Einführung in natürlichsprachliche Textgenerierung

Jörg Roth

Seminar „Natürlichsprachliche Systeme“, Universität Kaiserslautern, 1989

1 Einleitung und historische Übersicht

1.1 Einleitung

Natürlichsprachliche Generierung kann nie losgelöst von der Umgebung betrachtet werden. In der Regel bildet sie mit der Sprachanalyse ein Interface zu einer Datenbasis oder einem Expertensystem. Als solches dient es als einfache Eingabeeinheit, um auch für den Laien den Umgang mit solchen Systemen zu ermöglichen. Als zweites Ziel sollen mit natürlichsprachlicher Generierung und Analyse intelligente Denkprozesse des Menschen nachgeahmt werden, um so Einblicke in die Vorgehensweise von Sprachverstehen zu gewinnen.

1.2 Historische Übersicht über Generierungssysteme

Erste Phase: einfache Textmanipulationen auf der Basis von Mustern:

1963: BASEBALL: Interface zu den Baseballdaten der amerikanischen Baseballiga. Sehr geringer Wortschatz. (Green)
1963: SAD SAM: Eingabe von Verwandtschaftsbeziehungen, antwortete auf Fragen. (Lindsay)
1965: SYNTHETIK: Antwortete auf Fragen zu einem gegebenen Text durch Textvergleich mithilfe eines Punktesystems. (Simmons)
1966: ELIZA: Simuliert eine Sitzung bei einem Psychiater auf der Basis einfacher Textmanipulationen. (Weizenbaum)

Zweite Phase: Wissen jetzt nicht mehr in Texten abgelegt sondern in Fakten und Regeln:

1972: LUNAR: Interface zu der Datenbank über die Mondprobensammlung der Apollo 11 Mission. Syntaktische Analyse über ATN-Grammatik. (Woods)
1972: SHRDLU: Abfrage und Manipulation einer Bauklötzchenwelt. (Winograd)
1975: PARRY: Simuliert einen Paranoiden in einem Psychiatergespräch. (Bestand "modifizierten" Turingtest) (Colby)
1975: PLANES: Interface zu der Datenbank der US Navy. (Waltz)
1977: ROBOT: Erstes kommerzielles Frage-Antwort-System. (Harris)
1982: VIE-LANG: Dialogsystem in deutscher Sprache. Semantische Netze als Wissensrepräsentation. (Buchberger)
1983: HAM-ANS: Dialogsystem in deutscher Sprache (simuliert beispielsweise einen Hotelmanager). (Hoeppner)
1983: MUMBLE: Textgenerierung mit Schwerpunkt how to say it. (Mc Donald)
1985: TEXT: Textgenerierung mit Schwerpunkt what to say (Mc Kewn)

2 Frage-Antwort-Systeme

2.1 Einleitung

Frage-Antwort-Systeme waren die ersten sprachverarbeitende Programme. Während man im allgemeinen Fall dauernd den Gesprächskontext aufrechterhalten muß, um passende Antworten zu geben, kann dies bei Frage-Antwort-Systemen entfallen. Sie sind lediglich dazu da, um natürlichsprachliche Anfragen in eine Datenbankanfrage umzuwandeln, diese durchzuführen, und die Ergebnisse dem Benutzer in natürlichsprachlicher Form anzugeben. Obwohl diese Systeme keine reinen Generierungssysteme sind, lohnt es sich sie näher zu betrachten, da sie Probleme aufzeigen, die auf kompliziertere Systeme übertragen werden können.

2.2 Prozedurale Semantik am Beispiel von LUNAR

2.2.1 Syntax von LUNAR

LUNAR ist das natürlichsprachliche Interface zu der Mondgesteinsammlung der Apollo 11 Mission. Es benutzt einen ATN-Parser der einen großen Wortschatz für englische Fragen besitzt. Die Ausgabe des Parsers ist noch nicht eine Datenbankanfrage. Er erzeugt vielmehr eine Zwischenrepräsentation, die sehr nah an der logischen Form ist, und als Programm ausgeführt werden kann. Diese Methode, nämlich Anfragen nicht direkt, sondern in eine ausführbare Zwischenrepräsentation umzuwandeln, nennt man *prozedurale Semantik*. Neben dieser Methode gibt es noch logikbasierte Systeme, die auf prolog-ähnlichen Sprachen aufbauen.
In LUNAR gibt es Relationen, Typen, Terme und Quantoren. Quantoren können geschachtelt werden. Ein quantifizierter Ausdruck sieht allgemein so aus:

```
(FOR quantor var typ restriktion;formel)
```

wobei folgende Quantoren erlaubt sind:

```
DEF/SING oder THE
SOME
EVERY, EACH oder ALL
```

aber auch numerische Quantoren wie

```
(GREATER-THAN n)
(LESS-THAN n)
(EXACTLY n)
```

Als Formel bzw. Restriktion wird ein Prädikat oder wieder ein quantifizierter Ausdruck eingesetzt. Zusätzlich kann die Formel eine Prozedur wie PRINTOUT sein.
Beispielsweise wird die Anfrage

"Liste alle Proben, die Silizium enthalten"

in die Form

```
(FOR ALL s SAMPLE (CONTAIN s SILICON);
                  (PRINTOUT s))
```

umgewandelt. Komplizierter sind geschachtelte Ausdrücke. Zum Beispiel wird die Anfrage

"Liste das Gewicht aller Proben, die Silizium enthalten"

zu

```
(FOR ALL s SAMPLE (CONTAIN s SILICON);
                  (FOR THE w WEIGHT (WEIGHS s w);
                                    (PRINTOUT w)))
```

2.2.2 Die Abarbeitung von Quantoren

An einem Beispiel soll die Abarbeitung von Quantoren aufgeführt werden und zwar bei

```
(FOR DEF/SING var typ restriktion;formel)
```

Folgende Schritte werden ausgeführt:

- Finde alle Variablen des angegebenen Typs, für die die Restriktion erfüllt ist
- Ist die Anzahl der gefundenen Lösungen 1, dann führe formel aus mit var an den entsprechenden Wert gebunden.
- Ist die Anzahl 0 oder größer 1 dann schlägt die Anfrage fehl.

