Featuregrammatiken

Vorlesung “Computerlinguistische Techniken”

Alexander Koller

21. November 2014

Musterlösung Ü2 A4

- Erste 100 Wörter des Brown-Korpus ausgeben:

```
import nltk
words = nltk.corpus.brown.words()

for i in range(100):
    print(words[i])
```

- Wie viele Tokens im Brown-Korpus?

  `len(words)` → 1161192

- Wie viele Types im Brown-Korpus?

  `len(set(words))` → 56057

Musterlösung Ü2 A4

- 100 häufigste Wörter, absteigend nach Häufigkeit:

```
def add(dictionary, word):
    if word in dictionary:
        dictionary[word] = dictionary[word] + 1
    else:
        dictionary[word] = 1

dictionary = {}
for word in words:
    add(dictionary,word.lower())

print sorted(dictionary.items(), key=itemgetter(1),
             reverse=True)[:100]
```

Übersicht

- Features
- Featurestrukturen und Unifikation
- Featuregrammatiken
- Parsing
- Expressivität

Agreement

- Wir haben in unseren Mini-kfGs bisher Kongruenz völlig ignoriert:

The boy sleeps.	* The boy sleep.
The boys sleep.	* The boys sleeps.

- Wie modelliert man Kongruenz am effektivsten in einer Grammatik?

Subkategorisierung

- Verschiedene Verben haben verschiedene Subkategorisierungsrahmen, die angeben, welche Komplemente das Verb braucht.

  The boy sleeps.
  The boy eats a sandwich.
  The boy gives the girl a book.
  * The boy sleeps a sandwich.

- Wie modelliert man das effektiv?

1. Versuch: mit kfGs

S → NP VP
VP → IV
VP → TV NP
VP → DV NP NP
IV → sleeps
TV → eats
DV → gives

1. Versuch: mit kfGs

S → NPsg VPsg
VPsg → IVsg
VPsg → TVsg NP
VPsg → DVsg NP NP
IVsg → sleeps
TVsg → eats
DVsg → gives

S → NPpl VPpl
VPpl → IVpl
VPpl → TVpl NP
VPpl → DVpl NP NP
IVpl → sleep
TVpl → eat
DVpl → give

Das Problem

- KfGs: Kongruenzartige Information muss in Nichtterminalen codiert werden.

- Kongruenz und Subkategorisierung nur zwei von vielen Constraints, die interagieren können.

Das Problem

- Für große Grammatiken bedeutet das: Brauchen extrem viele NTs und Regeln, die (für uns) alle fast das gleiche aussagen.

- Aufwendig in Entwicklung und Wartung.

- Grundsatz: In Programmen und Grammatiken Redundanz vermeiden!

Features

- Alternative: An Nichtterminale *Merkmale* (engl. *Features*) anhängen, deren Werte separat verarbeitet werden können.

- In Grammatik enthalten Features Variablen, die durch *Unifikation* ihre Werte bekommen.

- Wir werden dann *Featuregrammatiken* daraus bauen.

2. Versuch: mit Features

S → NP[num:sg] VP[num:sg]
VP[num:sg] → IV[num:sg]
VP[num:sg] → TV[num:sg] NP
VP[num:sg] → DV[num:sg] NP NP
IV[num:sg] → sleeps
TV[num:sg] → eats
DV[num:sg] → gives

S → NP[num:pl] VP[num:pl]
VP[num:pl] → IV[num:pl]
VP[num:pl] → TV[num:pl] NP
VP[num:pl] → DV[num:pl] NP NP
IV[num:pl] → sleep
TV[num:pl] → eat
DV[num:pl] → give

2. Versuch: mit Features

S → NP[num:X] VP[num:X]
VP[num:X] → IV[num:X]
VP[num:X] → TV[num:X] NP
VP[num:X] → DV[num:X] NP NP
IV[num:sg] → sleeps
TV[num:sg] → eats
DV[num:sg] → gives

S → NP[num:X] VP[num:X]
VP[num:X] → IV[num:X]
VP[num:X] → TV[num:X] NP
VP[num:X] → DV[num:X] NP NP
IV[num:pl] → sleep
TV[num:pl] → eat
DV[num:pl] → give

2. Versuch: mit Features

S → NP[num:X] VP[num:X]
VP[num:X] → IV[num:X]
VP[num:X] → TV[num:X] NP
VP[num:X] → DV[num:X] NP NP

