
Semantische Sprachverarbeitung

Carola Eschenbach
Universität Hamburg, FB Informatik
AB Wissens- und Sprachverarbeitung (WSV)

Sommersemester 2007

Semantische Sprachverarbeitung

Sitzung 2

- Produktivität von Bedeutung
- Techniken der Bedeutungserfassung
 - PROLOG & DCGs : Syntax
 - Logik : Semantik



Ausdruckskomplexitäten

in der natürlichen Sprache

- Morphem
- Wort, Lexem
- Phrase
- Satz
- Text, Diskurs, Dokument

**Bedeutungszuordnung kann auf all diesen Ebene
erfolgen.**

Produktivität von Bedeutung

- 1) Gestern sah ich einen gelben Wal auf dem Uni-Parkplatz.
 - 2) Farblose grüne Ideen schlafen heftig.
- 1) ist auch beim ersten Hören / Lesen verständlich
- Vgl. 93759,23 – 129,291 = 93629,939
- 2) ist syntaktisch korrekt aber unverständlich
- Vgl. $y = 0 \wedge x = y^{-1}$

Grundannahme der Semantik-Analyse

- es existieren Regeln (Algorithmen), die die Beziehung zwischen (komplexen) sprachlichen Ausdrücken und Bedeutungen festlegen.
- Es gibt semantische Wohlgeformtheitsbedingungen.

Kompositionalitätsprinzip (Frege)

Die Bedeutung eines komplexen Ausdrucks

ergibt sich aus

- der Bedeutung der Bestandteile und
- den (syntaktischen) Beziehungen zwischen ihnen.

Relevanz der Syntax

- Träger von Bedeutung sind syntaktische Einheiten: Konstituenten
- syntaktische Wohlgeformtheit / Analyse ist Ausgangspunkt semantischer Analyse
 - Aber: die Pragmatik kommt auch oft ohne Syntax klar.

Analysetechnik: Sprachfragmentbildung

systematischer Sprachausschnitt

- Lexikon (Domäne, Wortarten)
- Grammatik (Syntax)

Bedeutungsspezifikation in formalem Rahmenwerk

- Lexikalische Einträge
- Kompositionsregeln (Syntax-Nutzung)

Erfassung von Aussagekraft und Beschränkungen

Erweiterung von Sprachausschnitt und Spezifikation



Fragment 1

Einfache (Haupt-) Aussagesätze

- und ihre Komponenten
- einfache Nominalphrasen
 - Peter, Laura: Eigennamen
 - ein rotes Haus, jeder Junge, kein Junge: Artikel (+ Adjektiv) + Nomen
- einfache Verbalphrasen
 - lacht, winkt: intransitive Verben
 - sieht: transitive Verben

Beispiele in F1

- Peter lacht. Peter sieht ein rotes Haus. Kein Junge sieht Laura. Jeder Junge winkt.

Fragment 2

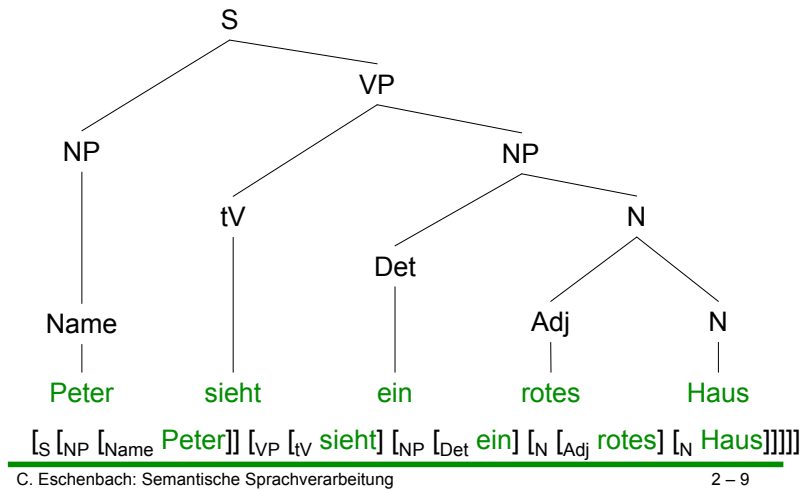
Einfache zusammengesetzte (Haupt-) Aussagesätze

