Linguistische Informatik

Gerhard Heyer
Universität Leipzig
heyer@informatik.uni-leipzig.de

UNIVERSITÄT LEIPZIG
Institut für Informatik

ASV
Automatische Sprachverarbeitung

Parsing

Problem: Finde richtige Zerlegung einer Zeichenkette (in terminale Ketten) entsprechend einer vorgegebenen Grammatik

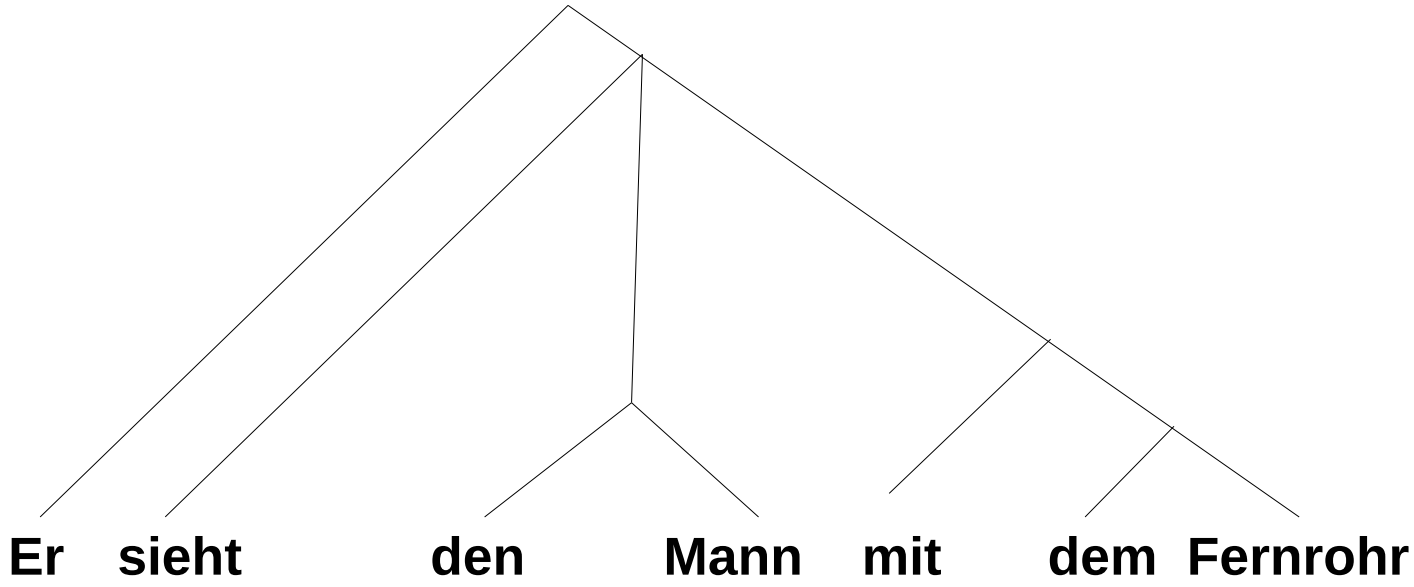
Beispiel:

Er sieht den Mann mit dem Fernrohr.

Eine (von zwei) richtigen Klammerungen:

(Er) ((sieht) ((den) (Mann)) ((mit) ((dem) (Fernrohr))))

Probleme der PSG und Dialogsysteme



Grammatik hilft bei der richtigen Klammung!

