
Semantische Sprachverarbeitung

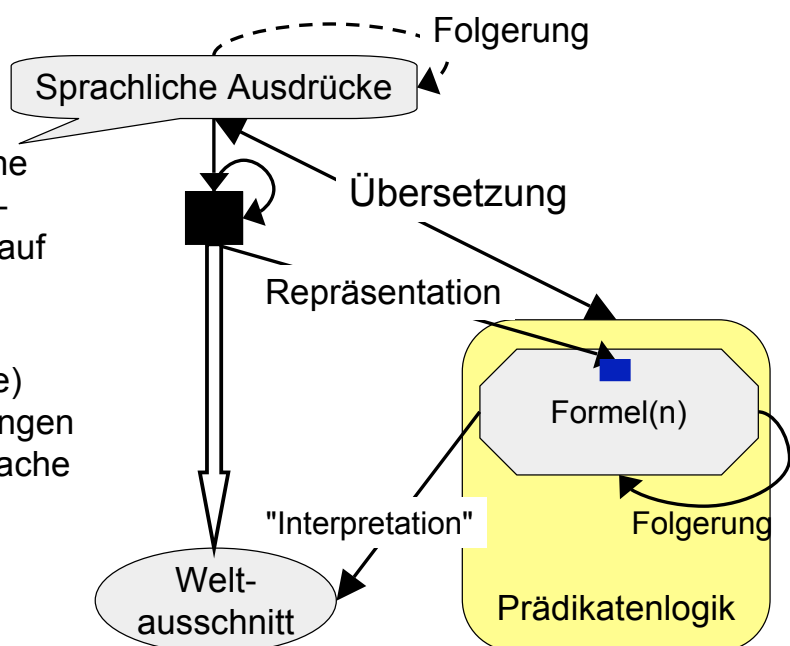
Sitzung 6

- Techniken der Bedeutungserfassung
 - Logik : Semantik
 - Syntaxgesteuerter Bedeutungsaufbau
-

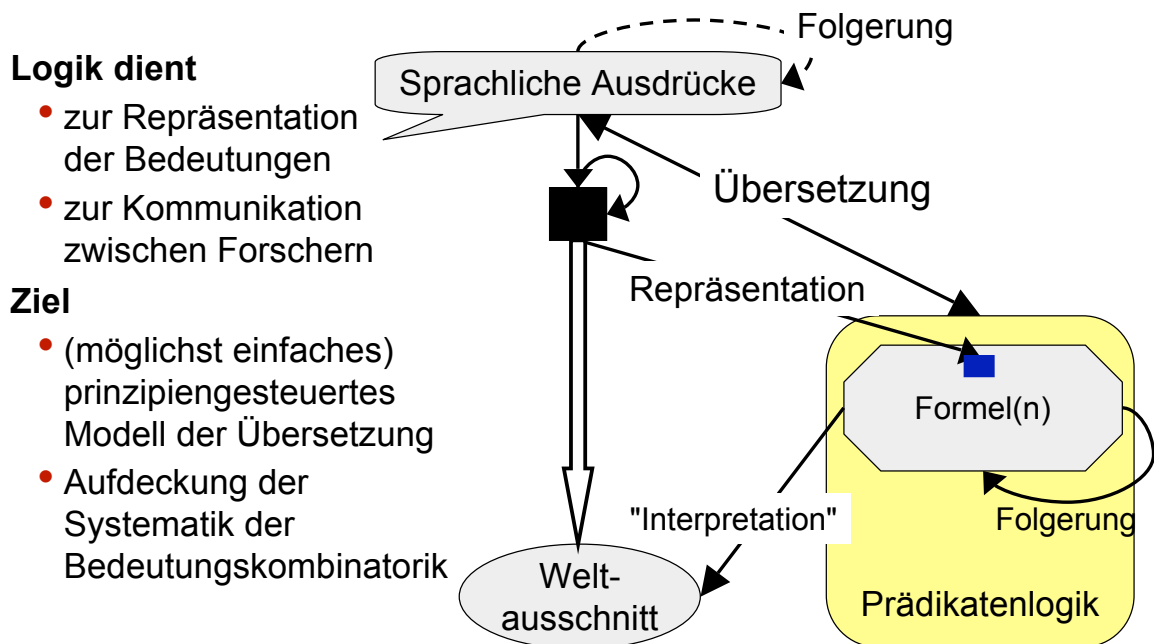
Formale Linguistik

Modellierung von Bedeutung

- durch systematische Abbildung natürlicher Sätze auf prädikatenlogische Formeln,
- so dass (empirische) Folgerungsbeziehungen der natürlichen Sprache als nachweisbare Folgerungen in der Logik resultieren



Formale Linguistik



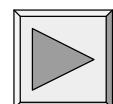
Formale Darstellung von Bedeutungen

Prädikatenlogik

- Formeln repräsentieren Wahrheitsbedingungen
- Prädikate, Funktions- und Relationssymbole repräsentieren Eigenschaften, Abbildungen und Relationen (die Inhalte)
- Konstanten repräsentieren Objekte
- Logische Symbole (Junktoren, Quantoren, Variablen) bieten Inventar für die Verknüpfung der Inhalte

Lambda-Kalkül

- Lambda-Abstraktion erlaubt die Bildung von komplexen Prädikaten, Funktions- und Relationsausdrücken.
- Mächtiges Werkzeug für die Darstellung der Kombinationsregeln



Korrespondenz Syntax - Semantik

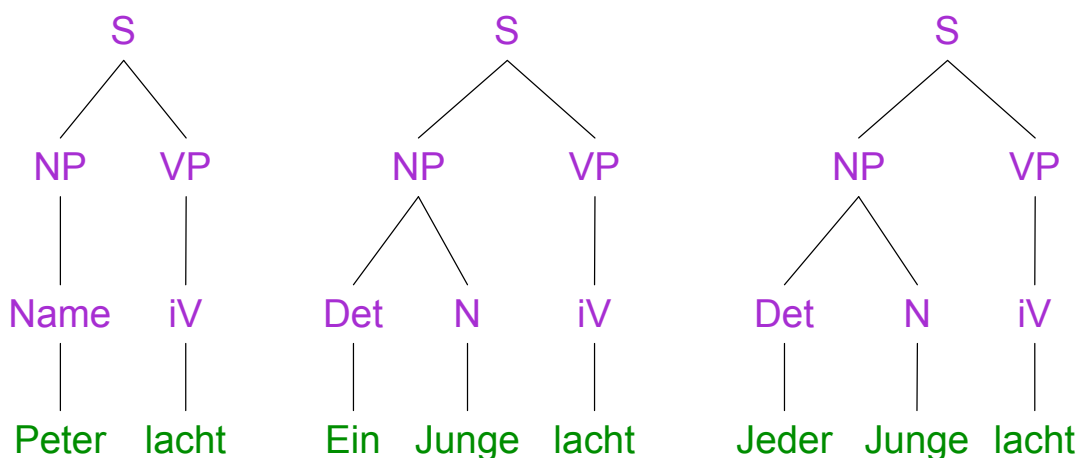
Syntax

- Ausdrücke (Wörter, Konstituenten, Phrasen) gehören **syntaktischen Kategorien** an, die ihren grammatischen Positionsmöglichkeiten entsprechen.
 - Verb (intransitiv, transitiv, di-transitiv, ...), Verbalphrase
 - Nomen, Nominalphrase, Artikel, Determinator, Pronomen

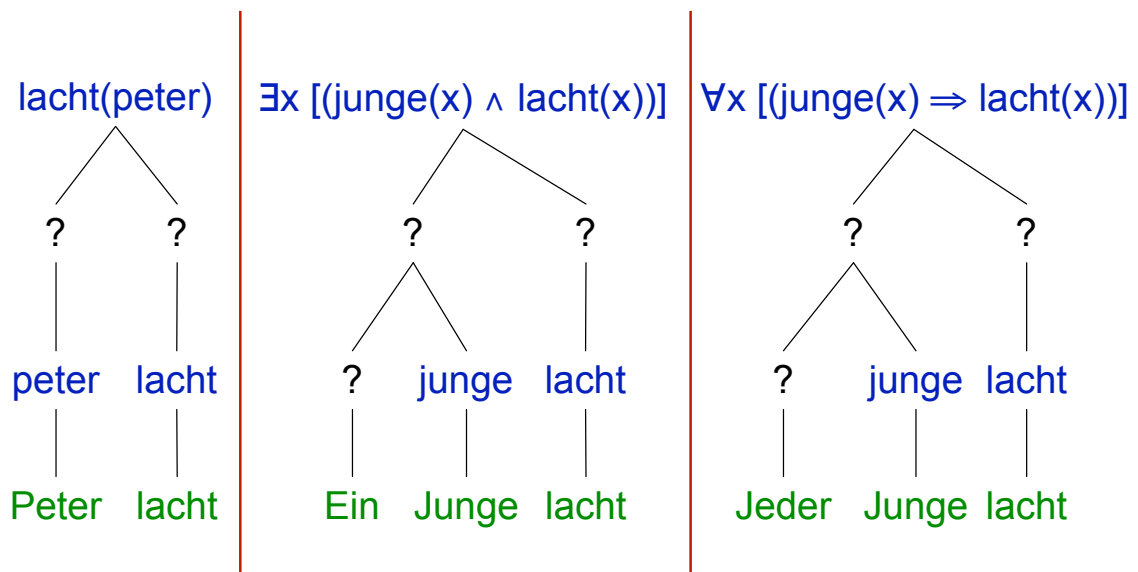
Semantik

- Die Bedeutungen der Ausdrücke haben unterschiedliche **semantische Typen**, die ihrer Funktion im Bedeutungsaufbau entsprechen
 - Proposition, Menge / Eigenschaft, Relation, Quantor, ...

