Left-Corner-Parsing

Übersicht

- Parsen von kontextfreien Grammatiken
- Parsing-Strategien
  - Bottom-Up: datengesteuert
  - Top-Down: hypothesengesteuert
- Left-Corner-Relation (Linke Ecken von Grammatikregeln)
- Left-Corner-Parsing-Algorithmus
- Vorteile gegenüber Bottom-Up und Top-Down
  - Problemregeln
- Left-Corner-Parsing mit Links
- Left-Corner-Links berechnen
  - Links als transitive und reflexive Hülle der Left-Corner-Relation

Motivation

Reine Parsing-Strategien

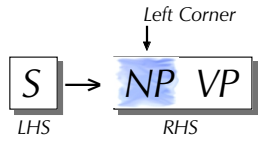
- Top-Down (links-rechts)
  - Probleme mit links-rekursiven Grammatikregeln (direkt oder indirekt)
- Bottom-Up (links-rechts)
  - Probleme mit Tilgungsregeln

NP → NP CONJ NP

X → ε

Gemischte Strategie

- Top-Down und Bottom-Up
- Wechsel bei Left-Corner (linker Ecke) der rechten Regelseite



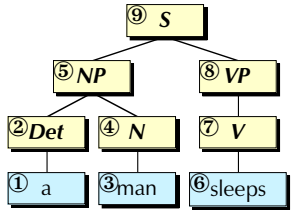
Strategie des Bottom-Up-Parsing

Vgl. Folien "Shift-Reduce-Parser"

Stack	Wortsequenz
_	a man sleeps
a	man sleeps
Det	man sleeps
Det man	sleeps
Det N	sleeps
NP	sleeps
NP sleeps	_
NP V	_
NP VP	_
S	_

S → NP VP
NP → Det N
VP → V

Det → a
N → man
V → sleeps



➥ *datengesteuert*

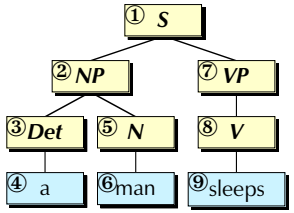
Strategie des Top-Down-Parsing

Vgl. Folien "DCG" im WS

Ziele	Wortsequenz
S	a man sleeps
NP VP	a man sleeps
Det N VP	a man sleeps
a N VP	a man sleeps
N VP	man sleeps
man VP	man sleeps
VP	sleeps
V	sleeps
sleeps	sleeps
_	_

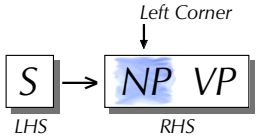
S → NP VP
NP → Det N
VP → V

Det → a
N → man
V → sleeps



➥ *hypothesengesteuert*

Left-Corner-Relation



Das Grammatiksymbol *LC* ist linke Ecke (*left corner*) des Grammatiksymbols *LHS,* gdw. gilt, es gibt eine Grammatikregel

LHS → LC …

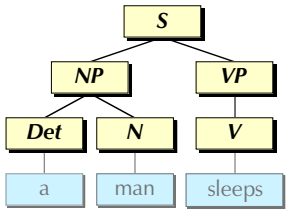
```
left_corner(np, s).
left_corner(det, np).
left_corner(v, vp).
```

S → NP VP
NP → Det N
VP → V
Det → ε

Left-Corner im Baum

Syntaktische Left-Corner-Relation in jeden Teilbaum

- Man zeichne…



S → NP VP
NP → Det N
VP → V

Det → a
N → man
V → sleeps

Left-Corner-Algorithmus

Eine Konstituente *C* left-corner-parsen heisst:

1. **Nimm** ein Wort von der Eingabe und bestimme seine Kategorie *LC*
2. **Vervollständige** *LC* zu *C*.
   a. Abbruchbedingung: **Stoppe**, falls *LC* = *C* ist
      (Achtung: Hier ist die Bezeichnung *LC* ein "falscher Freund". Diese Klausel akzeptiert gerade alle Nicht-Left-Corner-Knoten!)
   b. Rekursionsschritt
      i. **Finde** eine Syntaxregel *LHS → LC Sisters*
      ii. **Left-corner-parse** alle Schwester-Kategorien *Sisters* von *LC*
      iii. **Vervollständige** *LHS* zu *C*