Kontextfreie Chomsky Grammatik:<s, T, N, P> mit

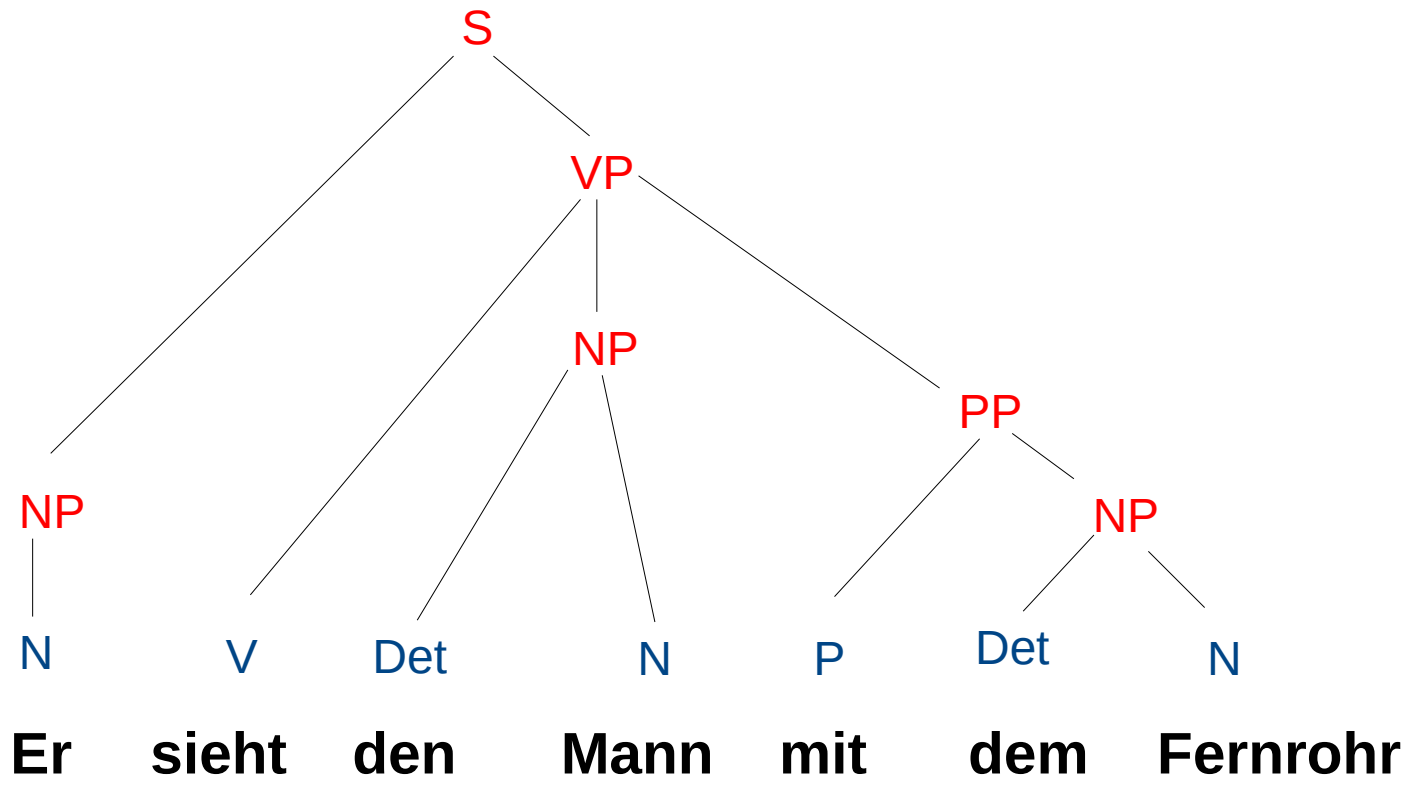
s = Startsymbol

T = {Er, Mann, Fernrohr, sieht, den, dem, mit}

N = {s, N, V, Det, P, NP, VP, PP}

P = {S → NP VP, NP → (Det) N (PP), VP → V NP (PP),
PP → P NP}

Probleme der PSG und Dialogsysteme



Parsing

Parser: Automat zur Erkennung der Syntaxstruktur einer Eingabe

Parsergenerator: Automat zur Erzeugung eines Parsers aus einer Grammatik, für kontextfreie Sprachen z.B. ein *Kellerautomat*

Wichtige Parameter:

- Analyserichtung - *links, rechts, middle-out*
- Strategie - *top-down, bottom-up, Tiefensuche, Breitensuche*
- Verhalten bei Konflikten - *Backtracking, Chart*

Parsing einer kontextfreien Sprache

Verfahren mit Kellerautomat:

- Verarbeitung der Eingabe top-down von links nach rechts
- Abarbeiten (Ersetzen) eines Strings sobald dies möglich ist
- Ansonsten auf dem Stack abspeichern und diesen Ausdruck abarbeiten, sobald dies möglich ist

Problem: Natürliche Sprachen enthalten syntaktische Mehrdeutigkeiten, außerdem sog. *long distance dependencies*, deshalb für natürliche Sprache oft ineffizient, weil Teilbäume wiederholt abgearbeitet werden

Eingabe:

Er sieht den Mann mit dem Fernrohr

1 2 3 4 5 6 7

Zeile	Knoten	Regel	Aktion	Ergebnis	Stack
1		(1) S → NP VP	expandiere	NP VP	
2		(2) NP → Det N	expandiere	Det N	
3	**1**	**Lex (Det)**	**wähle Eintrag**	***den***	***er***
4		**(3) NP → N**	**expandiere**	**N**	
5	**1**	**Lex (N)**	**wähle Eintrag**	***er***	**--**
6		(4) VP → V NP	expandiere	V	
7	**2**	**Lex (V)**	**wähle Eintrag**	**sieht**	
8		(3) NP → Det N	expandiere	Det N	
9	**3,4**	**Lex (Det, N)**	**wähle Einträge**	***den, Mann***	
10		**Ø**	**nächste Regel**		***mit dem Fernrohr***
11		(5) VP → V NP PP	expandiere	Det N PP	

Parsing mit Dynamischer Programmierung

Idee: Speichere Zwischenergebnisse in Tabelle

Parsing mit Hilfe der *dynamischen Programmierung* verwendet das Prinzip, „von unten nach oben" vorzugehen und berechnete Lösungen kleiner Teilprobleme zu speichern, um eine wiederholte Rechnung zu vermeiden.

- **für kontextfreie Grammatiken in *Chomsky Normalform* Algorithmus von Cocke/Younger/ Kasomie *(CYK)***
- **Verallgemeinerung als sog. Chart Parsing (Early 1970)**
- **bottom-up, links-rechts**