- Fragmente 1 und
- einfache Zusammensetzungen
 - und, oder: Konjunktionen auf Satz- und Verbalphrasenebene

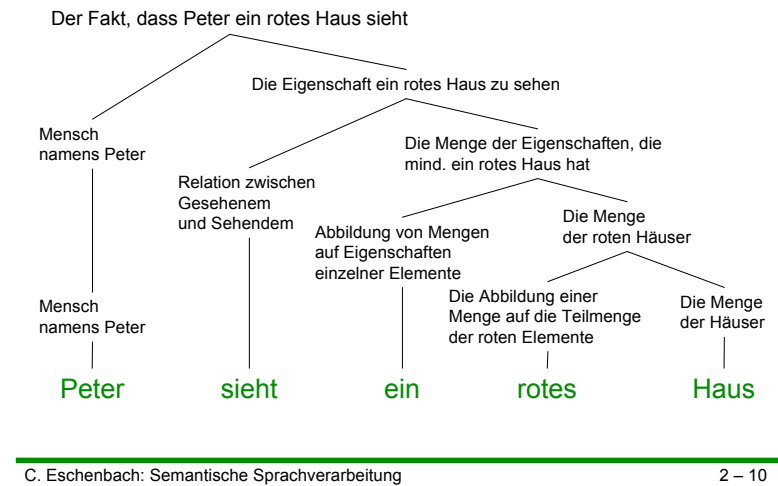
Beispiele in F2

- Peter lacht und winkt.
- Jeder Junge lacht oder winkt.
- Kein Junge lacht und Laura winkt.
- Ein Junge lacht und ein Junge winkt.

Syntax: Peter sieht ein rotes Haus.



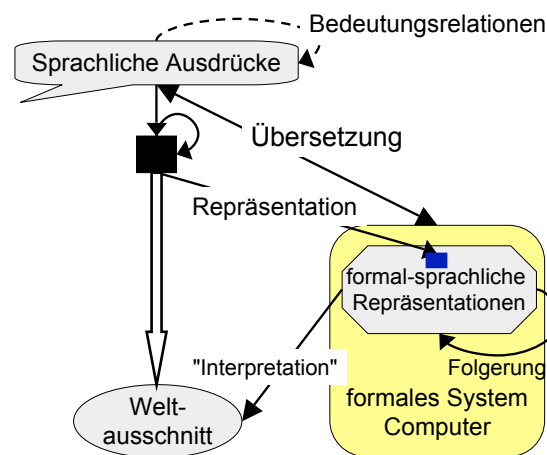
Denotation: Peter sieht ein rotes Haus.



Formale Linguistik

Modellierung von Bedeutung

- durch Abbildung auf formal-sprachliche Ausdrücke mit fester Semantik
- deren semantische Relationen die Bedeutungsrelationen widerspiegeln



Syntax und Parsing

Grundannahme

- Syntaktische Konstituenten sind die bedeutungstragenden Einheiten
- Wörter haben Bedeutung
- den syntaktischen Regeln entsprechen semantische Regeln

Voraussetzung für Beschreibung der Semantik

- Syntax-Analyse
 - Grammatik
 - Parser

Kontextfreie Grammatik

Mathematische Sicht

- Eine Grammatik ist ein 4-Tupel $(G = (N, T, R, S))$, wobei
 - N = Menge von Symbolen: die nichtterminalen Symbole
 - T = Menge von Symbolen: die terminalen Symbole
 - R = Menge von Regeln
 - S = Startsymbol ($\in N$)
- Regeln werden dargestellt als $A \rightarrow w$.
- Eine Regel ist kontextfrei, wenn für alle Regeln $A \rightarrow w \in R$ gilt, dass $A \in N$ und $w \in (N \cup T)^*$
- (Im allgemeinen Fall ist $A \in (N \cup T)^* N (N \cup T)^*$)

- (\rightarrow F2/FGI1-Vorlesung)