Beispiel: Syntaktische Strukturen



Beispiel



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

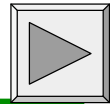
Fragment 1: Einfache (Haupt-) Aussagesätze

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

Beispiele in F1

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



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 FGI1-Vorlesung)

Eine einfache Grammatik mit Lexikon für F1

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

Eine DCG Grammatik für F1

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

Exkurs DCG in Prolog

Ein einfaches Prolog-Programm für Grammatik und Lexikon für F1

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

`np(List, Rest) :- pn(List, Rest).
np(List, Rest) :- det(List, Mid),
n(Mid, Rest).`

`n(List, Rest) :- adj(List, Mid),
n(Mid, Rest).`

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

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

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

`det([ein | Rest], Rest).`

`det([jeder | Rest], Rest).`

`det([kein | Rest], Rest).`

`n([haus | Rest], Rest).`

`n([junge | Rest], Rest).`

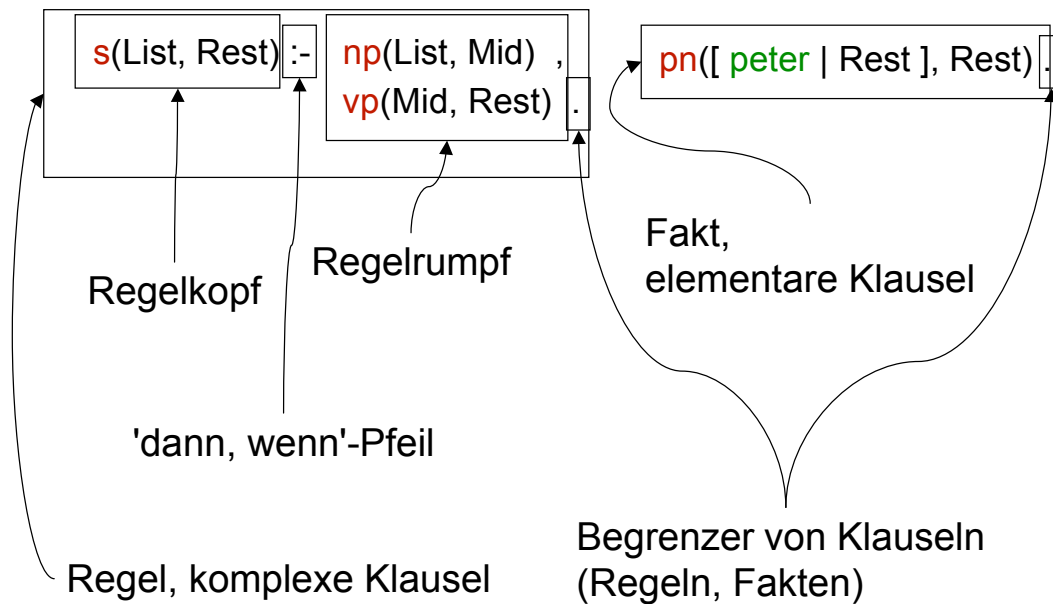
`adj([rotes | Rest], Rest).`

`tv([sieht | Rest], Rest).`

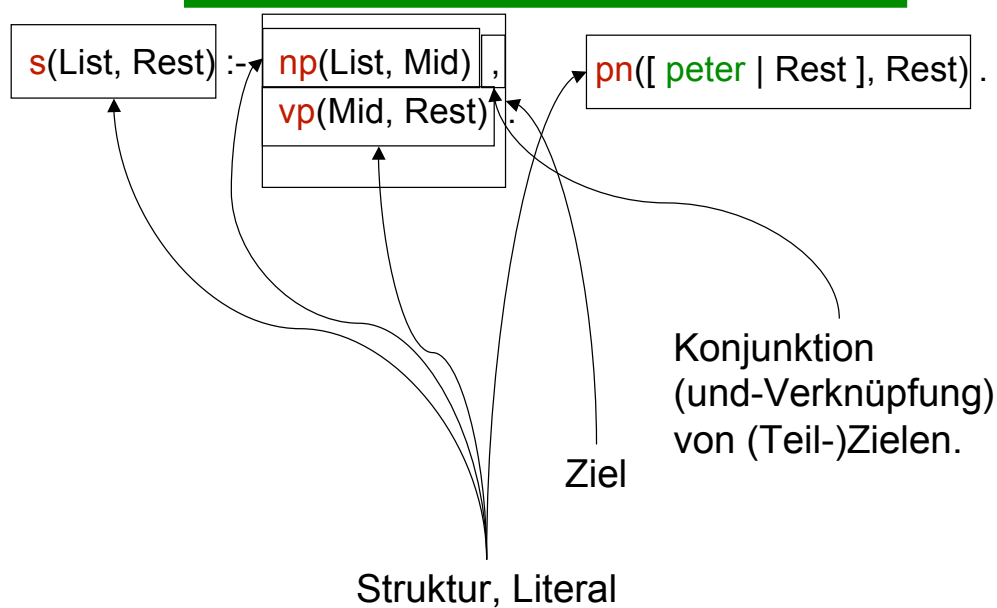
`iv([lacht | Rest], Rest).`

`iv([winkt | Rest], Rest).`

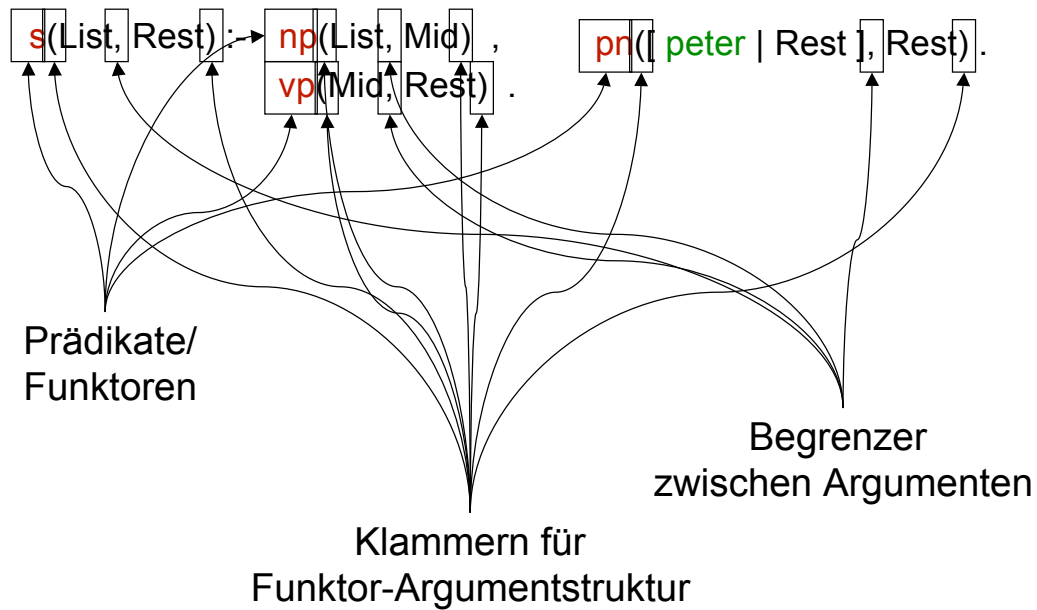
Ein paar Erläuterungen zu Prolog (1)



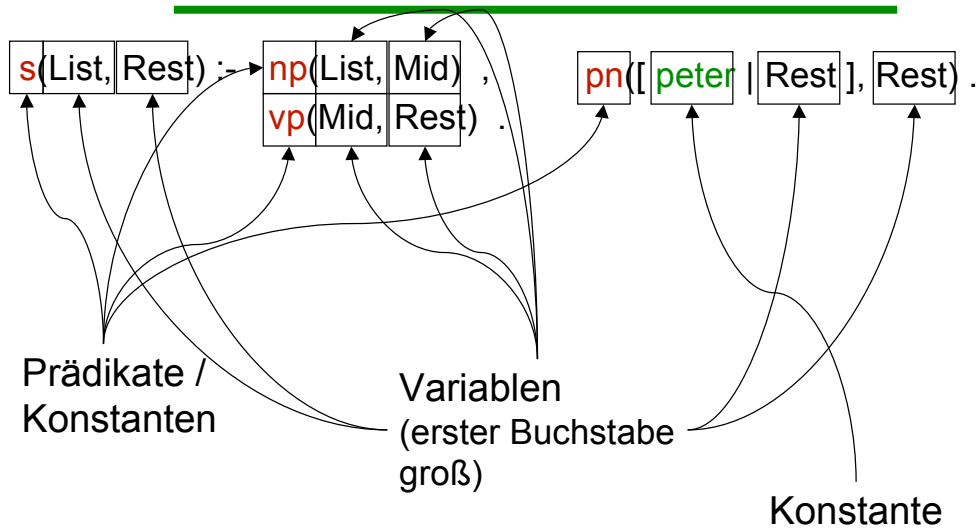
Ein paar Erläuterungen zu Prolog (2)



Ein paar Erläuterungen zu Prolog (3)

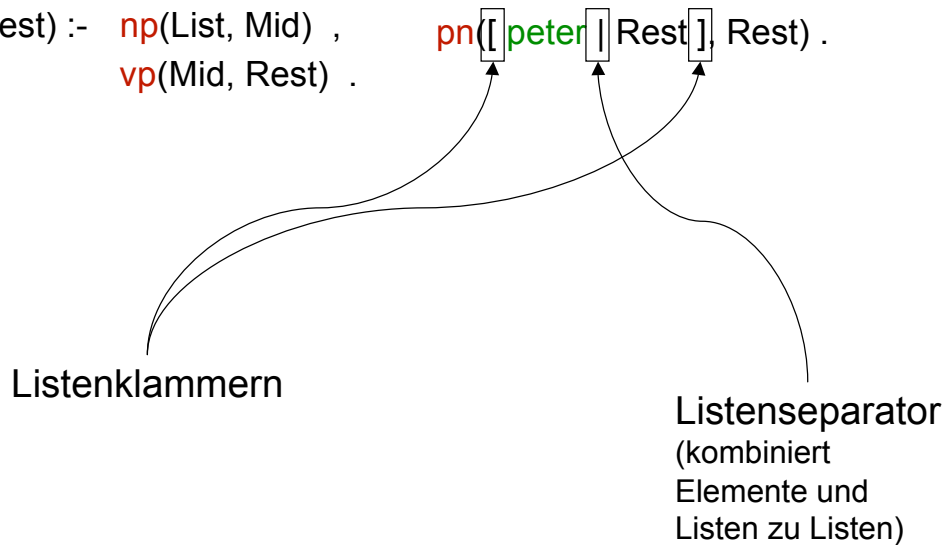


Ein paar Erläuterungen zu Prolog (4)



Ein paar Erläuterungen zu Prolog (5)

`s(List, Rest) :- np(List, Mid) , pn([peter|Rest], Rest) .`
`vp(Mid, Rest) .`



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

Leere Liste

- []

Einelementige Liste

- [element]
- [element | []]