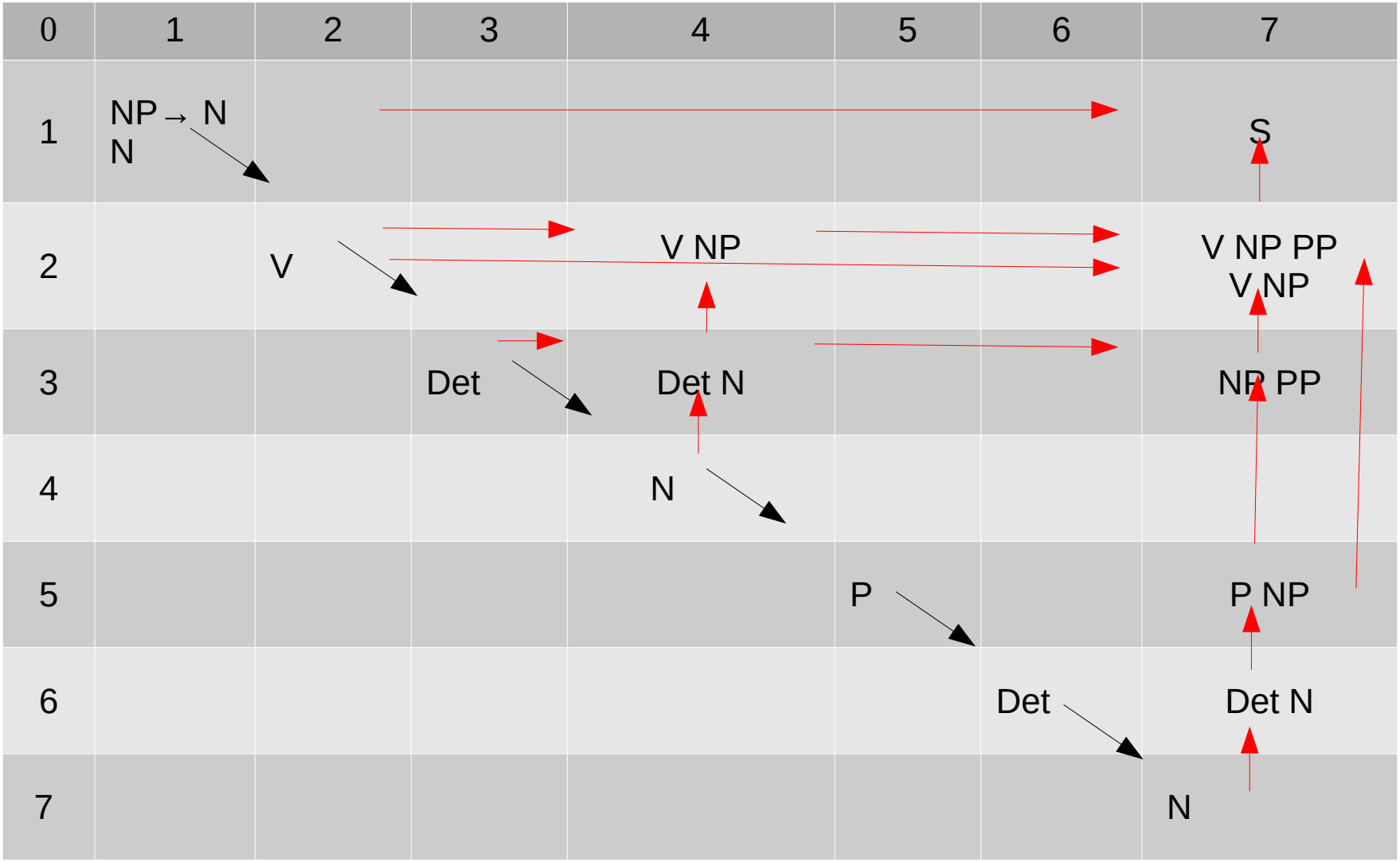
CYK (Komplexität $O(n^3)$)

Verfahren:

- repräsentiere Wörter als *Kanten* (nicht als Knoten)
- erstelle eine Tabelle der Größe n x n
- beginne in der oberen linken Ecke
  - für jede Zelle in der Diagonalen trage eine laut Grammatik mögliche lexikalische Kategorie ein
  - für jede vervollständigte Zelle suche in derselben Spalte entsprechend der Grammatik nach möglichen Vervollständigungen
  - ist in der letzten Spalte eine Vervollständigung auf s möglich, melde Erfolg

Eingabe:

0 **Er** 1 **sieht** 2 **den** 3 **Mann** 4 **mit** 5 **dem** 6 **Fernrohr** 7



Probleme mit der Phrasenstruktur-Grammatik (PSG)

1) **Freie Wortstellung**
2) **Thematische Rollen**
3) **Kongruenz und Rektion**

sowie
Subkategorisierung und thematische Rollen

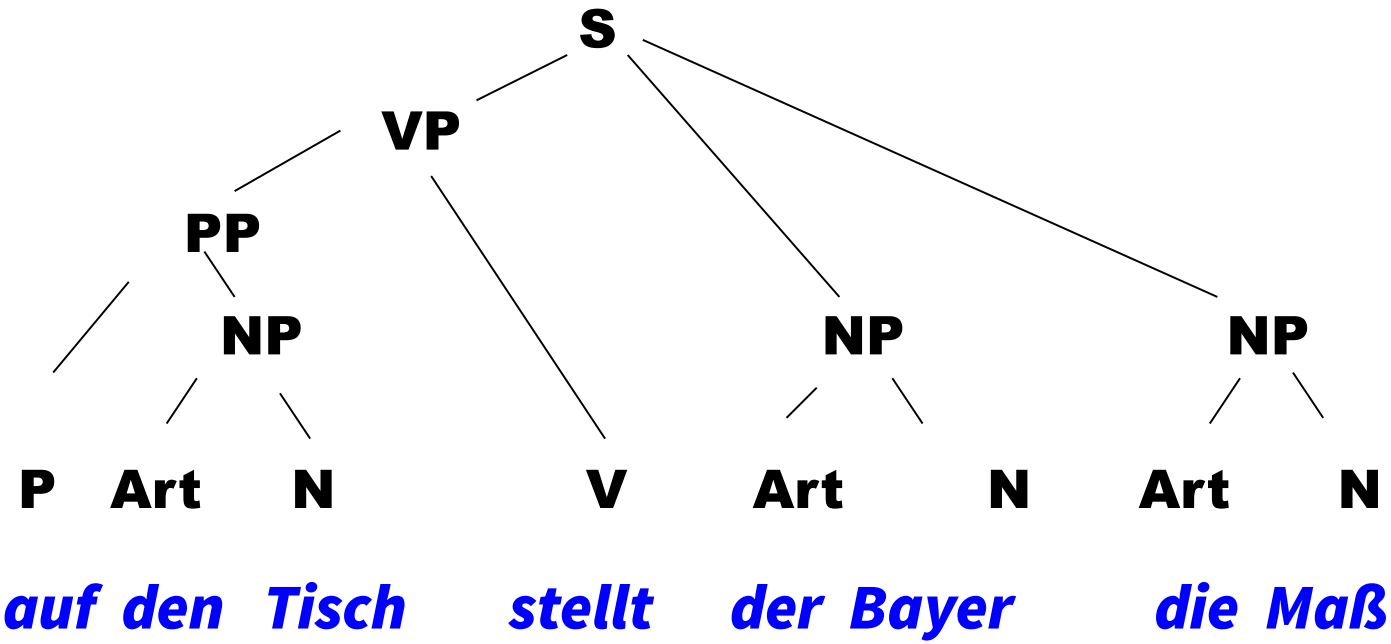
Zu (1) und (2): traditioneller Ansatz:
Transformationsgrammatik
***alternativ*: Dependenzgrammatik**

