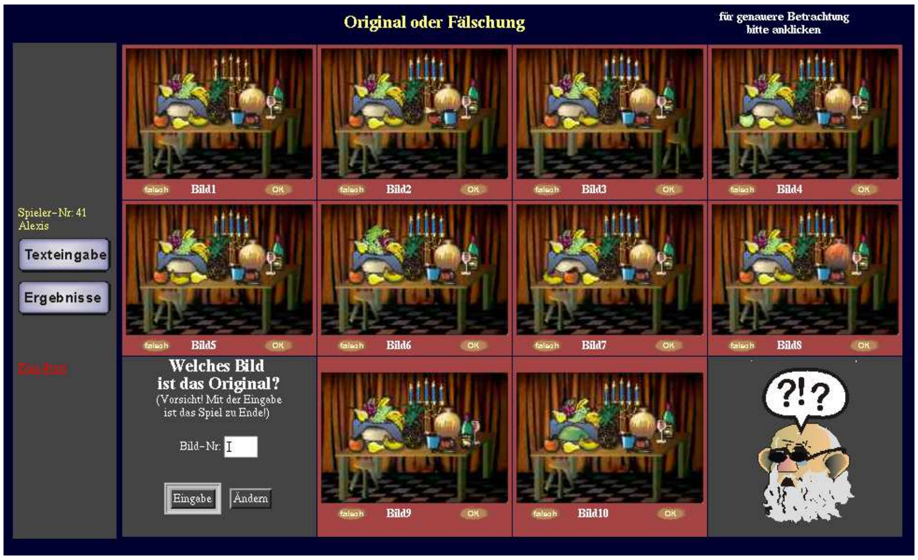


Abbildung 1: Hauptseite

der neun Fälschungen, zu sehen. Das Originalbild, das der Schüler herausfinden soll, ist für das ganze Spiel das per Zufallsgenerator ausgewählte Bild. Als Beispiel sei hier das vierte Bild das Original.

Der Schüler wird die Bildergalerie betrachten und z.B. bemerken, dass nur im ersten Bild die Kerzen golden sind. Er kann das Bild anklicken und gelangt so zu einer Eingabeseite (vgl. Abbildung 2). Auf dieser Seite ist ein Textfeld vorgesehen, in das der Schüler eine Bildbeschreibung eingegeben soll. In diesem Beispiel wird er z.B. den Satz „*Die Kerzen sind golden*“ eingeben. Der Meister wird sagen, dass die Kerzen nicht golden sind, da sie auf dem Originalbild (Nr. 4) blau sind. Daraufhin kann der Schüler durch anklicken von *Bildauswahl* zur

Abbildung 2: Eingabeseite

Hauptseite zurückkehren und das Bild mit den goldenen Kerzen als Fälschung markieren, denn er weiß jetzt, dass dieses Bild nicht das Original sein kann.

Die Antwort des Meister Albrecht und die Kommentare des Papageien werden auch auf der Eingabeseite gegeben. Auf der Hauptseite soll der Schüler Stück für Stück die Fälschungen markieren und so das Originalbild herausfinden. Hat er es erkannt, kann er die entsprechende Nummer auf der Hauptseite in das Feld mit der Überschrift „*Welches Bild ist das Original?*“ eingeben.

4 CDG-Diagnose und Fehlersimulation

Zur Analyse der vom Schüler eingegebenen Sätze wird das von Niels Stockfleth beschriebene und programmierte Verfahren von CDG-Parsing und Fehlersimulation verwendet, eine detaillierte Beschreibung findet sich in [Stockfleth 2000]. Das CDG-Parsing (*Constraint Dependency Grammar*) hat die Aufgabe für jede Schülereingabe eine syntaktische Strukturbeschreibung zu generieren. Es eignet sich besonders für den Einsatz in einem Sprachlehrsystem, da es auch für grammatikalisch fehlerhafte Sätze eine Strukturbeschreibung generieren kann.

Das Verfahren besteht aus einem Lexikon, in dem Wörter mit Merkmalen versehen werden, einer CD-Grammatik, die die Grammatikregeln enthält, die beachtet werden sollen und dem CDG-Parsingverfahren, das Grammatik und Lexikon benutzt, um für einen Satz eine syntaktische Struktur zu generieren. Das Lexikon und die Grammatik wurden den Bedürfnissen der Sprachübung angepasst.

Die Fehlersimulation generiert für Kongruenzfehler Fehlererklärungen. Kongruenzfehler sind Fehler der Gestalt, dass zusammengehörende Wörter im Numerus, Kasus oder Genus nicht übereinstimmen. Dabei werden verschiedene Sichtweisen berücksichtigt. Für den Genusfehler in der Nominalphrase *Das Apfel* wird die Beschreibung `agr:gender` generiert, sie besagt, dass *Das* und *Apfel* im Genus nicht übereinstimmen. Die Beschreibung `val:gender.{neutr} der` gibt an, dass der Artikel *Das* in *Der* umgewandelt werden muss. Manche Fehler können auf verschiedene Weise gelesen werden, *Die Apfel* kann entweder zu *Der Apfel* oder zu *Die Äpfel* verbessert werden. In diesem Fall werden alternative Fehlerbeschreibungen generiert. Die Fehlerbeschreibungen werden in einer Form gegeben, die für den Schüler nicht hilfreich ist. Das Generieren von natürlichsprachlichen Fehlererklärungen ist die Aufgabe einer weiteren Komponente.

5 Generieren von Fehlererklärungen

Ein wesentlicher Bestandteil des Sprachlehrsystem ist, dass der Schüler auf seine grammatikalischen Fehler aufmerksam gemacht wird, sie ihm erklärt und gegebenenfalls verbessert werden. Die bei der Fehlersimulation des verwendeten Diagnoseverfahrens generierten Fehlerbescheibungen müssen so aufbereitet werden, dass sie dem Schüler nützlich sind. Hierzu werden zwei Fehlertypen unterschieden, Kongruenzfehler und Strukturfehler.

Wenn der Schüler z.B. den Satz „*Das Apfel ist rot*“ eingibt wird der Papagei ihn mit der Bemerkung „*Du hast einen Sprachfehler gemacht!*“ darauf aufmerksam machen. Klickt der Schüler dann auf *Anzeigen* wird ihm der Eingabesatz präsentiert indem der fehlerhafte Satzteil, hier *Das Apfel*, in rot erscheint. Reicht ihm das, um seinen Fehler zu verbessern, so geht das Spiel weiter. Weiß er sich nicht zu helfen, dann kann er auf *Erklären* klicken und sich die entsprechende Grammatikregel anzeigen lassen. Für das obige Beispiel wäre das „*Du hast einen Genusfehler gemacht*“. Reicht auch diese Erklärung nicht aus, kann der Schüler sich, indem er auf *Lösung* klickt, eine Berichtigung anzeigen lassen. Hier würde der Satz „*Das* ist sächlich und *Apfel* ist männlich: *der Apfel* wäre richtig gewesen“ ausgegeben werden. Der Schüler hat zu jeder Zeit die Möglichkeit die Fehler zu ignorieren und durch klicken auf den weiter-Button das Spiel fortzusetzen.

Bei Kongruenzfehlern werden die in den Diagnosen auftretenden value, agree und restrict-Komponenten zum generieren von einfachen sprachlichen Erklärungen genutzt. Dies geschieht indem zu einem value-Fehler (z.B. `val:gender.{neutr} der`) der passende agree-Knoten (z.B `agr:gender`) gesucht wird. Diese Informationen werden kombiniert, um die oben beschriebenen Fehlererklärungen zu erzeugen.

Bei Strukturfehlern wie z.B. „*Das Apfel auf dem Tisch liegt*“ werden vom CDG-Parser die Regeln angegeben die verletzt wurden. Für diesen Beispielsatz würde `CID_ERR_VERBSECOND` ausgegeben werden. Dieser Fehler wird auf die natürlichsprachliche Fehlererklärung „*Der zweite Satzteil im Aussagesatz ist immer das Prädikat. Die beiden Satzteile müßten also vertauscht werden*“ abgebildet. Für 28 verschiedene Strukturfehler stehen Fehlererklärungen zur Verfügung.

6 Kommunikation

Die Kommunikationskomponente ist ein Java-Programm, das die verschiedenen Teile des Sprachlehrsystems verbindet. Da die Kommunikationskomponente der PHP-Komponente dient, indem sie für Anfragen bestimmte Antworten zurückgibt, wird sie auch als Java-Server bezeichnet.

Von der PHP-Komponente wird der Eingabesatz des Schülers einschließlich der Nummer des Originalbildes an den Java-Server geschickt. Ungültige Eingaben werden vom Java-Server abgefangen und sofort beantwortet, das ist z.B. der Fall, wenn ein Fragezeichen im Eingabesatz auftaucht. Handelt es sich um einen gültigen Satz, dann wird er an den CDG-Parser und die Fehlersimulation weitergeleitet. Wenn kein Syntaxbaum generiert werden kann, wird eine Standardantwort zurückgegeben. Kann ein Syntaxbaum erzeugt werden, so wird dieser aufbereitet und an die Dialogsteuerung geschickt. Generierte Fehlerbeschreibungen werden der Fehlererklärung übergeben. Die von der Fehlererklärung und der Dialogsteuerung erzeugten Antworten werden zusammengefügt und an die PHP-Komponente zurückgeschickt.

Die Verbindungen zur PHP-Komponente und zur Dialogsteuerung laufen über Sockets. CDG-Parser, Fehlersimulation und Fehlererklärung werden im Java-Programm direkt aufgerufen. Ein Diagramm, das die Architektur veranschaulicht findet sich in Abbildung 3.