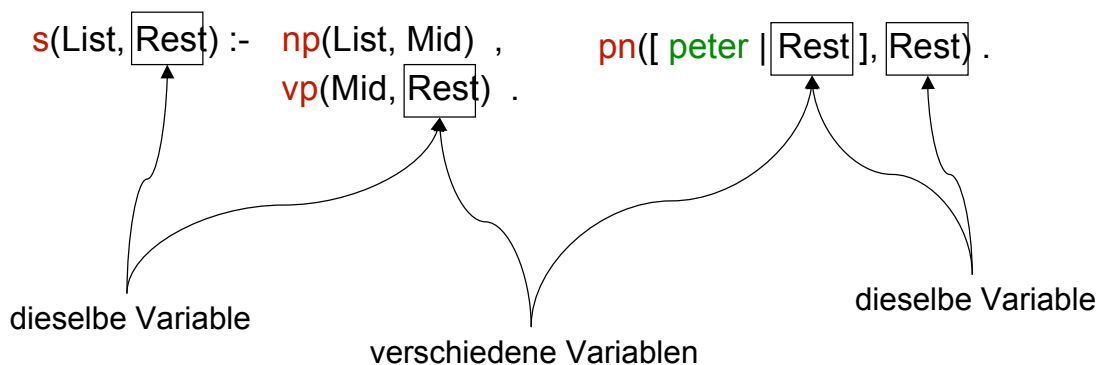
Zweielementige Liste

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

Dreielementige Liste: Mischung der Formate möglich

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

Ein paar Erläuterungen zu Prolog (6)



- Variablen sind (implizit) allquantifiziert.
- Lokale Namensräume: Jede/s Regel/Fakt hat eigene Variablen. (Variablenskopus: Regel, Fakt)

Instanziierung: Grundinstanzen

Regeln und Fakten enthalten Variablen

$np(\text{List}, \text{Rest}) :- \text{det}(\text{List}, \text{Mid}), n(\text{Mid}, \text{Rest}).$

- Variablen sind (implizit) allquantifiziert.
- Lokale Namensräume: Jede/s Regel/Fakt hat eigene Variablen. (Variablenskopus: Regel, Fakt)
- Grundinstanzen entstehen durch (Grund-)Substitution der Variablen durch Grundterme (Terme ohne Variablen)

(Grund-)substitution

- $\text{List} \rightarrow a; \text{Mid} \rightarrow b; \text{Rest} \rightarrow c$

$np(a, c) :- \text{det}(a, b), n(b, c).$

- $\text{List} \rightarrow [\text{das}, \text{haus}, \text{steht}]; \text{Mid} \rightarrow [\text{haus}, \text{steht}]; \text{Rest} \rightarrow [\text{steht}]$

$np([\text{das}, \text{haus}, \text{steht}], [\text{steht}]) :-$

$\text{det}([\text{das}, \text{haus}, \text{steht}], [\text{haus}, \text{steht}]), n([\text{haus}, \text{steht}], [\text{steht}]).$

Instanziierung: Grundinstanzen

$np(\text{List}, \text{Rest}) :- \text{det}(\text{List}, \text{Mid}),$
 $n(\text{Mid}, \text{Rest}).$

Instanzen

$np([\text{ein}], []) :- \text{det}([\text{ein}], []) ,$
 $n([], []).$

$np([\text{ein}, \text{haus}], []) :-$
 $\text{det}([\text{ein}, \text{haus}], [\text{haus}]) ,$
 $n([\text{haus}], []).$

$np([\text{ein}, \text{haus}, \text{steht}], [\text{steht}]) :-$
 $\text{det}([\text{ein}, \text{haus}, \text{steht}],$
 $[\text{haus}, \text{steht}]) ,$
 $n([\text{haus}, \text{steht}], [\text{steht}]) .$

$\text{det}([\text{ein} | \text{Rest}], \text{Rest}).$

$n([\text{haus} | \text{Rest}], \text{Rest}).$

Instanzen

$\text{det}([\text{ein} | []], []).$
 $\text{det}([\text{ein}], []).$

$\text{det}([\text{ein} , \text{haus}], [\text{haus}]) .$

$n([\text{haus}], []).$

$\text{det}([\text{ein}, \text{haus}, \text{steht}], [\text{haus}, \text{steht}]).$

$n([\text{haus} , \text{steht}], [\text{steht}]) .$

Unifikation: Bestimme gemeinsame Instanzen

Unifikatoren

- sind allgemeinste Substitutionen,
- die zwei Ausdrücke mit Variablen auf eine gemeinsame Instanz abbilden.

- Unifikatoren von $\text{det}(\text{List}, \text{Mid})$ und $\text{det}([\text{ein} | \text{Restd}], \text{Restd})$
List \rightarrow [ein | Restd]; Mid \rightarrow Restd: $\text{det}([\text{ein} | \text{Restd}], \text{Restd})$
List \rightarrow [ein | Mid]; Restd \rightarrow Mid: $\text{det}([\text{ein} | \text{Mid}], \text{Mid})$
List \rightarrow [ein | X]; Restd \rightarrow X; Mid \rightarrow X: $\text{det}([\text{ein} | X], X)$

- Unifikatoren von $n(\text{Mid}, \text{Rest})$ und $n([\text{haus} | \text{Restn}], \text{Restn})$
Mid \rightarrow [haus|Restn]; Rest \rightarrow Restn: $n([\text{haus} | \text{Restn}], \text{Restn})$
Mid \rightarrow [haus|Restn]; Restn \rightarrow Rest: $n([\text{haus} | \text{Rest}], \text{Rest})$
Mid \rightarrow [haus|X]; Restn \rightarrow X; Rest \rightarrow X: $n([\text{haus} | X], X)$

Unifikation

Formale Beobachtungen

- wenn zwei Ausdrücke A, B (ohne gemeinsame Variablen) eine gemeinsame (Grund-)Instanz I haben,
- dann gibt es eine Variablen-Substitution uni (Unifikator), für die gilt
 - 1) $\text{uni}(A) = \text{uni}(B)$
 - 2) jede gemeinsame Instanz von A und B ist Instanz von $\text{uni}(A)$ (also ist auch I Instanz von $\text{uni}(A)$)
 - 3) jede andere Variablen-Substitution subst , die (1) und (2) erfüllt, unterscheidet sich von uni nur in der Benennung von Variablen (lexikalische Variante).
- Unifikatoren sind (bis auf Variablenbenennung) eindeutig und einfach zu bestimmen (s. FGI-1).

Unifikation und Prolog: Prolog-Verarbeitungsschleife

Gegeben ein Ziel

- Wähle eine Regel, deren Kopf mit dem Ziel unifiziert, oder ein Fakt, das mit dem Ziel unifiziert.
- Wende den Unifikator auf die gesamte Regel, das Fakt an
- Die resultierenden Rumpf-Ausdrücke sind die weiter zu bearbeitenden Teilziele. Im Falle eines Faktes gibt es keine zu bearbeitenden Teilziele.
- Die aus der Bearbeitung eines Teilzieles resultierenden Unifikatoren sind auf die weiteren Teilziele anzuwenden, bevor diese bearbeitet werden.
- Sind alle Teilziele erfolgreich bearbeitet, dann ist auch das Ziel selbst erfolgreich bearbeitet.
- Der resultierende Unifikator für die Variablen des Zieles ergibt sich aus allen zwischenzeitlich verwendeten Unifikatoren.

Unifikation und Prolog: Fakten (1)

- Gegeben ein Ziel
 $pn([peter, schlaeft], Mid)$
- Wähle ein Fakt, das mit dem Ziel unifiziert.
 $pn([peter | Rest], Rest)$
 $uni_1: Rest \rightarrow [schlaeft]; Mid \rightarrow [schlaeft]$
- Wende den Unifikator auf das Fakt an
 $pn([peter, schlaeft], [schlaeft])$.
- Es ergeben sich keine zu bearbeitenden Teilziele.
- Das Ziel ist erfolgreich bearbeitet.
- Der resultierende Unifikator
 $uni: Mid \rightarrow [schlaeft]$

Unifikation und Prolog: Regeln (1)

- Gegeben ein Ziel
 $np([peter, schlaeft], Mid)$
- Wähle eine Regel, deren Kopf mit dem Ziel unifiziert.
 $np(List, Rest) :- pn(List, Rest).$
 $uni_1: List \rightarrow [peter, schlaeft]; Rest \rightarrow Mid$
- Wende den Unifikator auf die gesamte Regel an
 $np([peter, schlaeft], Mid) :- pn([peter, schlaeft], Mid).$
- weitere Teilziele: die resultierenden Rumpf-Ausdrücke
 $pn([peter, schlaeft], Mid)$
- Alle Teilziele erfolgreich bearbeitet:
 $pn([peter, schlaeft], [schlaeft]) ; uni_2: Mid \rightarrow [schlaeft]$
- Ziel erfolgreich bearbeitet. Der resultierende Unifikator
 $uni: Mid \rightarrow [schlaeft]$

Unifikation und Prolog: Regeln (2)

- Gegeben ein Ziel
 $s([peter, schlaeft], [])$
- Wähle eine Regel, deren Kopf mit dem Ziel unifiziert.
 $s(List, Rest) :- np(List, Mid), vp(Mid, Rest).$
 $uni_1: List \rightarrow [peter, schlaeft]; Rest \rightarrow []$
- Wende den Unifikator auf die gesamte Regel an
 $s([peter, schlaeft], []) :- np([peter, schlaeft], Mid), vp(Mid, []).$
- Die resultierenden Rumpf-Ausdrücke sind die zu bearbeitenden Teilziele.
 $np([peter, schlaeft], Mid), vp(Mid, []).$
- Die aus der Bearbeitung eines Teilzieles resultierenden Unifikatoren sind auf die weiteren Teilziele anzuwenden, bevor diese bearbeitet werden.
 $np([peter, schlaeft], [schlaeft]) ; uni_2: Mid \rightarrow [schlaeft]$
 $vp([schlaeft], []).$
- Sind alle Teilziele erfolgreich bearbeitet, dann ist auch das Ziel erfolgreich bearbeitet.

Grundtechnik: Parsen mit Differenzlisten

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

Instanzen

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

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

pn([peter | Rest], Rest) .

