Nichtdeterministische Head-Corner-Analyse

In sog. „$\bar{X}$-Regeln“ (`XP --> ... X ...`) soll die Hauptkonstituente `X` die Kategorien (ggf.: und Anordnungen) der Nebenkonstituenten ... bestimmen, z.B. das Verb seine Objekte.

In jeder Grammatikregel sei eine *Hauptkonstituente* bestimmt.

⟨*Regelformat*⟩≡

```
(SynCat --> Cat1,...,Cati,...,Catn).
```

Wir definieren die Hauptkonstituente `B` durch ein Prädikat:

⟨*Hauptrelation*⟩≡

```
head((SynCat --> Alpha,C,Beta),Alpha,B,Beta).
```

Beim Head-Corner-Parser soll eine erkannte Phrase nicht als linke Ecke, sondern als Kopfkonstituente verwendet werden.

Vorteil

(i) Linguistisch attraktiv: Kopf-Komplement-Schema

(ii) Während die Kategorie der linken Ecke von der Anordnung abhängt, ist die Position des Kopfs (meist) fest, die Kopfrelation also unabhängig von Argumentstellungen berechenbar. (Daher auch für Metaregeln geeignet.)

Nachteil

(i) Kompliziert zu implementieren: Parserichtung rechts-links im Linkskontext, links-rechts im Rechtskontext des Kopfs

(ii) Praktische Effizienz?

Ein Head-Corner Erkenner in Prolog

Man definiert induktiv die Relation

$$headcorner(B, A, i, i', j', j) \ :\Longleftrightarrow \\ A \Rightarrow^* a_{i+1} \cdots a_{i'} B a_{j'+1} \cdots a_j.$$

⟨*headcorner.pl*⟩≡

```
parse(A,I,J) :-
        (head(B,A) ; B=A),
        leaf(B,I,I2,J2,J),
        headcorner(B,A,I,I2,J2,J).

headcorner(B,A,I,I2,J2,J) :-             %      A
    ( I=I2, J2=J, B=A                    %    /   \
    ; (C --> Gamma),                     %   /     \
      head((C --> Gamma),Alpha,B,Beta),  %  i-- C --j
      reverse(Alpha,AlphaRev),           %     / \
      parse_list_rev(AlphaRev,I,I3,I2),  %    /   \
      parse_list(Beta,J2,J3),            %   /     \
      headcorner(C,A,I,I3,J3,J)         % i3-i2 B j2-j3
    ).
```

Mit `leaf/5` sucht man im Teilwort $a_{i+1} \cdots a_j$ nach möglichen lexikalischen Köpfen $a_{i'+1} = a_{j'}$:

⟨*headcorner.pl*⟩+≡

```
leaf(A,I,I2,J2,J) :-
     ( (A --> []),
         I=I2, I2=J2, J2=J
     ; append(_,I2,I), I2 = [W|J2], append(_,J,J2),
        (word(W,A) ; A = [W])  % , lexhead(A)
     ).
```

Die Grammatik bestimmt die Hauptkonstituente(n) einer Regel, hier grob durch `head/2` allein aus der Phrasenkategorie:

⟨*headcorner.pl*⟩+≡
```
head((C --> Gamma),Alpha,B,Beta) :-
      tupleToList(Gamma,GammaList),
      head(B,C),
      append(Alpha,[B|Beta],GammaList).
```

In `parse_list_rev(AlphaRev,I+,K-,J+)` sucht man `K` zwischen `I` und `J` mit `parse(Alpha,K,J)`:

⟨*headcorner.pl*⟩+≡
```
parse_list_rev([],_,J,J).
parse_list_rev([A|As],I,K,J) :-
        parse_rev(A,I,L,J),
        parse_list_rev(As,I,K,L).

parse_rev(A,I,K,J) :-
        append(_,K,I), append(_,J,K),
        parse(A,K,J).

parse_list([B|Bs],I,J) :-
        parse(B,I,K), parse_list(Bs,K,J).
parse_list([],I,J) :- I = J.

⟨tupleToList⟩
```

Aufg. 3 Erweitere diesen Erkenner zu einem Parser, d.h. ergänze die Ausgabe eines Syntaxbaums.