Natürlich kann dieser Ablauf zur Effizienzsteigerung modifiziert werden. So müssen beispielsweise nicht alle Bindungen erfasst werden, wenn man weiß, daß schon ab der zweiten die Anfrage nicht mehr erfüllt werden kann.
Entsprechend werden die Quantoren im obigen Beispiel bearbeitet. Der ALL-Quantor wird ausgeführt, indem erst einmal alle Proben generell zusammengefasst werden. Dann werden die behalten, auf die die Restriktion zutrifft. Das sind alle, die Silizium enthalten. Als nächstes wird die Formel ausgewertet. Der THE-Quantor ermittelt jetzt "alle" Gewichte einer Probe. Da es (erfahrungsgemäß) immer nur ein Gewicht zu einer Probe gibt, ist der THE-Quantor immer erfolgreich, was zur Ausführung der Formel führt, und die Gewichte werden ausgegeben.
Zusätzlich zu den Prädikaten enthält LUNAR Funktionen, die auf Mengen angewendet werden wie AVERAGE, NUMBER, SUM. Sie erlauben Fragen wie "Wie lautet das durchschnittliche Gewicht aller Proben" oder ähnliches.
Obwohl die Antwort, wenn einmal die prozedurale Form der Frage vorliegt, einfach zu ermitteln ist, gibt es doch beträchtliche Probleme, wenn man intelligente Antworten fordert.

2.3 Intelligente Antworten

2.3.1 Informative Ausgaben

Wir haben gesehen, daß Antworten relativ einfach zu ermitteln sind. Erwartet man jedoch intelligente Antworten, so müssen noch weitere Vorkehrungen getroffen werden. Betrachten wir folgendes Beispiel:

"Was ist das Gewicht jeder Probe ?"

Dieser recht einfache Satz wird wie folgt umgewandelt:

```
(FOR EACH s SAMPLE;
     (FOR THE w WEIGHT (WEIGHS s w);
                       (PRINTOUT w)))
```

Führt das System diese Anfrage durch, würde man folgende Ausgabe erhalten:

20g
22g
18g
54g
...

Diese Ausgabe ist wertlos, da der Bezug zu der jeweiligen Probe fehlt. Die Konstukteure von LUNAR haben das Problem folgendermaßen behoben:
Die Prozedur PRINTOUT wurde so umgestaltet, daß sie bei jeder Variable, die sie ausgeben soll prüft, von welcher Variable sie abhängig ist. Im letzten Fall war die Variable w durch den Ausdruck (WEIGHS s w) von s abhängig. PRINTOUT stellt also fest, s mitanzuzeigen, wir erhalten folgendes Ergebnis:

SI001 20g
SI002 22g
SI003 18g
SI004 54g
... ...

2.3.2 Probleme mit nein-Antworten

Viele Probleme mit Antworten kommen daher, daß natürlichsprachliche Fragen nicht immer wörtlich aufzufassen sind ("Kannst Du mir sagen wieviel Uhr es ist"). Auf der anderen Seite muß der Antwortende erkennen worauf die Frage abzielt, und gegebenenfalls zusätzliche Informationen liefern.
Eine ganze Klasse von problematischen Antworten sind jene auf ja-nein-Fragen mit negativem Ausgang. Dazu ein Beispiel: ein Frage-Antwort-System auf einem Flugplatz bekommt folgende Frage gestellt:

"Kommt der Flug von Sidney nach Rom um 15 Uhr an ?"

Angenommen das System antwortet mit einem schlichten "nein". Das kann mehrere Gründe haben:

- Der Flug von Sidney nach Rom kommt zu einer anderen Zeit als 15 Uhr an.
- Es existieren keine Flüge von Sidney nach Rom.
- Es existieren keine Flüge.

Obwohl der dritte Fall unwahrscheinlich ist, erwartet man in den anderen Fällen doch informativere Antworten wie zum Beispiel "nein, aber um 16 Uhr" oder "nein, es geht kein Flug von Sidney nach Rom". Andererseits handelt das System die Anfrage nach rein logischen Gesichtspunkten ab. Die Frage ist: wie kann das System erkennen, wie es geschickt zu antworten hat ?

Eine einfache Technik ist es, bei negativer Antwort die Anfrage zu wiederholen, jedoch nachdem man einen Teil entfernt hat.
Beispielsweise sieht die letzte Anfrage (in Prolog-ähnlicher Notation) folgendermaßen aus:

```
Antwort <- (typ ?f Flug)
           (verbindet ?f Sidney Rom)
           (Ankunft ?f Rom 15Uhr)
```

Schlägt nun diese Anfrage fehl, so wird sie wiederholt, jedoch ohne den letzten Teil.

```
Antwort <- (typ ?f Flug)
           (verbindet ?f Sidney Rom)
```

Schlägt das wieder fehl, so kann das System "Es gibt keine Flüge von Sidney nach Rom" antworten. Natürlich könnte das System noch weitergehen und die Anfrage

```
Antwort <- (typ ?f Flug)
```

auswerten, was zu dem unerwarteten Ergebnis "Es gibt keine Flüge" führen könnte.
Eine andere Strategie wäre es, bei nein-Antworten die Anfrage komplett zu wiederholen, jedoch mit einer Konstanten ersetzt durch eine existenzquantifizierte Variable. In unserem Beispiel wäre das:

```
Antwort <- (typ ?f Flug)
           (verbindet ?f Sidney Rom)
           (Ankunft ?f Rom ?t)
```

Was zu der informativen Antwort "nein, um 16 Uhr" führen könnte.
Man kann diese Technik mit der vorigen koppeln, indem man Teile weglässt, und Konstanten durch Variablen ersetzt. Diese Strategie könnte

```
Antwort <- (typ ?f Flug)
           (verbindet ?f Sidney ?a)
```

auswerten, und zu dem Schluß kommen "nein, es gehen keine Flüge von Sidney aus".
So einfach diese Techniken vielleicht anmuten, so ist es jedoch immer ein Problem zu entscheiden, welche Teile man wegläßt, oder welche Konstanten man ersetzt.