IV[num:sg] → sleeps
TV[num:sg] → eats
DV[num:sg] → gives

IV[num:pl] → sleep
TV[num:pl] → eat
DV[num:pl] → give

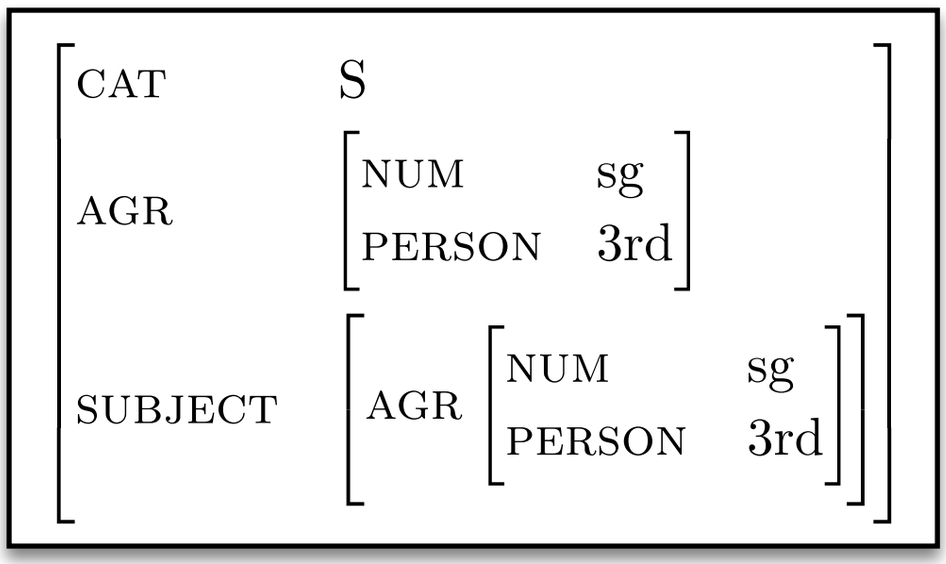
Feature-Strukturen

- Eine Featurestruktur (FS) gruppiert mehrere Zuweisungen von Werten an Features.

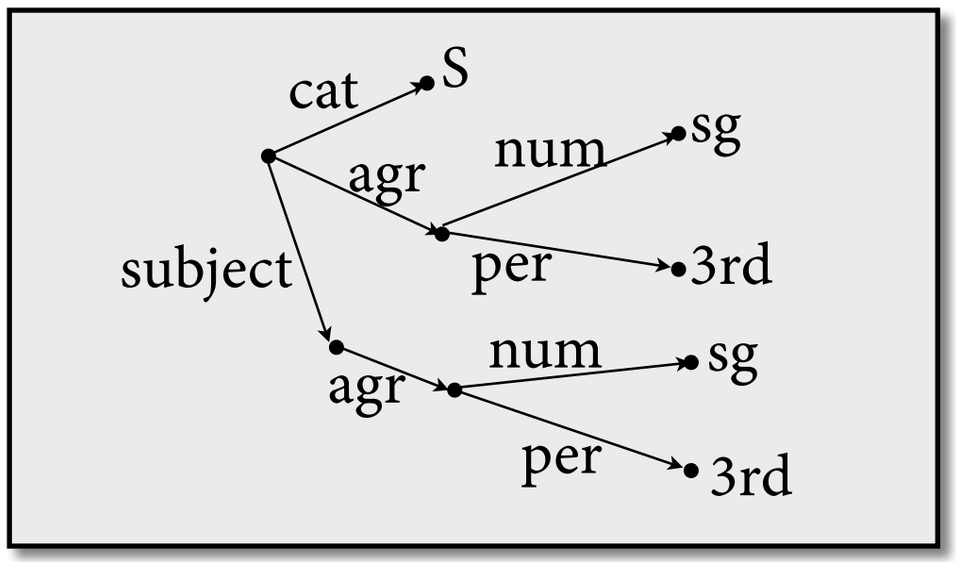
- Verschiedene Featurestrukturen können verschiedene Features enthalten.

- Werte von Features können sein:
  - atomare Werte
  - Featurestrukturen

Feature-Strukturen sind DAGs



Attribut-Wert-Matrix
(AVM)



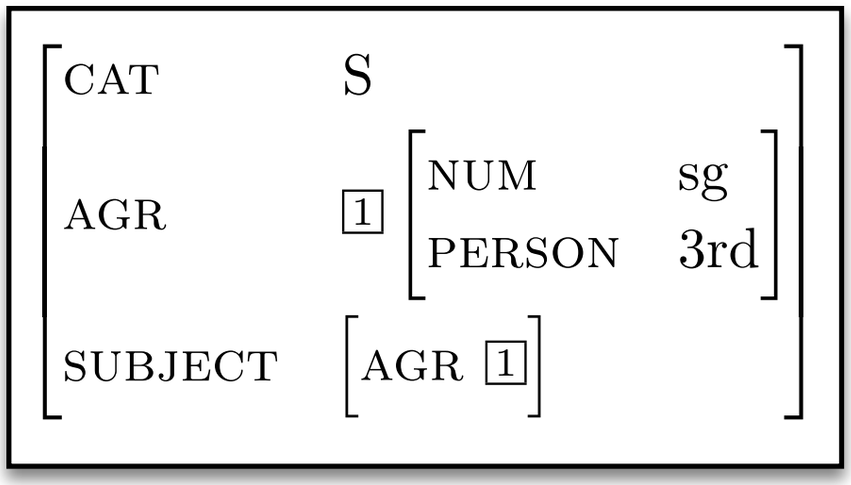
Feature-Struktur
(= gerichteter azyklischer Graph, DAG)