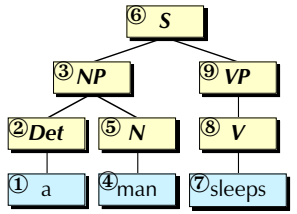
Strategie des Left-Corner-Parsing

Vorgehen beim Left-Corner-Parsing

- Um S zu parsen, nimm "a" (1); es ist ein Det (2).
- Um Det zu S zu vervollständigen, verwende NP-Regel (3) NP → Det N und parse N.
- Um N zu parsen, nimm "man" (4); es ist ein N (5). N vervollständigt direkt zu N.
- Um NP zu S zu vervollständigen, verwende S-Regel (6) S →NP VP und parse VP.
- Um VP zu parsen, nimm "sleeps" (7), es ist ein V (8).
- Um V zu VP zu vervollständigen, verwende VP-Regel (9) VP → V und parse nichts mehr (keine Schwestern). VP vervollständigt direkt zu VP.
- S vervollständigt direkt zu S.

S → NP VP
NP → Det N
VP → V

Det → a
N → man
V → sleeps

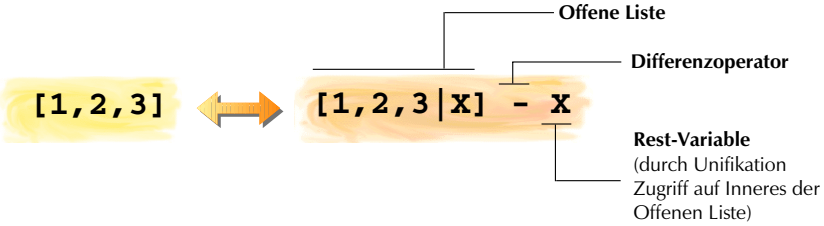


➥ *daten- und hypothesengesteuert*

Repräsentation der Eingabekette

Die Eingabekette als Differenzliste

- **Wort**: *man* ⇒ `[man,sleeps]-[sleeps]`
- **Phrase**: *a man* ⇒ `[a,man,sleeps]-[sleeps]`
- **Satz**: *a man sleeps* ⇒ `[a,man,sleeps]-[]`



Repräsentation der Regeln

Trennung von syntaktischen und lexikalischen Regeln

```
rule(s, [np, vp]).
rule(np, [det,n]).
rule(np, [np, conj, np]).
rule(vp, [v, adv]).
rule(adv, []).
```

```
word(a, det).
word(man, n).
word(sleeps, v).
word(and, conj).
word(now, adv).
```

S → NP VP
NP → Det N
NP → NP Conj NP
VP → V Adv
Adv → ε

Det → a
N → man
V → sleeps
Conj → and
Adv → now

lcp/2: Left-Corner-Parsen

- **lcp**(***Symbol***, ***Eingabekette***)

Eine Konstituente *C* left-corner-parsen heisst:

1. **Nimm** ein Wort von der Eingabe und bestimme seine Kategorie *LC*
2. **Vervollständige** *LC* zur Konstituente *C* mit dem Rest der Eingabe

```
lcp(C, [Word|Rest]-RestDiff) :-
    word(Word, LC),
    complete(LC, C, Rest-RestDiff).
```


complete/3: Vervollständigen

***LC* zur Kategorie *C* vervollständigen, heisst:**

a. Abbruchbedingung: **Stoppe**, falls LC = C ist

```
complete(C, C, S-S).
```

b. Rekursionsschritt:

i. **Finde** eine Syntaxregel *LHS* → *LC Sisters*
ii. **Left-corner-parse** alle Schwester-Kategorien *Sisters* von *LC*
iii. **Vervollständige** *LHS* zu *C*

```
complete(LC, C, S-Rest) :-
    rule(LHS, [LC|Sisters]),
    lcp_sisters(Sisters, S-SistersRest),
    complete(LHS, C, SistersRest-Rest).
```

lcp_sisters/2: Kategorien parsen

Eine Liste von Kategorien left-corner-parsen, heisst:

- Stoppen, falls die Liste leer ist
- rekursiv jedes Element der Liste left-corner-parsen

```
lcp_sisters([], S-S).

lcp_sisters([C|Cs], S-Rest) :-
   lcp(C, S-SisterRest),
   lcp_sisters(Cs, SisterRest-Rest).
```

- *Rest* ist der Teil der Eingabekette, der nach dem Aufruf von lcp_sisters noch zu parsen bleibt, *SisterRest* der Teil der Eingabekette, der nach dem Aufruf von lcp noch bleibt.