2.3.3 ja-Antworten und Mißverständnisse

Wie schon angesprochen, ist die Intension einer ja-nein-Frage nicht immer, den Wahrheitswert einer Aussage festzustellen. Beispielsweise soll ein System jemanden instruieren, in ein Haus zu gehen und dort Dinge zu holen. Der Benutzer fragt:

"Komme ich überhaupt in das Haus hinein ?"

Das System könnte mit "ja" antworten, wenn es weiß, daß alle Türen verschlossen sind mit Ausnahme der hinteren Kellertür. Diese Antwort ist jedoch nicht sinnvoll, da sie beim Benutzer einen falschen Eindruck erweckt. Um jedoch eine informative Antwort zu geben ist ein umfassendes Bild vom Wissen des Benutzers nötig. Das System sollte also anhand einer Frage erkennen, was für Informationen verlangt werden. Genauso sollte eine System erkennen, inwieweit ein Benutzer Fragen unter falschen Annahmen stellt. Beispielsweise nimmt der Benutzer im letzten Beispiel an, daß die Dinge in einem bestimmten Zimmer zu finden sind. Er fragt deshalb:

"Die vordere Treppe führt doch ins Wohnzimmer ?"

Das System weiß nun, daß die Dinge nicht durch die Treppe zu erreichen sind. Jetzt einfach mit "ja" zu antworten, würde den Benutzer in seinem Mißverständnis noch bestärken. Das

System muß also, um Mißverständnisse zu erkennen, alle Schritte des Benutzers wissen, die zur Vervollständigung seines Ziels nötig sind.

3 Generierungssysteme

3.1 Einleitung

Im letzten Kapitel haben wir Systeme untersucht, die Fragen zu gegebenen Wissensbasen beantworteten. Sie zeichneten sich durch folgende Merkmale aus:

- Antwort nur von der jeweiligen Frage abhängig
- nur kleine Antworten (ein Satz oder eine Phrase)
- kleine "Welten" (z.B. Welt der Mondproben)
- mögliche Antworten wurden in das System hineinkodiert

In diesem Kapitel werden Generierungssysteme beschrieben, die der Antwortmöglichkeit eines Menschen schon eher entsprechen:

- Antwort auch vom Vorgespräch und Benutzer abhängig
- komplizierte Antworten (bis zu mehreren Sätzen)
- kompliziertere Welten
- flexible Antworten möglich

Natürlichsprachliche Sprachgenerierung ist als solche invers zu Sprachanalyse. Aus einer gegebenen Repräsentation soll ein Text generiert werden, der linguistischen Regeln entspricht. Betrachten wir einmal folgende Sätze:

1:Der Mann mit dem Hut steigt in das rote Auto.
2:Der Mann mit dem Hut steigt in das Auto, das rot ist.
3:Der Mann, der einen Hut trägt, steigt in das rote Auto.
4:Das Auto, in das der Mann mit dem Hut steigt, ist rot.
5:Der Mann, der in das rote Auto steigt, trägt einen Hut.

Obwohl alle Sätze das gleiche aussagen, unterscheiden sie sich doch mehr oder weniger in der Betonung. Trotzdem würden wohl die meisten Analysesysteme alle Unterschiede beim Verstehen wegabstrahieren. Ein Generierungssystem sollte allerdings wissen, welchen der Sätze es für die Beschreibung der Situation wählen sollte. Beispielsweise wird mit Satz 3 und 5 der Mann beschrieben, während mit Satz 4 das Auto im Blickpunkt steht. Satz 1 und 2 unterscheiden sich dagegen nur leicht in der Form. Ein Generierungsprogramm muß anhand der Sätze die es schon vorher ausgegeben hat und anhand des Gesprächsverlaufs erkennen, welche Randbedingungen es betrachten soll. Folgende Kriterien sind von Bedeutung:

- Was steht im Mittelpunkt?
- Wie wandelt man Sprachaktionen in Worte und Phrasen um?
- In welcher Reihenfolge werden Sätze oder Phrasen ausgegeben?
- Welche Zeitform soll benutzt werden?
- Soll der Satz in Aktiv oder Passiv sein?

Allgemein sind folgende Teilschritte bei allen Generierungssystemen vorzufinden:

- Am Anfang steht eine Intension. Das heißt, das System will eine Aufgabe erfüllen, die eine Sprachausgabe erfordert.
- Dann wird versucht, diese Aufgabe in Sprachaktionen zu unterteilen. Infrage kommen dafür z.B. INFORM, IDENTIFY, REQUEST usw. Diese Aktionen kann man sich als Funktionen vorstellen, die als Argumente Objekte aufnehmen, aber auch komplexe

Strukturen wie ganze Wissensbasen. Dieser Schritt wird allgemein als *"what to say"* bezeichnet.
- Als letztes versucht man lexikalische und syntaktische Entscheidungen zu treffen um so den eigentlichen Text zu erzeugen. Dieser Schritt wird *"how to say it"* genannt.