Instanzen

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

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),
 vp(Mid, Rest).

np(List, Rest) :- pn(List, Rest).
np(List, Rest) :- det(List, Mid),
 n(Mid, Rest).

n(List, Rest) :- adj(List, Mid),
 n(Mid, Rest).

vp(List, Rest) :- iv(List, Rest).
vp(List, Rest) :- tv(List, Mid),
 np(Mid, Rest).

pn([peter | Rest], Rest).

pn([laura | Rest], Rest).

det([ein | Rest], Rest).

det([jeder | Rest], Rest).

det([kein | Rest], Rest).

n([haus | Rest], Rest).

n([junge | Rest], Rest).

adj([rotes | Rest], Rest).

tv([sieht | Rest], Rest).

iv([lacht | Rest], Rest).

iv([winkt | Rest], Rest).

Parsen mit Prolog

Die Struktur der Regeln

- einheitlich, systematisch
- eigentlich sehr redundant

Abkürzendes Format DCG (Definite Clause Grammar)

- Statt
 - $c_0(L_1, \text{Rest}) :- c_1(L_1, L_2), c_2(L_2, L_3), \dots, c_n(L_n, \text{Rest}).$
- kurz
 - $c_0 \text{ --> } c_1, c_2, \dots, c_n.$
- Statt
 - $c([\text{konstante} \mid \text{Rest}], \text{Rest}).$
- kurz
 - $c \text{ --> } [\text{konstante}].$

Eine DCG Grammatik für F1

Grammatik

- $s \text{ --> } np, vp.$
- $s \text{ --> } s, conj, s.$
- $vp \text{ --> } tv, np.$
- $vp \text{ --> } iv.$
- $vp \text{ --> } vp, conj, vp.$
- $np \text{ --> } pn.$
- $np \text{ --> } det, n.$
- $n \text{ --> } adj, n.$

Lexikon

- $n \text{ --> } [\text{haus}].$
- $n \text{ --> } [\text{junge}].$
- $adj \text{ --> } [\text{rotes}].$
- $det \text{ --> } [\text{ein}].$
- $det \text{ --> } [\text{jeder}].$
- $det \text{ --> } [\text{kein}].$
- $pn \text{ --> } [\text{peter}].$
- $pn \text{ --> } [\text{laura}].$
- $iv \text{ --> } [\text{lacht}].$
- $iv \text{ --> } [\text{winkt}].$
- $tv \text{ --> } [\text{sieht}].$
- $conj \text{ --> } [\text{und}].$
- $conj \text{ --> } [\text{oder}].$

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] carolae% 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 ?

main_dcg.pl

DCG für ein einfaches Fragment des Deutschen

Über die Vorlesungs-Web-Seite zu erhalten

Ergebnisse des Parsers

Mit der bisherigen DCG-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.
 - Deren Struktur ist durch die nachfolgenden Verarbeitungsschritte bestimmt.

Prolog-Kodierung (eine von viele Möglichkeiten)

- Blatt: `kategorie/(- wort)`
- Baum: `kategorie/[teilbaum1, ..., teilbaumn]`

Beispiel

- `s/[np/[pn/ (-peter)], vp/[iv/ (-schlaeft)]]`

Einbettung in Grammatikregeln

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

Prolog: Terme und Operatoren

Allgemeine Struktur von Prolog-Ausdrücken

- Funktor-Argumentstruktur: `funktor(arg1, arg2, arg3)`
- Präfixnotation, Argumente in Klammern, durch Kommata getrennt.
- Diese Notation funktioniert immer !

Notationsvariante

- (für bestimmte Operatoren vorgesehen, für weitere durch Operatorenendeklaration möglich.)
- Infixnotation für zweistellige Funktoren : `a + b`, `a / b`, ...
(entspricht `+(a, b)`, `/(a, b)`, ...)
- Präfixnotation für einstellige Funktoren ohne Klammern: `- a`,
...
(entspricht `-(a)`, ...)

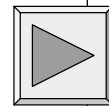
Erzeugung des Syntaxbaums (F1)

Grammatik

$s(s/[NP,VP]) \rightarrow np(NP), vp(VP).$
 $s(s/[S1,C,S2]) \rightarrow s(S1), conj(C), s(S2).$
 $vp(vp/[TV,NP]) \rightarrow tv(TV), np(NP).$
 $vp(vp/[IV]) \rightarrow iv(IV).$
 $vp(vp/[VP1,C,VP2]) \rightarrow vp(VP1), conj(C), vp(VP2).$
 $np(np/[PN]) \rightarrow pn(PN).$
 $np(np/[Det, N]) \rightarrow det(Det), n(N).$
 $n(n/[Adj, N]) \rightarrow adj(Adj), n(N).$

$n(n/(-haus)) \rightarrow [haus].$
 $n(n/(-junge)) \rightarrow [junge].$
 $adj(adj/(-rotes)) \rightarrow [rotes].$
 $det(det/(-ein)) \rightarrow [ein].$
 $det(det/(-jeder)) \rightarrow [jeder].$
 $det(det/(-kein)) \rightarrow [kein].$
 $pn(pn/(-peter)) \rightarrow [peter].$
 $pn(pn/(-laura)) \rightarrow [laura].$
 $iv(iv/(-lacht)) \rightarrow [lacht].$
 $iv(iv/(-winkt)) \rightarrow [winkt].$
 $tv(tv/(-sieht)) \rightarrow [sieht].$
 $conj(conj/(-und)) \rightarrow [und].$
 $conj(conj/(-oder)) \rightarrow [oder].$

Parserinformation
Syntaxbaum
Sprachlicher Ausdruck



Lexikon und Schnittstelle Grammatik/Lexikon

Unifikation und Prolog: Fakten (2)

- Gegeben ein Ziel
 $pn(PN, [peter, schlaeft], Mid)$
- Wähle ein Fakt, das mit dem Ziel unifiziert.
 $pn(pn/(-peter), [peter | Rest], Rest).$
 $uni_1: PN \rightarrow pn/(-peter); Rest \rightarrow [schlaeft]; Mid \rightarrow [schlaeft]$
- Wende den Unifikator auf das Fakt an
 $pn(pn/(-peter), [peter, schlaeft], [schlaeft]).$
- Es ergeben sich keine zu bearbeitenden Teilziele.
- Das Ziel ist erfolgreich bearbeitet.
- Der resultierende Unifikator
 $uni: PN \rightarrow pn/(-peter); Mid \rightarrow [schlaeft]$

Unifikation und Prolog: Regeln (3)

- Gegeben ein Ziel
 $np(NP, [peter, schlaeft], Mid)$
- Wähle eine Regel, deren Kopf mit dem Ziel unifiziert.
 $np(np/[PN], List, Rest) :- pn(PN, List, Rest).$
 $uni_1: NP \rightarrow np/[PN]; List \rightarrow [peter, schlaeft]; Rest \rightarrow Mid$
- Wende den Unifikator auf die gesamte Regel an
 $np(np/[PN], [peter, schlaeft], Mid) :- pn(PN, [peter, schlaeft], Mid).$
- weitere Teilziele: die resultierenden Rumpf-Ausdrücke
 $pn(PN, [peter, schlaeft], Mid)$
- Alle Teilziele erfolgreich bearbeitet:
 $pn(pn/(-peter), [peter, schlaeft], [schlaeft]) ;$
 $uni_2: PN \rightarrow pn/(-peter); Mid \rightarrow [schlaeft]$
- Ziel erfolgreich bearbeitet. Der resultierende Unifikator
 $uni: NP \rightarrow np/[pn/(-peter)]; Mid \rightarrow [schlaeft]$

Unifikation und Prolog: Regeln (4)

- Gegeben ein Ziel
 $s(T, [peter, schlaeft], [])$
- Wähle eine Regel, deren Kopf mit dem Ziel unifiziert.
 $s(s/[NP,VP], List, Rest) :- np(NP, List, Mid), vp(VP, Mid, Rest).$
 $uni_1: T \rightarrow s/[NP,VP]; List \rightarrow [peter, schlaeft]; Rest \rightarrow []$
- Wende den Unifikator auf die gesamte Regel an
 $s(s/[NP,VP], [peter, schlaeft], []) :- np(NP, [peter, schlaeft], Mid), vp(VP, Mid, []).$
- Die resultierenden Rumpf-Ausdrücke sind die zu bearbeitenden Teilziele.
 $np(NP, [peter, schlaeft], Mid), vp(VP, Mid, []).$
- Die aus der Bearbeitung eines Teilzieles resultierenden Unifikatoren sind auf die weiteren Teilziele anzuwenden, bevor diese bearbeitet werden.
 $np(NP, [peter, schlaeft], [schlaeft]) ;$
 $uni_2: NP \rightarrow np/[pn/(-peter)]; Mid \rightarrow [schlaeft]$
 $vp(VP, [schlaeft], []) ; uni_2: VP \rightarrow vp/[iv/(-schlaeft)];$
- Sind alle Teilziele erfolgreich bearbeitet, dann ist auch das Ziel erfolgreich bearbeitet. Der resultierende Unifikator:
 $uni: T \rightarrow s/[np/[pn/(-peter)], vp/[iv/(-schlaeft)]];$

Nutzung des Syntaxbaums

```
go(Ex):-  
  example(Ex, Phrase), % Waehle Beispiel  
  write(Ex), write(': '), % Gib Beispiel-Name aus  
  writeList(Phrase), nl, % Gib Wortfolge aus  
  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
```