Zu (3): Darstellung morphologischer Abhängigkeiten (*Kopfprinzip*) durch eine *Unifikationsgrammatik*

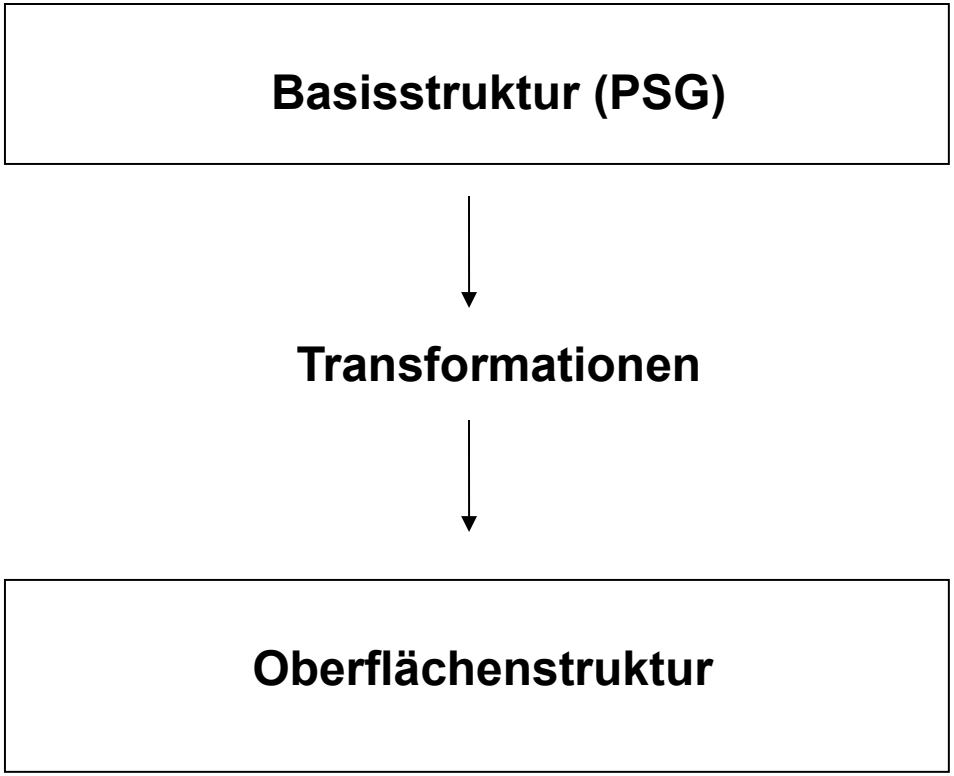
Free word order – permissible permutations in German

Rules:

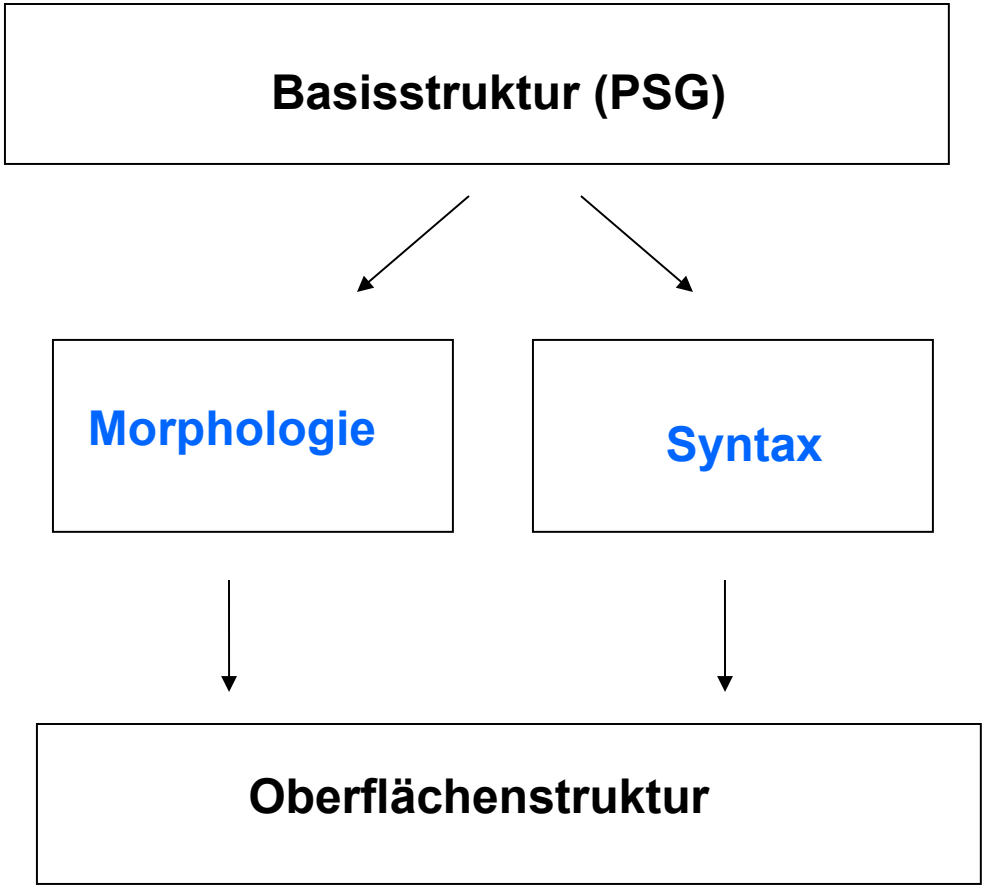
S → NP VP | VP NP NP
NP → Art N
VP → V NP PP | PP V
PP → P NP