Beschreibung

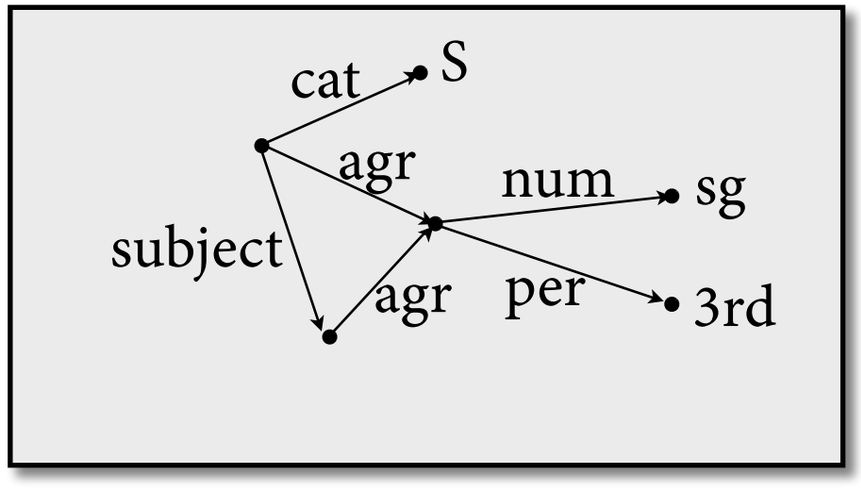
Modell

Oft nimmt man AVM und Feature-Struktur als alternative Schreibweisen; ist aber nicht ganz präzise.

Reentrancy



Attribut-Wert-Matrix
(AVM)



Feature-Struktur
(= gerichteter azyklischer Graph)