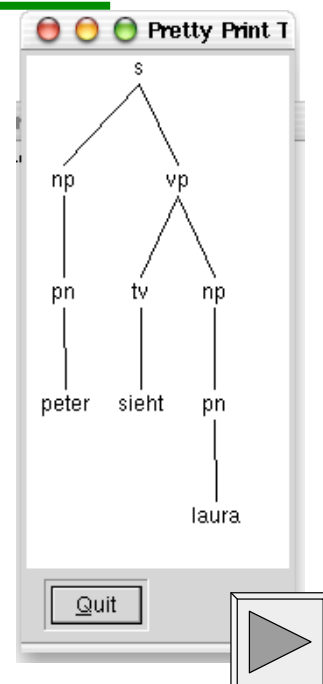
Beispiel

```
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?

DCGSyntaxbaum: main_dcg.pl

- Grammatik mit Erzeugung und Ausgabe der Syntaxbäume

Hilfsmodule

- readLine.pl, draw_tree.pl, print.pl
-

Probleme mit DCG-Parsern

Prolog-Interpretation

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

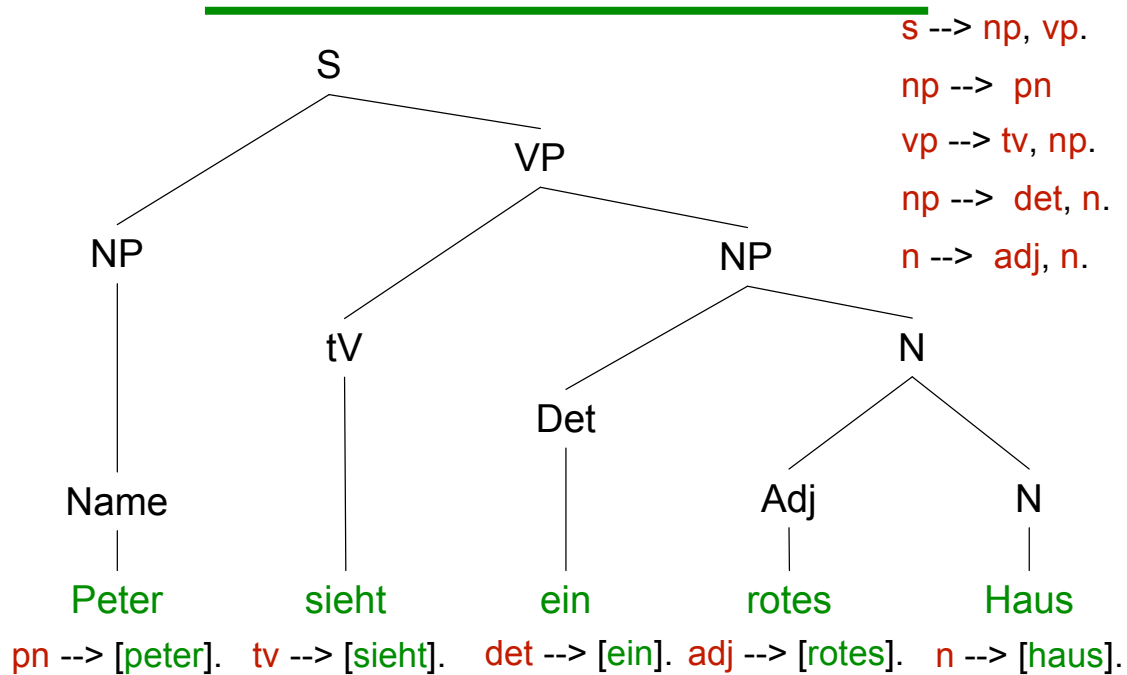
Konsequenzen

- Top-Down-Parser
 - muss die richtige Analyse 'raten'
- Linksrekursive Regeln führen zu unendlichen Suchpfaden
 - die Konjunktionsregeln in F1 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.



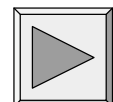
Grammatik mit Lexikon für eigenen Parser (F1)

Grammatik

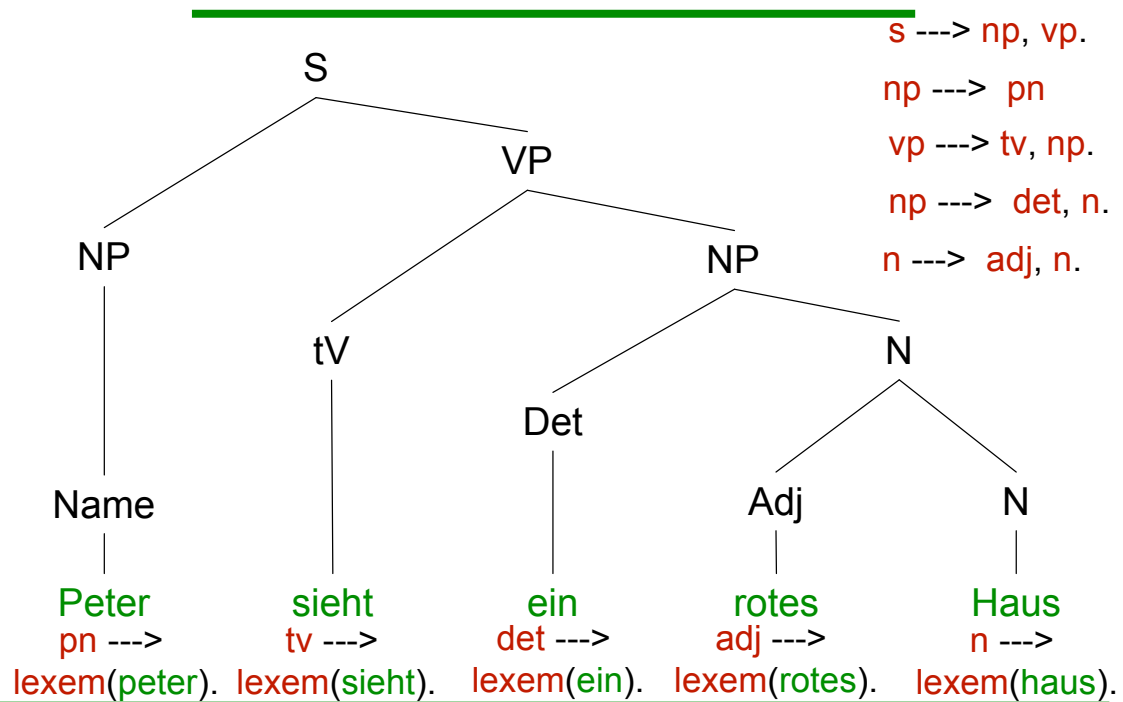
s ---> np, vp.
s ---> s, conj, s.
vp ---> tv, np.
vp ---> iv.
vp ---> vp, conj, vp.
np ---> pn.
np ---> det, n.
n ---> adj, n.

Lexikon

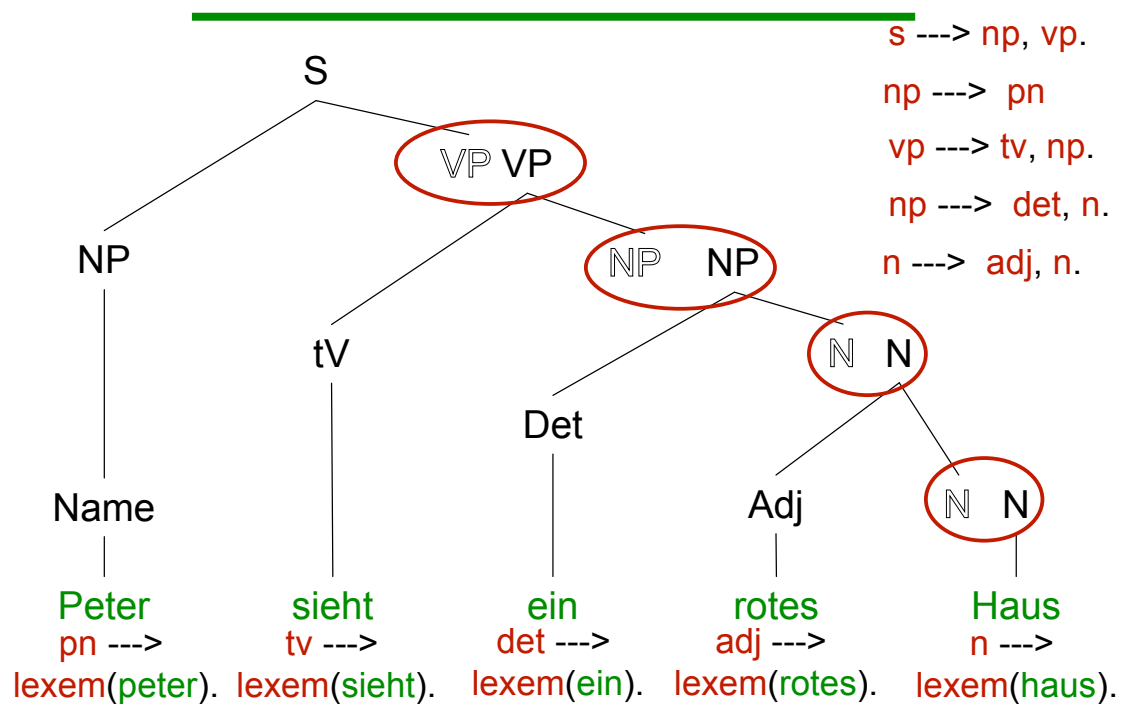
n ---> lexem(haus).
n ---> lexem(junge).
adj ---> lexem(rotes).
det ---> lexem(ein).
det ---> lexem(jeder).
det ---> lexem(kein).
pn ---> lexem(peter).
pn ---> lexem(laura).
iv ---> lexem(lacht).
iv ---> lexem(winkt).
tv ---> lexem(sieht).
conj ---> lexem(und).
conj ---> lexem(oder).



Bottom-Up: Peter sieht ein rotes Haus.



Bottom-Up mit Vorausschau



Erzeugung des Syntaxbaums (F1)

Grammatik

s(s/[NP,VP]) ---> np(NP), vp(VP).
s(s/[S1,C,S2]) ---> s(S1), conj(C),
s(S2).
vp(vp/[TV,NP]) ---> tv(TV), np(NP).
vp(vp/[IV]) ---> iv(IV).
vp(vp/[VP1,C,VP2]) ---> vp(VP1),
conj(C), vp(VP2).
np(np/[PN]) ---> pn(PN).
np(np/[Det, N]) ---> det(Det), n(N).
n(n/[Adj, N]) ---> adj(Adj), n(N).

Parserinformation
Syntaxbaum
Sprachlicher Ausdruck

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

Lexikon und Schnittstelle Grammatik/Lexikon

Lexikonzugriff systematisieren

Das Lexikon

- Appendix zur Grammatik
- Sammlung von Ausnahmen
- ungeordnete Liste von Worten / Lexemen
- Eigenständiges regelhaftes Modul (→ Morphologie)
- Schnittstelle zwischen
 - phonologischer
 - orthographischer
 - syntaktischer und
 - semantischer Ebene

Lexikalische Einträge (LE)

Oberflächenform (Wort) / phonologische Form / orthographische Form

grammatische Merkmale

- Wortart (Wortform)

Bedeutungsbeitrag (semantische Form)

Beispiel

lexem(junge, n, junge).	n(n/ (-Lex))	---> lexem(Lex, n, _).
lexem(winkt, iv, winkt).	iv(iv/ (-Lex))	---> lexem(Lex, iv, _).
lexem(ein, det, indef).	det(det/ (-Lex))	---> lexem(Lex, det, _).

Erzeugung des Syntaxbaums (F1)

Grammatik

Schnittstelle Grammatik/Lexikon