3.2 Was soll gesagt werden (what to say)

3.2.1 Erzeugen der Sprachaktionen

Ein Problem ist es, zu entscheiden welche Objekte man wie beschreibt. Es ist nämlich keineswegs selbstverständlich, daß eine eindeutige Beschreibung für ein Sprachsystem auch eindeutig für den Benutzer ist. Genauso verhält es sich bei ganzen Handlungsabläufen. Soll zum Beispiel ein System beim Zusammensetzen einer HIFI-Anlage behilflich sein, so könnte es auf die Frage "Wie verbinde ich Verstärker und Plattenspieler ?" folgenden Satz produzieren:

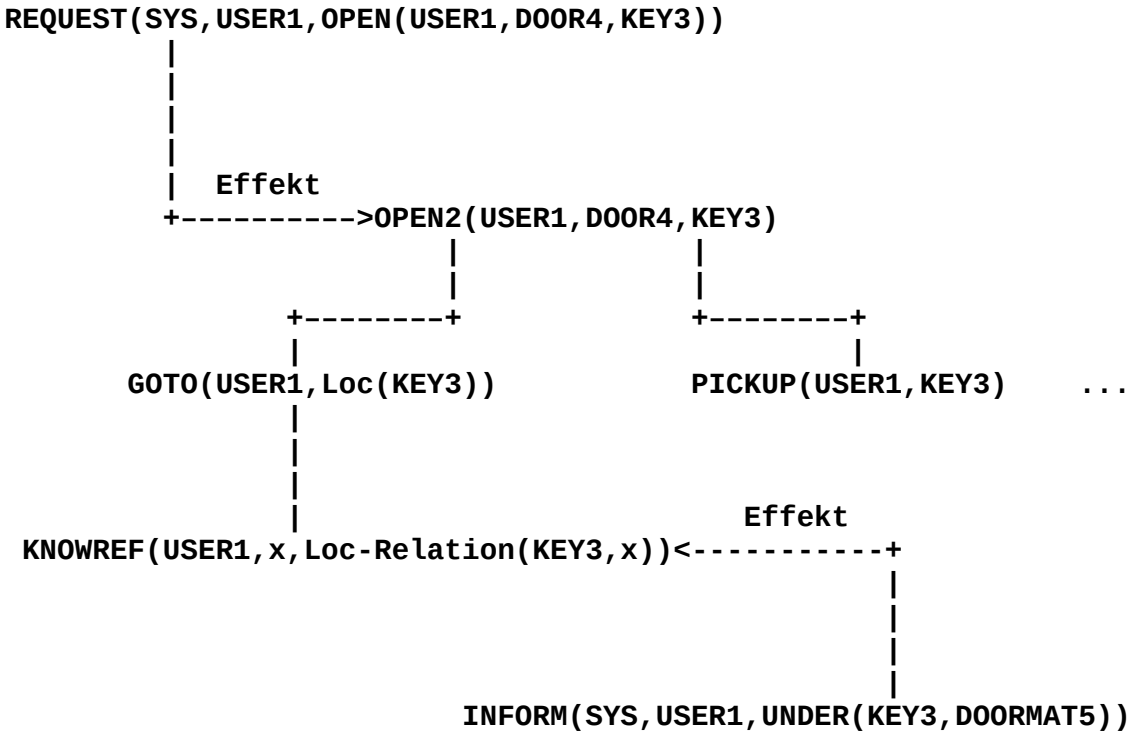
"Nehme das Kabel, das auf der einen Seite einen 5-poligen Stecker hat und stecke ihn in die Buchse auf der Rückseite des Verstärkers, die die Beschriftung "IN" trägt, dann ..."

Diese Beschreibung hilft dem Laien. Einen Fachmann würde sie allerdings langweilen. Würde er die Frage stellen, so zielte sie auf etwas ganz anderes ab. Hier wäre vielleicht die Antwort

"Das entsprechende Kabel liegt in der obersten Schublade"

angemessen. Ein System benötigt demnach zwei Informationen, um adäquate Antworten zu generieren:
- Ein Modell des Benutzers
- Ein Modell des Unterhaltungskontextes

Das Problem beim ersten Teilschritt der Generierung ist, welche Sprachaktionen mit welchen Argumenten ausgewählt werden sollen.
Angenommen, ein System will einen Benutzer anweisen, eine Tür mit einem Schlüssel zu öffnen. Ein Plan um die Sprachaktionen auszuwählen könnte so aussehen:



Die notwendigen Sprachaktionen festzustellen, ist nicht immer einfach zu ermitteln, da wie man in diesem Beispiel sieht, die Kenntnis aller Handlungen benötigt wird. In einfachen Fällen kann man jedoch auf einfachere Methoden übergehen.

3.2.2 Skriptartige Strukturen

Ein Weg, das Problem zu lösen, ist komplexe Sprachaktionen schon vorher ins System in Form von Skripten abzulegen. Zum Beispiel könnte ein Skript, um verschiedene Klassen von Objekten zu identifizieren, folgendermaßen aussehen:

```
Actionclass IDENTIFY(class):
Decompositon: IDENTIFY-CLASS(class,superclass) ->
              IDENTIFY-PROPS(class,superclass) ->
              GIVE-EXAMPLE(class)
```

Jede dieser Teile kann selbst wieder skriptartig aufgebaut sein.
Angenommen, es läge folgende Wissensbasis vor:

```
Hund(x)->Tier(x)
Säugetier(x)->AnzahlBeine(x,4)
Hund(x)->bellt(x)
Hund(x)->Haustier(x)
Seehund(x)->bellt(x)
Katze(x)->Haustier(x)
Hund(Fido)
```

Dann würde die Sprachaktion INFORM(Hund) folgende Ausgabe erzeugen:

"Hunde sind Tiere. Sie bellen. Sie sind Haustiere. Fido ist ein Hund"

Und zwar von der Aktion IDENTIFY-CLASS die Angabe "Hunde sind Tiere", von IDENTIFY-PROPS "Sie bellen. Sie sind Haustiere", und von GIVE-EXAMPLE "Fido ist ein Hund".