Vorteile von Left-Corner-Parsing

Bezüglich problematischer Regeln

- Kein Problem mit Linksrekursion!

NP → NP Conj NP

- Kein Problem mit Tilgungsregeln der Form (a), falls folgende Klausel zu lcp/2 hinzugefügt wird:

```
lcp(C, S-Rest) :-
   rule(LHS, []),
   complete(LHS, C, S-Rest).
```

VP → V Adv
Adv → ε

Tilgungsregel der Form (a)

- Aber Problem mit Tilgungsregeln der Form (b)

  Warum? Leere Produktionen an der Left-Corner-Stelle werden bottom-up geparst! Darum schlecht!

  Leere Produktionen an andern Stellen werden top-down geparst! Darum kein Problem!

NP → Det N
Det → ε

Tilgungsregel der Form (b)

Left-Corner-Links

Linking-Idee: Setze leere Kategorien nur dort ein, wo sie syntaktisch möglich sind!

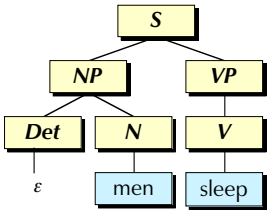
- Ein leerer Plural-Artikel ist ein möglicher Artikel, Beginn einer NP oder Beginn eines Satzes. Aber z.B. kein möglicher Beginn einer VP.

NP → Det N
Det → ε

- Die lc_link/2-Relation hält die Links fest, die an einer Left-Corner-Position möglich sind.

```
lc_link(np, s).
lc_link(det, np).
lc_link(det, s).
lc_link(v, vp).
```

Grammatikspezifische Links



lcp/3: Left-Corner-Parsing mit Links

lcp/3 mit Linking

- Der Aufruf von lc_link/2 erzwingt, dass Tilgungsregeln nur zulässig sind zur Bildung von grammatisch zulässigen Konstituenten.

```
lcp(C, [Word|Words]-Rest) :-
   word(Word, K),
   complete(K, C, Words-Rest).

lcp(C, S-Rest) :-
   rule(LHS, []),
   lc_link(LHS, C),
   complete(LHS, C, S-Rest).
```

Tilgungsproblem gelöst? Ja!

Keine unendlichen Suchbäume mehr bei Left-Corner-Tilgungsregeln

NP → Det N
Det → ε

- Sinnloses Anwenden der Tilgungsregel in der Left-Corner-Position wird verhindert
- Sinnvolles Anwenden wird gestattet

```
?- lcp(np, [men]-R).
R = [] ? ;
no
```

Anfrage

```
rule(np, [det,n]).
rule(det, []).

word(men, n).

lc_link(det, np).
```

Mini-Grammatik mit Links

Tilgungsproblem gelöst? Nein!

Leider keine Lösungen mehr bei andern Tilgungsregeln!

VP → V Adv
Adv → ε

- Ooops! Bisher problemlose Tilgungsregeln werden nicht mehr angewendet!

```
?- lcp(vp, [sleep]-R).
no
```

Anfrage

```
rule(vp, [v,adv]).
rule(adv, []).

word(sleep, v).

lc_link(v, vp).
```

Mini-Grammatik mit Links

Grund für Nein

Logischer Grund

- Adv steht zu Adv nicht in der lc_link-Relation

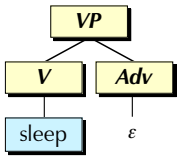
Beobachtung

VP → V Adv
Adv → ε

- Beim Parsen via lcp_sisters/2 müssen alle Kategorien mit ε-Expansion mit sich selbst gelinkt sein.