```
s(s/[NP,VP]) ---> np(NP), vp(VP).
s(s/[S1,C,S2]) ---> s(S1), conj(C),
                    s(S2).
vp(vp/[TV,NP]) ---> tv(TV), np(NP).
vp(vp/[IV]) ---> iv(IV).
vp(vp/[VP1,C,VP2]) ---> vp(VP1),
                        conj(C), vp(VP2).
np(np/[PN]) ---> pn(PN).
np(np/[Det, N]) ---> det(Det), n(N).
n(n/[Adj, N]) ---> adj(Adj), n(N).
```

```
n(n/ (-Lex)) ---> lexem(Lex, n, _).
adj(adj/ (-Lex)) ---> lexem(Lex, adj, _).
det(det/ (-Lex)) ---> lexem(Lex, det, _).
pn(pn/ (-Lex)) ---> lexem(Lex, pn, _).
iv(iv/ (-Lex)) ---> lexem(Lex, iv, _).
tv(tv/ (-Lex)) ---> lexem(Lex, tv, _).
conj(conj/ (-Lex)) ---> lexem(Lex, conj, _).
```

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

Parserinformation
Syntaxbaum/Wortart
Sprachlicher Ausdruck

Lexikon



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

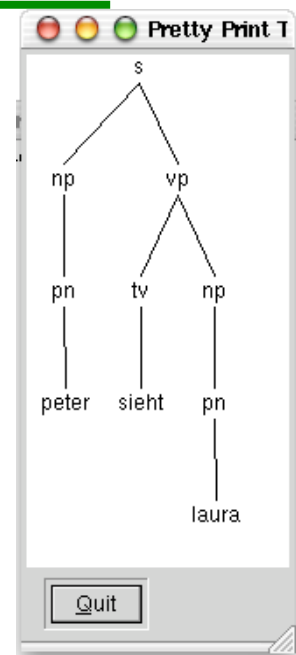
```
example(s2, [peter,sieht,  
laura]).
```

?- 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?

main_cp.pl

- Grammatik mit Erzeugung und Ausgabe der Syntaxbäume

Hilfsmodule

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

Erzeugung semantischer Repräsentationen

Prädikatenlogische Repräsentation: Inventar

Nicht-logisches Inventar

- (freie, verfügbare Symbole)
- Konstanten: peter, ...
- Prädikate (einstellig): junge, lacht, winkt, ...

Logisches Inventar

- (feste, logische Symbole)
- Junktoren, einstellig: \neg (Negation)
zweistellig: \wedge (Konjunktion), \vee (Disjunktion), \Rightarrow (Implikation)
- Quantoren: \forall (All-Quantor), \exists (Existenzquantor)
- Variablen: x, y, z, ...

Hilfssymbole

- Klammern, Komma

Prädikatenlogische Repräsentation: formale Sprache

Terme

- Konstanten
- Variablen
- (Kombinationen aus Funktionssymbolen und Termen)

Formeln

- Kombinationen aus einem Prädikat und einem Term:
 $lacht(peter)$, $winkt(x)$, $junge(y)$
- Kombinationen aus n-stelligen Relationssymbolen und n Termen: $lacht(laura, peter)$
- Negation einer Formel: $\neg lacht(peter)$
- Kombination von zwei Formeln mit einem zweistelligen Junktor: $(winkt(peter) \wedge \neg lacht(peter))$
- Kombination aus Quantor, Variable und Formel: $\exists x [lacht(x)]$

Prädikatenlogische Repräsentation: Interpretation

Eine Interpretation einer prädikatenlogischen Sprache

- ist gegeben durch eine Abbildung der **frei verfügbaren Symbole** auf passende Entitäten
 - Konstanten \rightarrow Objekte
 - Prädikate \rightarrow Mengen von Objekten
 - Relationssymbole \rightarrow Mengen von Objekt-Tupeln
 - Funktionssymbole \rightarrow Funktionen
- determiniert **eindeutig**, welchen Wert ein komplexer geschlossener Ausdruck erhält.
 - prädikatenlogische Grammatik ist eindeutig
 - Interpretation der logischen Symbole ist fest vorgegeben

Prädikatenlogik

Logische Eigenschaften und Relationen

- gelten in allen Interpretationen
- sind unabhängig von der Interpretation der freien Symbole
- **Tautologie**, gültige Formel
 - **wahr** unter jeder Interpretation
z.B. $(\text{lacht}(\text{peter}) \vee \neg \text{lacht}(\text{peter}))$
- **Kontradiktion**, widersprüchliche Formel
 - **falsch** unter jeder Interpretation
z.B. $(\text{lacht}(\text{peter}) \wedge \neg \text{lacht}(\text{peter}))$
- **Folgerung**: aus **F** folgt **G**, wenn unter jeder Interpretation, in der **F** **wahr** ist, auch **G** **wahr** ist
 - z.B. $(\text{lacht}(\text{peter}) \wedge \text{winkt}(\text{peter})) \models \text{lacht}(\text{peter})$

Logische Eigenschaften und Relationen

gelten unter allen Interpretationen

- in der Prädikatenlogik gibt es unendlich viele Interpretationen

können durch Beweisverfahren festgestellt werden

- z.B. Resolution (\rightarrow FGI1-Vorlesung)
- z.B. Tableau-Beweiser (\rightarrow FGI3-Logik-Vorlesung)

aber: Prädikatenlogik (PL) ist nur Semi-Entscheidbar

- für jedes Beweisverfahren gibt es eine Aufgabe, die es nicht (in endlicher Zeit) bearbeiten kann
- man muss damit rechnen, kein Ergebnis zu erhalten

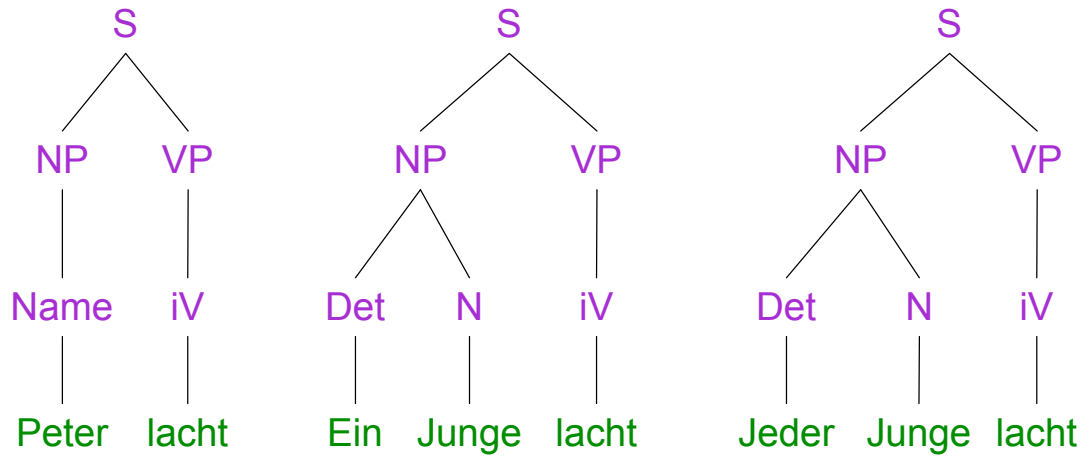
Bestimmte Fragmente der PL sind entscheidbar

Systematische Variationen

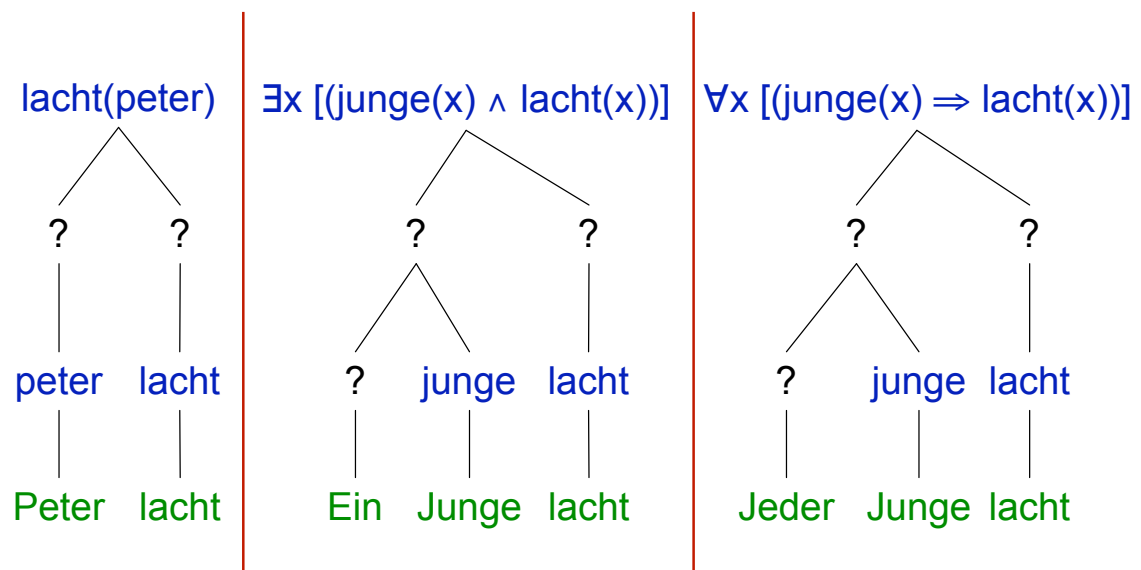
NP-Variation	VP-Konjunktion
<p>$[_{NP} \text{Peter}] [_{VP} \text{lacht}]$. lacht(peter)</p> <p>$[_{NP} \text{Ein Junge}] [_{VP} \text{lacht}]$. $\exists x [\text{junge}(x) \wedge \text{lacht}(x)]$</p> <p>$[_{NP} \text{Jeder Junge}] [_{VP} \text{lacht}]$. $\forall x [\text{junge}(x) \Rightarrow \text{lacht}(x)]$</p> <p>$[_{NP} \text{Kein Junge}] [_{VP} \text{lacht}]$. $\neg \exists x [\text{junge}(x) \wedge \text{lacht}(x)]$</p>	<p>$[_{NP} \text{Peter}] [_{VP} \text{lacht und winkt}]$. lacht(peter) \wedge winkt(peter)</p> <p>$[_{NP} \text{Ein Junge}] [_{VP} \text{lacht und winkt}]$. $\exists x [\text{junge}(x) \wedge (\text{lacht}(x) \wedge \text{winkt}(x))]$</p> <p>$[_{NP} \text{Jeder Junge}] [_{VP} \text{lacht und winkt}]$. $\forall x [\text{junge}(x) \Rightarrow (\text{lacht}(x) \wedge \text{winkt}(x))]$</p> <p>$[_{NP} \text{Kein Junge}] [_{VP} \text{lacht und winkt}]$. $\neg \exists x [\text{junge}(x) \wedge (\text{lacht}(x) \wedge \text{winkt}(x))]$</p>