3.3 Wie soll etwas gesagt werden (how to say it)

3.3.1 Lexikalische Wahl

Eine Menge von Aussagen, die zum Beispiel durch Frames oder ein semantisches Netz gegeben sind können auf viele Arten ausgedrückt werden, gelenkt durch syntaktische und lexikalische Wahl. Hat man zum Beispiel die Aussage PASST(SCHLÜSSEL3,TÜR4), dann hängt es von der syntaktischen Wahl ab, ob man daraus einen vollständigen Satz macht, nämlich "Der Schlüssel paßt zur Tür" oder die Nominalphrase "der Schlüssel, der zur Tür paßt".
Bei der lexikalischen Wahl steht man vor einem Problem: viele Worte beschreiben zwar dieselbe Aktion benutzen aber verschiedene Konstruktionen.
Liegt beispielsweise die Aussage KAUF(JACK1,SUE2,BUCH5) vor, so kann die Aktion durch folgenden Sätze realisiert werden, je nachdem ob man des Verb "verkaufen" oder "erwerben" für die Aktion auswählt:

"Jack erwirbt ein Buch von Sue"
"Sue verkauft ein Buch an Jack"

Die Wahl, ob man nun den einen oder anderen Satz nimmt, könnte im schlimmsten Fall zufällig erfolgen. Besser wäre es allerdings, wenn man Entscheidungsmechanismen besitzt.

a) Objekte in den Mittelpunkt stellen (focus)

Häufig wird im Lauf einer Sprachaktion ein bestimmtes Objekt in den Mittelpunkt gestellt. Solch ein Handlung wird *focusing* genannt. Wenn so ein Mechanismus vorhanden ist, kann man Entscheidungen über die lexikalische Wahl treffen. Wäre im letzten Beispiel Jack der Mittelpunkt der Aktion, so würde man den ersten Satz wählen. Die Regel für dieses Beispiel sieht so aus:

```
K1:KAUF  ----------------> erwerben  mit AGENT=Käufer(K1)
                                       THEMA=Objekt(K1)
                                       HERKUNFT=Verkäufer(K1)

         ----------------> verkaufen mit AGENT=Verkäufer(K1)
                                       THEMA=Objekt(K1)
                                       BESTIMMUNG=Käufer(K1)
```

b) Bewertung von Eigenschaften

Eine zweite Möglichkeit ist, in die lexikalische Wahl die Eigenschaften des betreffenden Objektes einfließen zu lassen. Ein Beispiel ist das Wort "einnehmen". Man könnte sich folgende Regeln vorstellen:

```
                Objekt ist fest
E1:EINNEHMEN ----------------> essen mit AGENT=Agent(E1)
                                        THEMA=Objekt(E1)
             Objekt ist flüssig
             ----------------> trinken mit AGENT=Agent(E1)
                                          THEMA=Objekt(E1)
             Objekt ist Luft
             ----------------> atmen mit AGENT=Agent(E1)

             Objekt ist Gas
             --------------->inhalieren mit Agent=Agent(E1)
                                          THEMA=Objekt(E1)
```

c) Auswahl nach der größten Übereinstimmung

Mit vielen Worten können mehrere Aktionen beschrieben werden. Ein Beispiel ist das Wort "geben". Man kann sich folgende Aktionen vorstellen:

verkaufen
kaufen
schenken
leihen
...

Hier wendet man für die Auswahl folgende Strategie an: man wählt diejenige Realisation, die die größtmögliche Anzahl von Übereinstimmungen hat, und die kleinstmögliche Anzahl von lexikalischen Entscheidungen. Angenommen, es liegen folgende Aussagen vor:

```
GIBT(JACK1,SUE2,BUCH3)
GIBT(SUE2,JACK1,STIFT2)
```

Hieraus könnte man folgenden Satz generieren:

"Jack gibt Sue ein Buch, Sue gibt Jack einen Stift"

Besser wäre allerdings:

"Jack tauscht mit Sue ein Buch gegen einen Stift"

Obwohl damit die lexikalischen Wahl getroffen ist, verbleiben noch Entscheidungen. Existiert zum Beispiel noch die Aussage

```
GEWICHT(BUCH3,700g)
```

so könnte das noch das Wort "schwer" erzeugen, mit Buch3 als Thema. Die ausstehende Wahl ist, was man in den Mittelpunkt des Satzes stellt. Man kann nämlich zwei verschiedene Sätze produzieren:

"Jack tauscht mit Sue das schwere Buch gegen einen Stift"

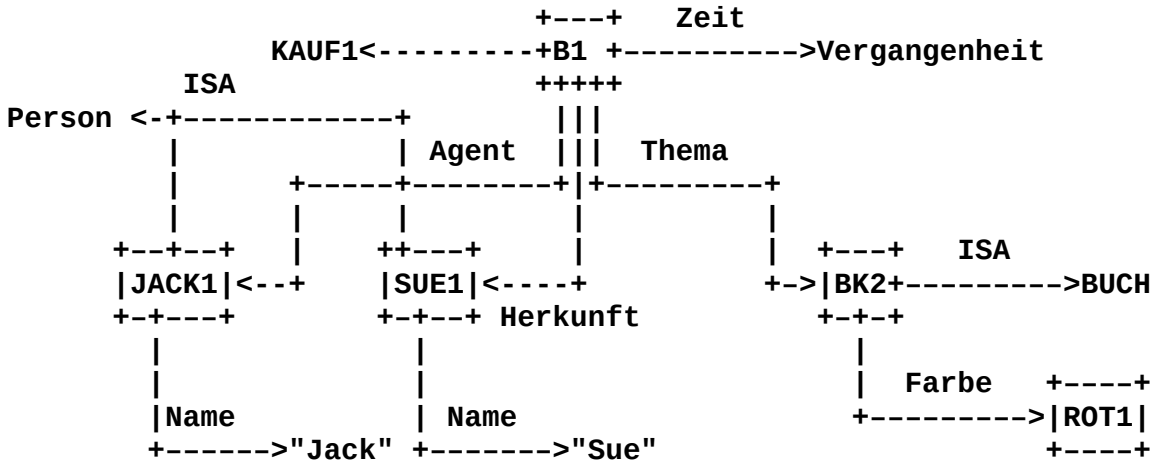
und

"Das Buch, das Jack mit Sue gegen einen Stift tauscht, ist schwer"