Eine einfache Grammatik mit Lexikon für F2

Grammatik

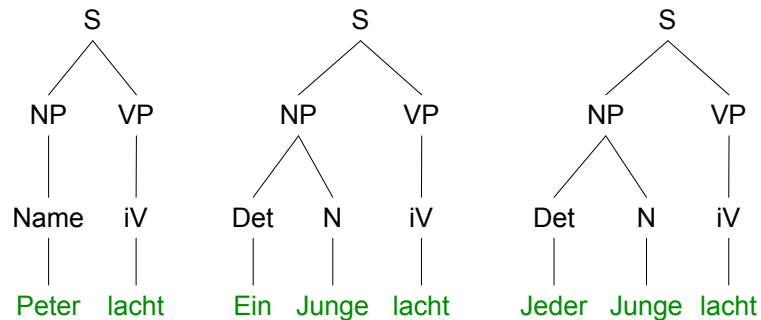
$s \rightarrow np\ vp$
 $s \rightarrow s\ conj\ s$
 $vp \rightarrow tv\ np$
 $vp \rightarrow iv$
 $vp \rightarrow vp\ conj\ vp$
 $np \rightarrow pn$
 $np \rightarrow det\ n$
 $n \rightarrow adj\ n$

Lexikon

$n \rightarrow$ haus
 $n \rightarrow$ junge
 $adj \rightarrow$ rotes
 $det \rightarrow$ ein
 $det \rightarrow$ jeder
 $det \rightarrow$ kein
 $pn \rightarrow$ peter
 $pn \rightarrow$ laura
 $iv \rightarrow$ lacht
 $iv \rightarrow$ winkt
 $tv \rightarrow$ sieht
 $conj \rightarrow$ und
 $conj \rightarrow$ oder



Beispiel



Zur Diskussion

Welche syntaktische Struktur ordnet unsere Grammatik für F2 folgenden Sätzen zu

- Peter sieht Laura und winkt.
- Peter sieht Laura und lacht oder winkt.
- Peter sieht Laura und lacht und winkt.

Fragment 3

zur Übung

- Fragment 2 und
- **zeigt**: ditransitive Verben
- **nicht**: (Verbalphrasen-) Negation

Beispiele in F3

- Laura zeigt Peter ein Haus.
- Ein Junge winkt nicht.
- Peter lacht und winkt nicht.
- Peter lacht nicht und Laura winkt.
- Peter lacht und Laura winkt nicht.
- Peter winkt nicht und lacht.



Zur Diskussion

Welche Modifikationen an der Grammatik von F2 sind für F3 erforderlich?

- Syntaktische Kategorien
- Lexikon
- Grammatik

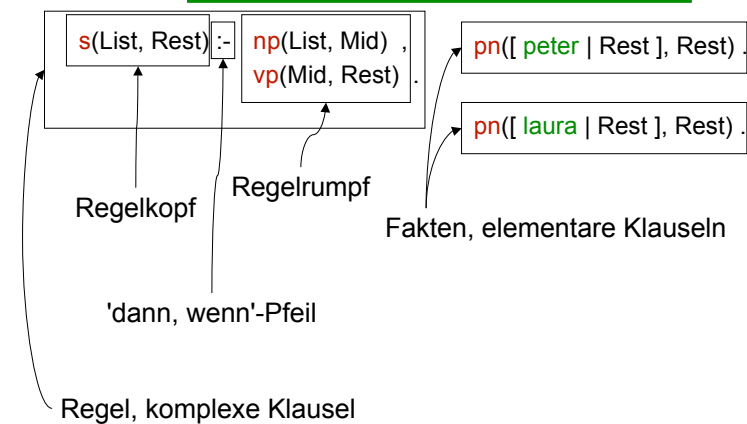
Ergebnis bitte notieren !

