

# Programmierkurs Python II

## Vorlesung 6: Parsing I

Michaela Regneri & Stefan Thater  
FR 4.7 Allgemeine Linguistik (Computerlinguistik)  
Universität des Saarlandes

Sommersemester 2012



## Grammatiken

- Grammatiken generieren Wörter (Sätze)

S → NP VP	DET → der
NP → DET N	DET → das
NP → NP PP	N → student
PP → P NP	N → buch
VP → V	N → bibliothek
VP → V NP	V → arbeitet
VP → VP PP	V → liest
	P → in

- *Der Student arbeitet*
- *Der Student arbeitet in der Bibliothek*
- *Der Student liest das Buch*
- *Der Student liest das Buch in der Bibliothek*
- *[...]*

# Kontextfreie Grammatiken

- Kontextfreie Grammatik  $G = \langle N, T, R, S \rangle$ 
  - Nichtterminalsymbole  $N$
  - Terminalsymbole  $T$
  - Startsymbol  $S \in N$
  - Endliche Menge von Regeln:  $R \subseteq N \times (N \cup T)^*$

3

# Übergangsrelation

- $x \Rightarrow_G y$  gdw. es gibt  $u, v, w, z \in (N \cup T)^*$  so dass
  - $x = uvw$
  - $y = uzv$
  - $v \rightarrow z \in R$
- Sprechweise:
  - $x$  geht (unter  $G$ ) unmittelbar in  $y$  über

4

# Ableitungen

- Eine **Ableitung** von  $y$  aus  $x$  (in  $G$ ) ist eine Folge
  - $x = w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n = y$
- Die durch  $G$  **erzeugte Sprache**:
  - $L(G) = \{ w \in T^* \mid S \Rightarrow_G^* w \}$
  - $\Rightarrow_G^*$  ist die reflexive, transitive Hülle von  $\Rightarrow_G$

5

# Linksableitungen

- **Linksableitung**: in jedem Ableitungsschritt wird das am weitesten links stehende Nichtterminalsymbol ersetzt.
- $x \Rightarrow_L y$  gdw. es gibt  $A \in N$ ,  $a, b \in (N \cup T)^*$ ,  $w \in T^*$ 
  - $x = wAb$
  - $y = wab$
  - $A \rightarrow a \in R$
- **Rechtsableitungen**: analog

6

# Ableitungsbäume

- Kontextfreie Grammatik  $G = \langle N, T, R, S \rangle$
- **Ableitungsbäume:** Bäume mit folgenden Eigenschaften
  - Innere Knoten sind mit Symbolen aus  $N$  beschriftet
  - Blattknoten sind mit Symbolen aus  $T \cup \{\varepsilon\}$  beschriftet
  - Wenn  $v$  ein Knoten mit Beschriftung  $A$  und Kindern  $v_1, \dots, v_n$  mit Beschriftungen  $A_1, \dots, A_n$  ist, dann ist  $A \rightarrow A_1 \dots A_n$  eine Regel aus  $G$
  - Wenn ein Blatt mit  $\varepsilon$  beschriftet ist, dann ist das Blatt das einzige Kind seiner Mutter

7

# Satz (siehe Lewis & Papamitriou)

- Sei  $G = \langle N, T, R, S \rangle$  eine kontextfreie Grammatik.
- Folgenden Aussagen sind äquivalent:
  - $A \Rightarrow_G^* w = w_1 \dots w_n$
  - Es gibt Ableitungsbau mit Wurzel  $A$  und Ertrag  $w$
  - Es gibt eine Linksableitung  $A \Rightarrow_L^* w$
  - Es gibt eine Rechtsableitung  $A \Rightarrow_R^* w$

8

# Mehrdeutigkeit

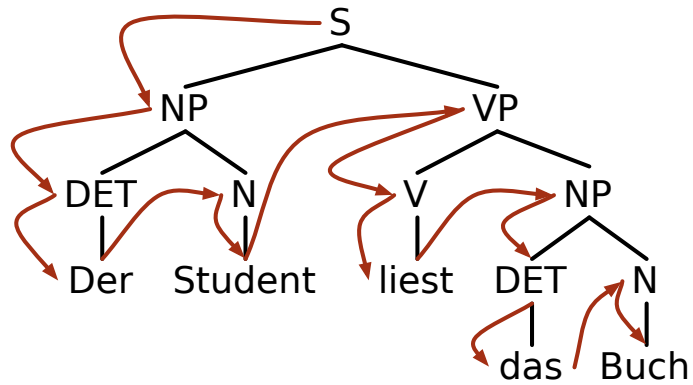
- Eine Grammatik  $G$  ist **eindeutig**, wenn es zu jedem Wort aus  $L(G)$  genau einen Ableitungsbaum gibt.
- Sonst heißt  $G$  **mehrdeutig**.

# Erkenner & Parser

- **Erkenner** (Recognizer)
  - Ist Eingabekette (Wort)  $w_1 \dots w_n \in L(G)$ ?
- **Parser**:
  - Welche Ableitungsstruktur(en) hat  $w_1 \dots w_n$ ?