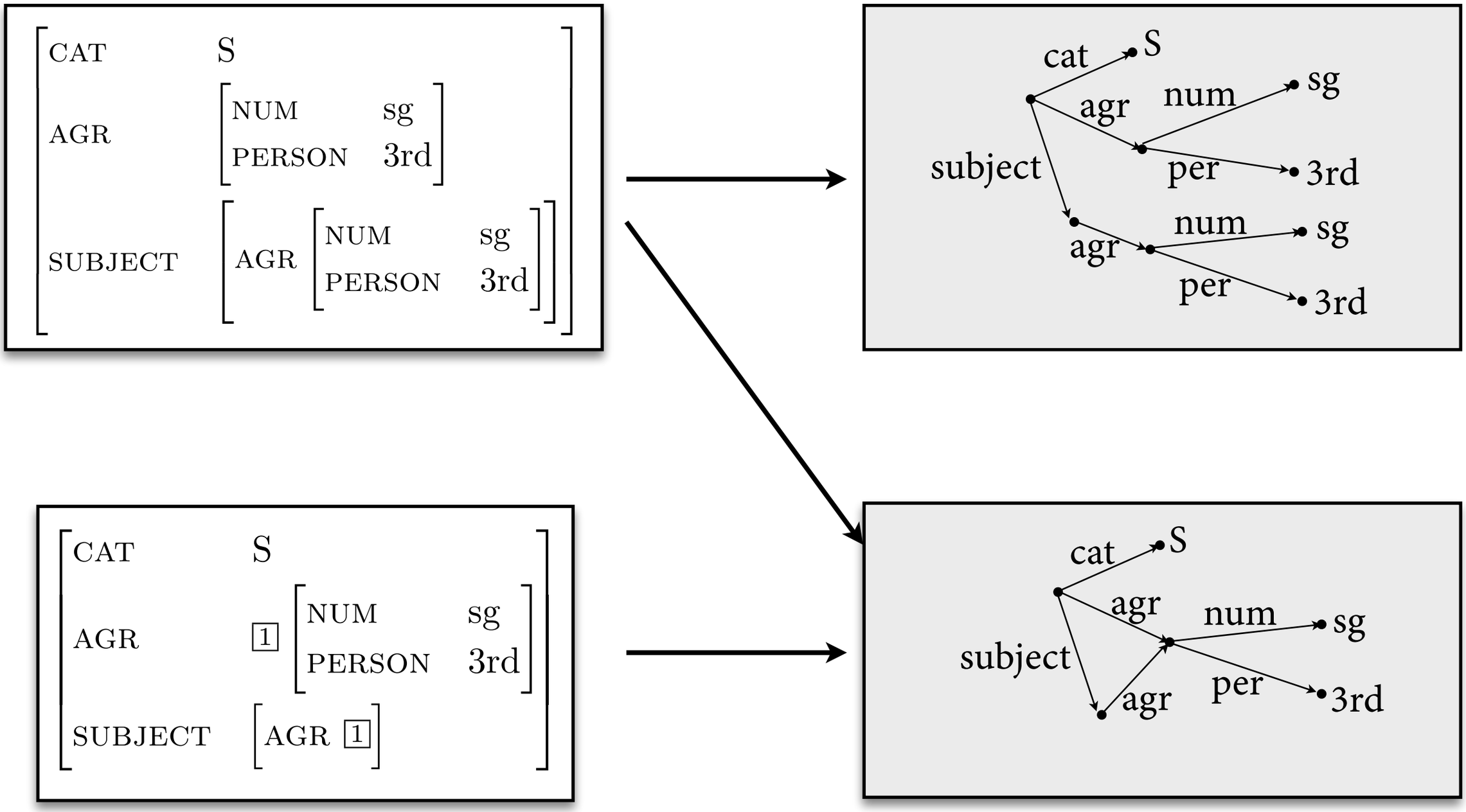
Beschreibung

Modell

Verschiedene Pfade im Graphen können zum gleichen Knoten führen.

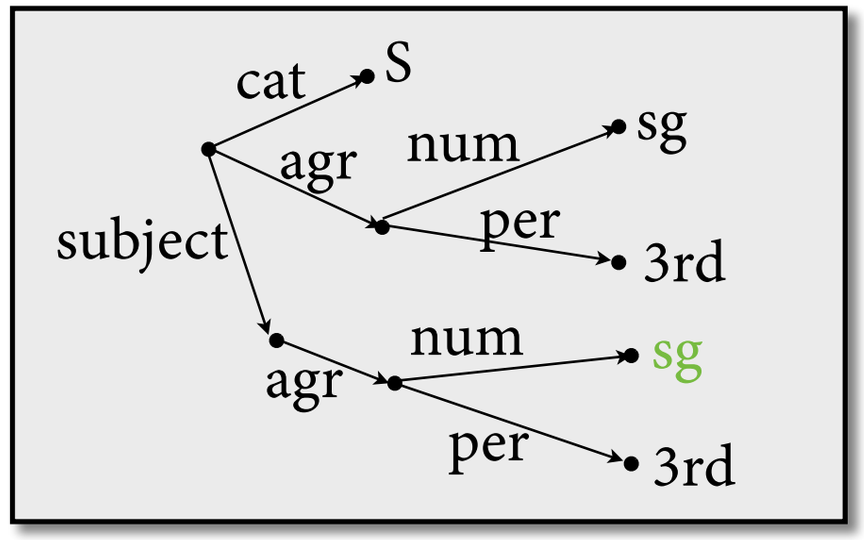
Identität wird in der AVM durch *Koindizierung* erzwungen.

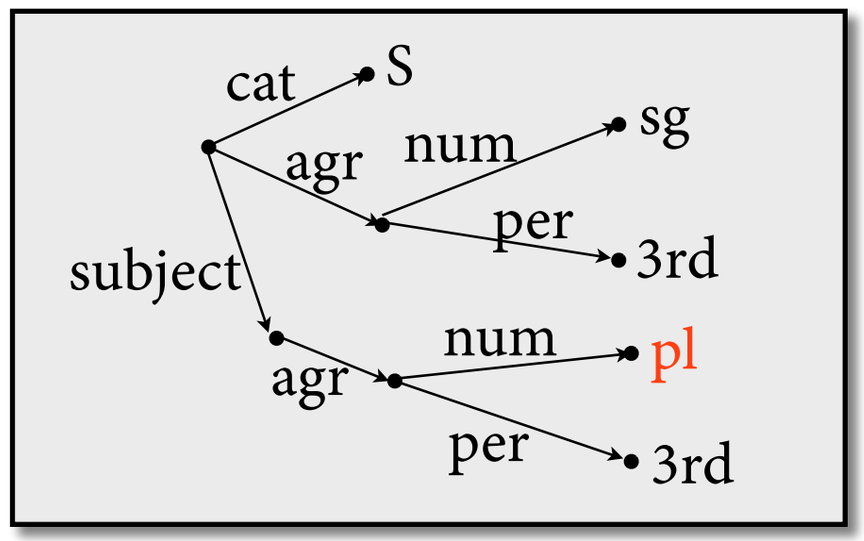
Knotengleichheit vs Wertgleichheit



Unterspezifizierte AVMs

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERSON} & \text{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERSON} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$





Beschreibung

Modelle

Unifikation

- In Unifikationsgrammatik trägt jede Grammatikregel nur ein bisschen Information über die FS bei.