- Wir werden es später noch brauchen

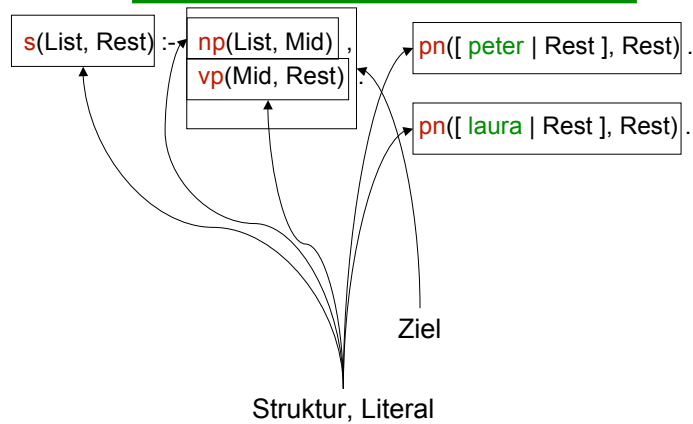
Ein einfaches Prolog-Programm für Grammatik und Lexikon für F1

<code>s(List, Rest) :- np(List, Mid), vp(Mid, Rest).</code>	<code>pn([peter Rest], Rest).</code>
<code>np(List, Rest) :- pn(List, Rest).</code>	<code>pn([laura Rest], Rest).</code>
<code>np(List, Rest) :- det(List, Mid), n(Mid, Rest).</code>	<code>det([ein Rest], Rest).</code>
<code>n(List, Rest) :- adj(List, Mid), n(Mid, Rest).</code>	<code>det([jeder Rest], Rest).</code>
<code>vp(List, Rest) :- iv(List, Rest).</code>	<code>det([kein Rest], Rest).</code>
<code>vp(List, Rest) :- tv(List, Mid), np(Mid, Rest).</code>	<code>n([haus Rest], Rest).</code>
	<code>n([junge Rest], Rest).</code>
	<code>adj([rotes Rest], Rest).</code>
	<code>tv([sieht Rest], Rest).</code>
	<code>iv([lacht Rest], Rest).</code>
	<code>iv([winkt Rest], Rest).</code>

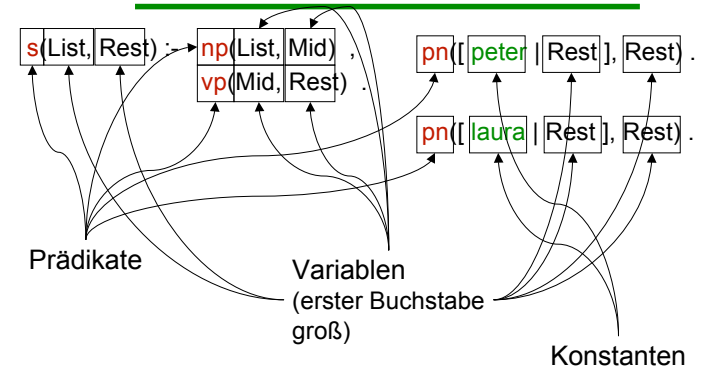
Ein paar Erläuterungen zu Prolog (1)



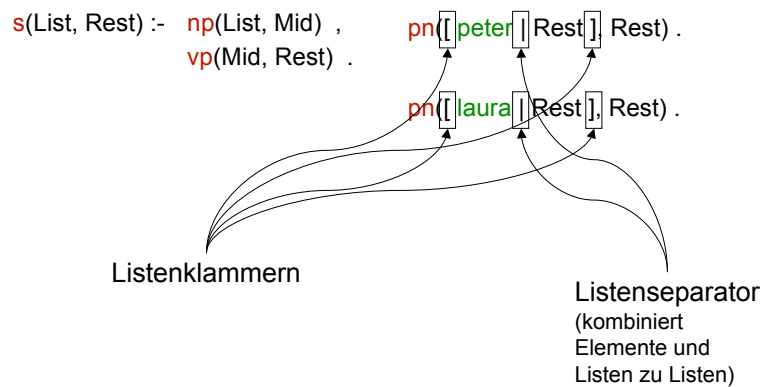
Ein paar Erläuterungen zu Prolog (2)



Ein paar Erläuterungen zu Prolog (3)



Ein paar Erläuterungen zu Prolog (4)



