Charts

Übersicht

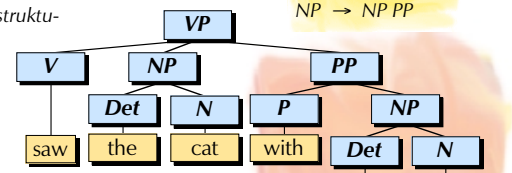
- Chart bzw. Well-Formed Substring Table (WFST)
  - Als azyklischer Graph, Tabelle und Relation
- Kantenbeschriftungen
  - Kategorien: WFST
  - Regeln: Passive Charts
  - Regelhypothesen: Aktive Charts
- Chart-Parsing-Verfahren
  - Interpretierender Top-Down-Parser
  - Top-Down mit WFST
  - Das Vollständigkeitsproblem
  - Top-Down mit Vollständigkeitsvermerk

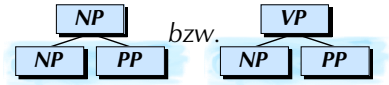
Motivation

Mehrfachanalysen sind oft unnötig!

- Die RHS für jede Phrasenstrukturregel (PSR) wird separat berechnet, auch wenn sie z.T. gleiche Konstituenten haben
  - *Wenn sich eine PSR als falsch erweist, 'vergisst' der Parser alles, was er schon erkannt hat, obwohl meist die Teilstrukturen korrekt analysiert wurden.*
  - *Wenn eine Grammatik mehrdeutig ist, werden auch die involvierten Konstituenten immer mehrfach analysiert.*

VP → V NP
VP → V NP PP
NP → Det N
NP → NP PP





Grundfrage

Frage

- Wie kann man verhindern, dass bei einer VP wie

  chased the cat with the bone

  und einer Grammatik wie

  VP → V NP
  VP → V NP PP
  NP → Det N
  NP → NP PP
  PP → P NP

  gemeinsame Bestandteile nicht mehrfach berechnet werden?

Antwort

- Durch Speichern der Teilkonstituenten in einer Chart bzw. einem Well-Formed Substring Table (WFST).

Chart als Graph

Visualisieren von Charts als azyklische Graphen

- Charts sind keine Bäume!
  - Zwischenpositionen sind Knoten
  - Terminale und Nicht-Terminale sind beschriftete verbindende Kanten

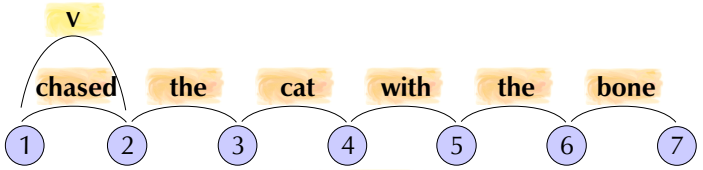


Chart als Tabelle

Charts in Tabellenform

- Zwischenpositionen bilden End- und Anfangspunkte.
- Tabelleneinträge sind die Kantenbeschriftungen.
- In den **Spalten** stehen Symbole mit gleichem **Endpunkt**.
- In den **Zeilen** stehen Symbole mit gleichem **Anfangspunkt**.
  - In einem Kästchen können mehrere Tabelleneinträge gemacht werden. Z.B. auch nötig bei kategoriell mehrdeutigen Wörtern

Endpunkte

Anfangspunkte

	2	3	4	5	6	7
1	v		vp			vp
2		det	np			np
3			n			
4				p		pp
5					det	np
6						n

Chart als Prolog-Relation

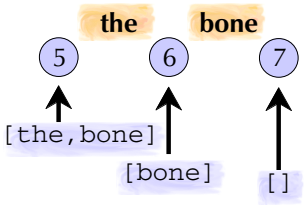
edge(*Kategorie, Startposition, Endposition*)

- Chart als edge/3 Prädikat
- Jede Klausel steht für eine Kante.
- Die Positionen können direkt als Zahlen repräsentiert werden.
- Falls zwischen zwei Punkten mehrere Kanten möglich sind, gibt es einfach mehrere Klauseln, die sich nur in der Kategorie unterscheiden.

```
edge(  v, 1, 2).
edge( vp, 1, 4).
edge( vp, 1, 7).
edge(det, 2, 3).
edge( np, 2, 4).
edge( np, 2, 7).
edge(  n, 3, 4).
edge(  p, 4, 5).
edge( pp, 4, 7).
edge(det, 5, 6).
edge( np, 5, 7).
edge(  n, 6, 7).
```