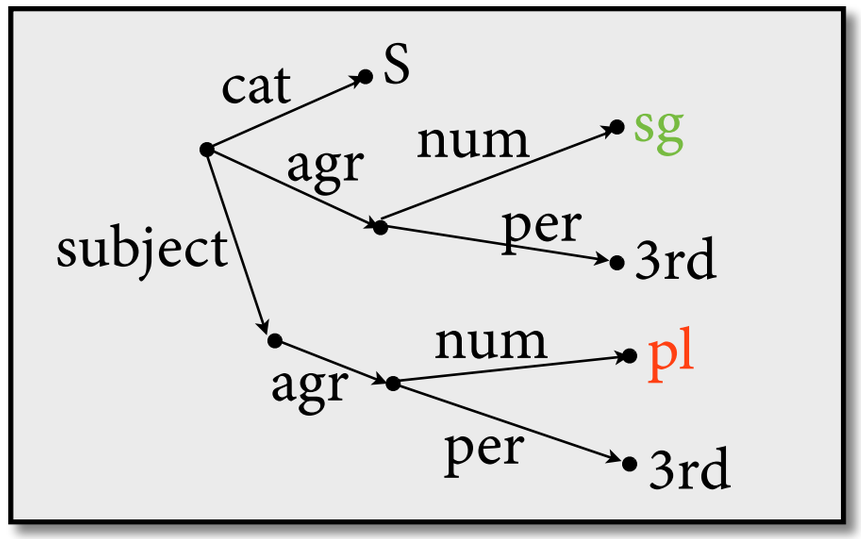
- *Unifikation:* Informationen aus zwei FSen zusammenführen.
  - kann fehlschlagen!

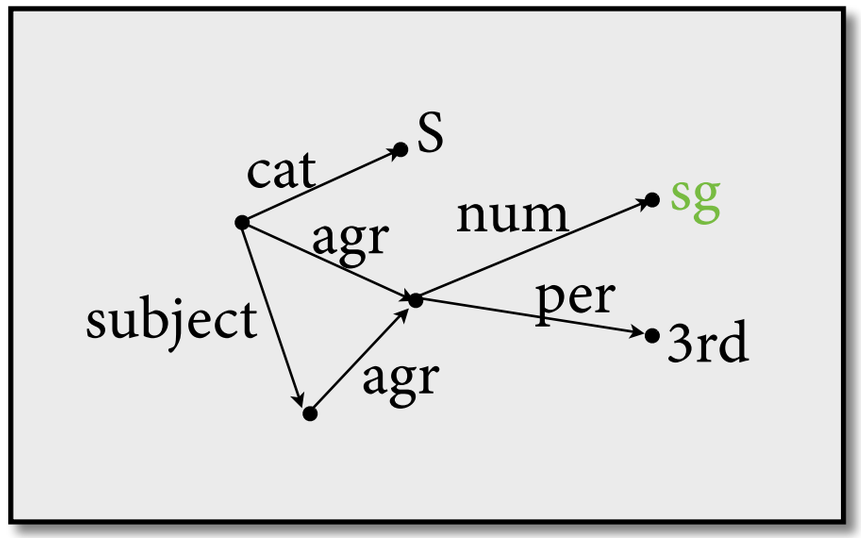
- Zentrale Operation im Parsingalgorithmus.

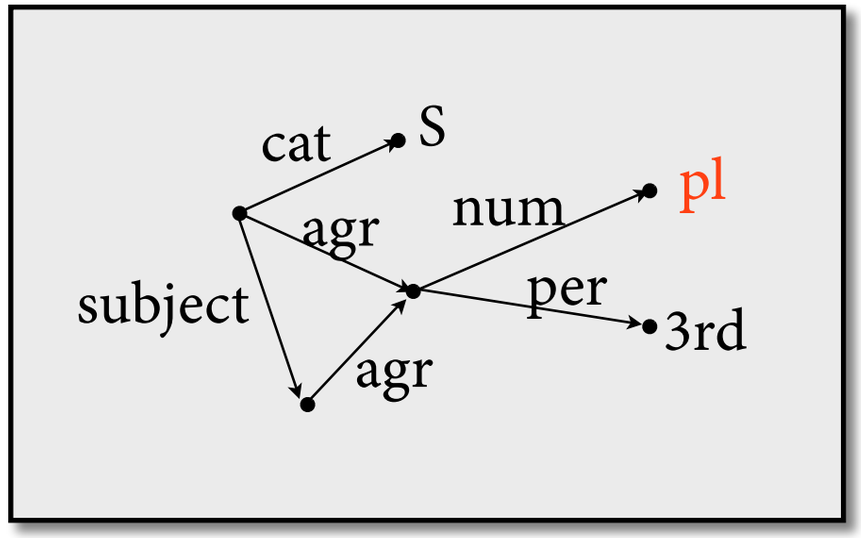
Unifikation

f

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERSON} & \text{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERSON} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$