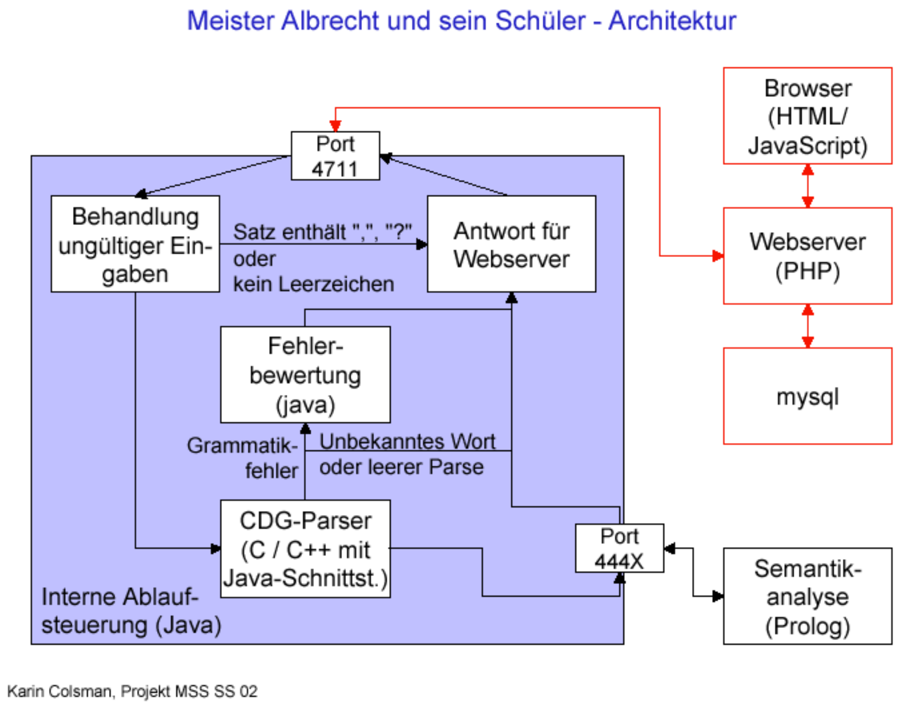


Abbildung 3: Architektur

7 Dialogsteuerung

Die Aufgabe der Dialogsteuerung ist es, den Dialog mit dem Schüler zu führen. Zu einer Bildbeschreibung des Schülers muss sich der Meister Albrecht äußern, er muss sagen, ob das durch den Schüler beschriebene mit dem Originalbild übereinstimmt oder nicht.

7.1 Bildrepräsentation

Die in der Sprachübung vorkommenden zehn Bilder werden alle durch Datalog-Fakten repräsentiert, so eine Repräsentation wird auch Wissensbasis genannt. Eine Einführung in Wissensbasierte Systeme findet sich in [Poole 1998]. Den Objekten auf den Bildern werden eindeutige Namen zugeordnet, z.B. `banane1`, `banane2`, `tasse1` oder `messer`. Es gibt 24 Objekte, zwischen ihnen bestehen folgende örtliche Beziehungen: `in`, `auf`, `unter`, `vor` und `links`. Das Prädikat `rechts` taucht in den Bildrepräsentationen nicht auf, da es aus `links` abgeleitet werden kann. Das gleiche gilt für `vor` und `hinter`. Beispiele für Einträge in der Bildbeschreibung eines bestimmten Bildes sind:

```
in(banane1, schale, 4).
auf(apfel, tisch, 4).
```

Die Nummer an der dritten Argumentstelle gibt an um welches Bild es sich handelt, in diesem Fall also um das vierte der zehn Bilder. Die Einträge heißen *Atome* oder *Fakten*. Zusätzlich zu den verwendeten Lokalpräpositionen und dem Adjektiv `links` gibt es noch das Prädikat `eigenschaft`. Hier steht an erster Argumentstelle ein Objekt, an zweiter eine Eigenschaft des Objekts und an dritter Stelle die Bildnummer. Beispiele sind:

```
eigenschaft(apfel, rot, 4). oder
eigenschaft(tasse1, groß, 4).
```

Das Prädikat `eigenschaft` taucht in der Repräsentation eines Bildes ungefähr 30 Mal auf. Es gibt zwar nicht so viele Gegenstände auf dem Tisch, es müssen aber alternative Beschreibungen zulässig sein. Zum Beispiel können die hellen Weintrauben auch als grün, hellgrün oder gelb beschrieben werden. Die Bildrepräsentation enthält aus diesem Grund die Einträge:

```
eigenschaft(trauben2, hell, 4).
eigenschaft(trauben2, hellgrün, 4).
eigenschaft(trauben2, gelb, 4).
eigenschaft(trauben2, grün, 4).
```

Eine vom Sprachschüler eingegebene Bildbeschreibung muss in Prädikate der oben beschriebenen Form gebracht werden, damit die semantische Richtigkeit im

Hinblick auf das Originalbild überprüft werden kann. Der Satz „*Der Apfel liegt auf dem Tisch*“, muss in `auf(apfel, tisch, 4)` umgewandelt werden, dann kann anhand der Bildbeschreibungen getestet werden, ob der eingegebene Satz mit dem Originalbild übereinstimmt. Aus dem Satz „*Der rote Apfel liegt auf dem Tisch*“, müssen die Atome `eigenschaft(apfel, rot, 4)` und `auf(apfel, tisch, 4)` erzeugt werden. Im zweiten Beispiel muss für beide Atome getestet werden, ob sie Teil der Bildbeschreibung sind.

Das Prädikat `links` wird benutzt um zu beschreiben wo, von einem bestimmten Gegenstand aus gesehen, sich ein anderer befindet. Wenn nun `rechts(banane, apfel, 4)` aus einem Schülersatz generiert wurde, findet die Regel `rechts(X,Y,N) :- links(Y,X,N)` Anwendung und es wird in der Wissensbasis nach `links(apfel, banane, 4)` gesucht. Das Prädikat `neben(X,Y,N)` kann aus `links(X,Y,N)` oder aus `links(Y,X,N)` abgeleitet werden. Ein Eintrag `links(X,Y,N)` steht genau dann in der Wissensbasis wenn der Gegenstand `X` direkt links von dem Gegenstand `Y` zu finden ist. Es gibt ungefähr 20 Einträge der Form `links(X,Y,N)` für jedes Bild. Jede Bildrepräsentation enthält um die 80 Fakten.

Die Dialogsteuerung ist in Prolog programmiert, Prolog ist eine Sprache die das Suchen nach Fakten in einer Wissensbasis übernimmt, eine Einführung in Prolog wird in [Clocksin 1987] gegeben. Der verwendete Compiler ist SWI-Prolog. SWI-Prolog wurde an der Universität von Amsterdam entwickelt und ist ein frei verfügbarer Prolog-Compiler. SWI steht für *Sociaal Wetenschappelijke Informatica*. Die Entwicklung von SWI-Prolog begann 1987 und war geleitet vom Kriterium der praktischen Anwendbarkeit. Da SWI-Prolog frei verfügbar ist, ist der Compiler für Ausbildungszwecke sehr verbreitet. Auch die Anzahl an kommerziellen Anwendungen wächst schnell. (vgl. `http://www.swi-prolog.org`)

7.2 Generieren der Fakten – einfaches Beispiel

Um einen vom Schüler eingegebenen Satz in Fakten umzuwandeln, die sich mit der Datenbasis vergleichen lassen, wird die Dependenzstruktur, die bereits für die Fehlerdiagnose erstellt wurde, benutzt. Zu dem eingegebenen Satz „*Der Apfel liegt auf dem Tisch*“ generiert der CDG-Parser folgenden Baum (vgl. Abbildung 4). Dieser Baum steht für eine Klasse von Sätzen. Der Satz „*Die Flasche steht vor den Tassen*“ hat die gleiche Struktur und also den gleichen Baum. Um die Baumstruktur in Prolog verwenden zu können, muss der Baum in Form von Atomen in die Datenbasis von Prolog eingefügt werden. Hierzu werden die Kanten in Prädikate umgewandelt, an der ersten Argumentstelle steht das, was unterhalb der Kante stand und an der zweiten das, was oberhalb stand. Der obige Baum wird für seine Verarbeitung in Prolog in

```
det(der, apfel)   subj(apfel, liegt)  vfin(liegt, [])
pmod(auf, liegt)  pn(tisch, auf)      det(dem, tisch)
```

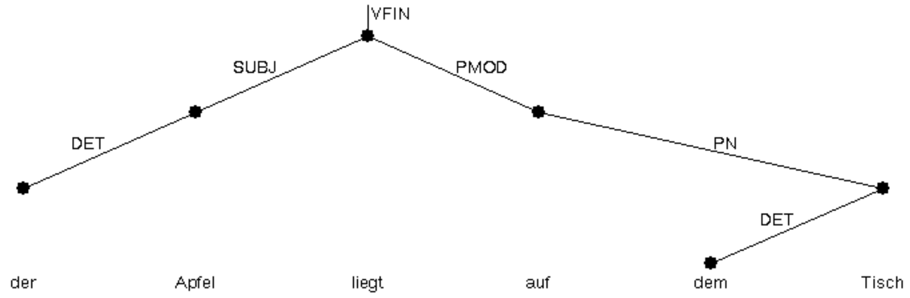


Abbildung 4: Syntaxbaum

umgewandelt. Das Zeichen `[]` steht für den Wurzelknoten. Anhand einer Vereinfachung der Regel `sem_trans` (für semantische Transformation), die das Atom `SemStrukt` (für semantische Struktur) zurückgibt, soll verdeutlicht werden, wie aus einem Satz (hier „*Der Apfel liegt auf dem Tisch*“) ein Atom (hier `auf(apfel, tisch)`) generiert wird.