Satz-Konjunktion	
<p>$[_{NP} \text{Peter}] [_{VP} \text{lacht}]$ und $[_{NP} \text{Peter}] [_{VP} \text{winkt}]$. lacht(peter) \wedge winkt(peter)</p> <p>$[_{NP} \text{Ein Junge}] [_{VP} \text{lacht}]$ und $[_{NP} \text{ein Junge}] [_{VP} \text{winkt}]$. $\exists x [\text{junge}(x) \wedge \text{lacht}(x)] \wedge \exists y [\text{junge}(y) \wedge \text{winkt}(y)]$</p> <p>$[_{NP} \text{Jeder Junge}] [_{VP} \text{lacht}]$ und $[_{NP} \text{jeder Junge}] [_{VP} \text{winkt}]$. $\forall x [\text{junge}(x) \Rightarrow \text{lacht}(x)] \wedge \forall y [\text{junge}(y) \Rightarrow \text{winkt}(y)]$</p> <p>$[_{NP} \text{Kein Junge}] [_{VP} \text{lacht}]$ und $[_{NP} \text{kein Junge}] [_{VP} \text{winkt}]$. $\neg \exists x [\text{junge}(x) \wedge \text{lacht}(x)] \wedge \neg \exists y [\text{junge}(y) \wedge \text{winkt}(y)]$</p>	

Bedeutungsaufbau

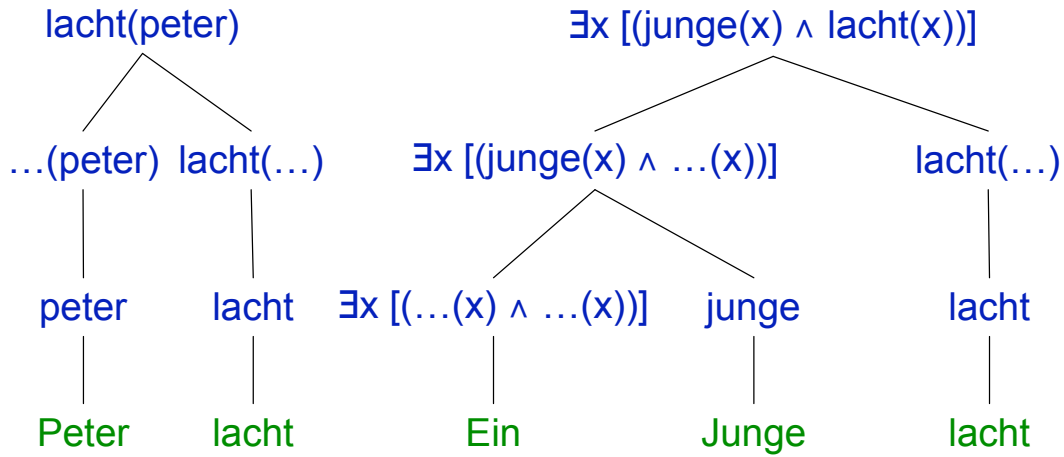
Beispiel: Syntaktische Strukturen



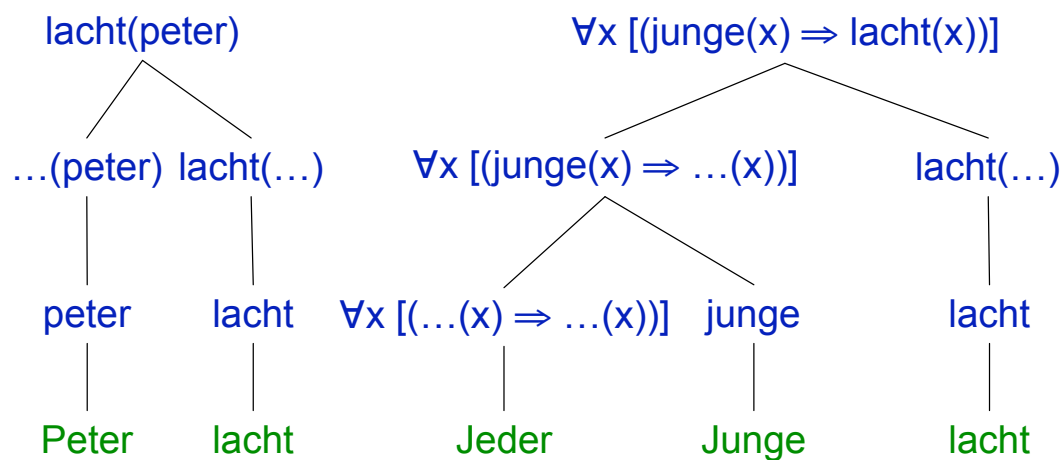
Beispiel



Beispiel



Beispiel



Grundidee

die Bedeutung eines Ausdrucks

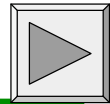
- beinhaltet einen eigentlichen, inhaltlichen Beitrag
- weist (wohldefinierte) Lücken auf, die der Kontext spezifiziert
- Beitrag und Lücken werden zu prädikatenlogischen Fragmenten zusammengesetzt

der Typ des Ausdrucks

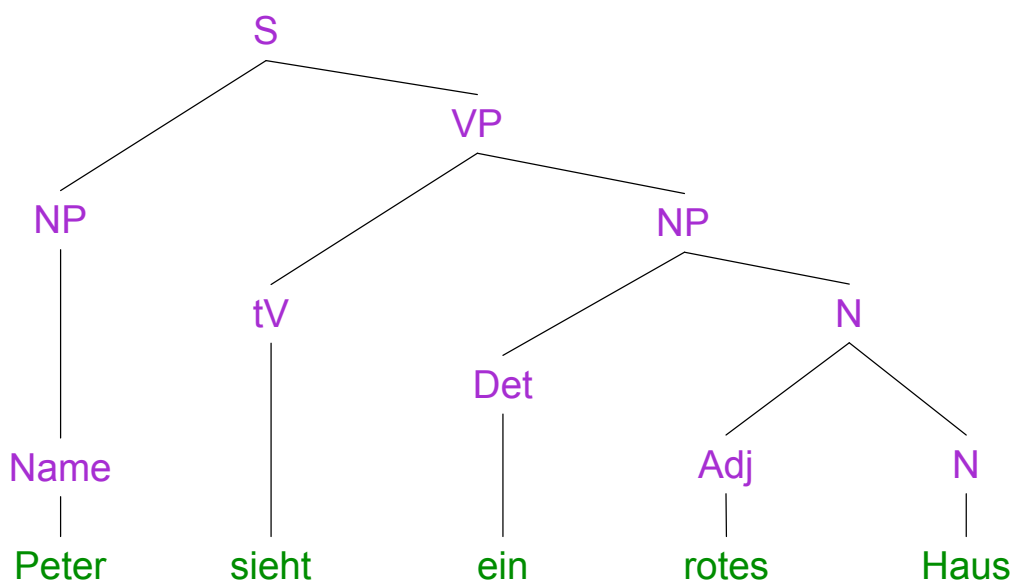
- determiniert, welche Lücken vom Kontext zu füllen sind

Regeln

- determinieren, wie die Lücken gefüllt werden

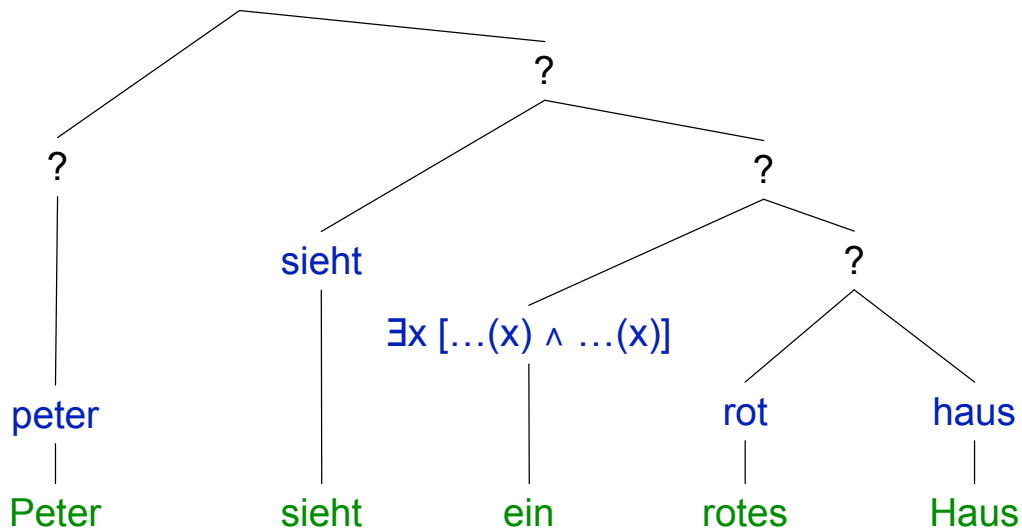


Peter sieht ein rotes Haus.



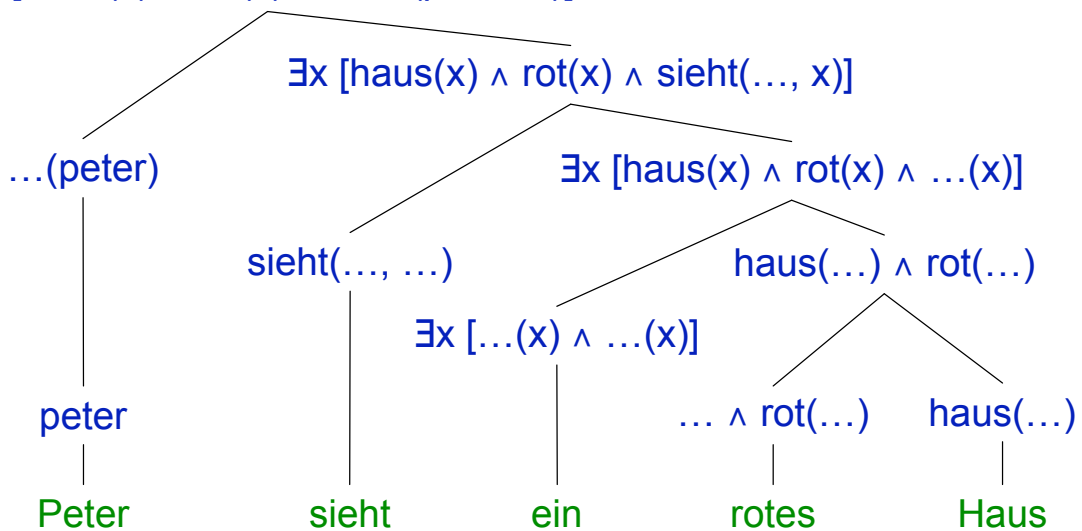
Wahrheitsbedingungen: Peter sieht ein rotes Haus.

$\exists x [\text{haus}(x) \wedge \text{rot}(x) \wedge \text{sieht}(\text{peter}, x)]$



Wahrheitsbedingungen: Peter sieht ein rotes Haus.

$\exists x [\text{haus}(x) \wedge \text{rot}(x) \wedge \text{sieht}(\text{peter}, x)]$



Grundidee

die Bedeutung eines Ausdrucks

- beinhaltet einen eigentlichen, inhaltlichen Beitrag
- weist (wohldefinierte) Lücken auf, die der Kontext spezifiziert
- Beitrag und Lücken werden zu prädikatenlogischen Fragmenten zusammengesetzt

der Typ des Ausdrucks

- determiniert, welche Lücken vom Kontext zu füllen sind

Regeln

- determinieren, wie die Lücken gefüllt werden

PROLOG-Umsetzung

- Lücken und Prädikatenlogik-Variablen werden durch Prolog-Variablen repräsentiert

Prädikatenlogik in PROLOG

- Formeln und Terme werden durch PROLOG-Terme repräsentiert
- Variablen
 - PROLOG-Variablen

Kein Symbol-Zeichensatz in PROLOG

- Junktoren
 - &, v, ->, ~ anstelle von \wedge , \vee , \Rightarrow , \neg
 - definiert als (Infix-)Operatoren
- Quantoren
 - exists(X, F), forall(Y, F) anstelle von $\exists x [F]$, $\forall y [F]$
- freie Symbole: Deklaration von Konstanten und Relationen
 - constant(laura). relation(junge,1).

Verknüpfung von Lexikon und Grammatik (F1)

$n(n/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, n, \text{Sem}).$
 $\text{adj}(\text{adj}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{adj}, \text{Sem}).$
 $\text{det}(\text{det}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{det}, \text{Sem}).$
 $\text{pn}(\text{pn}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{pn}, \text{Sem}).$
 $\text{iv}(\text{iv}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{iv}, \text{Sem}).$
 $\text{tv}(\text{tv}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{tv}, \text{Sem}).$
 $\text{conj}(\text{conj}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{conj}, \text{Sem}).$

Satz-Konjunktionen (F1)

verknüpfen zwei Sätze zu einem Satz

Lexikon

$\text{lexem}(\text{und}, \text{conj}, [F, G]/[F \& G]).$

$\text{lexem}(\text{oder}, \text{conj}, [F, G]/[F \vee G]).$

Regeln

$s(s([S1, C, S2], \text{For}) \text{ ---> } s(S1, F), \text{conj}(C, [F, G]/\text{For}), s(S2, G)).$

Verknüpfung von Lexikon und Grammatik

$\text{conj}(\text{conj}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{conj}, \text{Sem}).$

Parser-Information

Syntaxbaum

Semantik

Wortform

inhaltlicher Beitrag

interne Lücke

Lücken-Info nach außen geben

Lücken-Info nutzen und füllen

Resultat nach außen geben

Satz- und VP-Konjunktionen (F1)

verknüpfen zwei Sätze zu einem Satz

