Übungen 10: Subsumption

Programmiertechniken in der Computerlinguistik II · Sommersemester 2005

Programmtexte finden sich auf Homepage: http://www.cl.unizh.ch/siclemat/lehre/ss05/pcl2/

1. Subsumption

In der folgenden Grammatik werden Subkategorisierungen (Anzahl und Art von vom Verb abhängigen Satzgliedern) über einen Index im Lexikon kodiert, der die Auswahl der zulässigen Grammatikregeln steuert.

```
rule(s, [np,vp]).                    word(v(0), isst).
rule(np, [pron]).                    word(v(1), isst).
rule(np, [det,n]).                   word(v(2), gibt).
rule(vp,[verbal(0)]).                word(pron, er).
rule(vp, [verbal(X), objekte(X)]).   word(n, kuchen).
rule(verbal(X), [v(X)]).             word(det, den).
rule(objekte(1), [np]).
rule(objekte(2), [np,np]).
```

a) Lade "verbose_earleyp.txt" und „gram_subsume.txt" mit obiger Grammatik von der Homepage herunter. Parse den Satz "Er isst den Kuchen". Was passiert? Wieso? Baue den Subsumptions-Check ein und parse den Satz nochmals. (Modifiziere store/1 so, dass weiterhin die Kante auf den Bildschirm herausgeschrieben wird nach dem Assertieren!)

b) **Freiwillig**: Ersetze obige Grammatik durch

```
rule(s,[a(b)]).
rule(a(b),[b]).                      word(b, bla).
rule(a(X),[a(f(X))]).
```

Versuche nun folgende Anfragen:

```
?- parse(s,[bla]).

?- parse(a(B),[bla]).
```

Was passiert?

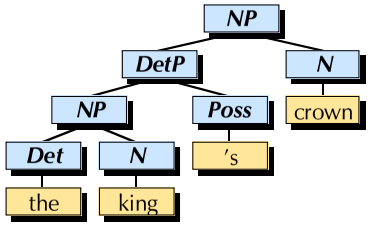
2. Penn-Treebank-Format erzeugen

Für syntaktisch annotierte Korpora – im Englischen oft als *treebank* bezeichnet – gibt es ein textbasiertes Standardformat, das sowohl für menschliche wie für informatische Bedürfnisse brauchbar ist. Populär wurde diese Format durch die Erstellung der Penn-Treebank (≈Annotation des *Wallstreet Journals* an der Universität von Pennsylvania) und ist letztlich eine ASCII-Variante der sogenannten „Indizierten Klammerung" (vgl. Stichwort „Strukturbaum" in H. Bussmann: Lexikon der Sprachwissenschaft).

Hinweis: Diese Aufgabe ist als Übung zu doppelter Rekursion gedacht, wie sie im Umgang mit Bäumen oft vorkommt.

Beispiel:

- Syntaxbaum



- Indizierte Klammerung:

$$[\,[\,[\,[\,\textit{the}\,]_{\text{Det}}\,[\,\textit{king}\,]_{\text{N}}\,]_{\text{NP}}\,[\,\textit{`s}\,]_{\text{Poss}}\,]_{\text{DetP}}\,[\,\textit{crown}\,]_{\text{N}}\,]_{\text{NP}}$$

- Penn-Treebank-Format:

```
(NP
   (DetP
      (NP
         (Det the)
         (N king))
      (Poss `s))
   (N crown))
```

- Prolog-Term-Format:

```
'NP'('DetP'('NP'('Det'(the),'N'(king)),'Poss'('`s')),'N'(crown))
```

Definiere das Prädikat print_penn_format/1, das als Argument einen Syntaxbaum im Prolog-Term-Format nimmt und auf dem Bildschirm das entsprechende Penn-Treebank-Format ausgibt.

```
?-
print_penn_format('NP'('DetP'('NP'('Det'(the),'N'(king)),'Poss'('`s')),'N'(crown))).
(NP
   (DetP
      (NP
         (Det the)
         (N king))
      (Poss `s))
   (N crown))

yes
```

Hinweise: Diese Übung ist für das Training deiner rekursiven Definitionsfähigkeiten gedacht. Gehe an Hand von einfachen Beispiel aus, am Anfang vielleicht nur ein Term wie *det(the)*, versuche erst dann den rekursiven Fall zu verstehen, zu formulieren und zu programmieren. Erinnere dich an das Prädikat =../2, das aus Termen beliebiger Arität eine Liste der Form [*Hauptfunktor*|*Argumente*] berechnet.