Bei einem Aufruf von `sem_trans(SemStrukt)` (vgl. Abbildung 5) wird zunächst

```
sem_trans(SemStrukt) :-
    pmod(P,_),
    subj(Arg1,_),
    pn(Arg2,_),
    SemStrukt=..[P,Arg1,Arg2].
```

Abbildung 5: Vereinfachte Version von `sem_trans`

`pmod(P,_)` aufgerufen, die Variable `P` wird durch die Konstante `auf` ersetzt. Weiter werden `subj(Arg1,_)` und `pn(Arg2,_)` aufgerufen; `Arg1` wird zu `apfel` und `Arg2` zu `tisch`. Nun wird mit Hilfe des Prologprädikats `=..` das Atom `SemStrukt` erzeugt. Das Prädikat `=..` hat zwei Argumente, das erste ist ein Atom und das zweite ist eine Liste, in die das Atom zerlegt oder aus der es zusammengesetzt wird. In diesem Beispiel wird aus der Liste `[auf, apfel, tisch]` das Atom `auf(apfel, tisch)` gebildet. Mit der gleichen Version von `sem_trans` lässt sich aus dem Satz „*Die Flasche steht vor den Tassen*“ `vor(flasche, tassen)` erzeugen. Auch grammatikalisch fehlerhafte Sätze wie z.B. „*Das Apfel auf das Tisch liegt*“ können verarbeitet werden.

7.3 Eingabestruktur und Vorverarbeitung

Die Kommunikationskomponente verarbeitet den vom CDG-Parser erstellten Baum bevor er an die Semantikkomponente geschickt wird. Die im Syntaxbaum vorkommenden Wörter werden durch Zahlen repräsentiert. Das erste Wort im Satz erhält dabei die Zahl Eins, das zweite die Zahl Zwei usw., hierdurch wird die

Eindeutigkeit der Wörter gewährleistet, wenn ein Wort in einem Satz mehrfach auftaucht.

Zur Verarbeitung wird für jede Schüleräußerung eine Liste mit drei Einträgen zur Verfügung gestellt. An erster Stelle der Liste steht der Syntaxbaum, wobei die Wörter durch Zahlen ersetzt wurden, an der zweiten Stelle steht der Eingabesatz und an der letzten Stelle steht die Nummer, die angibt um welches Originalbild es sich in diesem Spiel handelt.

Der Syntaxbaum wird in der oben beschriebenen Form dargestellt, wo eine Kante von einem Knoten zu einem anderen durch ein Atom repräsentiert wird. Die Atome stehen in einer Liste. Die Wörter des Eingabesatzes werden in der Reihenfolge in der sie vorkommen in eine Liste geschrieben. Bei groß geschriebenen Wörtern ist der erste Buchstabe durch den entsprechenden kleinen Buchstaben ersetzt. Dies ist notwendig, da in Prolog groß geschriebene Wörter Variablen sind; klein geschriebene sind Konstanten. Zum Beispiel wird, wenn der eingegebene Satz „*Der Apfel liegt auf dem Tisch*“ ist, an die Semantikkomponente die Liste

```
[[det(1,2), subj(2,3), vfin(3,0), pmod(4,3), pn(6,4),
det(5,6)], [der, apfel, liegt, auf, dem, tisch],4]
```

geschickt. Das Originalbild ist in diesem Beispiel das vierte Bild.

Um den Syntaxbaum und die Wörter des Satzes verarbeiten zu können, werden die Elemente der Liste, die den Baum darstellt, in die Datenbasis von Prolog eingefügt. Hierfür stellt Prolog das Prädikat `assert` zur Verfügung. Alle Adjektive die in der Wortliste vorkommen werden in ihre Grundform gebracht. Für diese Aufgabe gibt es die Regel `gf` (für Grundform). Sie übernimmt eine Wortliste, wandelt die Adjektive in ihre Grundform um und gibt die Liste zurück. Die so erzeugte Liste wird in die Datenbasis eingefügt, indem sie als einziges Argument dem Prädikat `lautung` übergeben wird. Der Fakt `lautung(Liste)` wird dann in die Datenbasis geschrieben.

Da immer neue Sätze verarbeitet werden sollen, müssen die Fakten die zum vorangegangenen Baum gehören vor dem Einfügen neuer Fakten gelöscht werden. Die oben beschriebene Vorverarbeitung wird mit der Regel `sem_int` ausgeführt, anschließend ruft sie die Regel `sem_trans` auf.

7.4 Kommunikation über Sockets

Die Kommunikation zwischen der Dialogsteuerung und der Kommunikationskomponente läuft über Sockets. Dies geschieht mit Hilfe von XPCE, einem Programm für die Entwicklung von graphischen Anwendungen und graphischen Nutzeroberflächen (graphical user interfaces GUIs) in Prolog (und anderen logischen Sprachen). Die Entwicklung von XPCE begann 1987 zeitgleich mit der von SWI-Prolog und ist eng mit ihr verbunden. XPCE stellt unter anderem die hier verwendeten Funktionen für die Kommunikation über Sockets zur Verfügung. Eine genaue Beschreibungen von XPCE findet sich unter `http://www.swi.psy.uva.nl/projects/xpce/home.html`.

Ein Aufruf der Regel `communication` (vgl. Abbildung 6) bewirkt, dass zunächst

```
communication :-
    socketnummer(Socketnummer),
    new(@albrechtsocket, socket(Socketnummer)),
    send(@albrechtsocket,
          accept_message,
          message(@prolog, accept_handler, @arg1)),
    send(@albrechtsocket,
          input_message,
          message(@prolog, input_handler, @arg1)),
    send(@albrechtsocket,listen).
```

Abbildung 6: `communication`

mit `socketnummer` die Nummer des Sockets über den die Verbindung laufen soll aus der Datei `config.txt` gelesen wird. Das von XPCE zur Verfügung gestellte Prädikat `new` wird verwendet, um mit `new(@albrechtsocket,socket(Nummer))` ein Objekt vom Typ Socket mit Namen `albrechtsocket` und der Adresse `Nummer` zu erstellen.

Mit dem ersten Aufruf von `send` bekommt das Objekt `albrechtsocket` die Eigenschaft, dass es einen Verbindungaufbau von außen zulässt. Wenn eine Verbindung aufgebaut wird, wird die Adresse des Ports, von dem aus die Verbindung aufgebaut wird, mit `@arg1` dem Prädikat `accept_handler` übergeben, dieses wird anschließend ausgeführt. Das hat zur Folge, dass der Fakt `endpoint(N)` in die Datenbasis von Prolog eingefügt wird, wobei das `N` die Portnummer ist. Der zweite Aufruf von `send` bewirkt, dass wenn eine Nachricht an `albrechtsocket` geschickt wird, diese an das Prädikat `input_handler` weitergeleitet wird. Das letzte `send` setzt den Socket `albrechtsocket` in einen wartenden Zustand. Die Reihenfolge in der die drei Aufrufe von `send` erfolgen ist unerheblich, sie versehen lediglich das Objekt `albrechtsocket` mit bestimmten Zuständen.

Die Berechnung der Antwort von Meister Albrecht geschieht beim Aufruf von `input_handler` (vgl. Abbildung 7). Zunächst wird die an Prolog geschickte Zeichenkette mit `get` in ein Prolog-Atom umgewandelt, dann mit `atom_to_term` in einen Prolog-Term, der der Regel `super_sem_int` übergeben wird. Die Regel `super_sem_int` berechnet die Antwort und reicht sie an `respond` weiter.

Die Regel `respond` (vgl. Abbildung 8) hat die Aufgabe die Antwort an die Adresse zurückzuschicken von der die Daten kamen. Mit `endpoint` wird diese aus der Prolog Datenbasis gelesen. Die Antwort, die zu diesem Zeitpunkt noch ein Term ist, wird in ein Atom umgewandelt, das Zeichen `\n` wird angehängt und mit `send` wird der String an die mit `endpoint` gefundene Adresse geschickt.

```
input_handler(X) :-
     get(X,value,Y),
     term_to_atom(Z1,Y),
     super_sem_int(Z1,Z),
     write(Z),nl,
     respond(Z).
```

Abbildung 7: `input_handler`

```
respond(X):-endpoint(E),
     term_to_atom(X,S),
     atom_concat(S,'\n',R),
     send(E, format, R).
```

Abbildung 8: `respond`

Bei einem Aufruf des Programms `albrecht.pl` wird mit `:-communikation` die Kommunikation sofort gestartet, d.h. es wird auf einen Verbindungsaufbau von außen gewartet.

7.5 Beschreibung von `sem_trans`

Nachdem anhand eines einfachen Beispiels die Idee der Regel `sem_trans` verdeutlicht wurde (vgl. Abschnitt 7.2), wird sie in diesem Abschnitt in ihren Einzelheiten beschrieben. Die Sätze, die das Programm verarbeiten kann, lassen sich in drei Klassen einteilen.

1. Sätze, deren Syntaxbaum eine Präposition direkt unterhalb des Verbs anordnet und deren Verb lokalen Charakter hat (z.B. *stehen*, *liegen*, *befinden* und *sein*).
2. Sätze, bei denen ebenfalls eine Präposition direkt unterhalb des Verbs angeordnet ist, aber das Verb sich wie das Verb sehen verhält (z.B. *geben*, wenn es in der Form *es gibt* gebraucht wird).
3. Sätze, bei denen der Syntasbaum das Verb mit einem Adjektiv verbindet.

Beispiele zu den verschiedenen Satztypen finden sich in Anhang B.

Die Regel `sem_trans` berechnet eine Liste von Fakten die den Inhalt des Satzes wiedergibt. Es gibt zwei Versionen der Regel, eine ist für den ersten und zweiten Satztyp zuständig, die andere Version verarbeitet Sätze vom dritten Typ. Zuerst wird die Version für den ersten und zweiten Satztyp vorgestellt. Der Text des Hauptprogramms ist in Anhang A abgedruckt.

7.5.1 sem_trans für den ersten und zweiten Satztyp

Bei einem Aufruf von `sem_trans` (vgl. Abbildung 9) wird zunächst das Verb

```
sem_trans([SemStrukt|List2],Original):-
   vfin(V,_),
   find_predicate(V,Pred),
   find_first_arg(Arg1W,V),
   pmod(P,V),
   pn(Arg2W,P),
   find_object(Arg1W,Arg1C,[],List1,Original),
   find_object(Arg2W,Arg2C,List1,List2,Original),
   SemStrukt=..[Pred,Arg1C,Arg2C,Original].
```

Abbildung 9: sem_trans

gesucht. Das geschieht mit einem Aufruf von `vfin`. Dabei wird eine Zahl zurückgegeben, die angibt an welcher Stelle im Satz das Verb steht. Das ist notwendig, um im nächsten Schritt herauszufinden wie das Prädikat heißen soll.