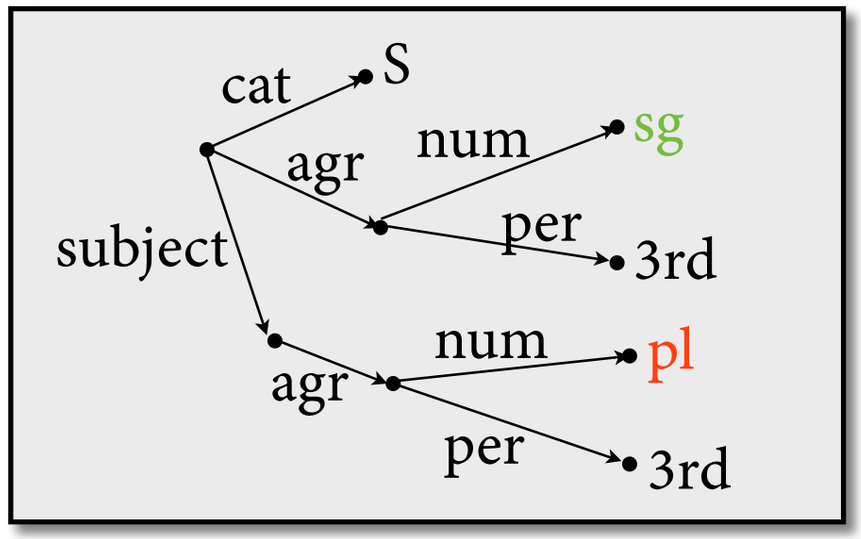




Unifikation

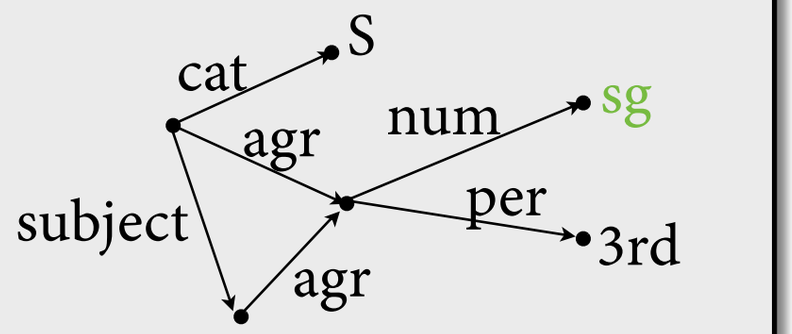
f

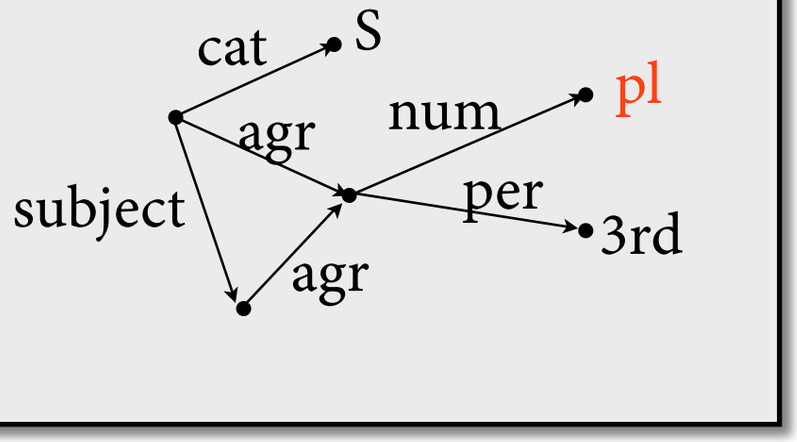
$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERSON} & \text{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERSON} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$