Ein paar Erläuterungen zu Prolog: Listen

- eine zentrale Datenstruktur
- Leere Liste : []**
- nicht-leere Listen: zwei Notationsformen**
- Gemeinsamkeit: Umschließung mit eckigen Klammern
- Rekursionsformat**
- Eine Liste besteht aus einem Kopfelement und der Restliste (getrennt durch den Separator ' | ')
- Beispiel: [Kopfelement | Restliste]
- Aufzählungsformat**
- Eine Liste besteht aus eine Folge von Elementen (getrennt durch den Separator ' , ')
- Beispiel: [Element1, Element2, Element3, ... , Elementn]

Beispiel: Listennotation

Einelementige Liste

- [element]
- [element | []]

Zweielementige Liste

- [element1, element2]
- [element1 | [element2]]
- [element1 | [element2 | []]]

Dreielementige Liste: Mischung der Formate möglich

- [e1, e2, e3]
- [e1, e2 | [e3]]
- [e1 | [e2, e3]]

Parsen mit Differenzlisten

```
s(List, Rest) :-  
  np(List, Mid) ,  
  vp(Mid, Rest) .
```

```
pn([ peter | Rest ], Rest) .
```

Instanzen

```
s([peter, lacht], [ ] ) :-  
  np([peter, lacht], [lacht]) ,  
  vp([lacht], [ ] ) .
```

Instanzen

```
pn([ peter | [ ] ], [ ] ) .  
pn([ peter ], [ ] ) .
```

```
s([peter, lacht, laura], [laura]) :-  
  np([peter, lacht, laura],  
     [lacht, laura]) ,  
  vp([lacht, laura], [laura]) .
```

```
pn([ peter | [lacht] ], [lacht]) .  
pn([ peter, lacht ], [lacht]) .
```

```
pn([ peter|[lacht,laura],[lacht,laura] ] .  
pn([ peter, lacht, laura ], [lacht, laura]) .
```

Ein einfaches Prolog-Programm für Grammatik und Lexikon für F1

```
s(List, Rest) :- np(List, Mid),      pn([ peter | Rest], Rest).  
                 vp(Mid, Rest).     pn([ laura | Rest], Rest).  
  
np(List, Rest) :- pn(List, Rest).   det([ ein | Rest], Rest).  
np(List, Rest) :- det(List, Mid),   det([ jeder | Rest], Rest).  
                 n(Mid, Rest).      det([ kein | Rest], Rest).  
  
n(List, Rest) :- adj(List, Mid),    n([ haus | Rest], Rest).  
                 n(Mid, Rest).      n([ junge | Rest], Rest).  
                 adj([ rotes | Rest], Rest).  
  
vp(List, Rest) :- iv(List, Rest).   tv([ sieht | Rest], Rest).  
vp(List, Rest) :- tv(List, Mid),    iv([ lacht | Rest], Rest).  
                 np(Mid, Rest).     iv([ winkt | Rest], Rest).
```

Parsen mit Prolog

Die Struktur der Regeln

- einheitlich, systematisch
- eigentlich sehr redundant

Abkürzendes Format DCG (Definite Clause Grammar)

- Statt
 - $c_0(L_1, Rest) :- c_1(L_1, L_2), c_2(L_2, L_3), \dots, c_n(L_n, Rest).$
- kurz
 - $c_0 \rightarrow c_1, c_2, \dots, c_n.$
- Statt
 - $c([konstante | Rest], Rest).$
- kurz
 - $c \rightarrow [konstante].$

Eine DCG Grammatik für F1

Grammatik

s --> np, vp.
vp --> tv, np.
vp --> iv.
np --> pn.
np --> det, n.
n --> adj, n.

Lexikon

n --> [haus].
n --> [junge].
adj --> [rotes].
det --> [ein].
det --> [jeder].
det --> [kein].
pn --> [peter].
pn --> [laura].
iv --> [lacht].
iv --> [winkt].
tv --> [sieht].