7.5.2 find_predicate

Mit der Regel `find_predicate` wird das Prädikat berechnet. Das Prädikat entspricht der Präposition die am Verb hängt, wenn diese nicht genauer beschrieben wird. Der Satz „*Der Apfel liegt links neben der Banane*" enthält eine genauere Erklärung von *neben* und zwar durch *links*. Hier muss aus *links neben* das Wort *links* ausgewählt werden, da es die genauere Beschreibung ergibt. Ist das Wort *links* aus dem Satz gestrichen, dann muss statt `links(apfel, banane, 4)` das Atom `neben(apfel, banane, 4)` generiert werden. Bei der Berechnung wird ausgenutzt, dass Beschreibungen einer Präposition mit der Kante ADV (für Adverb) an der Präposition hängen (vgl. Abbildung 10). Die Adverben werden in einer

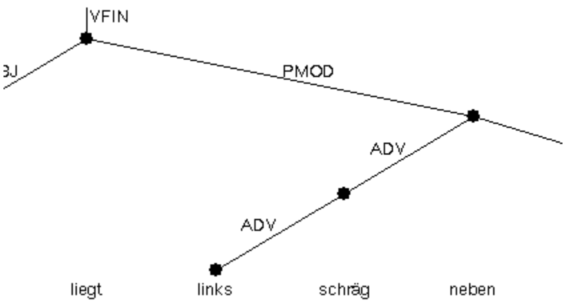


Abbildung 10: Teilbaum mit Adverben

Liste gesammelt; wenn *links* bzw. *rechts* in ihr vorkommt, wird *links* bzw. *rechts* der Prädikatname. Zum Beispiel wird aus *schräg links neben* das Wort *links* ausgewählt. Kommt weder *links* noch *rechts* in der Liste vor, so wird die Präposition selber als Prädikat verwendet. Aus *schräg neben* wird *neben*.

Die Suche ob die Präposition durch Adverben genauer beschrieben wird übernimmt die Regel `find_adv`. Zunächst werden mit `find_all_adv` alle Adverben gefunden die zu dieser Präposition gehören, diese Liste von Zahlen muss in die entsprechende Wortliste umgewandelt werden, was mit Hilfe von `num_to_wordl` geschieht. Dann wird aus dieser Liste *links* oder *rechts* ausgewählt. Ein Ausnahmefall ist hier berücksichtigt. Wenn es sich um die Präpositionen *unter* und *auf* handelt werden genauere Beschreibungen ignoriert.

„*Der Hocker steht links unter dem Tisch*" wird zu `unter(hocker, tisch, 4)`. „*Rechts auf dem Tisch liegt ein Apfel*" wird zu `auf(apfel, tisch, 4)`. Obwohl hier Information nicht berücksichtig wird und es so zu falschen Äußerungen von Meister Albrecht kommen kann, ist diese Verfahrensweise sinnvoll. Die Bilder unterscheiden sich nicht darin, dass sich Gegenstände einmal *links* und in einem anderen Bild *rechts* auf dem Tisch befinden. Auch der Hocker unter dem Tisch macht keine Außnahme. Er befindet sich zwar in fast allen Bildern *links unter* dem Tisch, in dem Bild, in dem das nicht der Fall ist befindet er sich aber *rechts neben* und nicht mehr *unter* dem Tisch. Wenn der Schüler nun den Satz „*Der Hocker steht rechts unter dem Tisch*" eingibt, antwortet der Meister zustimmend, wenn der Hocker im Originalbild links unter dem Tisch steht. Es erscheint gerechtfertigt anzunehmen, dass es unwahrscheinlich ist, dass ein Schüler etwas beschreibt, das auf keinem Bild so dargestellt ist.

7.5.3 `find_first_arg`

Im nächste Schritt wird das erste Argument des Atoms berechnet, diese Aufgabe übernimmt die Regel `find_first_arg`. Handelt es sich um einen Satz vom ersten Typ z.B. „*Der Apfel liegt auf dem Tisch*", dann steht an erster Argumentstelle das Subjekt des Satzes, hier also *apfel*. Handelt es sich um den zweiten Satztyp z.B. „*Ich sehe einen Apfel auf dem Tisch*", dann steht an erster Argumentstelle das direkte Objekt, also auch *apfel*. Die Regel nutzt dazu, dass zwei Verbtypen unterschieden werden. Es ist in Form von Fakten direkt angegeben welchen Typ ein Verb hat. Hat es lokalen Charakter wird das Prädikat `verb_loc` verwendet; verhält sich das Verb wie das Verb sehen, dann wird es mit `verb_see` gekennzeichnet. Es ist nur für sechs Verben (*stehen, liegen, sich befinden, sein, geben und sehen*) angegeben von welchem Typ sie sind.

Der nun folgende Aufruf von `pmod` und `pn` berechnet das zweite Argument des Atoms. Das ist das Wort, das über die Präposition mit dem Verb verbunden ist. Hier muss nicht zwischen erstem und zweitem Satztyp unterschieden werden. Das erste Argument von `pn` wird das zweite Argument des Atom, das generiert wird.

7.5.4 find_object

Die mit find_first_arg und pn berechneten Argumente sind noch nicht in ihrer entgültigen Form. Bis jetzt sind es noch die Zahlen durch die die Wörter ersetzt wurden. Wenn nun zunächst die Zahlen in Wörter zurückübersetzt werden, erhält man die Wörter die im Satz vorkommen und nicht die Objekte die in den Bildbeschreibungen verwendet werden. Diese Übersetzungsarbeit, insbesondere, die vom Wort zum Objekt, leistet die Regel find_object und zwar sowohl für das erste als auch für das zweite Argument. Die Regel find_object hat zwei Aufgaben. Erstens wird eine Zahl in ein Objekt umgewandelt, und zweitens werden aus Beschreibungen des Wortes, für das die Zahl steht, Fakten generiert. Wenn z.B. das Objekt apfel im Satz „*Der rote Apfel vor der Schale liegt ...*“ berechnet wird, dann generiert find_object auch die Fakten die den Apfel genauer beschreiben, hier also eigenschaft(apfel, rot, 4) und vor(apfel, schale, 4).

Die Regel find_object hat fünf Argumente. An erster Stelle steht die Zahl die in ein Objekt umgewandelt werden soll, an zweiter stelle wird das Objekt zurückgegeben. An der dritten Argumentstelle steht eine Liste, sie enthält alle bis zu diesem Zeitpunkt aus dem Satz generierten Fakten, an der vierten Stelle wird dann die Liste zurückgegeben die, die bis dahin generierten Fakten enthält, einschließlich derer, die das Objekt genauer beschreiben. An der fünften Stelle kann die Originalbildnummer oder eine Variable stehen.

Wenn find_object aufgerufen wird, wird mit num_to_word die Zahl in das entsprechende Wort umgewandelt, dann wird mit inst das Wort in ein Bildobjekt umgewandelt. Es ist angegeben für welche Objekte ein Wort stehen kann. Das Wort *Tasse* kann z.B. für das Objekt tasse1 oder tasse2 stehen. In die Datei ist also u.a.

```
inst(tasse,tasse1).
inst(tasse,tasse2).
```

eingetragen. Andererseits kommt es häufig vor, dass es mehrere Beschreibungen für ein Objekt gibt. Die Schale kann auch als Schüssel, Gefäß, Obstschüssel, Obstschale oder Fruchtschale bezeichnet werden. Aus diesem Grund sind die Instanziierungen

```
inst(schüssel,schale).
inst(gefäß,schale).
inst(obstschüssel,schale).
inst(obstschale,schale).
inst(fruchtschale,schale).
```

angegeben.

Nachdem eine Zahl in ein Bildobjekt umgewandelt wurde, ist die zweite Aufgabe der Regel find_object, für eventuelle Beschreibungen des Objekts Fakten zu generieren. Für diese Aufgabe gibt es die Regeln test_prop und test_loc.

Die Regel `test_prop` generiert aus Beschreibungen durch Adjektive die entsprechenden Fakten. Für alle Adjektive, die an einem Wort hängen wird ein Fakt der Form `eigenschaft(Objekt, Adjektiv, N)` erzeugt. Da die Adjektive nach der in Abschnitt 7.3 beschriebenen Vorverarbeitung in ihrer Grundform vorliegen können sie direkt in den Fakt eingefügt werden. Die Fakten entsprechen so den Eintragungen in den Bildbeschreibungen.

Die Regel `test_loc` verarbeitet Beschreibungen eines Objekts, die durch eine Präposition eingeleitet werden, z.B. „*Der Apfel vor der Schale liegt ...*“. Hier wird das Objekt `apfel` durch seine Position im Vergleich zur Schale beschrieben. Die Funktionsweise von `test_loc` ähnelt der von `sem_trans` sie verwendet die gleichen Regeln. Die Schale kann ihrerseits auch genauer beschrieben sein. Aus diesem Grund wird innerhalb von `test_loc` die Regel `find_object` wieder aufgerufen, das führt zu einer Rekursion bei verschachtelten Sätzen.

Im letzten Schritt von `sem_trans` wird, nachdem der Prädikatname gefunden und das erste und zweite Argument mit `find_object` in ein Bildobjekt umgewandelt wurden, mit dem von Prolog zur Verfügung gestellten Prädikat `=..` das Atom erzeugt. Anschließend wird es an die bei den Aufrufen von `find_object` generierte Liste von Fakten angehängt.