Position: Numerisch oder differentiell

Die Repräsentation der Positionen als Zahlen oder Differenzlisten ist äquivalent.




```
edge(det, 5, 6).
```

```
edge(det, [the,bone], [bone]).
```

- Ob wir eine Differenzliste als 2 Terme oder via Operator '-' verknüpfen ist einerlei…

```
edge(det, [the,bone], [bone]).  ⇔  edge(det, [the,bone]-[bone]).
```


WFST vs. Ableitungsbaum

Aus WFST kann der Ableitungsbaum nicht immer eindeutig rekonstruiert werden…

- Aus untenstehendem abstrakten WFST allein wird nicht ersichtlich, ob *E* wegen einer Regel $E \rightarrow A\ D$ oder $E \rightarrow C\ B$ eingefügt wurde!

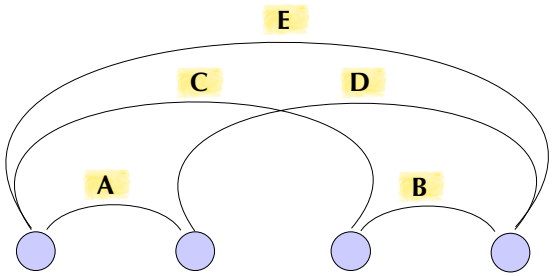


Chart mit Regeln

Damit sich Ableitungsbäume aus Charts ablesen lassen, beschrifte jede Kante mit ihrer/n erzeugenden Regel/n!

- Solche Kanten heissen auch **passive** oder **inaktive** Kanten
- Charts, die aus passiven Kanten bestehen, heissen **passive Charts**.

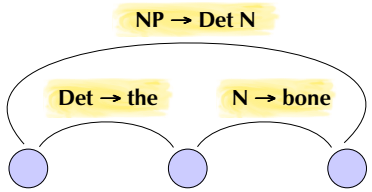
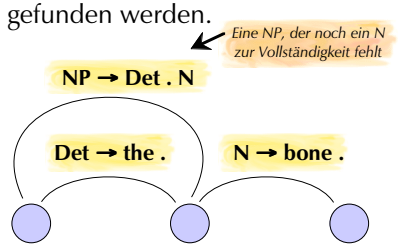


Chart mit Regelhypothesen

Aktive Kanten drücken Strukturhypothesen aus

- **Aktive Kanten** zeigen, welche Konstituente analysiert ist, falls zu den bereits analysierten Strukturen noch bestimmte hypothetische (= noch nicht erkannte) Strukturen gefunden werden.
- **Vor** dem Punkt:
  Bereits gefundener Teil
- **Nach** dem Punkt:
  Noch fehlender Teil



- Charts, die aktive Kanten enthalten, heissen **aktive Charts.**
  - *Siehe Folien "Earley-Parser"*

Gepunktete Regeln

Eine gepunktete Regel *(dotted rule)* steht für ein Zwischenresultat des Parsers.

Mögliche Parsezustände der Regel ***S → NP VP***

S → . NP VP

- Ein Satz wird gesucht, aber es wurde noch nichts erkannt.

S → NP . VP

- Vom gesuchten Satz wurde die NP, aber noch keine VP erkannt.

S → NP VP .

- Ein Satz aus NP und VP wurde vollständig erkannt.

Nutzen von Charts

Dank Charts (mit passiven Kanten)

- wird jede Konstituente nur einmal berechnet
- verfügen wir über eine einfache Datenstruktur, die alternative Analysen repräsentieren kann.
- verfügen wir über einen Grad an Robustheit: Auch wenn nicht der ganze Satz analysiert werden kann, werden alle erkennbaren Teilstücke in die Chart eingetragen.

Dank Charts mit aktiven Kanten

- wird sowohl das Resultat als auch der Fortschritt des Parsevorgangs einheitlich repräsentiert.
- kann bei ungrammatischem Input ev. Fehlendes erkannt werden.