Prolog-Programm

go(Ex):-

example(Ex, Phrase), % Waehle Beispiel
write(Ex), write(' '), % Gib Beispiel-Name aus
writeList(Phrase), nl, % Gib Wortfolge aus
s(Phrase, []), % Parse die Wortfolge als Satz
writeln('ist ein deutscher Satz.').

```
[Carola-Eschenbachs-Computer:~/Testprogramme/SemSprachProgramme/01DCG] carolaes% swipl
Welcome to SWI-Prolog (Version 5.0.10)
Copyright (c) 1990-2002 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
```

```
For help, use ?- help(Topic), or ?- apropos(Word).
```

```
?- [dcg_frag1de].
% print compiled into print 0,00 sec, 4,200 bytes
% dcg_frag1de compiled 0,00 sec, 8,912 bytes
```

```
Yes
?- go(s5).
s5: peter sieht ein rotes haus
ist ein deutscher Satz.
```

```
Yes
?- go(s1).
s1: peter lacht
ist ein deutscher Satz.
```

```
Yes
?- go(s7).
s7: jeder junge winkt
ist ein deutscher Satz.
```

```
Yes
?- █
```

Ausprobieren ?

dcg_frag1de.pl

DCG für ein einfaches Fragment des Deutschen

Über die Vorlesungs-Web-Seite zu erhalten

Probleme mit DCG-Parsern

Prolog-Interpretation

- basiert auf Tiefensuche
- hängt von der Reihenfolge der Klauseln im Regel-Rumpf ab

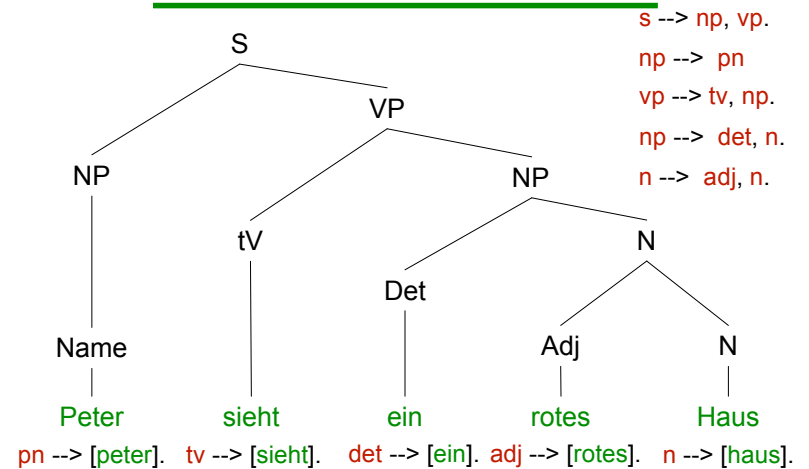
Konsequenzen

- Top-Down-Parser
 - muss die richtige Analyse 'raten'
- Linksrekursive Regeln führen zu unendlichen Suchpfaden
 - die Konjunktionsregeln in F2 sind linksrekursiv
- Im Backtracking werde Teilanalysen mehrfach durchgeführt

Ausweg

- eigener Parser, der aber Regeln ähnlich dem DCG-Format verarbeitet.

Top-Down: Peter sieht ein rotes Haus.



Grammatik mit Lexikon für eigenen Parser (F2)