g

$$\begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

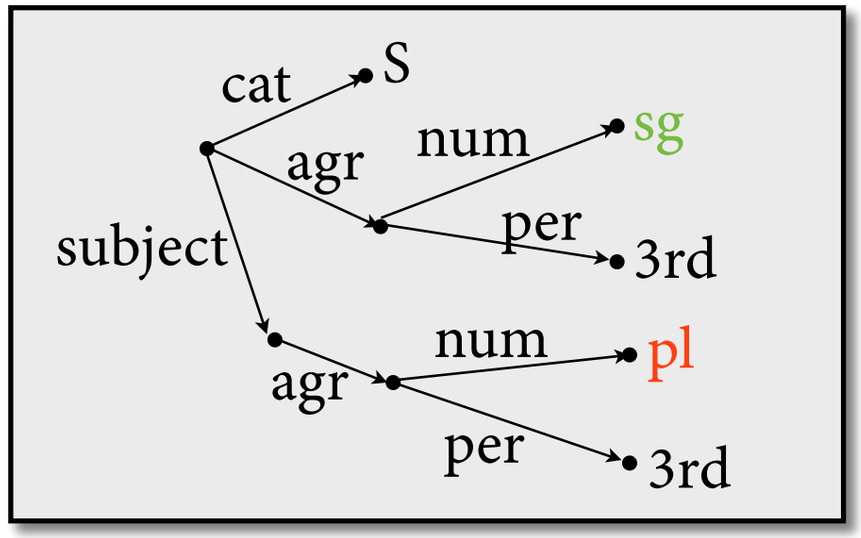




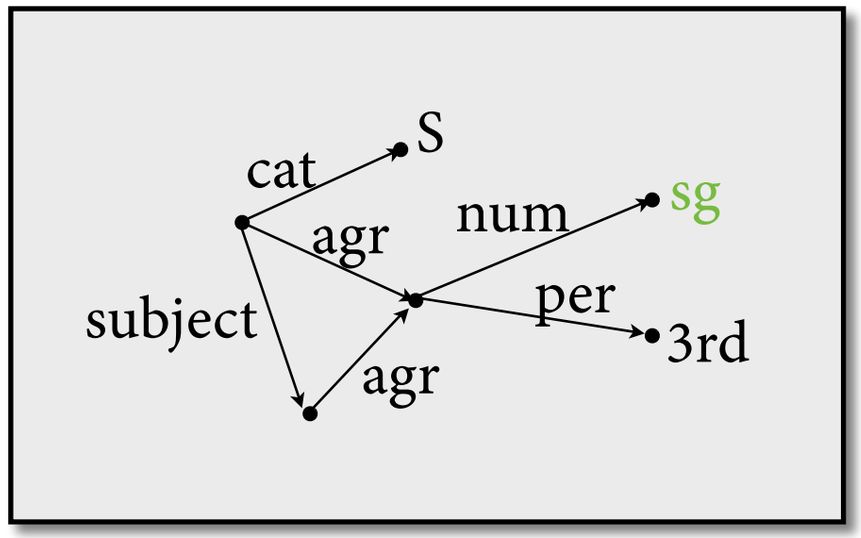
Unifikation

f

$$\begin{bmatrix} \text{CAT} & \text{S} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERSON} & \text{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERSON} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

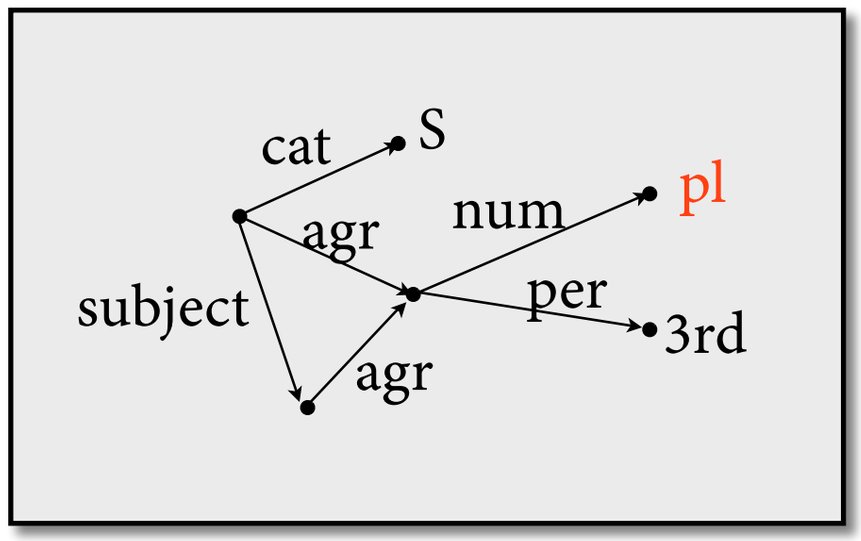


$$f \sqcup g = \begin{bmatrix} \text{CAT} & \text{S} \\ \text{AGR} & \boxed{1} \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERSON} & \text{3rd} \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \boxed{1} \begin{bmatrix} \text{PERSON} & \text{3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$



g

$$\begin{bmatrix} \text{AGR} & \boxed{1} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$



Unifikation

- Unifikation kann fehlschlagen:

  [num pl] ⊔ [num sg] → fail

- Unifikation sensibel für Reentrancy:

$$\begin{bmatrix} \text{agr} & [\text{pers 3rd}] \\ \text{subject} & [\text{agr [pers 3rd]}] \end{bmatrix} \sqcup \begin{bmatrix} \text{agr} & [\text{num sg}] \\ \text{subject} & [\text{agr [num pl]}] \end{bmatrix} = \begin{bmatrix} \text{agr} & \begin{bmatrix} \text{pers} & \text{3rd} \\ \text{num} & \text{sg} \end{bmatrix} \\ \text{subject} & \begin{bmatrix} \text{agr} & \begin{bmatrix} \text{pers} & \text{3rd} \\ \text{num} & \text{pl} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{agr} & \boxed{1}\,[\text{pers 3rd}] \\ \text{subject} & [\text{agr } \boxed{1}[\text{pers 3rd}]] \end{bmatrix} \sqcup \begin{bmatrix} \text{agr} & [\text{num sg}] \\ \text{subject} & [\text{agr [num pl]}] \end{bmatrix} \rightarrow \text{fail}$$

Featuregrammatiken

- Eine *Featuregrammatik* (FCFG) ist eine kfG, in der jedes Auftreten eines Nichtterminals in einer Produktionsregel mit einer Featurestruktur versehen sein kann.

S → NP[num 1] VP[num 1]
VP[num 1] → IV[num 1]
IV[num: sg] → sleeps

Ableitungen in FCFGs

- Regel A[F0] → B[F1] C[F2] kann man als größere Featurestruktur sehen:

$$F = \begin{bmatrix} A & F0 \\ B & F1 \\ C & F2 \end{bmatrix}$$

- Denke Ableitungen bottom-up:
  - um B[G1] und C[G2] zu kombinieren,
  - berechne G = F ⊔ [B G1] ⊔ [C G2] (darf nicht fehlschlagen)
  - und leite daraus A[G.A] ab.

