Parsing-Einführung

Übersicht

- Falsifizierbarkeit, oder: Sind Grammatiken wissenschaftlich?
- Grammatik, Formalismus
- Kontextfreie Grammatiken
- Ableitungen

Ziel

- Verstehen der linguistischen Motivation
- Intuitives Verständnis, was ein Parser macht
- Benutzen des in Prolog eingebauten Parsers

Grammatik

Lexikon-Definitionen für »Grammatik«

- Wissen/Lehre von den Regularitäten einer natürlichen Sprache
- Strukturelles Regelsystem
- Systematische Beschreibung der formalen Regularitäten einer natürlichen Sprache

Sind Grammatiken wissenschaftlich?

Falsifizierbarkeit

- Bei einer wissenschaftlichen Theorie soll klar sein, durch welche Beobachtungen sie widerlegt *(falsifiziert)* werden könnte.

Womit wird eine Grammatik widerlegt?

- Bei vielen linguistischen Theorien unklar (!)
- Umdenken seit den 1950er Jahren
  - Theorie über die Struktur einer Sprache = *formales* System von Grammatikregeln
- Beobachtbar bei einer Grammatik:
  - Ein korrekter Satz entspricht nicht der Grammatik
  - Ein falscher Satz entspricht der Grammatik

    → klares, nachprüfbares Falsifikations-Kriterium

Grammatik, Formalismus

Grammatiken sollen so festgehalten werden, dass nachprüfbar ist, ob ein Satz der Grammatik entspricht.

- Das heisst auch: In einer Notation, deren Bedeutung klar ist
  - Natürliche Sprache ist jedoch unscharf und mehrdeutig
  - Daher eignet sich eine formale Schreibweise besser

Formalismus

- Formalisierung: Verwendung formaler Sprachen der Mathematik zur Beschreibung der Grammatik einer natürlichen Sprache
- Vorteile eines Formalismus mit mathematischer Grundlage:
  - klarere Terminologie, grössere Präzision — »es ist klar, worüber geredet wird«
  - leichtere und zuverlässigere Überprüfbarkeit der Argumente

Grammatik, Formalismus

Die Grammatik ist in idealerweise in einem mathematisch fundierten Formalismus gehalten.

- Sonst wird schnell unklar, was genau die Theorie umfasst
- Sonst sind zwei Theorien kaum miteinander vergleichbar
- Sonst ist es schwer, Sätze der Grammatik entsprechend maschinell zu verarbeiten
- Sonst kann leicht verschleiert werden, durch welche Beobachtungen die Theorie widerlegt würde

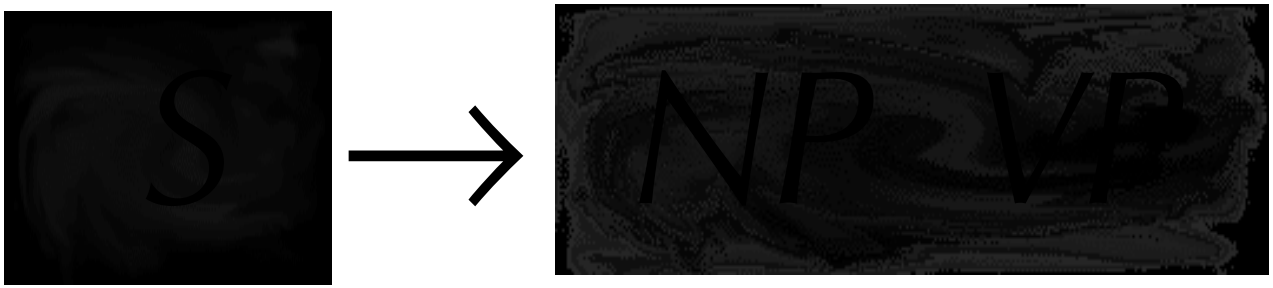
Kontextfreie Grammatiken (CFG)

Kontextfreie Grammatiken *(Context-Free Grammars, CFG)* sind ein solcher Formalismus