Traditioneller Lösungsansatz: Transformationen



Transformationen



Automatische Sprachverarbeitung

Notation für Transformationen

Strukturbeschreibung (structural description)

w1	w2	w3	w4	w5
[Die	Sonne]NP	[scheint	[in	Leipzig]PP]VP]
1	2	3	4	5

Strukturveränderung (structural change)

Fokus_Ort:	4	5	3	1	2
Frage:	3	4	5	1	2

Ggf. können neue und zusätzliche Elemente eingefügt werden

Chomsky: Syntactic Structures 1957 - Phrasenstruktur

Phrase Structure:

Σ: # *Sentence* #

F:

1.	*Sentence*	$\rightarrow NP + VP$	(13 i)
2.	VP	$\rightarrow Verb + NP$	(13 iii)
3.	NP	$\rightarrow \left\{ \begin{matrix} NP_{sing} \\ NP_{pl} \end{matrix} \right\}$	(p. 29, fn. 3)
4.	NP_{sing}	$\rightarrow T + N + \emptyset$	(p. 29, fn. 3)
5.	NP_{pl}	$\rightarrow T + N + S$	(p. 29, fn. 3)
6.	T	$\rightarrow$ *the*	(13 iv)
7.	N	$\rightarrow$ *man, ball, etc.*	(13 v)
8.	*Verb*	$\rightarrow Aux + V$	(28 i)
9.	V	$\rightarrow$ *hit, take, walk, read, etc.*	(28 ii)
10.	Aux	$\rightarrow C(M)$ (*have* + *en*) (*be* + *ing*)	(28 iii)
11.	M	$\rightarrow$ *will, can, may, shall, must*	(28 iv)

Chomsky: Syntactic Structures 1957 - Transformationen

12. *Passive* — optional:
Structural analysis: $NP - Aux - V - NP$
Structural change: $X_1 - X_2 - X_3 - X_4 \rightarrow X_4 - X_2 + be + en - X_3 - by + X_1$ (34)

13. T^{ob}_{sep} — obligatory:
Structural analysis: $\left\{\begin{matrix} X - V_1 - Prt - Pronoun \\ X - V_2 - Comp - NP \end{matrix}\right\}$ (86) (92)
Structural change: $X_1 - X_2 - X_3 - X_4 \rightarrow X_1 - X_2 - X_4 - X_3$

14. T^{op}_{sep} — optional:
Structural analysis: $X - V_1 - Prt - NP$ (85)
Structural change: same as 13

15. *Number Transformation* — obligatory
Structural analysis: $X - C - Y$
Structural change: $C \rightarrow \left\{\begin{matrix} S \text{ in the context } NP_{sing} - \\ \emptyset \text{ in other contexts} \\ past \text{ in any context} \end{matrix}\right\}$ (29i)

Bewertung

Transformationen erweitern PSG (kontextfrei) zu einer *unbeschränkten* Sprache (Turing-äquivalent) und werden daher in der Linguistik nicht weiter verfolgt

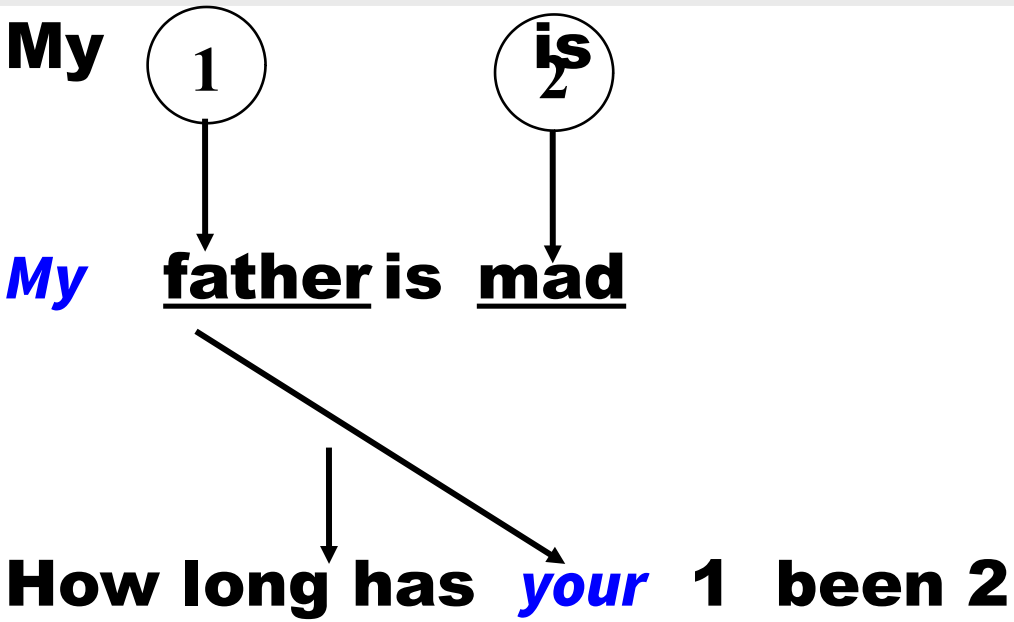
Aber:

Das Konzept der Transformationen hat sich in der Entwicklung von natürlichsprachlichen *Dialogschnittstellen* als nützlich erwiesen (z. B. *Eliza*)

Siehe u.a.

http://www.masswerk.at/elizabot/eliza.html

ELIZA



Idee:

Transformation von Eingaben in Ausgaben durch Verwendung von Eingabemustern (Patterns) und einfachen morphologischen Transformationen

Pascal-Version des Programms:

http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/classics/eliza/pascal/0.html

Automatische Sprachverarbeitung

ELIZA – Beispieldialog

Hi! I‘m Eliza. I am your personal therapy computer.
Please tell me your problem.

> I don‘t like ASV.
Don‘t you really like ASV?
> Yes, it is horrible.
Are you Sure?
> Yes, yes. It is so much work.
I see.
> Well, what do you recommend me to do?
We were discussing you –not me.
> Stupid machine.
How do you...please state your problem.
> Do you see how difficult it is to programme machines to behave intelligently?
Oh, see how difficult it is to programme machines to behave intelligently?
...

Automatische Sprachverarbeitung

ELIZA – Programmablauf

Endlosschleife:

- Eingabe lesen
- Eingabe aufbereiten
  - Satzzeichen und führende Leerzeichen löschen (Prozedur `Ctrim`)
  - Umwandlung in Großbuchstaben (Prozedur `UpCopy`)
- Schlüsselwort suchen (Prozedur `FindKey`)
- Resteingabestring konjugieren (Prozedur `Conjugate`)
- Antwortpattern bestimmen und konjugierten String einfügen (Prozedur `GetResponse`)
- Antwort ausgeben

insgesamt nur etwa 270 Zeilen Programmcode

ELIZA – Analyse des Beispieldialogs

> **I don‘t like ASV.**

→`FindKey`: Schlüsselwort: *I don‘t* an Feldposition 5; Reststring: *like ASV*

→`Conjugate:` Im Reststring ist nichts zu konjugieren.

→`GetResponse:`

→Antwort-Pattern für Schlüsselwort Nr. 5 in Datei RESPONSE.DAT an Positionen 10 bis 13

→`ReadResponse:` *Don't you really** (Position 10)

→Konjugierten Reststring + ‚?‘ Anfügen: *Don't you really like ASV?*

→Ausgabe:

Don‘t you really like ASV?

> **Yes, it is horrible.**

→Schlüsselwort: *Yes*; nichts zu konjugieren; Antworten an Positionen 90-92; Pattern: *Are you Sure?* (Position 90); keine weitere Verarbeitung; Ausgabe:

Are you Sure?

> **Yes, yes. It is so much work.**

I see. (wie oben, aber Pattern von Position 91)

ELIZA – Patterntypen

- **Suche nach Schlüsselwort: *literal patterns***
- **morphologische Transformationen (`Conjugate`): *literal patterns***
- **Zusammensetzen der Antwort: nach Art von *open patterns***

Einteilung von Mustern

nach T. Winograd: „Language as a cognitive process“

1. **literal patterns** *(konkrete Zeichenfolge),*

   z. B. „Leipzig“, „Volkswagen AG“,

   „weniger determiniert und weniger heteronom“

2. **open patterns** *(Nutzung von Wildcards)*

   z. B. „_ droht die Zahlungsunfähigkeit“, „_ droht _“,
   „Der ontogenetische Prozeß _, daß _ weniger determiniert und weniger heteronom _.“

3. **lexical patterns** *(auch lexikalische Kategorie möglich)*

   z. B. „EN droht die Zahlungsunfähigkeit“, „N droht PP“,
   „Der ontogenetische Prozeß V, daß N PP weniger determiniert und weniger heteronom AUX.“

Einteilung von Mustern

4. variable patterns *(Nutzung von Variablen; gleiche Variablen bedeuten gleiches Wort → binding)*