Lexikon

$\text{lexem}(\text{und}, \text{conj}, [F, G]/(F \ \& \ G)).$

$\text{lexem}(\text{oder}, \text{conj}, [F, G]/(F \ \vee \ G)).$

Regeln

$s(s/[S1,C,S2], \text{For}) \text{ ---> } s(S1, F), \text{conj}(C, [F, G]/\text{For}), s(S2, G).$

$vp(vp/[VP1,C,VP2], [X]/\text{For}) \text{ ---> } vp(VP1, [X]/F),$
 $\text{conj}(C, [F, G]/\text{For}), vp(VP2,[X]/G).$

Die Regel für Verbalphrasenkonjunktion unterscheidet sich nur in der Behandlung von Lücken.

Nomen und Adjektive (F1)

Nomen drücken Eigenschaften aus

Adjektive modifizieren Eigenschaften

Lexikon

$\text{lexem}(\text{haus}, n, [X]/(\text{haus}(X))).$

$\text{lexem}(\text{junge}, n, [X]/(\text{junge}(X))).$

$\text{lexem}(\text{rotes}, \text{adj}, [X, F]/(\text{rot}(X) \ \& \ F)).$

Regel

$n(n/[Adj, N], [X]/\text{For}) \text{ ---> } \text{adj}(\text{Adj}, [X, F]/\text{For}),$
 $n(N, [X]/F).$

AN IMPORTANT JOB

This is a story about four people named Everybody, Somebody, Anybody and Nobody.

There was an important job to be done and Everybody was sure that Somebody would do it.

Anybody could have done it, but Nobody did it.

Somebody got angry about that, because it was Everybody's job.

Everybody thought Anybody could do it, but Nobody realised that Everybody wouldn't do it.

It ended up that Everybody blamed Somebody when Nobody did what Anybody could have.

Nominalphrasen (F1)

Determinatoren verknüpfen Nomen und VPs

Lexikon

lexem(ein, det, [X, F, G]/exists(X, F & G)) .

lexem(jeder, det, [X, F, G]/forall(X, F -> G)).

lexem(kein, det, [X, F, G]/(~ exists(X, F & G))).

lexem(peter, pn, peter).

lexem(laura, pn, laura).

Regel

np(np/[Det, N], [X, G]/For) ---> **det**(Det, [X, F, G]/For), **n**(N,[X]/F).

np(np/[PN], [Name, For]/For) ---> **pn**(PN, Name).

Verbalphrasen und Sätze (F1)

Lexikon

lexem(lacht, iv, [X]/lacht(X)).

lexem(winkt, iv, [X]/winkt(X)).

lexem(sieht, tv, [X, Y]/sieht(X, Y)).

Regeln

vp(vp/[IV], [X]/For) ----> iv(IV, [X]/For).

vp(vp/[TV,NP], [X]/For) ----> tv(TV, [X, Y]/G),
np(NP, [Y, G]/For).

s(s/[NP,VP], For) ----> np(NP, [X, G]/For), vp(VP, [X]/G).

Grammatik mit Syntaxbaum und Semantik (F1)

s(s/[NP,VP], For) ----> np(NP, [X, G]/For), vp(VP, [X]/G).

np(np/[PN], [Name, For]/For) ----> pn(PN, Name).

np(np/[Det, N], [X, G]/For) ----> det(Det, [X, F, G]/For), n(N, [X]/F).

n(n/[Adj, N], [X]/For) ----> adj(Adj, [X, F]/For), n(N, [X]/F).

vp(vp/[IV], [X]/For) ----> iv(IV, [X]/For).

vp(vp/[TV,NP], [X]/For) ----> tv(TV, [X, Y]/G),
np(NP, [Y, G]/For).

vp(vp/[VP1,C,VP2], [X]/For) ----> vp(VP1, [X]/F),
conj(C, [F, G]/For), vp(VP2, [X]/G).

s(s/[S1,C,S2], For) ----> s(S1, F), conj(C, [F, G]/For),
s(S2, G).

Verknüpfung von Lexikon und Grammatik (F1)

$n(n/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, n, \text{Sem}).$
 $\text{adj}(\text{adj}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{adj}, \text{Sem}).$
 $\text{det}(\text{det}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{det}, \text{Sem}).$
 $\text{pn}(\text{pn}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{pn}, \text{Sem}).$
 $\text{iv}(\text{iv}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{iv}, \text{Sem}).$
 $\text{tv}(\text{tv}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{tv}, \text{Sem}).$
 $\text{conj}(\text{conj}/ (-\text{Lexem}), \text{Sem}) \text{ ---> } \text{lexem}(\text{Lexem}, \text{conj}, \text{Sem}).$

Lexikon mit Syntax und Semantik (F1)

$\text{lexem}(\text{haus}, n, [X]/(\text{haus}(X))).$
 $\text{lexem}(\text{junge}, n, [X]/(\text{junge}(X))).$
 $\text{lexem}(\text{rotes}, \text{adj}, [X, F]/(\text{rot}(X) \ \& \ F)).$
 $\text{lexem}(\text{ein}, \text{det}, [X, F, G]/\text{exists}(X, F \ \& \ G)).$
 $\text{lexem}(\text{jeder}, \text{det}, [X, F, G]/\text{forall}(X, F \rightarrow G)).$
 $\text{lexem}(\text{kein}, \text{det}, [X, F, G]/(\sim \text{exists}(X, F \ \& \ G))).$
 $\text{lexem}(\text{peter}, \text{pn}, \text{peter}).$
 $\text{lexem}(\text{laura}, \text{pn}, \text{laura}).$
 $\text{lexem}(\text{lacht}, \text{iv}, [X]/\text{lacht}(X)).$
 $\text{lexem}(\text{winkt}, \text{iv}, [X]/\text{winkt}(X)).$
 $\text{lexem}(\text{sieht}, \text{tv}, [X, Y]/\text{sieht}(X, Y)).$
 $\text{lexem}(\text{und}, \text{conj}, [F, G]/(F \ \& \ G)).$
 $\text{lexem}(\text{oder}, \text{conj}, [F, G]/(F \vee G)).$

Ausprobieren ?

Paket PL

main.pl

- Grammatik mit Erzeugung und Ausgabe der Syntaxbäume und prädikatenlogischen Formeln (Semantik)
-

Theoretischer Hintergrund

Prolog-Umsetzung

- aber was wird hier umgesetzt ?
- was ist die Theorie dahinter ?
- wie ist das Beispiel zu verallgemeinern ?

Typen und Lambda-Kalkül

Nächster Foliensatz