- von Noam Chomsky in den 1950er Jahren zur Modellierung der Syntax des Englischen vorgeschlagen
- gut erforschte mathematische Eigenschaften
- jahrzehntelange Diskussion, ob für natürliche Sprachen mächtig genug
  - endgültige Klärung erst 1985: Zürichdeutsch und Bambara können nicht mit CFG beschrieben werden
- Es gibt andere, besser geeignete Formalismen als CFG
- dennoch wichtig für Linguistik, Computerlinguistik, Informatik

Kontextfreie Regeln

Bestandteile einer kontextfreien Regel:

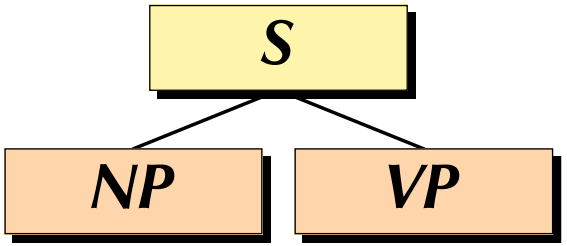


linke Seite
left-hand side

rechte Seite
right-hand side

Zwei gleichwertige Lesarten:

- Ein S besteht aus einer NP und einer VP
- Eine NP, gefolgt von einer VP, ergibt ein S



Kontextfreie Grammatiken

Beispiele für Regeln einer kontextfreien Grammatik:

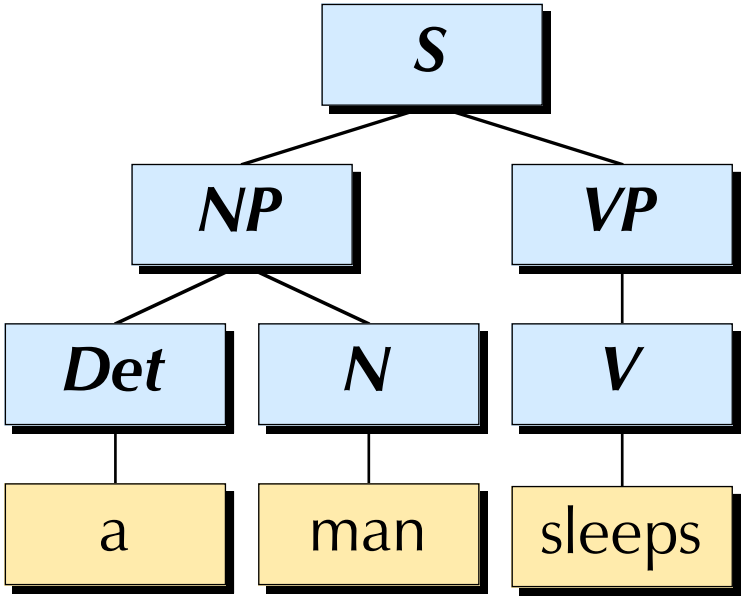
$S \rightarrow NP\ VP$	$Det \rightarrow the$	$V \rightarrow loves$
$NP \rightarrow Det\ N$	$Det \rightarrow a$	$V \rightarrow sleeps$
$VP \rightarrow V\ NP$	$N \rightarrow man$	$V \rightarrow sees$
$VP \rightarrow V$	$N \rightarrow woman$	$V \rightarrow thinks$

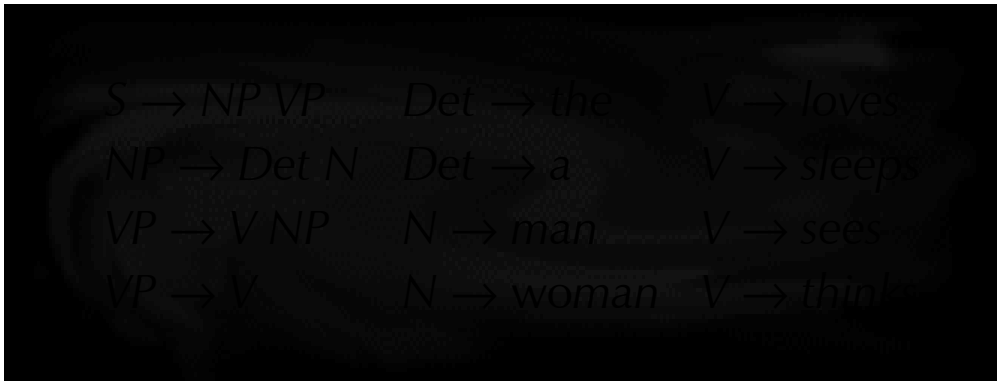
Gemäss diesen Grammatikregeln sind beispielsweise die folgenden Sätze (S) erlaubt:

- a man sleeps
- the woman sees a man
- a woman loves a woman

Kontextfreie Grammatiken

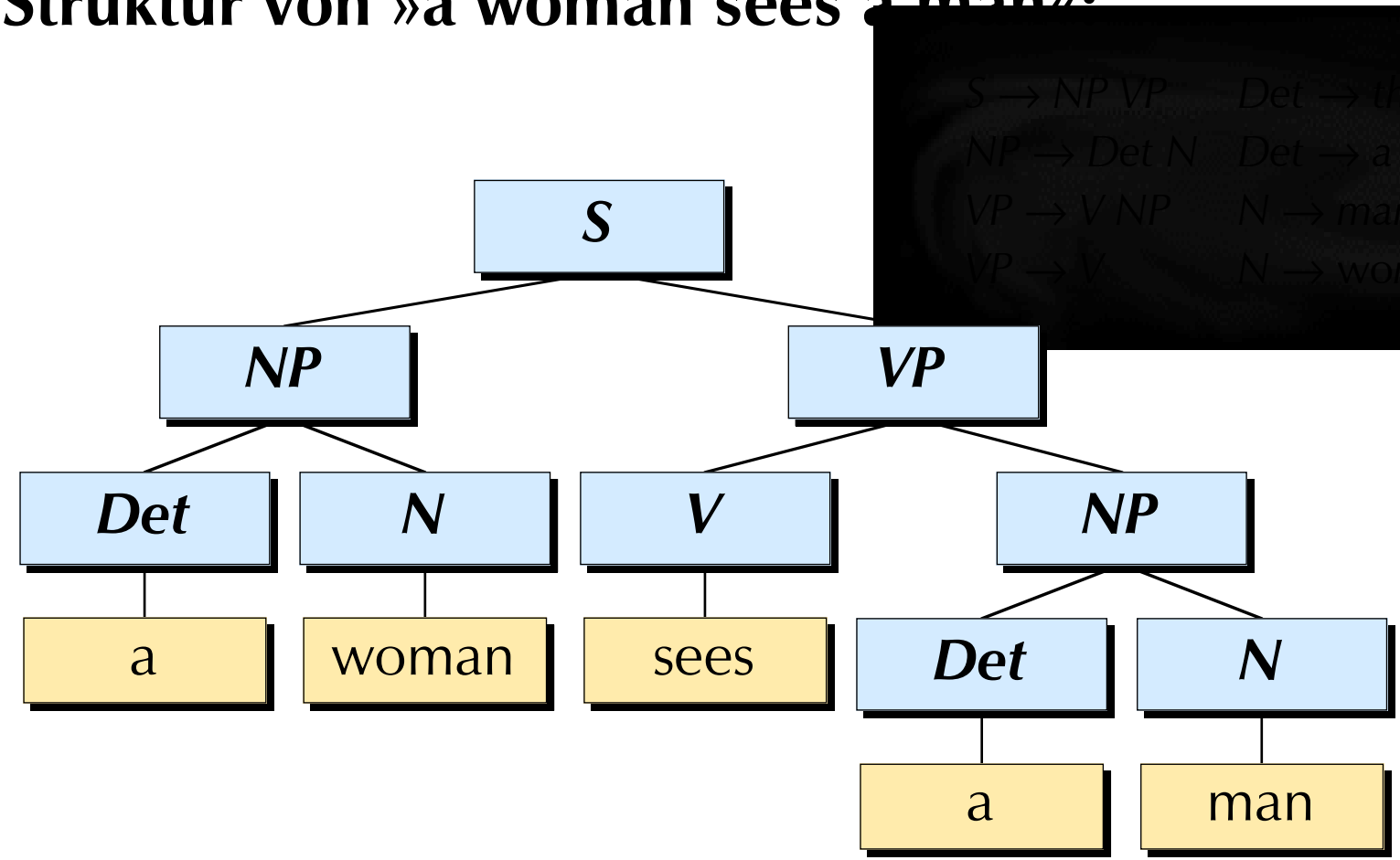
Struktur von »a man sleeps«:





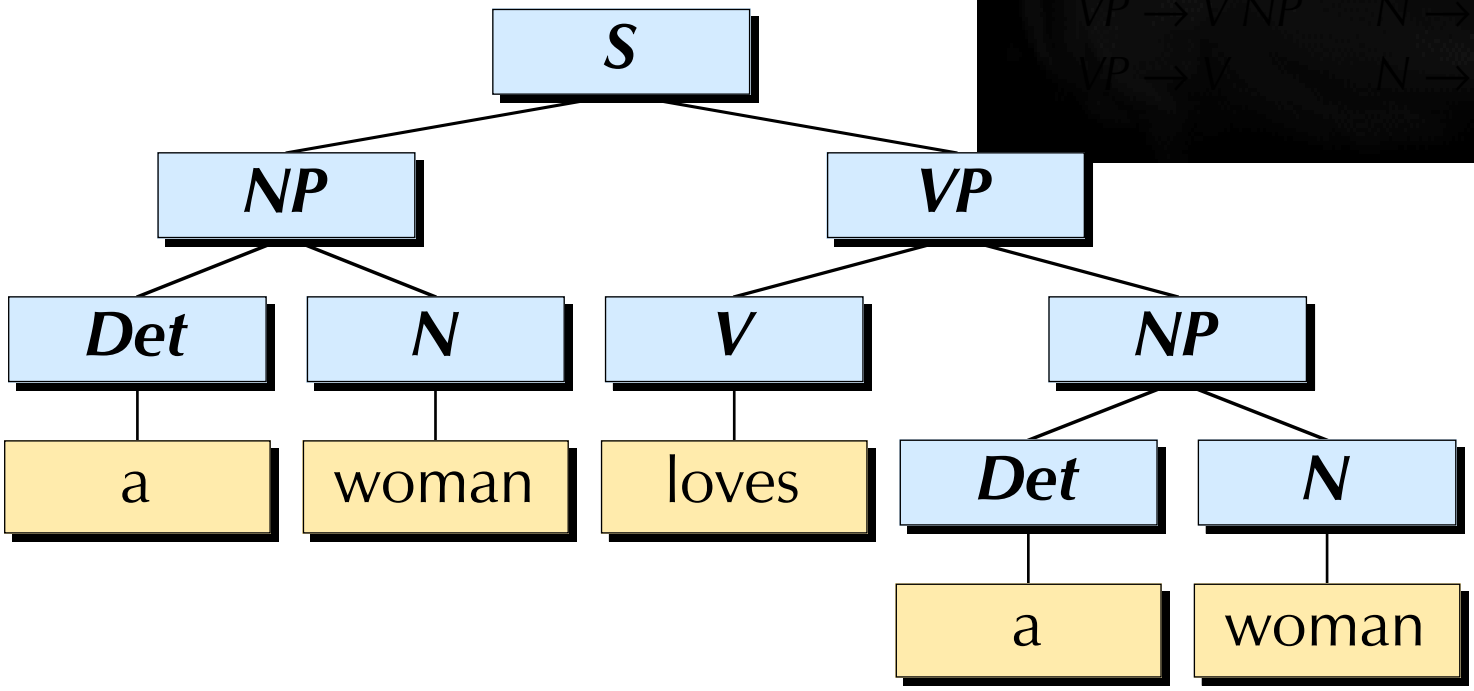
Kontextfreie Grammatiken

Struktur von »a woman sees a man«:



Kontextfreie Grammatiken

Struktur von »a woman loves a woman«:



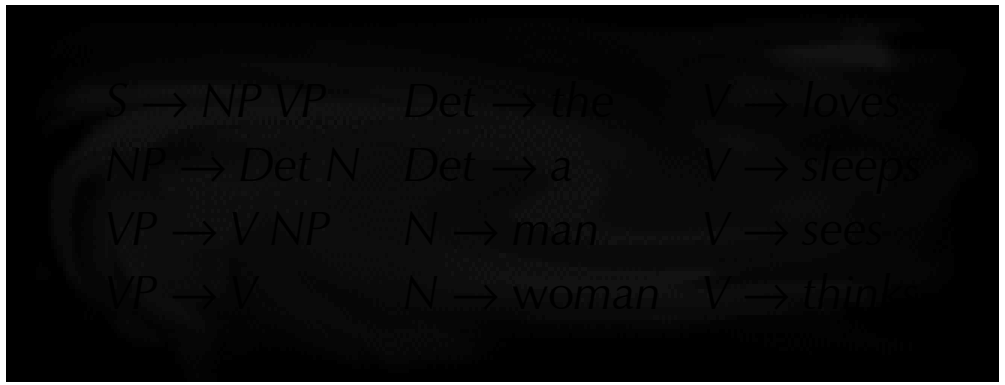
Bestandteile einer CFG

Symbole — V

Startsymbol — S
(Element von V)

Terminalsymbole — Σ
(Teilmenge von V)

Regeln — R



Bestandteile einer CFG

Eine kontextfreie Grammatik ist ein 4-Tupel $\langle V, \Sigma, R, S \rangle$, wobei

- V ist eine endliche Menge von **Symbolen**
- Σ ist eine endliche Menge von **Terminalsymbolen** $\subset V$
- R — endliche Menge von **Regeln** $\subseteq (V - \Sigma) \times V^*$
- S — **Startsymbol** $\in V - \Sigma$

Nicht-Terminale

- $V - \Sigma$ (die Menge der Elemente von V, die nicht in Σ sind)

Wir schreiben für die Regeln $A \rightarrow u$

- statt $\langle A, u \rangle$, um die Regeln lesbarer zu machen

Ableitung

Für $u, v \in V^*$ schreibt man $u \Rightarrow v$ genau dann wenn

- es gibt Ketten $x, y, v' \in V^*$ und $A \in V - \Sigma$ so dass
  - $u = x\,A\,y$
  - $v = x\,v'\,y$
  - $A \rightarrow v'$

Beispiele

- *a man V a woman* $\Rightarrow$ *a man sees a woman*
- *Det man V a woman* $\Rightarrow$ *a man V a woman*
- *a man VP* $\Rightarrow$ *a man V NP*
- *S* $\Rightarrow$ *NP VP*

S → *NP VP*	*Det* → *the*	*V* → *loves*
NP → *Det N*	*Det* → *a*	*V* → *sleeps*
VP → *V NP*	*N* → *man*	*V* → *sees*
VP → *V*	*N* → *woman*	

Ableitung

Eine Folge von durch ⇒ verbundenen Ketten heisst Ableitung *(derivation)*.

Da in jedem Schritt *irgendein* Nichtterminal ersetzt wird, kann dieselbe Kette auf verschiedene Weisen vom Startsymbol abgeleitet werden:

- *S ⇒ NP VP ⇒ Det N VP ⇒ a N VP ⇒ a man VP ⇒ a man V ⇒ a man sleeps*
- *S ⇒ NP VP ⇒ NP V ⇒ Det N V ⇒ Det man V ⇒ Det man sleeps ⇒ a man sleeps*

Parser, Akzeptor

Ein Programm zur syntaktischen Analyse (»Parser«)