Beispiel zur Head-Corner-Erkennung

⟨*cfg.hc.grm*⟩≡

```
start_symbol(s).
start_symbol(nom).

% Grammatikregeln:
:- op(1100,xfx,'-->').
s --> nom, v, akk, rs.
s --> nom, v, akk.
rs --> rnom, akk, v.
rs --> rakk, nom, v.

nom --> det(nom), n(nom)).
akk --> det(akk), n(akk)).

% Hauptrelation:
head(v,s).               head(v,rs).
head(n(nom),nom).        head(n(akk),akk).

% Lexikon:
word(er,nom).            word(ihn,akk).
word(der,det(nom)).      word(den,det(akk)).
word('Mann',n(nom)).     word('Mann',n(akk)).
word(der,rnom).          word(den,rakk).
word(kennt,v).           word(grüßt,v).
```

Bem. Zur Beschleunigung sollte man die transitive Hülle von `{(B,C)| head((C --> _),_,B,_)}` vorberechnen und in `parse/3` ausnutzen.

Beispiel zur Head-Corner-Erkennung (Forts.)

⟨*Tracer einschalten*⟩≡

```
100 ?- trace([headcorner,parse],[+call,+exit]).
```

⟨*Eingabesatz*⟩≡

```
101 ?- parse(s,[der,'Mann',kennt,
                den,'Mann',der,ihn,grüßt],[]).
```

⟨*Parse-Aufruf*⟩≡

```
C (7) parse(s, [der,'Mann',...], [])
C (8) headcorner(v, s,
         [der,'Mann',...], [kennt,..],[..], [])
E (9) s-->nom, v, akk, rs
```

⟨*Linkskontext* `nom` *von* `v` *aus (9)* ⟩≡

```
C (11) parse(nom, [der,'Mann',kennt,...],
                              [kennt,...])
C (12) headcorner(n(nom),nom,
          [der,'Mann',kennt,...],
          ['Mann',..],[..], [kennt,...])
E (13) ... E (13) nom-->det(nom), n(nom)
C ... E (15) parse(det(nom), [der,'Mann',...],
                                  ['Mann',...])
C E (13) headcorner(nom, nom,
            [der,'Mann',kennt,...],
            [der,'Mann',..],[..], [kennt, ...])
E (12) headcorner(n(nom),nom,
          [der,'Mann',kennt,...],
          ['Mann',..],[..], [kennt,...])
E (11) parse(nom, [der,'Mann',kennt,...],
                              [kennt,...])
```

Beispiel zur Head-Corner-Erkennung (Forts.)

⟨*Teil* `akk` *des Rechtskontexts* `akk,rs` *von* `v` *aus (9)* ⟩≡

```
C (10) parse(akk, [den,'Mann',der,ihn,grüßt],_G933)
... (analog)
E (10) parse(akk, [den,'Mann',der,ihn,grüßt],
                                      [der,ihn,grüßt])
```

⟨*Teil* `rs` *des Rechtskontexts* `akk,rs` *von* `v` *aus (9)* ⟩≡

```
C (11) parse(rs, [der, ihn, grüßt], _G972)
C (12) headcorner(v, rs,
          [der,ihn,grüßt], [grüßt],[], [])
E (13) ...
E (13) rs-->rnom, akk, v
```

⟨*Teil* `akk` *des Linkskontexts* `rnom,akk` *von* `v` *aus (13)* ⟩≡

```
C (15) parse(akk, [der,ihn,grüßt], [grüßt])
F ...
C (15) parse(akk, [ihn,grüßt], [grüßt])
E (15)
```

⟨*Teil* `rnom` *des Linkskontexts* `rnom,akk` *von* `v` *aus (13)* ⟩≡

```
C (16) parse(rnom, [der,ihn,grüßt], [ihn,grüßt])
C (17) headcorner(rnom, rnom,
         [der,ihn,grüßt],[der,..],[..],[ihn,grüßt])
E (17)
E (16) parse(rnom, [der,ihn,grüßt], [ihn,grüßt])
```

⟨*Teil* `rs` *des Rechtskontexts* `akk,rs` *von* `v` *aus (9)* ⟩+≡

```
C (13) headcorner(rs, rs,
       [der,ihn,grüßt], [der,ihn,grüßt],[], [])
E (13)
```

Beispiel zur Head-Corner-Erkennung (Ende)

⟨*Rückgabe der Ergebnisse*⟩≡

```
   E (12) headcorner(v, rs,
                      [der,ihn,grüßt], [grüßt],[], [])
   E (11) parse(rs, [der,ihn,grüßt], [])

   C (9) E (9) headcorner(s, s,
           [der,'Mann',...], [der,'Mann',...],[], [])
   E (8) headcorner(v, s,
           [der,'Mann',kennt,...], [kennt,...],[...], [])
   E (7) parse(s,
          [der,'Mann',kennt,den,'Mann',der,ihn,grüßt],
          [])
  Yes
```

GRAMMATIKÜBERSETZUNG

Um die Analyse zu beschleunigen, soll das Suchen nach einer Regel $(A \to \alpha) \in P$ so geändert werden, daß man nicht immer alle Grammatikregeln durchläuft, sondern

(i) die Kategorien des nächsten Eingabesymbols und

(ii) die für die Resteingabe „erwarteten“ Kategorien

benutzt, um Teilmengen $P' \subseteq P$ zu bestimmen, die alle in der jeweiligen Analysesituation noch nötigen Regeln umfassen.