3.3.2 Sätze direkt aus der Repräsentation generieren

a) Generierung mithilfe einer Generierungsfunktion

Die direkteste Technik, einen Text aus einer Repräsentation zu generieren ist, eine Phrase mit jedem Faktentyp zu assoziieren. Diese Technik ist dann am effektivsten, wenn die Repräsentation sehr nahe an der lexikalischen Form ist. Angenommen, es liegt folgende Situation in Form eines semantischen Netzes vor:



Der Prozess der schrittweisen Generierung kann nun mit einer Generierungsfunktion G beschrieben werden, die jedem Netzfragment eine Phrase zuordnet und zwar nach beispielsweise folgenden Regeln:

```
G(KAUF1)= G(>Agent) kaufte G(>Thema) { von G(>Herkunft }
G(Person)= G(>Name)
G(Buch)= das G(>Farbe) Buch { geschrieben von G(>Autor) }
G(ROT1)= rote
```

Elemente in geschweiften Klammern sollen optional sein, Angaben mit ">" erhält man, indem man dem entsprechenden Pfeil im Netz folgt. Ein Satz wird nun generiert, indem man G(B1) evaluiert. Die einzelnen Schritte der Auswertung sehen dann so aus:

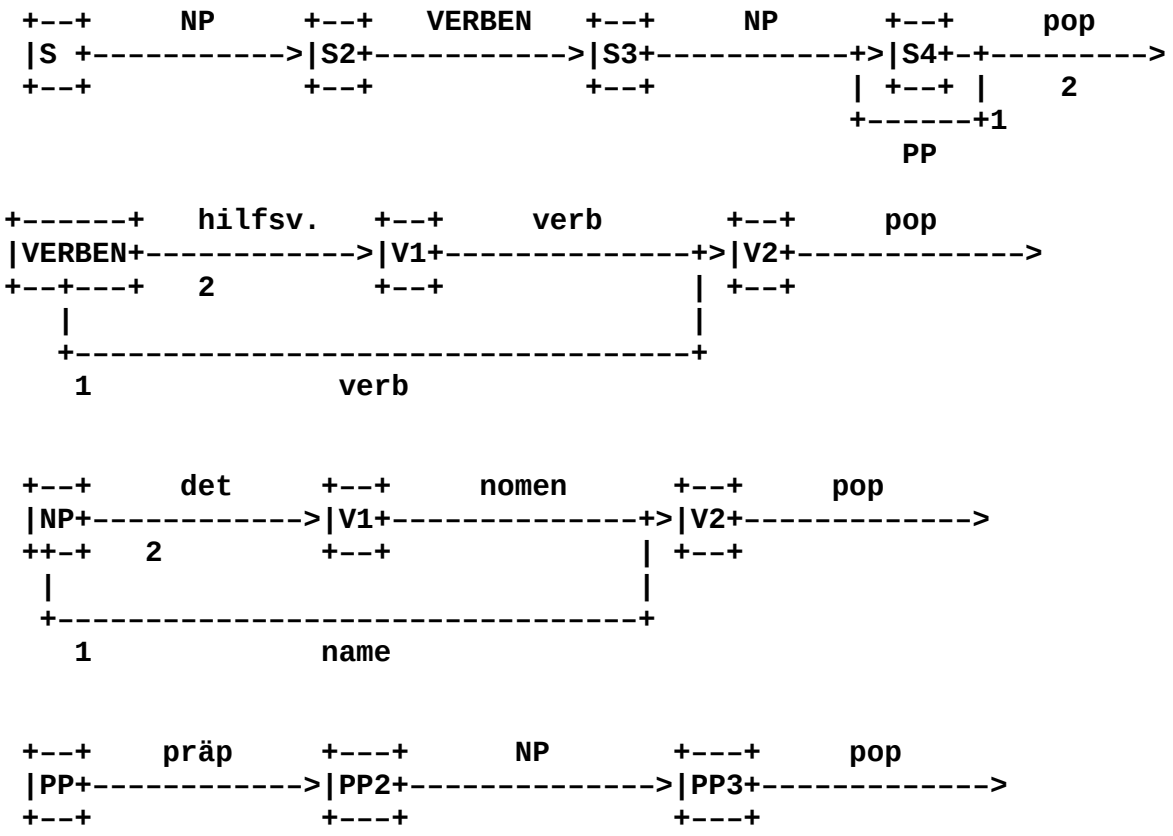
```
G(B1)
G(JACK1) kaufte G(BK2) von G(SUE1)
Jack kaufte das G(ROT1) Buch von Sue
Jack kaufte das rote Buch von Sue
```

Natürlich können die Regeln viel komplizierter sein. Man kann auch Text in einer anderen Zeitform erzeugen. Man erweitert einfach die G-Funktion um ein Argument, das die gewünschte Zeitform aufnimmt. Auch andere Steuerinformationen lassen sich auf diese Weise unterbringen.