7.5.5 `sem_trans` für den dritten Satztyp

Die zweite Version der Regel `sem_trans` (vgl. Abbildung 11) verarbeitet Sätze

```
sem_trans([SemStrukt|List],Original) :-
   vfin(V,_),
   adj(Adj,V),
   subj(Sub,V),
   find_object(Sub,Sub1,[],List,Original),
   num_to_word(Adj,Adj1),
   SemStrukt=..[eigenschaft,Sub1,Adj1,Original].
```

Abbildung 11: sem_trans

bei denen ein Adjektiv unterhalb des Verbs angeordnet ist. Dieses Adjektiv bezieht sich auf das Subjekt im Satz. Es muss also ein Fakt mit Prädikatnamen `eigenschaft` generiert werden. An erster Argumentstelle des Fakts steht dann das Subjekt und an zweiter das Adjektiv. Auch in so einem Satz, können für das Subjekt weitere Beschreibungen vorkommen. Diese werden, wie in der ersten Version von `sem_trans`, mit der Regel `find_object` berechnet.

7.6 Des Meisters Antworten

Die verbale Reaktion des Meister Albrecht ist auf sehr einfache Weise realisiert. Die Prologregel `super_sem_int` berechnet hierzu, mit Hilfe der aus dem Satz generierten Fakten, zunächst ob die Beschreibung auf das Originalbild zutrifft oder nicht. Dabei wird entweder `ja`, `nein` oder `weissnicht` zurückgegeben. Die Konstante `weissnicht` wird in dem Fall zurückgegeben in dem keine Entscheidung gefällt werden kann. Für jeden der drei Antworttypen gibt es eine Liste von vorgefertigten Antworten aus der eine Antwort ausgewählt wird.

An die Kommunikationskomponente wird eine Liste mit zwei Einträgen zurückgeschickt. An der ersten Stelle der Liste steht der Antworttyp, also `ja`, `nein` oder `weissnicht` und an der zweiten Stelle steht der Antwortsatz als String.

Die Regel `super_sem_int` existiert für fünf verschiedene Fälle. Bei einem Aufruf wird ihr die von der Kommunikationskomponente geschickte Liste (siehe Abschnitt 7.3) übergeben. Diese Liste enthält den Syntaxbaum, die Wörter des Satzes und die Nummer des Originalbildes. Im ersten Fall wird diese Liste der Regel `sem_int`, die mit Hilfe der Regel `sem_trans` eine Liste von Fakten generiert, übergeben. An der dritten Argumentstelle eines jeden Fakts steht die Nummer des Originalbildes. Mit `call_list` wird dann getestet ob alle Fakten zutreffend sind, ist dies der Fall wird eine Antwort vom Typ `ja` ausgewählt.

Die zweite und dritte Version von `super_sem_int` bearbeitet den Fall in dem eine verneinende Antwort gegeben wird, die beiden Versionen unterscheiden sich nur in dem Verfahren wie eine Antwort ausgewählt wird. Bei diesen Versionen wird die Liste mit Syntaxbaum, Wörtern und Bildnummer an die Regel `sem_int` weiter gegeben indem die Bildnummer durch eine Variable ersetzt wird. Das hat zur Folge, dass die Fakten, die von `sem_int` zurückgegeben werden, an der dritten Argumentstelle eine Variable aufweisen. Bei einem Aufruf von `call_list` wird in diesem Fall also getestet ob die Fakten für irgend eines der zehn Bilder zutreffen, dabei müssen nicht alle Fakten im gleichen Bild gefunden werden, es reicht, wenn zu jedem Fakt ein Bild existiert, das dieses Merkmal aufweist. Ist das der Fall, d.h. `call_list` gelingt, so wird eine verneinende Antwort zurückgegeben.

Diese Vorgehensweise stellt sicher, dass der eingegebene Satz verstanden wurde, wenn eine Antwort vom Typ `ja` oder `nein` zurückgegeben wird, also eine Antwort, die für den Spielverlauf entscheidend ist. Die alternative Idee, immer dann, wenn die aus dem Eingabesatz generierten Fakten nicht mit der Originalbild übereinstimmen, verneinend zu antworten, hat den Nachteil, dass auch wenn ein Satz nur nicht verstanden wurde negativ geantwortet wird. Das kann unter Umständen den Spielverlauf sehr stören. Da es in dieser Sprachlernübung die Aufgabe ist, die zehn Bilder zu beschreiben, kann man davon ausgehen, dass eine Beschreibung entweder mit dem Originalbild oder mit einem der übrigen neun Bilder übereinstimmt, wenn dies nicht der Fall ist, wurde die Äußerung wahrscheinlich nicht verstanden.

Wenn keine der drei ersten Versionen der Regel `super_sem_int` gelingt kann es sein, dass zwar eine Liste von Fakten generiert wurde, diese aber nicht alle in den Bildbeschreibungen wiedergefunden werden konnten. Für diesen Fall steht die vierte Version zur Verfügung, bei der eine Antwort vom Typ `weissnicht` gegeben wird. Diese Antworten haben ausweichenden Charakter; sie sind allgemein gehalten. In dem Fall, dass keine Fakten generiert werden können, d.h. dass `sem_trans` fehl schlägt, wird mit „*Das verstehe ich nicht*“ geantwortet.

In der ersten der beiden verneinenden Versionen wird in der Antwort Bezug auf den Eingabesatz genommen. Das heißt, wenn der Schüler den Satz *„Die Schale ist grün.“* eingibt, und das ist auf dem Originalbild nicht der Fall, dann antwortet der Meister: „*Eine grüne Obstschale habe ich nie und nimmer gemalt*“, im Gegensatz zu den Standardantworten, z.B. „*Nein, so war das nicht*“. Die spezifischeren Antworten werden gegeben, indem zu bestimmten Faktenlisten Antworten vorgegeben sind.

Wenn aus einem Satz eine Liste von Fakten generiert wurde und diese Fakten nicht mit dem Originalbild übereinstimmen, dann wird eine negative Antwort gegeben. In dem Fall, in dem dann die Faktenliste einer der vorgegebenen Listen entspricht, wird die zugehörige Antwort gegeben. Für das obige Beispiel findet sich folgender Eintrag:

```
antw_nein('Eine grüne Obstschale habe ich nie und nimmer
gemalt.',[eigenschaft(schale,grün,_)]).
```

Immer wenn aus einem Eingabesatz, `eigenschaft(schale,grün,_)` generiert wird und das auf dem Originalbild nicht der Fall ist, wird mit „*Eine grüne Obstschale habe ich nie und nimmer gemalt*“ geantwortet.

Wenn keine spezielle Antwort gefunden werden kann, dann muss wie bei den Antworten vom Typ `ja` und `weissnicht` eine Antwort aus einer Liste allgemeinerer Antworten ausgewählt werder. Diese Antworten werden der Reihe nach gegeben, so wie sie in der Liste stehen. Hierzu gibt es für jeden Antworttyp einen Zähler, der immer wenn eine Antwort gegeben wurde um eins erhöht wird. Diese Zahl, modulo der Anzahl der Antworten in der Liste, gibt an, welche Antwort ausgewählt wird. Wenn die letzte Antwort in einer Liste gegeben wurde, dann ist die nächste wieder die erste. Wenn häufig Antworten vom Typ `weissnicht` gegeben werden müssen, dann wird der Meister immer gesprächiger.

7.7 Organisation des Programms

Der Programmtext ist auf 17 Dateien verteilt. Im Hauptprogramm das in der Datei **albrecht.pl** steht, sind die Regeln zu finden, die die Umwandlung von der syntaktischen in die semantische Struktur vornehmen. Regeln oder Fakten die nicht unmittelbar mit dieser Aufgabe verbunden sind, finden sich in anderen Dateien. Diese 16 Hilfsprogramme werden bei einem Start von **albrecht.pl** mitgestartet.

wiba1.pl bis wiba10.pl Diese Dateien enthalten jeweils eine der zehn Bildbeschreibungen.

allwiba.pl Information die für alle Bildrepräsentationen gilt, steht in dieser Datei. Das sind die Angaben für welche Objekte ein Wort stehen kann und die Regeln die *links* in *rechts* und *vor* in *hinter* umwandeln.

grundformen.pl Hier werden Grundform von Adjektiven und Substantiven angegeben.

antworten.pl Es wird berechnet welche Antwort eines bestimmten Typs ausgewählt wird.

antwortenlisten.pl Diese Datei enthält die Antwortenlisten für die drei Antworttypen.

socketnummer.pl Die Socketnummer wird aus der Datei `config.txt` eingelesen.

kommunikation.pl Hier sind die Kommunikationsaufrufe für die Socketkommunikation gespeichert.

7.8 Problemsammlung

Beim Durchspielen der Übung wird die Begrenzung der freien Spracheingabe sofort deutlich.

Test

Testperson: Hallo ich bin Alexis
D. Albrecht: Ich verstehe Dich nicht. Was soll denn „hallo Alexis“ heißen?
Testperson: Alexis ist mein Name
D. Albrecht: Ich verstehe Dich nicht. Was soll denn „alexis Name“ heißen?
Testperson: Was verstehst Du denn überhaupt?
D. Albrecht: Bitte stelle mir keine Fragen. Beschreibe mir ein Bild.
Testperson: Auf dem Bild gibt es viel Obst
D. Albrecht: Ich verstehe Dich nicht. Was soll denn „viel“ heißen?
Testperson: Auf dem Bild gibt es Obst
Papagei: Du hast einen Sprachfehler gemacht!
Testperson: Fehler Anzeigen
Papagei: Hier ist etwas falsch: Auf dem Bild gibt es *Obst*
Testperson: Erklären
Papagei: Grammatische Erklärung: Ein Nomen darf nicht ohne Artikel stehen.
Testperson: Lösung
Papagei: Versuch es noch einmal.

Testperson: Weiter
D. Albrecht: Das ist jetzt nicht interessant. Was siehst Du auf dem Bild.