Ausweg

- Die Anfrage **?- lc_link(X, X).** muss gelingen.




```
1  1  Call: lcp(vp,[sleep]-[]) ?
2  2  Call: word(sleep,K) ?
2  2  Exit: word(sleep,v) ?
3  2  Call: complete(v,vp,[]-[]) ?
4  3  Call: rule(LHS,[v|R]) ?
4  3  Exit: rule(vp,[v,adv]) ?
5  3  Call: lcp_sisters([adv],[]-S1)
?
6  4  Call: lcp(adv,[]-S2) ?
7  5  Call: rule(LHS2,[]) ?
7  5  Exit: rule(adv,[]) ?
8  5  Call: lc_link(adv,adv) ?
8  5  Fail: lc_link(adv,adv) ?
```


lc_link/2 transitiv und reflexiv!

lc_link(Cat1, Cat2) beinhaltet drei Arten von Information

- Cat1 steht unmittelbar in der Left-Corner-Relation zu Cat2
- oder Cat1 ist mittelbar durch Left-Corner-Relationen mit Cat2 verbunden (*transitive Hülle*)
- oder Cat1 und Cat2 sind identisch (*reflexive Hülle*)

```
lc_link(np, s).
lc_link(det, np).
lc_link(v, vp).
```

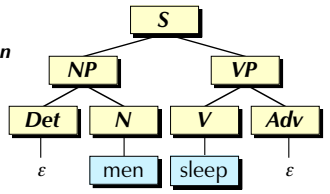
Left-Corner-Relation

```
lc_link(det, s).
```

Transitive Hülle

```
lc_link(X, X).
```

Reflexive Hülle



lc_link/2 berechnen

Berechnen von lc_link/2 aus Syntaxregeln

```
lc_link(LC, LHS) :-
   rule(LHS, [LC|_]).
```
⇐ ***Left-Corner-Relation***

```
lc_link(LC, C) :-
   rule(LHS, [LC|_]),
   lc_link(LHS, C).
```
⇐ ***Transitive Hülle***

```
lc_link(C, C).
```
⇐ ***Reflexive Hülle***

- **Probleme**: Was passiert bei rekursiven Regeln? Wie lassen sich mehrfache gleiche Lösungen verhindern?

Funktionalität und Effizienz

Left-Corner-Parsen mit (LC-)Links erlaubt insbesondere

- linksrekursive Regeln
- Tilgungsregeln
- *Damit können LinguistInnen arbeiten!*

Effizienteres Left-Corner-Parser durch

- direktes Implementieren der Left-Corner-Parsingstrategie (Stichwort BUP)
- verfeinertes Steuern der Suche

Effizienz und Backtracking

Identische Subkomponenten von Grammatikregeln

- In grösseren Grammatiken treten oft gleiche Konstituenten in verschiedenen Regeln auf.

  $VP \rightarrow V\ NP\ PP$
  $VP \rightarrow V\ NP\ NP$

- Beim Parsen eines Satzes wie wird die lange NP zweimal vollständig geparst, obwohl sie bei beiden potentiellen Analysen das Gleiche umfasst.

  I sent [$_{NP}$the very pleasant salesman that I met on holiday in Marbella last year] [$_{NP}$a postcard]

- *Die Speicherung von Zwischenresultaten (z.B. in einer sog. Chart) kann benutzt werden, um dieselbe Arbeit nicht mehrfach zu leisten.*

Literatur

Zum Left-Corner-Parsen

- Covington, M.(1994: 158ff.)

  Unser Kode stammt von dort. Zusätzlich gibt er Beschreibungen des sog. BUP-Algorithmus, der die Regeln noch effizienter repräsentiert

- Naumann, S./ Langer H. (1994): Parsing. Eine Einführung in die maschinelle Analyse natürlicher Sprache. Teubner, Stuttgart.

  Das (anspruchsvolle) **Parsing-Buch** auf Deutsch. Unterschiedlichste Parsing-Algorithmen werden vorgestellt, analysiert und in PROLOG und LISP implementiert. Zum Left-Corner-Parser Kapitel 5 (S. 83-100)

- Matthews, C.(1998): An Introduction to Natural Language Processing Through Prolog. Longman, London. (S. 220ff)
- Müller, St.(1998): Prolog und Computerlinguistik: Teil I – Syntax. (S. 78ff.) http://www.dfki.de/~stefan/Pub/prolog.html