Merkmalstrukturen in PROLOG

Übersicht

- Merkmalstruktur- vs. Termunifikation
  - Normale Terme
  - Normale Listen: nicht destruktive Unifikation
- Merkmalstrukturen als offene Listen
  - destruktive Unifikation
- Merkmalstrukturen als Termschemata
  - Übersetzungsrelationen
- GULP von M. Covington
  - Äussere und Innere Repräsentation
  - Verwendung
- Literatur

Motivation

Merkmalstrukturbasierte Grammatikformalismen

- Unifikationsgrammatiken zur Beschreibung der Syntax, Lexikons und anderer Ebenen der Sprache
  - Lexikalisch-Funktionale Grammatik (LFG)
  - Generalisierte Phrasenstruktur-Grammatik (GPSG)
  - Unifikationsbasierte Varianten der Kategorialgrammatik
  - Kopf-getriebene Phrasenstruktur-Grammatik (HPSG)
  - PATR I-II für Syntax, DATR für Lexikon

Vgl. Vorlesung "Formale Grammatiken und Syntaxanalyse"

Computerlinguistik

- Entwicklungsumgebungen/Formalismen für Unifikationsgrammatiken
  - ALE, GULP, YAP, PATR, ProFIT und viele andere

Term vs. Merkmalstruktur

```
f(a,b)
```

functor	*f*
arg1	*a*
arg2	*b*

Jeder (Prolog-)Term kann als Merkmalstruktur kodiert werden.

- In HPSG sieht man hin und wieder etwa folgende Listen…

```
[a,b]
```

first	*a*	
rest	*first*	*b*
	rest	*nil*

Leider ist es nicht so einfach, Merkmalstrukturen sinnvoll und effizient als Prolog-Terme zu kodieren…

Term- vs. Merkmalstruktur-Unifikation

Syntax	*Genus*	*feminin*	*Syntax*	*Kasus*	*dativ*
	Kasus	*dativ*		*Numerus*	*plural*

```
(syntax(
    genus(feminin),
    kasus(dativ)))
```

```
(syntax(
    kasus(dativ),
    numerus(plural)))
```

Merkmalstrukturen und komplexe Prolog-Terme haben unterschiedliche Unifikation

- Prolog-Terme haben fixe Anzahl Unterterme und fixe Reihenfolge
- Merkmalstrukturen haben freie Anzahl Unterstrukturen und freie Reihenfolge
  - Keine direkte Strukturähnlichkeit!

Term- vs. Merkmalstruktur-Unifikation


```
[syntax:
    [genus:feminin,
    kasus:dativ]]
```

```
[syntax:
    [kasus:dativ,
     numerus:plural]]
```

Merkmalstrukturen und Prolog-Listen haben unterschiedliche Unifikation

- Prolog-Listen haben freie Anzahl Elemente, aber fixe Reihenfolge
- Merkmalstrukturen haben freie Anzahl Unterstrukturen und freie Reihenfolge
  - ineffiziente Merkmalstrukturunifikation mit Listen ist relativ leicht möglich

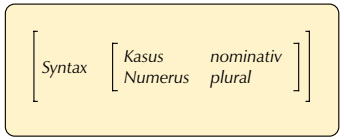
Merkmalstrukturen als offene Listen

Offene Liste

```
[num:sg,pers:3|_]
```

Manchmal erlauben komplexere Datenstrukturen einfachere Algorithmen…

- Merkmalstrukturen werden durch offene Listen repräsentiert
- Unifikation von Merkmalstrukturen wird merklich einfacher…




```
[syntax:[kasus:nominativ,
         numerus:sg|_]|_]
```


Destruktive Unifikation

unify_fs/2 erlaubt destruktive Unifikation von Merkmalstrukturen als offene Listen

- Die 1. Klausel erledigt die Fälle, wo Termunifikation korrekt ist
  - atomare Werte oder variable Werte (bei koreferenten Pfaden)
- Die 2. Klausel macht die eigentliche Merkmalstrukturunifikation
  - doppelte Rekursion und geschicktes Ausnützen von select/3
  - vorhandene bzw. fehlende Merkmal-Wert-Paare werden abgeglichen

```
unify_fs(FS, FS) :- !.
unify_fs([F:V1|Rest1], FS)
:-
   select(FS, F:V2, Rest2),
   unify_fs(V1, V2),
   unify_fs(Rest1, Rest2).
```


Auswählen von Elementen

select(L1, E, L2)

- ist wahr, falls die Liste *L1* gleich *L2* ist, wobei das 1. Auftreten von Element *E* entfernt ist
- beim Aufruf mit offenen Listen wird *E* dank Unifikation in *L1* eingefügt, wenn es nicht vorhanden ist!

```
select([X|Tail], X, Tail):- !.
select([Head|Tail], Elem, [Head|Rest])
:-
   select(Tail, Elem, Rest).
```

```
?- select([num:sg,pers:3|_], pers:3,  Rest).
Rest = [num:sg|_A] ?
yes
```

```
?- FS = [num:sg|_], select(FS, pers:3, Rest).
FS = [num:sg,pers:3|_A],
Rest = [num:sg|_A] ?
yes
```

Korrektes Scheitern der Unifikation

Nicht widerspruchsfrei unifizierbare Merkmalstrukturen

- korrektes Scheitern dank der 1. Klausel von unify_fs/2

```
?- unify_fs([a:b|_],[a:c|_]).
   1  1  Call: unify_fs([a:b|_84],[a:c|_112]) ?
   2  2  Call: select([a:c|_112],a:_1063,_1067) ?
   2  2  Exit: select([a:c|_112],a:c,_112) ?
   3  2  Call: unify_fs(b,c) ?
   3  2  Fail: unify_fs(b,c) ?
   1  1  Fail: unify_fs([a:b|_84],[a:c|_112]) ?
no
```