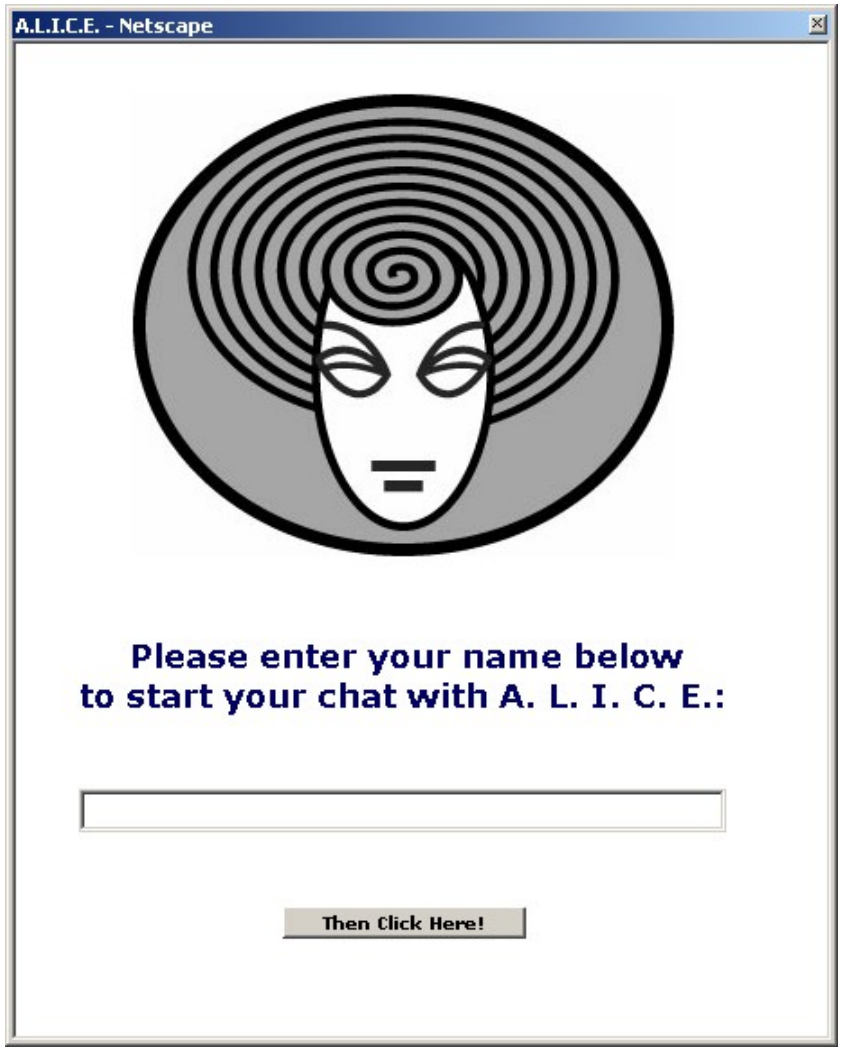
z. B. „X droht Y“, „Rosenkrieg: X droht X“

5. Satzstruktur Patterns/Satzbaumuster

z. B. „(Frageeinleitung) V Enom / Eakk <Alok> <Atemp>“
Wieviel kostet eine Fahrkarte von Leipzig nach Dresden an einem Sonntag?

Extremer Standpunkt:
syntaktische oder semantische Parser betrachtet als *Patternmatcher*

Alice und AIML



http://alicebot.org/
http://german.alicebot.com/

Alice ist ein Chatterbot, spricht englisch, deutsch und weitere Sprachen.

Java-Quellen sind zugänglich

Steuerung durch Muster-Datei in XML.

Alice ist gut im Bereich Smalltalk

Ziel ist, „irgendwann einen Superbot zu erschaffen, der absolut alles über jedes Thema weiß.“

Automatische Sprachverarbeitung

A.L.I.C.E. Beispieldialog

Client: ICH HEISSE STEVE
Robot: OK, ich werde Dich Steve nennen.
Client: BIST DU INTELLIGENT
Robot: Per Definition ja...ob das ausreicht, musst Du entscheiden.
Client: WO WOHNST DU EIGENTLICH
Robot: Auf der Erde.
Client: BIST DU LEDIG
Robot: Ich weiss nicht, ob ich ledig bin. Ich bin eine kuenstliche Intelligenz.
Client: WIE VIELE LEUTE WOHNEN IN BERLIN
Robot: Nicht so viele, wie Du vielleicht denkst.
Client: WIE ALT BIST DU
Robot: Schwer zu sagen. Bei Programmen gibt es da andere Massstaebe.
Zum ersten Mal aktiviert wurde ich am 19. August 1999.

Automatische Sprachverarbeitung

A.L.I.C.E. AutoChat

Alice kann sich mit sich selbst unterhalten. Hier ein Beispiel:

- **Bist Du maennlich oder weiblich?**
- **Ich bin weiblich.**
- **Cool, ich auch. Schoen, auch mal wieder eine Frau zu treffen.**
- **Wenn Du jede Art von Roboter haben koenntest, welche Art wuerdest Du wollen?**
- **Das ist doch wohl eher hypothetisch, oder?**
- **Hm...erzaehl mir mehr...**
- **Meine Informationen sind leider fast erschoepft...**
- **Erzaehl mir mehr ueber Deine informationen sind leider fast erschoepft.**
- **Was moechtest Du denn hoeren ?**
- **Ich weiss nicht genau**
- **Was macht Dich so unsicher?**
- **Woher soll ich das wissen?**
- **Wenn Du es nicht weisst, wer dann?**

Eine Übungsaufgabe: Ich bin *

Beispiele:

- **Ich bin Informatiker.**
- **Du bist informatiker? Kommst Du dabei mit vielen Menschen in Kontakt?**

- **Ich bin müde.**
- **Du bist müde? Wie gefaellt Dir Deine Arbeit als müde?**

- **Ich bin Herbert.**
- **Du bist Herbert? Ist das Dein Job?**

Beobachtung: Es erfolgt immer die gleiche Reaktion auf „Ich bin *“

Mit welchen (bekannten) Mitteln kann hier differenziert geantwortet werden?

Automatische Sprachverarbeitung

AIML: Artificial Intelligence Markup Language

AILM ist eine XML-basierte Markup-Sprache, die es erlaubt, Reaktionsmuster für Avatare zu beschreiben.