Chart-Parsing-Verfahren

Die Chart ist eine Datenstruktur

- … und kein bestimmtes Parsing-Verfahren!
- Es gibt ganz verschiedene Verfahren, die mit einer Chart arbeiten
  - Top-Down-Chart-Parsing
  - Bottom-Up-Chart-Parsing
  - BUP-Parsing (Left-Corner mit Charts)
  - Earley-Algorithmus
  - … und zahlreiche mehr

Die Verwendung von Charts beim Parsen ist eine wichtige und weit verbreitete Technik in der Computerlinguistik!

Top-Down-Parser

Parser

```
% parse/2: Parsen einer Kategorie C

parse(C, [Word|S]-S) :-
   word(C, Word).

parse(C, S-Rest) :-
   rule(C, RHS),
   parse_daughters(RHS, S-Rest).


% parse_daughters/2: Parsen einer
% Liste von Kategorien

parse_daughters([C|Cs], S-Rest) :-
   parse(C, S-S1),
   parse_daughters(Cs, S1-Rest).

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

Grammatik

```
% Phrasen-Struktur-Regeln

rule(s, [np,vp]).
%...

% Lexikon

word(d, the).
%...
```

Parsen

```
?- parse(vp, [chased,the,cat,with,the,bone]-[]).
```


Naiver Top-Down-Parser mit WFST

Parser

```
% parse/2: Parsen einer Kategorie C

parse(C, [Word|S]-S) :-
   word(C, Word).

parse(C, S-Rest) :-
    edge(C, S, Rest).

parse(C, S-Rest) :-
   rule(C, RHS),
   parse_daughters(RHS, S-Rest),
   asserta(edge(C,S,Rest)).

% parse_daughters/2: Parsen einer
% Liste von Kategorien

parse_daughters([C|Cs], S-Rest) :-
   parse(C, S-S1),
   parse_daughters(Cs, S1-Rest).

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

Grammatik

wie bisher…

Chart

```
% Kanten
:- dynamic edge/3.

% Chart saeubern
clear_chart :-
    retract(edge(_,_,_)),
    fail.
clear_chart.
```

Parsen

```
?- clear_chart.
?- parse(vp,
     [chased,the,cat,with,the,bone]-[]).
```


Probleme des Top-Down-Erkenners

Normales Parsen und Nachschlagen in der Chart konkurrenzieren sich.

Vollständigkeit der Chart ist unsicher! Weil…

- Unsichere positive Information: Fehlt die Konstituente in der Chart, weil sie noch nicht analysiert wurde?
- Unsichere negative Information: Fehlt die Konstituente in der Chart, weil sie gar nie erscheinen kann?

Ausweg

- Halte die Vollständigkeit der Charteinträge einer Kategorie für eine Eingabekette explizit fest, nachdem alle möglichen Konstituenten dieser Kategorie analysiert sind.

Vollständigkeitsvermerk

Das dynamische Prädikat complete/2 vermerkt Vollständigkeit.

2 Falls eine Konstituente ab Position nicht vollständig ist, ignoriere Chart. Falls Information vollständig ist bzgl. Kategorie und Position, aber trotzdem kein Chart-Eintrag existiert, dann scheitere hier endgültig (Cut!).

3 Suche **alle** Konstituenten C beliebiger Länge und füge sie in die Chart ein, aber gelinge nur bei der gesuchten Länge.

4 Falls via Klausel 3 keine Konstituente mehr gefunden wurde, d.h. alle positive Information in die Chart eingefügt ist, hat dies negative Information ergeben. C ist ab dieser Position vollständig geparst.

```
:- dynamic complete/2.

parse(C, [Word|S]-S) :-
   word(C, Word).

parse(C, S-Rest) :-                    2
   complete(C, S),
   !,
   edge(C, S, Rest).

parse(C, S-Rest) :-                    3
   rule(C, RHS),
   parse_daughters(RHS, S-S1),
   asserta(edge(C, S, S1)),
   S1 = Rest.

parse(C,S-_) :-                        4
   asserta(complete(C,S)),
   fail.
```


Literatur

Zu Charts und Chart-Parsing

- Covington, M.(1994: 167ff.): Natural Language Processing for PROLOG Programmers.

  Unser Kode stammt von dort.
- Naumann, S./ Langer H. (1994): Parsing. Eine Einführung in die maschinelle Analyse natürlicher Sprache. Teubner, Stuttgart.
- Müller, St. (1998: 84ff.): Prolog und Computerlinguistik: Teil I – Syntax.
- Gazdar, G./Mellish, Ch. (1989: 179ff.): Natural Language Processing in PROLOG.