Um die G-Funktion flexibler zu gestalten, kann man eine Programmiersprache für ihre Implementierung zulassen. Diese erlaubt dann auch, eine Liste der in Vergangenheit benutzten Worte und Phrasen anzulegen, sowie eine Festlegung des Blickpunktes. Damit können die Objekte immer in der geeigneten Weise ausgegeben werden.

b) Generierung mithilfe einer ATN-Grammatik

Die *ATN-Grammatik* (augmented transition network) eignet sich sowohl für die Sprachanalyse als auch für die Generierung. Hier soll an einem Beispiel die Textgenerierung untersucht werden. Dabei übernimmt die Grammatik die Rolle der G-Funktion aus dem ersten Beispiel.
Die Grammatik zur Generierung enthält einen Übergangsgraphen und Regeln. Einer Kante des Graphs zu folgen heißt dabei die Worte bzw. Phrasen zu generieren, die an der Kante stehen. Wird beispielsweise der Kante "verb" gefolgt, so muß im Lexikon ein Verb stehen, das in den Satz eingefügt wird. Folgt man Kanten, die den Namen eines Unternetzes tragen, so entspricht das einem "Unterprogrammsprung" d.h. dieses Netz wird abgearbeitet bis man zu einer "pop"-Kante gelangt.
Oft stehen mehrere Kanten zur Auswahl. Außerdem muß bestimmt werden, welche Worte man jeweils dem Lexikon entnimmt. Dazu existieren zu Kanten entsprechende Regeln. In jeder kann eine Bedingung stehen, die erfüllt sein muß, um der Kante zu folgen. Diese Bedingungen fragen Register ab, die vorher gesetzt werden, und die Form des Textes bestimmen.
Das *-Register spielt eine besondere Rolle: es zeigt auf den aktuellen Knoten im semantischen Netz und bestimmt die Worte, aus denen der fertige Satz zusammengesetzt wird. Dabei werden schon behandelte Knoten markiert.
Eine ATN-Grammatik, die einfache Sätze in aktiver und passiver Sprache generiert, sieht beispielsweise so aus:



Kante	Test	Vor-Aktion	Aktion
S	SPRACHE=Aktiv	*:= >AGENT	mark. AGENT-Kante
S	SPRACHE=Passiv	*:= >THEMA	mark. THEMA-Kante
S2		FORM:= >ZEIT	
		SPRACHE:=SPRACHE	
S3/1	SPRACHE=Aktiv	*:= >THEMA	mark. THEMA-Kante
S3/2	SPRACHE=Passiv		
S4	unmark. Kante L	*:= L	mark. Kante L
VERBEN/1	SPRACHE=Aktiv		
VERBEN/2	SPRACHE=Passiv	*:= werden	
V1		FORM:=passiv,Verg.	
NP/1	unmark name-Kante	*:= >name	
NP/2		*:= das	
PP		*:= *	

Angenommen, diese Grammatik soll benutzt werden, um die Situation zu beschreiben, die im semantischen Netz aus dem ersten Beispiel beschrieben wurde, und zwar in aktiver Sprache. Dazu setzt man das *-Register auf B1 und startet bei Knoten S. Da SPRACHE auf Aktiv gesetzt ist, wird die erste Regel zu S genommen und man springt zum NP-Netz mit dem *-Register gesetzt auf >AGENT also Jack1 und markiert die AGENT-Kante. Im NP-Netz folgt man der name-Kante, und generiert das Wort "Jack" als erstes Wort des Satzes. Über "pop" kehrt man zum NP-Netz zurück. Hier steht das *-Register immer noch auf B1.
Der Knoten S2 wird nun bearbeitet. Dort steht keine Bedingung, deswegen wird FORM auf >ZEIT (hier Vergangenheit) und SPRACHE auf SPRACHE (Aktiv) gesetzt. Die Verwirrung in der Namensgleichheit ist durch die Verwendung des lokalen Namens "SPRACHE" im VERBEN-Netz begründet. Dort folgt man der verb-Kante und erzeugt "kaufte". Nach Rückkehr aus diesem Netz befindet man sich im Knoten S3. Die erste Regel ist erfolgreich und * wird auf >THEMA also auf BK2 gesetzt. Damit wird das NP-Netz aufgerufen. Die erste Regel schlägt fehl, somit wird die zweite angewendet was "das" erzeugt. NP2 generiert dann das Wort "Buch" und man kehrt zum S-Netz zurück.
Der Test von S4 fällt positiv aus, womit * auf Sue1 gesetzt wird. Man bearbeitet das PP-Netz ab, generiert "von" als Präposition und "Sue" als NP. Damit wird das PP-Netz verlassen und endlich auch das S-Netz. Die Generierung ist damit abgeschlossen und man erhält den Satz "Jack kaufte das Buch von Sue".

3.4 Generierung mithilfe unabhängiger Module

Wie schon erwähnt findet die Erzeugung von Text in zwei Phasen statt:

- Selektion des Inhalts aus der Repräsentation
- Generierung des Textes aus dem Inhalt

In vielen Fällen sind diese beiden Schritte nicht zeitlich zu trennen, weil sie nicht unabhängig voneinander bearbeitet werden können. Viele Aspekte des Inhalts ergeben sich erst im Laufe der Generierung. Bisher war Interaktion auf diesem Gebiet unmöglich oder würde einen gewaltigen Aufwand an Backtracking erfordern; beispielsweise mußte die Zeitform und die aktiv-passiv Entscheidung vorher erfolgen.
Ein Ansatz, eine eher zeitverteilte Generierung zu erlauben, sind die *systemic grammers*, auf die hier nur kurz eingegangen wird. In ihnen existieren verschiedene Teile, die nach bestimmten Kriterien Entscheidungen treffen, und diese im System speichern. Die syntaktische Entscheidungskomponente ist dort nur eine von vielen. Die Reihenfolge der Entscheidungen ist nicht vorgeschrieben. Erst die Summe aller Informationen ergibt das Gesamtbild.