- Sprache, Wort-, Parsingproblem wie für kfGs.

Ein Beispiel

S → NP[num 1] VP[num 1]
VP[num 1] → IV[num 1]
IV[num: sg] → sleeps
NP[num: sg] → John

S ⇒ NP[num: sg] VP[num: sg]
⇒ NP[num: sg] IV[num: sg]
⇒ John IV[num: sg] ⇒ John sleeps

$$\begin{bmatrix} S & \\ NP & [num\ \boxed{1}] \\ VP & [num\ \boxed{1}] \end{bmatrix} \sqcup [NP\ [num\ sg]] \sqcup [VP\ [num\ sg]] = \begin{bmatrix} S & \\ NP & [num\ \boxed{1}\ sg] \\ VP & [num\ \boxed{1}\ sg] \end{bmatrix}$$

$$\begin{bmatrix} VP & [num\ \boxed{1}] \\ IV & [num\ \boxed{1}] \end{bmatrix} \sqcup [IV\ [num\ sg]] = \begin{bmatrix} VP & [num\ \boxed{1}\ sg] \\ IV & [num\ \boxed{1}\ sg] \end{bmatrix}$$

Ein Beispiel

S → NP[num 1] VP[num 1]
VP[num 1] → IV[num 1]
IV[num: sg] → sleeps
NP[num: sg] → John

S ⇏ NP[num: sg] VP[num: pl]

(also S ⇏* John sleep)

$$\begin{bmatrix} S & \\ NP & [num\ \boxed{1}] \\ VP & [num\ \boxed{1}] \end{bmatrix} \sqcup [NP\ [num\ sg]] \sqcup [VP\ [num\ pl]] \rightarrow fail$$

Listen als Featurestrukturen

- Man kann in FSen ganze Listen von Werten codieren:
  - atomarer Wert “Nil” = leere Liste
  - FS für Liste a|R:

$$\begin{bmatrix} \text{FIRST} & \text{a} \\ \text{REST} & \text{R} \end{bmatrix}$$

- Ich kürze Listen mit Schreibweise der Form (a, b, c) ab.

Subkategorisierung

- In Featuregrammatiken können wir jetzt Subkategorisierung von Verben aufräumen:

V[subcat: [1]] → V[subcat: NP | [1]] NP
S → NP V[subcat: (NP)]

V[subcat: (NP)] → sleeps
V[subcat: (NP, NP)] → eats
etc.

Subcat mit Kongruenz

V[subcat: 1 , agr: 2] → V[subcat: NP |1 , agr: 2] NP
S → NP[agr: 1] V[subcat: (NP), agr: 1]

V[subcat: (NP), agr: [num: sg]] → sleeps
V[subcat: (NP, NP), agr: [num: pl]] → eat
etc.

S ⇒ NP[agr:sg] V[subcat: (NP), agr:sg] ⇒* John sleeps

S ⇒ NP[agr:sg] V[subcat: (NP), agr:sg]
⇒ NP[agr:sg] V[subcat: (NP, NP), agr:sg] NP ⇒* John likes donuts

CKY-Parser für FCFGs

Items um FS erweitern

[B, G1, i, j] [C, G2, j, k]

A[F0] → B[F1] C[F2] ist Regel

$$G = \begin{bmatrix} A & F0 \\ B & F1 \\ C & F2 \end{bmatrix} \sqcup [B\ G1] \sqcup [C\ G2]$$

G0 = FS in G unter A

[A, G0, i, k]

teuerste Operation:
Unifikation berechnen

Expressivität

- Im allgemeinen sind FCFGs expressiver als kfGs:

S → A[X] B[X] C[X]

A[s: X] → a A[X] B[s: X] → b B[X] C[s: X] → c C[X]

A[s: end] → a B[s: end] → b C[s: end] → c

- Akzeptiert nicht-kf Sprache $a^n b^n c^n$.

- Im allgemeinen gibt es keinen Parsingalgorithmus für FCFGs, der garantiert terminiert.

Eingeschränkte FCFGs

- Problem sind Regeln, die mittels Reentrancy immer tiefere Strukturen bauen: A[s: X] → a A[X]

- Lösungsidee: Wertebereiche von Features auf endliche Mengen einschränken (also nicht Listen!).

- Dann kann man Kombinationen von Nichtterminal + FS in endlich viele Nichtterminale übersetzen.

- Folge: Expressivität & Komplexität kontextfrei.

Zusammenfassung

- Featuregrammatiken: Erlaubt kompakte Repräsentation von grammatischen Informationen.
- Unifikation: Fasst Information aus verschiedenen Featurestrukturen zusammen.
- KfG-Parsingalgorithmen können zu Parsingalgorithmen für FCFGs erweitert werden.