Die hier vom Schüler eingegebenen Sätze erscheinen durchaus realistisch. Der Dialog zeigt zunächst, dass Sätze nicht verarbeitet werden können, wenn sie nicht unmittelbar im Kontext der Sprachübung stehen. Wenn die Spielanleitung nicht gelesen wurde und das Spielziel nicht klar ist, ist auch nicht zu erwarten, dass die Eingaben durch den Kontext der Übung begrenzt sind. Ein Spiel mit freier Spracheingabe lädt dazu ein, mit einer Eingabe zu beginnen, auch ohne die Spielanleitung gelesen zu haben. Nachdem der Schüler aufgefordert wurde, ein Bild zu beschreiben, gibt er, in diesem Beispiel, eine Beschreibung ein, die ihn dem Ziel des Spiels nicht näher bringen kann, das rechtfertigt die ausweichende Antwort des Meisters.

Ein weiterer Fehler wird deutlich. Die verwendete Grammatik berücksichtigt bestimmte Ausnahmen nicht, hier z.B. das Nomen in einigen Fällen auch ohne Artikel stehen können. Für den eingegebenen Satz wird fälschlicher Weise eine Fehlererklärung gegeben, zusätzlich ist die Erklärung auch noch falsch. Der CDG-Parser benötigt für einige Sätze, z.B. Sätze mit vielen Adjektiven, sehr lange um die syntaktische Struktur zu erstellen. Der Schüler muss dann einige Minuten auf eine Antwort des Meisters oder des Papageien warten. Das stört den Spielverlauf.

Die Reaktion von Meister Albrecht auf die Aussage „*Auf dem Bild gibt es Obst*“ ist vom Typ `weissnicht`. Es wird das Atom `auf(Obst, Bild, 4)` generiert. In den Bildbeschreibungen tauchen aber nur Beziehungen zwischen Gegenständen auf den Bildern auf, folglich wird eine ausweichende Antwort gegeben. Es ist leicht die Bildbeschreibungen so zu erweitern, dass Meister Albrecht in diesem Fall eine Antwort vom Typ `ja` gibt. Es muss lediglich der Fakt `auf(obst, bild, N)` für jedes der zehn Bilder eingetragen werden, das N steht für die Nummer des jeweiligen Bildes.

Durch das Ja-Nein-Weißnicht-Schema tauchen auch für Sätze die vom System verarbeitet werden können Probleme auf. Ein interessanter Dialog kann nicht entstehen. Es kommt zwar vor, dass Antworten ein tieferes Verständnis suggerieren, z.B. wenn sich der Meister über irgendeine *Geschmacklosigkeit* empört, dies geschieht aber zufällig. Es wäre wünschenswert die Antworten des Meisters so zu erzeugen, dass sie mehr auf den Satz des Schülers Bezug nehmen und dass der Dialog insgesamt eine einheitliche Form annimmt.

Literatur

[**Clocksin 1987**] W. F. Clocksin, C. S. Mellish: *Programming in Prolog*, Springer, Berlin, Heidelberg 1987.

[**Poole 1998**] David Poole, Alan Mackworth, Randy Goebel: *Computational Intelligence: A Logical Approach*, Oxford University Press, New York 1998.

[**Stockfleth 2000**] Niels Stockfleth: *Integration von CDG-Parsing und Fehlersimulation* (Diplomarbeit), Universität Hamburg, 2000.

A albrecht.pl

```
:-['grundformen.pl'].
:-['allwiba.pl'].
:-['antworten.pl'].
:-['antwortenlisten.pl'].
:-['socketnummer.pl'].

:-dynamic(adv/2).
:-dynamic(adj/2).
:-dynamic(endpoint/1).
:-dynamic(zwischen/3).

:-multifile(auf/3).
:-multifile(in/3).
:-multifile(vor/3).
:-multifile(unter/3).
:-multifile(links/3).
:-multifile(eigenschaft/3).

:-['wiba1.pl'],['wiba2.pl'],['wiba3.pl'],['wiba4.pl'],['wiba5.pl'],
  ['wiba6.pl'],['wiba7.pl'],['wiba8.pl'],['wiba9.pl'],['wiba10.pl'].

:-['kommunikation.pl'].

% Die Regel sem_int(+ListIn,-ListOut) fuegt den Syntaxbaum und
% die Woerter des Satzes in die Datenbasis. Dabei werden
% Adjektive in ihre Grundform umgewandelt.
% sem_trans wird aufgerufen.
sem_int([SyntStrukt,Lautung1,Original],SemStrukt):-
   retract_dependencies([lautung(_),det(_,_),subj(_,_),dobj(_,_),
      pmod(_,_),pn(_,_),vfin(_,_),adv(_,_),adj(_,_)]),
   assert_dependencies(SyntStrukt),
   gf(Lautung1,Lautung2),
   assert(lautung(Lautung2)),
   sem_trans(SemStrukt,Original).

% retract_dependencies(+List) loescht Altes aus der Datenbasis.
retract_dependencies([]).
retract_dependencies([D|Ds]):-
   retractall(D),
   retract_dependencies(Ds).
```

```
% assert_dependencies(+List) fuegt den Syntaxbaum in die
% Datenbasis ein.
assert_dependencies([]).
assert_dependencies([D|Ds]):-
   assert(D),
   assert_dependencies(Ds).

% sem_trans(-List,?Original) erstellt die semantische
% Struktur aus der syntaktischen.
sem_trans([SemStrukt|List2],Original):-
   vfin(V,_),
   find_predicate(V,Pred),
   find_first_arg(Arg1W,V),
   pmod(P,V),
   pn(Arg2W,P),
   find_object(Arg1W,Arg1C,[],List1,Original),
   find_object(Arg2W,Arg2C,List1,List2,Original),
   SemStrukt=..[Pred,Arg1C,Arg2C,Original].

sem_trans([SemStrukt|List],Original) :-
   vfin(V,_),
   adj(Adj,V),
   subj(Sub,V),
   find_object(Sub,Sub1,[],List,Original),
   num_to_word(Adj,Adj1),
   SemStrukt=..[eigenschaft,Sub1,Adj1,Original].

% find_predicate(+Verb,-Predicate) berechnet das Praedikat.
find_predicate(Verb,Pred):- pmod(Z,Verb), not(find_adv(_,Z)),
       num_to_word(Z,Pred).
find_predicate(Verb,Adv):- pmod(Z,Verb), find_adv(Adv,Z).

% find_adv(-Predicate,+Preposition) waehlt Adverbien als Praedikat aus.
find_adv(unter,Z) :- num_to_word(Z,unter).
find_adv(auf,Z) :- num_to_word(Z,auf).
find_adv(Adj,Z) :-
   find_all_adv(Z,List1),
   num_to_wordl(List1,List2),
   preferred_adv(List2,Adj).

% find_all_adv(+Preposition,-AdvList) sammelt alle Adverbien auf, die an
% der Praeposition haengen.
find_all_adv(X,[Y|L]) :- adv(Y,X), find_all_adv(Y,L).
```

```
find_all_adv(X,[]) :- not(adv(_,X)).

% preferred_adv(+List,-Adverb) berechnet das bevorzugte Adverb.
preferred_adv(List,rechts) :- member(rechts,List).
preferred_adv(List,links) :- member(links,List).

% find_first_arg(-W,+Z) berechnet das erste Argument.
find_first_arg(W,Z) :- subj(W,Z),num_to_word(Z,Wort),verb_loc(Wort),
       not(adj(_,Z)).
find_first_arg(W,Z) :- dobj(W,Z),num_to_word(Z,Wort),verb_see(Wort).

% find_object(+Z,-W,+ListIn,-ListOut,?Originalbild) wandelt
% eine Zahl in ein Bildobjekt um und generiert aus
% Beschreibungen desselben die entsprechenden Fakten.
find_object(Z,C,ListIn1,ListOut,Original) :-
   num_to_word(Z,W),
   inst(W,C),
   test_prop(Z,C,ListIn1,ListIn2,Original),
   test_loc(Z,C,ListIn2,ListOut,Original).
find_object(Z,W,ListIn1,ListOut,Original) :-
   num_to_word(Z,W),
   not(inst(W,_)),
   test_prop(Z,W,ListIn1,ListIn2,Original),
   test_loc(Z,W,ListIn2,ListOut,Original).

% test_prop(+Z,+C,+ListIn,-ListOut,?Original) generiert
% fuer adverbiale Beschreibungen des Objekts die Fakten.
test_prop(Z,C,ListIn,ListOut,Original) :-
   findall(A,adj(A,Z),L),
   num_to_wordl(L,L1),
   gen_prop(C,Original,L1,List),
   append(List,ListIn,ListOut).
test_prop(Z,_,List,List,_) :- not(adj(_,Z)).

% gen_prop(+C,?Original,+ListIn,-ListOut) erstellt das
% Praedikat eigenschaft.
gen_prop(_,_,[],[]).
gen_prop(C,Original,[X|L],[eigenschaft(C,X,Original)|List]) :-
   gen_prop(C,Original,L,List).

% test_loc(+Z,+C,+ListIn,-ListOut,?Original) generiert
% fuer lokale Beschreibungen des Objekts die Fakten.
test_loc(Z,_,ListIn,ListIn,_) :- not(pmod(_,Z)).
```