Eine andere Realisation, die diese Vorteile besitzt, ist die *functional unification grammer* (FUG). Sie erlaubt es neben der Generierung semantische und Diskursinformationen zu speichern. Nehmen wir nocheinmal das vorige Beispiel einer Situation. Um sie für die FUG geeignet darzustellen, werden alle Spezifikationen in Form von Slots abgelegt:

```
(S    ZEIT      Vergangenheit
      TYP       KAUF1
      AGENT     Jack1
      THEMA     Buch2
      HERKUNFT  Sue1)
```

Um nun den Prozeß auszuführen, versucht man diese Liste mit vorhandenen der Grammatik zu unifizieren. Dabei bestimmt der Pattern-Slot, in welcher Reihenfolge die Bestandteile im fertigen Satz erscheinen müssen. Punkte bedeuten, daß dort beliebige Komponenten eingefügt werden dürfen.
Der syntaktische Teil sieht beispielsweise folgendermaßen aus

```
(S    ERSTE-NP  (NP)
      HAUPT-V   (VERB)
      NUM=NUM_ERSTE-NP=NUM_HAUPT-V
      OBJ       (NP)
      PATTERN   (ERSTE-NP HAUPT-V OBJ))

(S    MODS      (PP)
      PATTERN   (... MODS))


(NP   DET       (ART)
      KOPF      (NOMEN)
      PATTERN   (DET ... KOPF))

(NP   PRO       (PRO)
      PATTERN   (PRO))

(NP   NAME      (NAME)
      PATTERN   (NAME))

(PP   PRÄP      (PRÄP)
      POBJ      (NP)
      PATTERN   (PRÄP POBJ))
```

Ein Zusatz um Aktiv-Passiv-Entscheidungen zu steuern könnte so aussehen:

```
(S    ERSTE-NP = (NP REF=^AGENT)
      OBJ      = (NP REF=^THEMA)
      SPRACHE Aktiv)

(S    ERSTE-NP = (NP REF=^THEMA)
      HILFS-V werden
      MODS       (PP PRÄP durch
                     POBJ (NP REF=^AGENT))
      SPRACHE Passiv)
```

```
(S   ERSTE-NP = (NP REF=^THEMA)
     HILFS-V werden
     SPRACHE Passiv)
```

Angenommen, ein Modul ermittelt Aktiv für die Aktiv-Passiv-Entscheidung. Es fügt den Term (SPRACHE Aktiv) zu der bestehenden Liste hinzu und unifiziert sie mit der ersten Regel. Das ^-Symbol steht für den Inhalt des entsprechenden Slots also ^AGENT zum Beispiel für Jack1. Nach der Unifizierung sieht unsere Struktur folgendermaßen aus:

```
(S   ZEIT     Vergangenheit
     TYP      KAUF1
     AGENT    Jack1
     THEMA    Buch2
     HERKUNFT Sue1
     ERSTE-NP (NP REF Jack1)
     OBJ      (NP REF Buch2)
     SPRACHE  Aktiv)
```

Natürlich ist es möglich die Wahl auf aktive Sprache erst später zu treffen. Andere Festlegungen können dann vorgezogen werden und Bedingungen hinzufügen, um diese Entscheidung zu vereinfachen. Zum Beispiel könnte ein anderes Modul herausfinden, den Agent nicht im Satz zu nennen. Hier müsste auf jeden Fall die dritte Regel (und damit Passiv) genommen werden, weil sie die einzige ist, die nicht ^Agent enthält.
Verschiedene Module könnten jetzt in beliebiger Reihenfolge zusätzliche Bedingungen hinzufügen:

- Das Hauptverb soll "kaufen" sein:

```
(HAUPT-V (VERB WURZEL kaufen))
```

- Das erste NP soll als Pronom realisiert werden, weil Jack zum Beispiel schon vorher erwähnt wurde:

```
(ERSTE-NP (NP PRO er))
```

- Das Thema wird benannt:

```
(OBJ (NP DET  ein
         KOPF Buch))
```

- Die Herkunft wird als PP realisiert:

```
(S MODS (PP   PRÄP von
         POBJ (NP REF=^HERKUNFT)))
```

Die um diese Einträge erweiterte Struktur kann jetzt mit der ersten Regel unifiziert werden. Es entsteht folgende fertige Struktur:

```
(S   ZEIT     Vergangenheit
     TYP      KAUF1
     AGENT   Jack1
     NUM=NUM_ERSTE-NP=NUM_HAUPT-V
     THEMA    Buch2
     HERKUNFT Sue1
     HAUPT-V  (VERB WURZEL kaufen
                    ZEIT   Vergangenheit
                    NUM    {3s})
     ERSTE-NP (NP PRO er
                  NUM {3s}
                  REF Jack1)
     OBJ      (NP DET  ein
                  KOPF Buch
                  PATTERN (DET KOPF)
                  REF Buch2)
     MODS     (PP PRÄP von
                  POBJ (NP NAME Sue
                           REF  Sue1)
                  PATTERN (PRÄP POBJ))
     PATTERN (ERSTE-NP HAUPT-V OBJ MODS))
```

Aus dieser Struktur kann jetzt sehr einfach der Satz generiert werden, da alle Entscheidungen getroffen wurden. Es entsteht:

"Er kaufte das Buch von Sue".

Dieses Beispiel zeigt einen Trend in der Sprachgenerierung, nämlich viele verschiedenartige Module zusammenwirken zu lassen, wobei jedes einzelne Entscheidungen trifft, die zu einem Gesamten zusammengesetzt werden.

"Wohlan, wir wollen hinabsteigen und dort
ihre Sprache verwirren, so daß keiner mehr
die Sprache des anderen versteht !"
(gen 11,7)

Literatur

[1] James Allen, "natural language understanding".
Benjamin/Cummings Publishing Company Inc, 1987

[2] Harry Tennant, "natural language processing"
Petrocelli book 1981

[3] J. Retti u.a, "artificial intelligence"
Teubner 1984