- nimmt eine Kette von Wörtern entgegen
- beurteilt, ob die Eingabe gemäss den Regeln einer Grammatik zulässig ist
  - Wenn ja, geben richtige Parser die Struktur der Eingabekette aus
  - Akzeptoren antworten nur mit »akzeptabel« bzw. »nicht akzeptabel«

Für denselben Grammatik-Formalismus kann es verschiedene Parsingverfahren geben

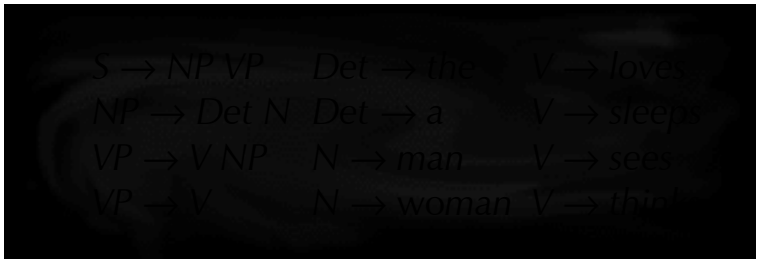
- Für kontextfreie Grammatiken: Dutzende Verfahren

Top-Down-Parsing

Ein Top-Down-Parser für eine kontextfreie Grammatik

- fängt mit dem Startsymbol an
- führt wiederholt Ableitungsschritte durch
- Ziel: Ableiten der zu analysierenden Kette

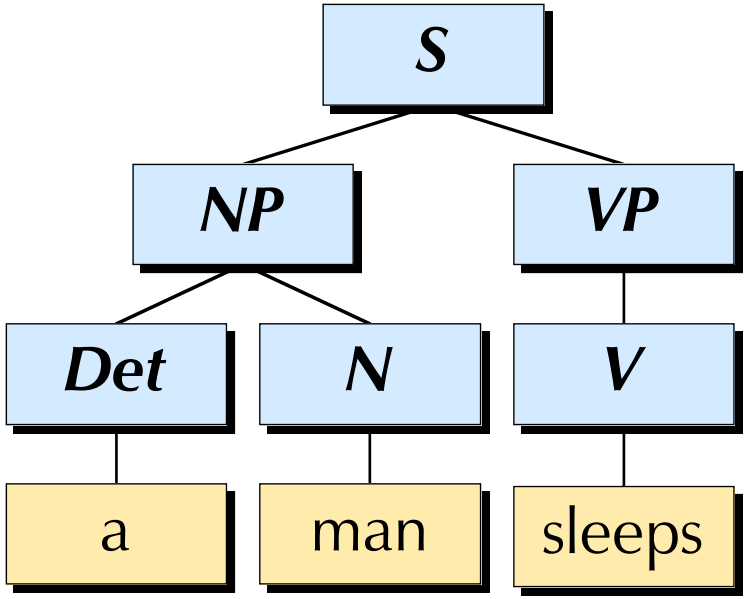
$S \Rightarrow NP\ VP \Rightarrow Det\ N\ VP \Rightarrow a\ N\ VP \Rightarrow a\ man\ VP \Rightarrow a\ man\ V \Rightarrow a\ man\ sleeps$

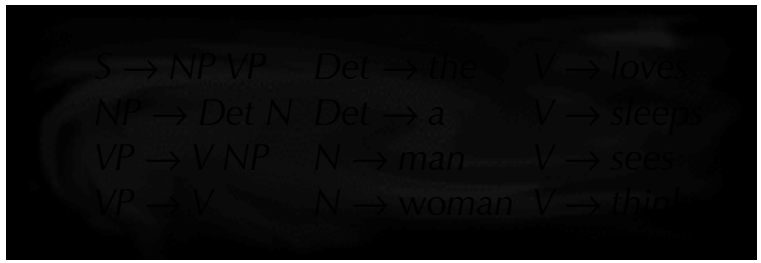


Top-Down-Parsing

Vorgehen beim Top-Down-Parsing:

- Ich suche ein S
- Um ein S zu erhalten, brauche ich eine NP und eine VP
- Um eine NP zu erhalten, brauche ich ein Det und ein N
- Um ein Det zu erhalten, kann ich das Wort »*a*« verwenden — gefunden
- Um ein N zu erhalten, kann ich das Wort »*man*« verwenden — gefunden
- Damit ist die NP vollständig
- Um eine VP zu erhalten, brauche ich ein V
- Um ein V zu erhalten, kann ich das Wort »*sleeps*« verwenden — gefunden
- Damit ist die VP vollständig
- Damit ist das S vollständig



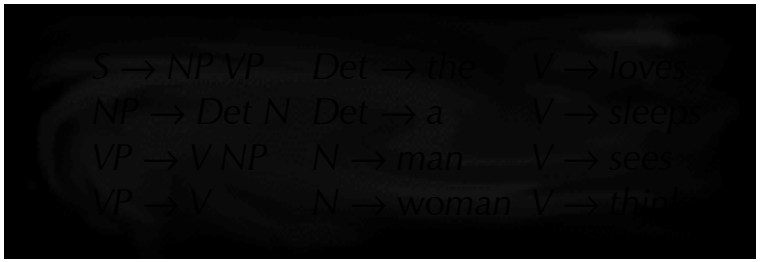


Bottom-Up-Parsing

Ein Bottom-Up-Parser für eine kontextfreie Grammatik

- fängt mit der zu analysierenden Kette an
- führt wiederholt Ableitungsschritte *»rückwärts«* durch
- Ziel: Erreichen des Startsymbols

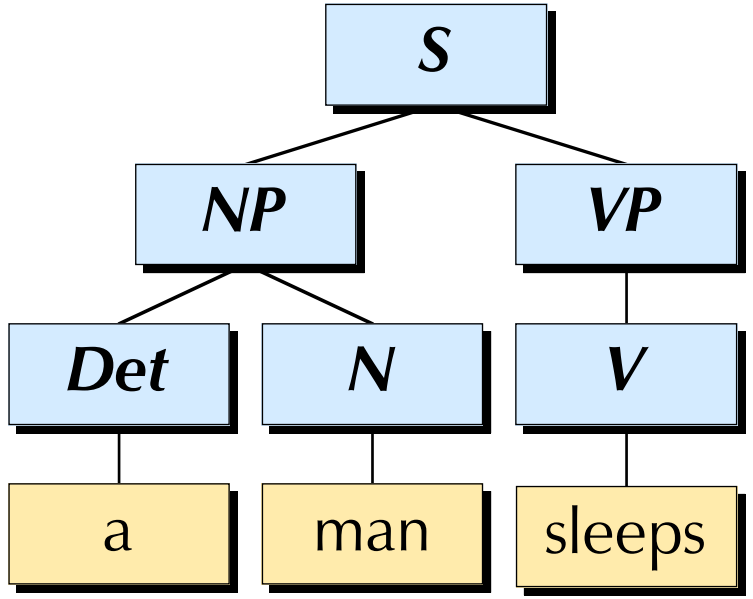
$S \Rightarrow NP\ VP \Rightarrow NP\ V \Rightarrow NP\ sleeps \Rightarrow Det\ N\ sleeps \Rightarrow Det\ man\ sleeps \Rightarrow$ *a man sleeps*

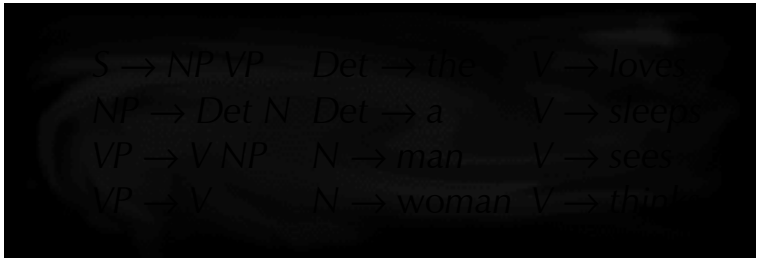


Bottom-Up-Parsing

Vorgehen beim Bottom-Up-Parsing:

- Nimm ein Wort — es ist *»a«*
- *»a«* ist ein Det
- Nimm ein weiteres Wort — es ist *»man«*
- *»man«* ist ein N
- Det und N bilden zusammen eine NP
- Nimm ein weiteres Wort — es ist *»sleeps«*
- *»sleeps«* ist ein V
- V bildet (für sich alleine) eine VP
- NP und VP bilden zusammen ein S





Definit-Klausel-Grammatiken (DCGs)

In Prolog ist ein einfacher Top-Down-Parser bereits eingebaut.