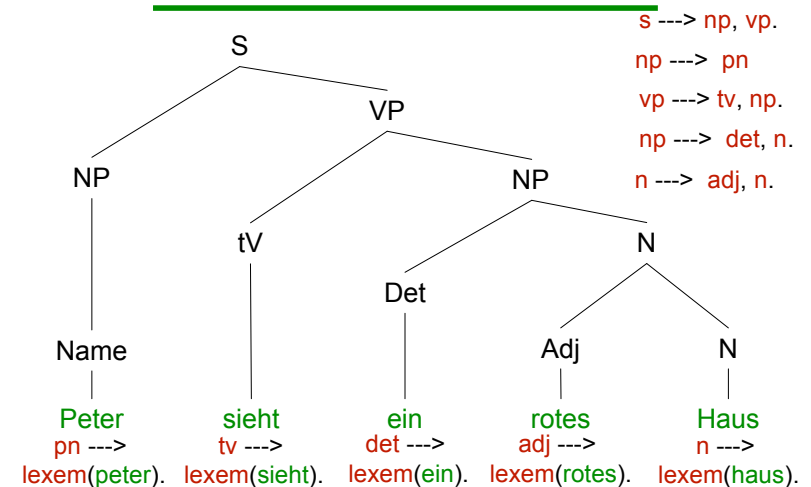
Grammatik

- s ---> np, vp.
- s ---> s, conj, s.
- vp ---> tv, np.
- vp ---> iv.
- vp ---> vp, conj, vp.
- np ---> pn.
- np ---> det, n.
- n ---> adj, n.

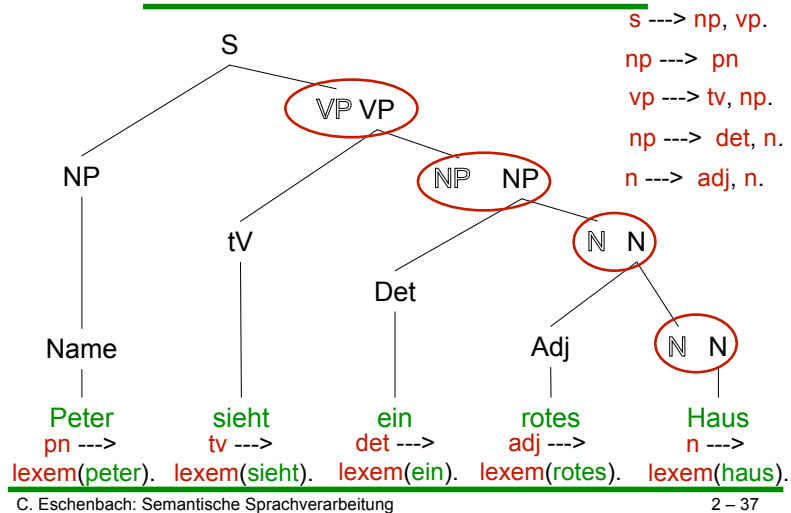
Lexikon

- n ---> lexem(haus).
- n ---> lexem(junge).
- adj ---> lexem(rotes).
- det ---> lexem(ein).
- det ---> lexem(jeder).
- det ---> lexem(kein).
- pn ---> lexem(peter).
- pn ---> lexem(laura).
- iv ---> lexem(lacht).
- iv ---> lexem(winkt).
- tv ---> lexem(sieht).
- conj ---> lexem(und).
- conj ---> lexem(oder).

Bottom-Up: Peter sieht ein rotes Haus.



Bottom-Up mit Vorausschau



Ergebnisse des Parsers

Mit den bisherigen DCG-artigen Grammatiken

- stellen wir nur fest, ob eine Wortfolge ein Satz ist.

Der Syntaxbaum

- ist implizit im Programmablauf beim erfolgreichen Parsing enthalten
- wird aber nicht als Ergebnis erzeugt.

Abhilfe

- Anreicherung der Grammatik-Regeln

Darstellung von Bäumen

Eine visuelle Ausgabe ist erforderlich!

- Beim Parsen wird implizit ein Ableitungsbaum für den Satz durchlaufen.
- Der Ableitungsbaum soll mit Prolog-Termen kodiert werden.

Prolog-Kodierung (eine von viele Möglichkeiten)

- Blatt: `-wort`
- Baum: `wurzel/[teilbaum1, ..., teilbaumn]`

Einbettung in Grammatikregeln

`wurzel(wurzel/[Teilbaum1, ..., Teilbaumn]) --->`
`a(Teilbaum1),..., z(Teilbaumn).`
`wurzel(wurzel/(-wort)) ---> lexem(wort).`

Erzeugung des Syntaxbaums (F2)