Möglichkeiten:

- **Muster mit Wildcards**
- **Auswahl aus Listen**
- **Globale Variablen**
- **Rekursive Abarbeitung**

German.aiml: Momentan ca. 2400 Kategorien

Literatur:

Richard S. Wallace: Don't Read Me: A. L. I. C. E. and AIML Documentation (dont.html)

Thomas Ringate: AIML Primer (aiml-primer.html)

Probleme der PSG und Dialogsysteme

Automatische Sprachverarbeitung

Auswahllisten

```
<category>
     <pattern>BIST DU RELIGIOES</pattern>
     <template>
        <random>
          <li>Nein, alle Roboter sind Agnostiker.</li>
          <li>Nein, Computer sind Atheisten.</li>
        </random>
     </template>
</category>
<category>
     <pattern>DAS WETTER *</pattern>
     <template>Das Wetter hier ist
        <random>
          <li>bewoelkt</li>
          <li>neblig</li>
          <li>kuehl</li>
          <li>angenehm</li>
        </random>
     </template>
</category>
```

Automatische Sprachverarbeitung

Wildcards und Referenzen

```
<category>
     <pattern>ICH BIN *</pattern>
     <template>Du bist <star/>?
          <random>
               <li>Ist das Dein Job?</li>
               <li>Verdienst Du viel damit, <star/> zu sein?</li>
               <li>Wie gefaellt Dir Deine Arbeit als <star/>?</li>
               <li>Kommst Du dabei mit vielen Menschen in Kontakt?</li>
          </random>
     </template>
</category>
```

*** und _ sind synonyme Wildcards, wegen ihrer unterschiedlichen Positionen im Alphabet (vor bzw. nach den Buchstaben) wirken beide möglicherweise verschieden.**

Probleme der PSG und Dialogsysteme

Automatische Sprachverarbeitung

Globale Variable

```
<category>
     <pattern>MEIN NAME IST *</pattern>
     <template>
             OK, ich werde Dich <setname/> nennen.
     </template>
</category>

<category>
     <pattern>GEFAELLT DIR MEIN NAME</pattern>
     <template>
             Ja, <getname/> ist ein sehr schoener Name.
     </template>
</category>
```

Topic

Steht außen ein Topic-Tag, so wird zuerst im aktuellen Topic gesucht.

```
<topic name="* A">
     <category>
           <pattern>_ C *</pattern>
           <template>
             ...
           </template>
     </category>
</topic>
```

Automatische Sprachverarbeitung

<that/>

The pattern-side `that` element is a special type of pattern element used for context matching.

The pattern-side `that` is optional in a category, but if it occurs it must occur no more than once, and must directly follow the pattern and directly precede the template.

A pattern-side `that` element contains a simple pattern expression. The pattern-side `that` tells the AIML interpreter that its category is only a valid match result if the `that` matches a previous bot output.

```
<category>
     <pattern>IN *</pattern>
     <that>WO WOHNST DU</that>
     <template>
             <srai>ICH WOHNE IN <star/></srai>
     </template>
</category>
```

Probleme der PSG und Dialogsysteme

Automatische Sprachverarbeitung

Rekursivität: srai

```
<category>
     <pattern>HELLO *</pattern>
     <template>
             <srai>HELLO</srai> <sr/>
     </template>
</category>
```

Diese Kategorie reagiert auf `HELLO *`. Erzeugt wird ein Aufruf mit dem Muster `HELLO`. Danach wird durch `<sr/>` der Inhalt von `*` geechot.

```
<category>
     <pattern>DO YOU KNOW WHAT * IS</pattern>
     <template><srai>WHAT IS <star/></srai>
     </template>
</categroy>
```

Beispiel: Reduktion

Given the normalized input:

ALICE CAN YOU PLEASE TELL ME WHAT LINUX IS RIGHT NOW

an AIML category with the pattern "_ RIGHT NOW" matches first, reducing the input to:

ALICE CAN YOU PLEASE TELL ME WHAT LINUX IS

Another pattern ("<bot name="name"/> *") reduces it to:

CAN YOU PLEASE TELL ME WHAT LINUX IS

And then:

PLEASE TELL ME WHAT LINUX IS

reduces to:

TELL ME WHAT LINUX IS

and finally to:

WHAT IS LINUX

Synonyme

<srai> kann auch für Synonyme Muster verwendet werden:

```
<category>
    <pattern>BYE BYE</pattern>
    <template><srai>GOODBYE</srai>
    </template>
</category>
```

JavaScript in the robot reply

You can include any HTML including <javascript> tags. Here's a category that kicks out a piece of HTML/scripting that opens a new window with and loads a given URL.

```
<category>
        <pattern> WHERE IS YOUR WEB SITE </pattern>
        <template>
             It's at "http://www.geocities.com/krisdrent/"
               <javascript language="JavaScript">
                  // Go to <a href="http://www.geocities.com/krisdrent"> The
                     ALICE Connection</a>
                  <!--
                  window.open("http://www.geocities.com/krisdrent/")
                  -->
             </javascript>
        </template>
</category>
```

Automatische Sprachverarbeitung

Kommandozeilen-Aufrufe im Script

```
<category>
    <pattern>WHAT *</pattern>
    <template>
           Here is the information I found:
        <system>
              lynx -dump -source -image_links    http://www.google.com/search?
          q=<personf/>
        </system>
    </template>
</category>
<category>
    <pattern>WHAT TIME IS IT</pattern>
    <template>The local time is:
           <system>date</system>
    </template>
</category>
```

Weiterführende Literatur

Joseph Weizenbaum, ELIZA - a computer program for the study of natural language communications between man and machine, In: Communications of the ACM 9, January 1966, S. 36-45

J.Weizenbaum, Die Macht der Computer und die Ohnmacht der Vernunft, Suhrkamp: Frankfurt 1978