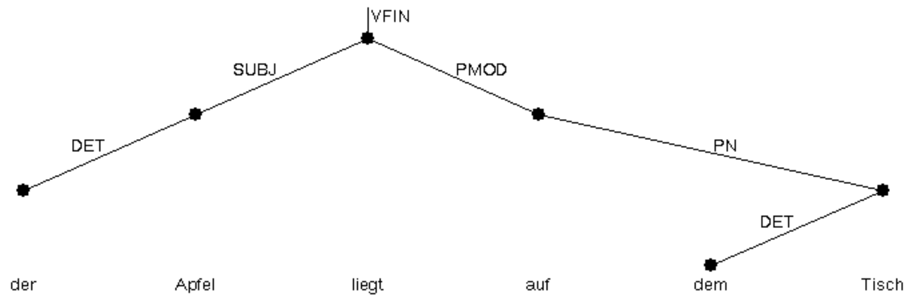
```
test_loc(Z,C,ListIn,[T|ListOut],Original) :-
   find_predicate(Z,Pred),
   pmod(N,Z),
   pn(O,N),
   find_object(O,O1,ListIn,ListOut,Original),
   T=..[Pred,C,O1,Original].

% num_to_word(+X,-Y) ordnet einer Zahl X das entsprechende
% Wort Y zu.
num_to_word(X,Y) :- lautung(L),nth1(X,L,Y).

% num_to_wordl(+ListIn,-ListOut) wandelt eine Liste
% von Zahlen in die Liste der entsprechenden Woerter.
num_to_wordl([],[]).
num_to_wordl([X|Rest1],[Y|Rest2]) :- num_to_word(X,Y),
       num_to_wordl(Rest1,Rest2).

verb_loc(steht).
verb_loc(liegt).
verb_loc(liegen).
verb_loc(befindet).
verb_loc(ist).
verb_loc(sind).

verb_see(gibt).
verb_see(sieht).
verb_see(sehe).

% call_list(+List) ein erweitertes call.
call_list([]).
call_list([X|List]) :- call(X), call_list(List).

% super_sem_int(+ListIn,-ListOut) wandelt die syntaktische
% Struktur in eine Antwort des Meister Albrecht um.
super_sem_int([X,Y,Z],[ja,Ja]) :-
   sem_int([X,Y,Z],Sem),
   call_list(Sem),
   antworten(Ja,ja),
   write(Sem), nl.

super_sem_int([X,Y,_],[nein,Nein]) :-
   sem_int([X,Y,_],Sem),
   call_list(Sem),
```

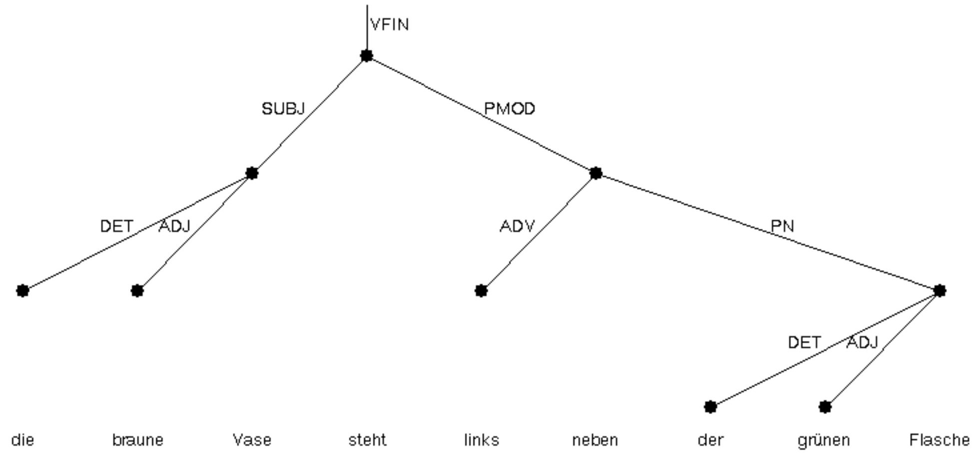
```
   antw_nein(Nein,Sem),
   write(Sem), nl.

super_sem_int([X,Y,_],[nein,Nein]) :-
   sem_int([X,Y,_],Sem),
   call_list(Sem),
   antworten(Nein,nein),
   write(Sem), nl.

super_sem_int([X,Y,_],[weissnicht,WN]) :-
   sem_int([X,Y,_],Sem),
   antworten(WN,wn),
   write(Sem), nl.

super_sem_int([X,Y,_],[weissnicht,'Das verstehe ich nicht.']) :-
   not(sem_int([X,Y,_],_)).
```

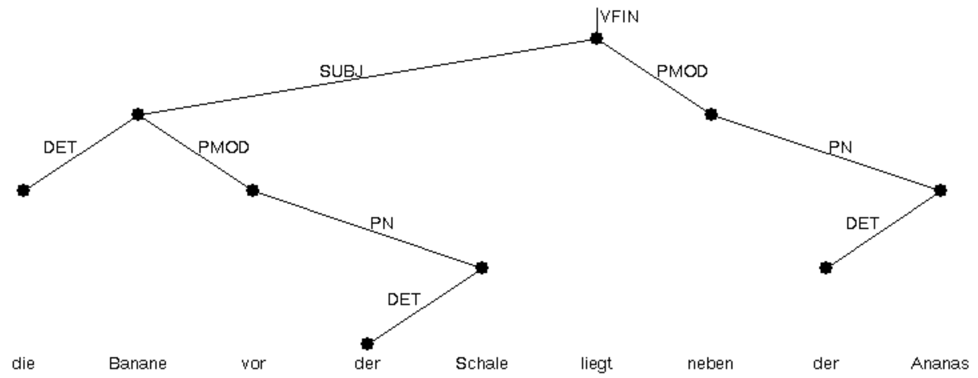
B Syntaxbäume

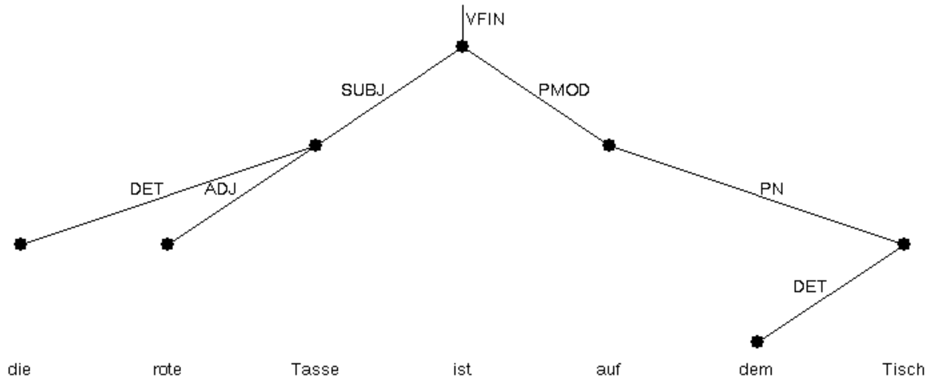
Beispielsätze mit zugehörigem Syntaxbaum.

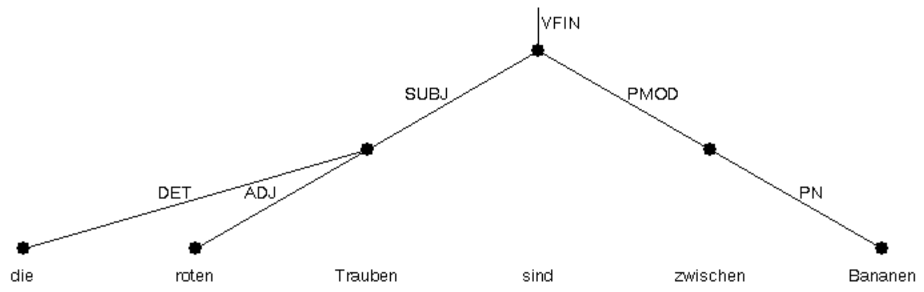
B.1 Erster Satztyp

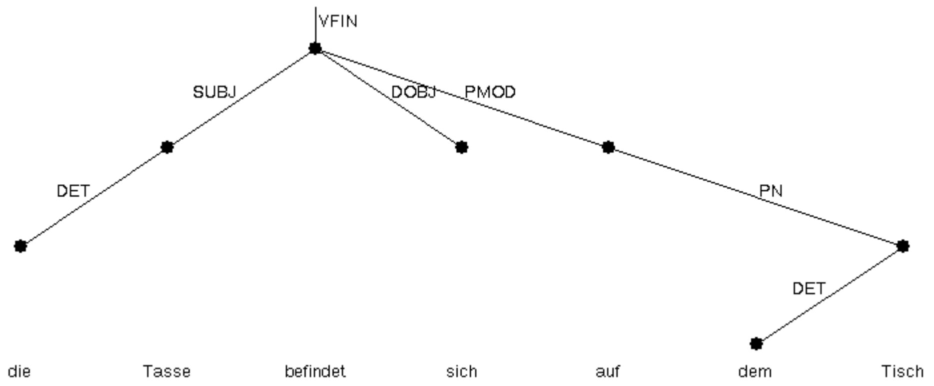


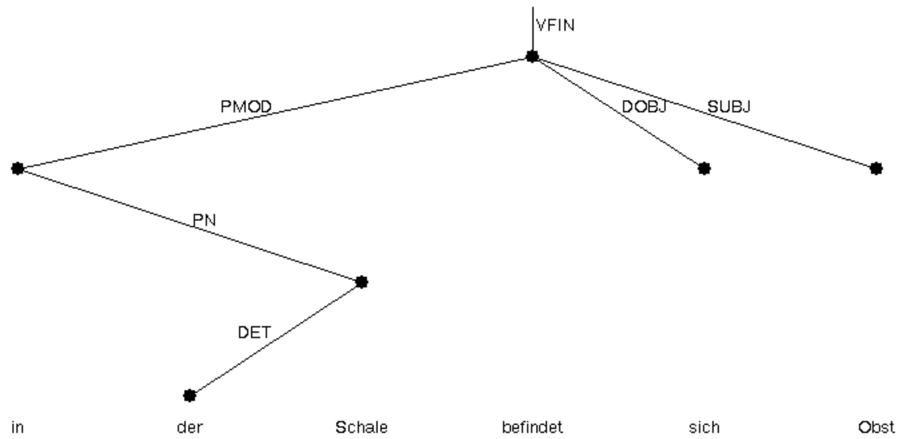

VFIN
SUBJ
PMOD
DET
ADJ
ADV
PN
DET
ADJ
die braune Vase steht links neben der grünen Flasche


VFIN
SUBJ
PMOD
DET
PMOD
PN
PN
DET
DET
die Banane vor der Schale liegt neben der Ananas