- Semantisch nicht wohlgeformte Merkmalstrukturen führen selbstverständlich zu problematischem Verhalten (von Reihenfolge der Merkmale abhängig)

```
?- unify_fs([a:b,a:c|_],[a:b|_]).
   1  1  Call: unify_fs([a:b,a:c|_323],[a:b|_351]) ?
   1  1  Exit: unify_fs([a:b,a:c|_323],[a:b,a:c|_323]) ?
yes
```


Merkmalstrukturen als Termschemata


```
translate(syntax, FS, fs(FS)).
translate(genus, Gen, fs(fs(Gen,_,_))).
translate(numerus, Num, fs(fs(_,Num,_))).
translate(kasus, Kas, fs(fs(_,_,Kas))).
```

```
fs(fs(feminin,_,dativ))          fs(fs(_,plural,dativ))
```

Term-Unifikation ist möglich, wenn Merkmalstrukturen in ein fixes Schema übersetzt werden!

- Verschachtelte fs/*n*-Funktoren: Wert von "Syntax" ist 1. Argument, Wert von "Genus" inneres 1. Argument usw.
- Einfacher, wenn für jede Sorte von Merkmalstrukturen festgelegt ist, welche Merkmale sie maximal haben kann. (getypte Merkmalstrukturen)

Merkmalstrukturen als Termschemata

Merkmalstrukturen als Prolog-Terme mit fest zugewiesenen Argumentstellen

Vorteile

- normale Prolog-Term-Unifikation verwendbar
- sehr effizient

Nachteile

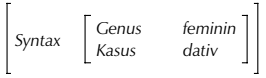
- mühsam zum Eingeben
- Terme völlig unleserlich fürs menschliche Auge

Ausweg

- Trennung von innerer und äusserer Repräsentation

Automatische Übersetzung

Direkte Term-Unifikation von Merkmalstrukturen dank automatischer Übersetzung:



- beim Einlesen der Grammatik
  - leserliche Ausdrücke → Prolog-Terme
- beim Ausgeben der Ergebnisse
  - Prolog-Terme → leserliche Ausdrücke

```
term(syntax:(genus:feminin..kasus:dativ))
```

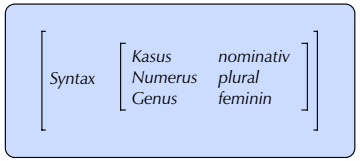
```
term(g__([_,_,g_(g__([g_(feminin),g_(dativ)|_]))|_])).
```

Ein Beispiel für eine solche Übersetzung ist *Graph Unification Logic Programming* (GULP) von M. Covington

Konstruktoren von GULP

Konkrete Syntax von GULP

- Als Trenner von Merkmal und Wert wird der Infix-Operator :/2 (xfy 600) verwendet.
- Merkmal-Wert-Paare werden mit dem Infix-Operator ../2 (xfy 602) zusammengefügt.



Kästchen-Notation

```
(syntax:
   (kasus:nominativ
   ..numerus:plural
   ..genus:feminin))
```

GULP-Notation

GULP-Integration

Zuerst muss GULP konsultiert werden

```
?- consult('gulp.pl').
```

- Danach können beliebige Prolog-Dateien mit Merkmalstrukturen verarbeitet werden.

```
s --> np(case::nom..num::N), vp(num::N).

np(num::N) --> d(num::N), n(num::N).
np(num::N..case::C) --> pronoun(num::N..case::C).

vp(num::N) --> v(subcat::1..num::N).
vp(num::N) --> v(subcat::2..num::N), np(case::acc).

v(num::sg..subcat::1) --> [barks].
v(num::pl..subcat::1) --> [bark].

d(_)       -->  [the].
d(num::pl)  -->  [two].

pronoun(num::sg..case::acc) --> [him].
...
```

Die Verwendung von ::/2 mit der automatischen Übersetzung der Merkmalstrukturen führt in SICStus Prolog manchmal zu Konflikten mit dem Modulsystem. Insbesondere mit der emacs-Schnittstelle. Deshalb verwenden wir hier eine modifizierte Version, die mit dem Operator ::/2 arbeitet.

Literatur zu GULP

GULP 3.1

- Michael A. Covington (1994): GULP 3.1: An Extension of Prolog for Unification-Based Grammar. Research Report AI-1994-06. Artificial Intelligence Center. The University of Georgia

  Dokumentation zur Implementation und Anwendung von GULP 3.1

Mini-Gulp

- Michael A. Covington(1994): Natural Language Processing for Prolog Programmers. Prentice-Hall. (130-140)

  Gut verständliche Erklärung zu einer vereinfachten Version von GULP.

URL: `http://www.ai.uga.edu/~mc/#PROLOG`

Literatur

Unifikationsbasierte Grammatiken

- Shieber, Stuart M.: An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes, vol. 4. Center for the Study of Language and Information: Palo Alto, 1986.
  - gute Einführung in praktische linguistische Anwendungen von Merkmalstrukturen
  - bespricht den Einsatz von Merkmalstukturen in den damals gebräuchlichen linguistischen Theorien
  - empfehlenswert, auch wenn das Buch etwas an der Oberfläche bleibt

Formalismen

- Eine Suchanfrage zu den Akronymen im Internet…