<code>s(s/[NP,VP]) ---> np(NP), vp(VP).</code>	<code>n(n/ (-haus)) ---> lexem(haus).</code>
<code>s(s/[S1,C,S2]) ---> s(S1), conj(C), s(S2).</code>	<code>n(n/ (-junge)) ---> lexem(junge).</code>
<code>vp(vp/[TV,NP]) ---> tv(TV), np(NP).</code>	<code>adj(adj/ (-rotes)) ---> lexem(rotes).</code>
<code>vp(vp/[IV]) ---> iv(IV).</code>	<code>det(det/ (-ein)) ---> lexem(ein).</code>
<code>vp(vp/[VP1,C,VP2]) ---> vp(VP1), conj(C), vp(VP2).</code>	<code>det(det/ (-jeder)) ---> lexem(jeder).</code>
<code>np(np/[PN]) ---> pn(PN).</code>	<code>det(det/ (-kein)) ---> lexem(kein).</code>
<code>np(np/[Det, N]) ---> det(Det), n(N).</code>	<code>pn(pn/ (-peter)) ---> lexem(peter).</code>
<code>n(n/[Adj, N]) ---> adj(Adj), n(N).</code>	<code>pn(pn/ (-laura)) ---> lexem(laura).</code>
	<code>iv(iv/ (-lacht)) ---> lexem(lacht).</code>
	<code>iv(iv/ (-winkt)) ---> lexem(winkt).</code>
	<code>tv(tv/ (-sieht)) ---> lexem(sieht).</code>
Parserinformation	<code>conj(conj/ (-und)) ---> lexem(und).</code>
resultierender Syntaxbaum	<code>conj(conj/ (-oder)) ---> lexem(oder).</code>
Sprachlicher Ausdruck	

Nutzung des Syntaxbaums

```
go(Ex):-  
  example(Ex, Phrase), % Waehle Beispiel  
  write(Ex), write(': '), % Gib Beispiel-Name aus  
  writeList(Phrase), nl, % Gib Wortfolge aus  
  recognise(s(T),Phrase), % Parse die Wortfolge als Satz  
  writeln(T),nl, % gib den Syntaxbaum als  
  % Prolog-Term aus  
  
  ppsyn(T, 0), nl, % gib den Syntaxbaum als  
  % formatierten Text aus  
  
  pp_tree(T). % male den Syntaxbaum in  
  % ein separates Fenster
```

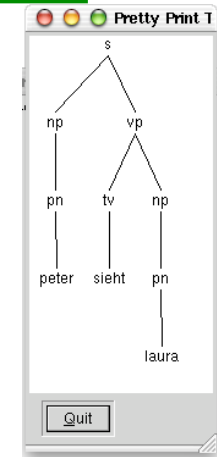
Beispiel

```
example(s2, [peter,sieht,  
  laura]).
```

?- go(s2).

```
s2: peter sieht laura  
s/[np/[pn/(-peter)], vp/[tv/  
  (-sieht), np/[pn/(-laura)]]]
```

```
s(np(pn(peter))  
  vp(tv(sieht)  
    np(pn(laura))))
```



Ausprobieren?

syn_frag2de.pl

- Grammatik mit Erzeugung und Ausgabe der Syntaxbäume

Hilfsmodule

- comsemOperators.pl, achartparser.pl, draw_tree.pl, print.pl

Zur Diskussion

Wie lassen sich die F3-Modifikationen an der F2-Grammatik in die Prolog-Grammatik übertragen?

- Syntaktische Kategorien
- Lexikon
- Grammatik

Zur Diskussion

Welche Rolle spielen

- Kasusinformation
- Kongruenzmerkmale
- Wortformen

Wie lassen sich entsprechende Informationen in der Grammatik berücksichtigen ?

Syntax ist (hier) nur Mittel zum Zweck

Der Zweck ist

- der Aufbau einer semantischen Struktur

Wie werden Bedeutungen repräsentiert ?

Wie werden Bedeutungen berechnet ?

Zusammenfassung

Syntaktische Struktur

- ist Basis für Bedeutungsaufbau
- wird durch Grammatik beschrieben
- wird durch Parser analysiert

Prolog-Syntaxanalyse

- Grundidee
- DCG
- Aufbau von Syntaxbäumen

Nächste Sitzung

- Bedeutungsrepräsentation durch Prädikatenlogik
- Aufbau der semantischen Struktur