- Grammatik besteht aus Prolog-Klauseln (Definiten Klauseln)
- nützlich zum schnellen Ausprobieren einer Mini-Grammatik

Allerdings:

- eher ineffizientes Verfahren
  - bei mehrdeutigen Grammatiken werden unter Umständen Teile des Satzes mehrmals analysiert
- »Aufhängen« bei bestimmten Grammatiken
- $\Rightarrow$ für richtige Sprachverarbeitungsprojekte so gut wie unbrauchbar

DCGs: Formalismus

Regeln mit Nichtterminalen auf der rechten Seite:

```
s --> np, vp.
np --> det, n.
```

```
vp --> v.
vp --> v, np.
```

Regeln mit Terminal-Symbolen auf der rechten Seite:

```
det --> [the].
det --> [a].
```

```
n --> [cat].
v --> [sees].
v --> [sings].
```

DCGs: Benutzung

Ein eingebautes Prädikat phrase/2 überprüft, ob von einem Nichtterminal eine Kette von Terminalsymbolen abgeleitet werden kann.

```
?- phrase(s, [the, cat, sings]).
yes

?- phrase(np, [a, cat]).
yes

?- phrase(s, [a, cat]).
no
```

Aufgaben: Parsing-Einführung

Programmiertechniken der Computerlinguistik 1 · Wintersemester 1998/99

1. Kontextfreie Grammatik erstellen

Schreibe eine Grammatik für einfache deutsche Sätze. Benutze hierzu folgende Nichtterminal-Symbole:

Symbol	Bedeutung	Beispiel(e)
S	Satz	Die Katze spielt mit dem gelben Käse.
NP	Nominal-Phrase	die Katze · dem gelben Käse
VP	Verbal-Phrase	spielt mit dem Käse · singt
PP	Präpositional-Phrase	mit dem Käse
N	Nomen	Käse
V	Verb	spielt · singt · sieht
P	Präposition	mit · für · unter
A	Adjektiv	gelben · rotes · grün
Det	Artikel (Determiner)	der · die · das · dem

2. Formulieren als DCG

Überprüfe für einige Beispielsätze, ob sie Deiner Grammatik aus der ersten Aufgabe (also Deiner Theorie über die Syntax des Deutschen) entsprechen, indem Du die Grammatik als DCG formulierst und den in Prolog eingebauten Parser (der in der bisher vorgestellten Form ja eigentlich nur ein Akzeptor ist) benutzt.

Gibt es eine Möglichkeit, wie Du Prolog nach sämtlichen Sätze fragen kannst, die der Grammatik entsprechen?

3. Über-/Untergenerierung

Eine Grammatik heisst *übergenerierend,* wenn zu ihrer Sprache auch Sätze gehören, die nicht Teil der modellierten natürlichen Sprache sind.

Umgekehrt heisst eine Grammatik *untergenerierend,* wenn sie bestimmte Sätze nicht akzeptiert, auch wenn diese Bestandteil der natürlichen Sprache wären, die mit der Grammatik modelliert wird.

a) Ist Deine Grammatik aus Aufgabe 1 übergenerierend? Nenne einige fehlerhafte deutsche Sätze, welche Deine Grammatik akzeptiert.

b) Ist Deine Grammatik untergenerierend? Nenne einige korrekte deutsche Sätze, welche Deine Grammatik nicht akzeptiert.

c) Überlege Dir, ob es für die maschinelle Verarbeitung natürlicher Sprache schlimmer ist, wenn eine Grammatik über- oder wenn sie untergeneriert.