Hilfsrelationen zwischen Kategorien

Aus den DCG-Regeln der Grammatik berechnen wir

(i) die *löschbaren Kategorien* $\{A \mid A \Rightarrow^+ \epsilon\}$,

(ii) die *Kettenrelation* $\{\,(A, B) \mid\ A \Rightarrow^+ B\,\}$,

(iii) die *Linke-Ecken-Relation*

$$<_{lc} \quad := \quad \{(B, A) \mid (A \to \alpha B \gamma) \in P,\ \alpha \Rightarrow^* \epsilon\}$$

und deren reflexive, transitive Hülle

$$<^*_{lc} \quad := \quad \{(B, A) \mid \exists \gamma \in Cat^*\ A \Rightarrow^* B\gamma\}$$

(iv) ggf. die *Hauptkategorie-Relation*

$$<_{hc} \quad := \quad \{(B, A) \mid (A \to \alpha \underline{B} \gamma) \in P\}$$

und ihre reflexive, transitive Hülle $<^*_{hc}$.

Löschbare atomare Kategorien

`del/1` $= \{A \mid A \Rightarrow^+ \epsilon\}$

Voraussetzung: alle Bedingungen `Cond` von Löschregeln `(A --> B) :- Cond.` oder `(A --> B, {Cond}).` sind zur Vorberechnungszeit auswertbar.

⟨*Löschbare Kategorien*⟩≡

```
:- dynamic del/1, aNewDeletable/0.

compute_deletable_categories :-
        retractall(del(_)),
        retractall(aNewDeletable),
        compute_del_basecase,
        compute_del_iteration.
```

Basis: Kategorien mit einer (bedingten) Löschregel:

⟨*Löschbare Kategorien*⟩+≡

```
compute_del_basecase :-
    (clause(A --> [],Cond)
    ; clause(A --> ([],{Cond}),true)
    ),
    add_del(A,Cond),
    fail.
compute_del_basecase.

add_del(A,Cond) :-
    clause(del(A),Cond),!.     % schon vorhanden!
add_del(A,Cond) :-
    assertz((del(A) :- Cond)), % kompakt als Schema
    (aNewDeletable -> true ; assert(aNewDeletable)).
```

Abschluß: Kategorien `A` mit einer Regel `(A --> Alpha)`, deren rechte Seite löschbar ist:

⟨*Löschbare Kategorien*⟩+≡

```
compute_del_iteration :-
    repeat,
    comp_del_iteration,
    not(aNewDeletable),!.
compute_del_iteration.

comp_del_iteration :-
    retract(aNewDeletable),
    clause(A --> Body,Cond),
    deletable(Body),
    simplify(Cond,NewCond),
    add_del(A,NewCond),
    fail.
comp_del_iteration.
```

⟨*Löschbare komplexe Kategorien*⟩

Löschbare komplexe DCG-Kategorien

Für Kategorienvariable (atomarer Kat., nur in Metaregeln):

⟨*Löschbare komplexe Kategorien*⟩≡
```
deletable(A) :-
    var(A), !, fail.
```

⟨*Löschbare komplexe Kategorien*⟩+≡
```
deletable((A,B)) :-
    !,deletable(A),deletable(B).
deletable((A;B)) :-
    !,(deletable(A);deletable(B)).
```

Für optionale Konstituenten, auch komplexe, z.B. alternative Objekte `?(dat;akk)` oder Attribute `?(komma,A,komma)`:

⟨*Löschbare komplexe Kategorien*⟩+≡
```
deletable(?(A,B)) :-
        !, deletable(?A) , deletable(?B).
deletable(?(A;B)) :-
        !, (deletable(?A) ; deletable(?B)).
deletable(?C) :- !, del(?C).
```

⟨*Löschbare komplexe Kategorien*⟩+≡
```
deletable({Goal}) :- !, call(Goal). % Testauswertung!
deletable([_W|_Ws]) :- !, fail.
deletable([]) :- !, true.
deletable(A) :- !, del(A).
```

`deletable(A)` belegt (durch `call(Goal)` und `del(A)`) meist alle freien Variable in `A`; freie Variable bleiben nur, wenn Löschregeln `(B --> []) :- C` unbeschränkte freie Variable hatten.