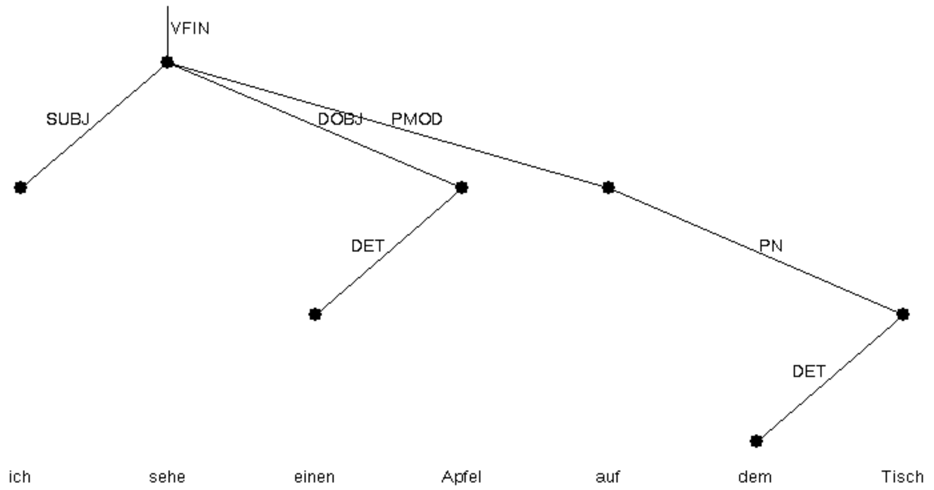

VFIN
SUBJ
PMOD
DET
ADJ
PN
DET
die rote Tasse ist auf dem Tisch

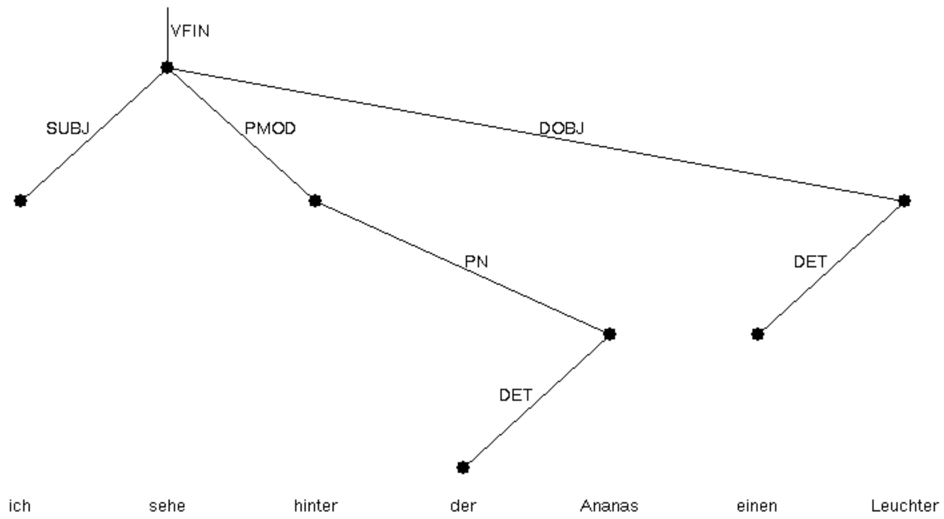

VFIN
SUBJ
PMOD
DET
ADJ
PN
die
roten
Trauben
sind
zwischen
Bananen

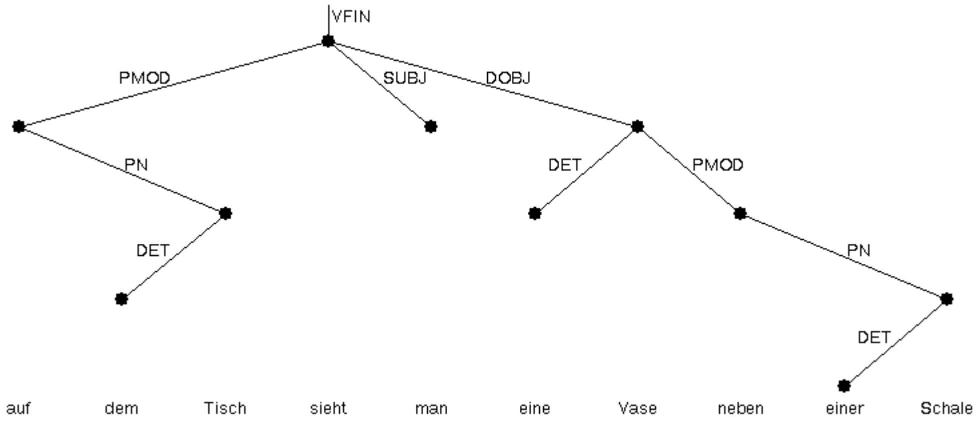

VFIN
SUBJ
DOBJ
PMOD
DET
PN
DET
die
Tasse
befindet
sich
auf
dem
Tisch

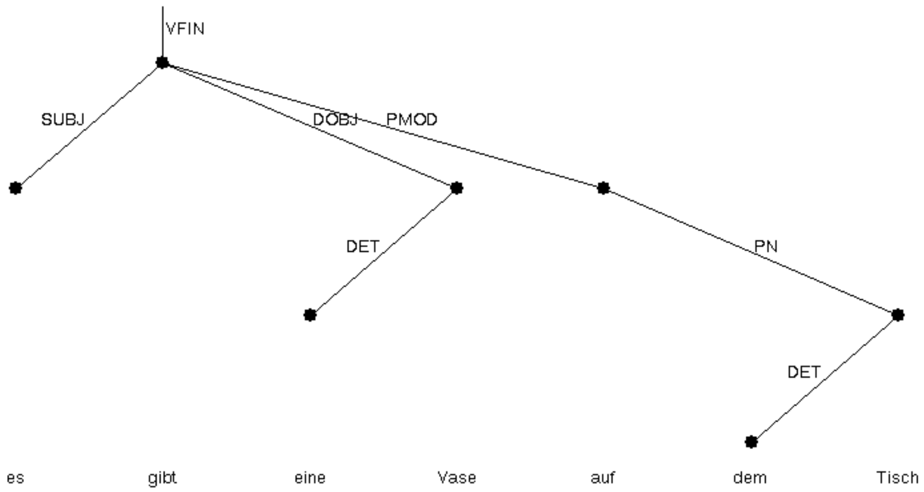

VFIN
PMOD
DOBJ
SUBJ
PN
DET
in
der
Schale
befindet
sich
Obst

B.2 Zweiter Satztyp

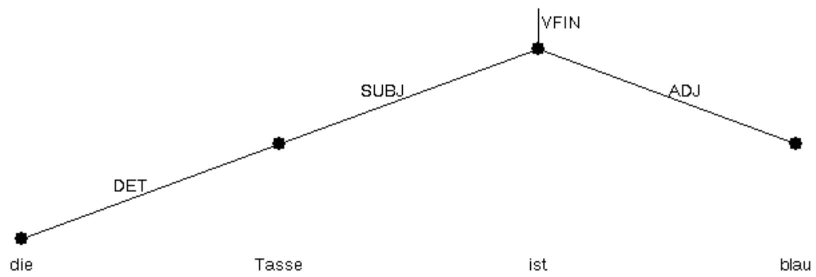


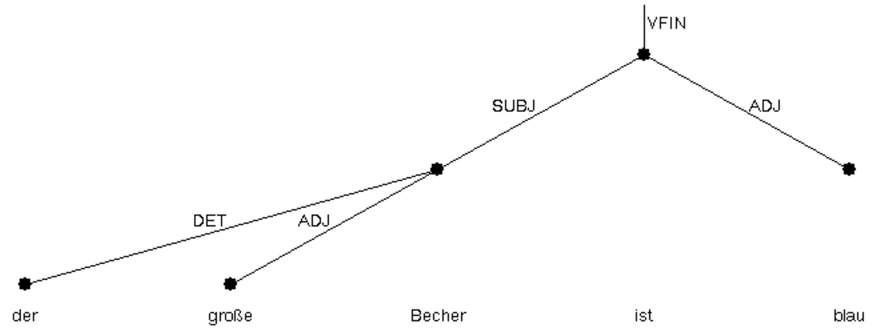


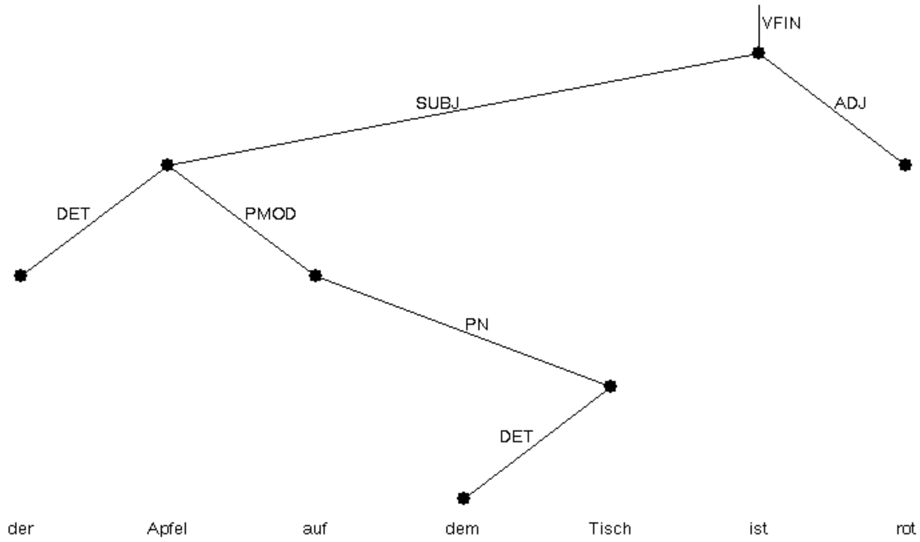




B.3 Dritter Satztyp




VFIN
SUBJ
ADJ
DET
ADJ
der
große
Becher
ist
blau


VFIN
SUBJ
ADJ
DET
PMOD
PN
DET
der
Apfel
auf
dem
Tisch
ist
rot