# Elementare Parsing-Strategien

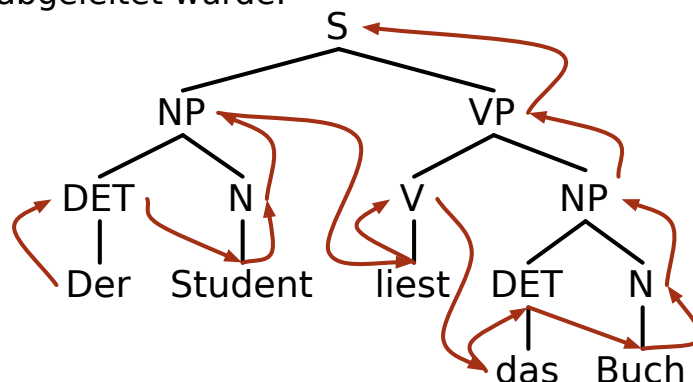
- Ein **Top-Down** Algorithmus ...
  - geht vom Startsymbol aus,
  - wendet Regel von links nach rechts an,
  - und gleicht Terminalsymbole mit der Eingabekette ab.



11

# Elementare Parsing-Strategien

- Ein **Bottom-Up** Algorithmus ...
  - geht von der Eingabekette aus
  - wendet Regeln „rückwärts“ an, wenn die rechte Seite der Regel auf die Eingabe passt,
  - bis das Startsymbol abgeleitet wurde.



12

## Shift-Reduce Parsing (Bottom-up)

- **Initiale Konfiguration** für Eingabekette  $w_1 \dots w_n$ :
  - $\langle [], [w_1, \dots, w_n] \rangle$
- **Akzeptierende Konfiguration:**
  - $\langle [S], [] \rangle$
- In jedem Schritt kann eine Shift- oder Reduce-Operation ausgeführt werden (nichtdeterministisch)
  - **Shift:** verschiebe ein Wort auf den Stapel (Stack)
  - **Reduce:** wende eine passende Regel auf die obersten Elemente des Stapels an.

13

## Shift

- Shift verschiebt ein Symbol auf den Stapel
- **Konfiguration:**
  - $\langle [A_1, \dots, A_k], [w_i, w_{i+1}, \dots, w_n] \rangle$
- **Neue Konfiguration:**
  - $\langle [A_1, \dots, A_k, w_i], [w_{i+1}, \dots, w_n] \rangle$

14

## Reduce

- Reduce ersetzt die obersten Symbole des Stapels durch die linke Seite einer passenden Regel.
- **Konfiguration:**
  - $\langle [A_1, \dots, A_{j-1}, A_j, \dots, A_k], [w_i, \dots, w_n] \rangle$
- **Regel:**
  - $B \rightarrow A_j, \dots, A_k$
- **Neue Konfiguration:**
  - $\langle [A_1, \dots, A_{j-1}, B], [w_i, \dots, w_n] \rangle$

15

## Nichtdeterminismus

- Wie entscheiden wir, ob eine Shift oder eine Reduce-Operation angewendet werden soll?
  - Für bestimmte (eindeutige) Grammatiken: vorberechnete Aktions/Sprung-Tabellen.
  - Im Allgemeinen Fall: Suche & Backtracking

16



## Implementierung in Python

```
rules = [ ( 'S', ['NP', 'VP']), ('NP', ['D', 'N']), ... ]
def shift(stack, sntnc):
    return ([sntnc[0]] + stack, sntnc[1:])
def rdce(stack, sntnc, lhs, rhs):
    return ([lhs] + stack[len(rhs):], sntnc)
def matches(stack, rhs):
    if len(stack) < len(rhs):
        return False
    for (s, r) in zip(stack, reversed(rhs)):
        if s != r:
            return False
    return True
```

17

## Implementierung in Python

```
def recognize(sntnc):
    agenda = [( [], sntnc)]
    while agenda:
        (stack, sntnc) = agenda.pop()
        if sntnc == [] and stack == ['S']:
            return True
        if sntnc != []:
            agenda.append(shift(stack, sntnc))
        for (lhs, rhs) in rules:
            if matches(stack, rhs):
                agenda.append(rdce(stack, sntnc, lhs, rhs))
    return False
```

18

## Problematische Regeln

- Reine Bottom-up Verfahren können nicht ohne Weiteres mit bestimmten Grammatiken verwendet werden.
  - Parser terminiert ggf. nicht
- Regeln der Form  $A \rightarrow \varepsilon$ 
  - $\langle [A_1, \dots, A_k], [w_i, \dots, w_n] \rangle$
  - $\langle [A_1, \dots, A_k, A], [w_i, \dots, w_n] \rangle$  (reduce)
  - $\langle [A_1, \dots, A_k, A, A], [w_i, \dots, w_n] \rangle$  (reduce)
  - [...]

19

## Problematische Regeln

- Reine Bottom-up Verfahren können nicht ohne Weiteres mit bestimmten Grammatiken verwendet werden.
  - Parser terminiert ggf. nicht
- Zyklische Regeln:  $A \rightarrow B, B \rightarrow A$ 
  - $\langle [A_1, \dots, A_k, A], [w_i, \dots, w_n] \rangle$
  - $\langle [A_1, \dots, A_k, B], [w_i, \dots, w_n] \rangle$  (reduce)
  - $\langle [A_1, \dots, A_k, A], [w_i, \dots, w_n] \rangle$  (reduce)
  - $\langle [A_1, \dots, A_k, B], [w_i, \dots, w_n] \rangle$  (reduce)
  - [...]

20

# Top-Down Parsing

- **Initiale Konfiguration** für Eingabekette  $w_1 \dots w_n$ 
  - $\langle [S], [w_1, \dots, w_n] \rangle$
- **Akzeptierende Konfiguration:**
  - $\langle [], [] \rangle$
- Operationen:
  - **Predict:** Anwendung einer Regel auf das linkeste Nicht-terminalsymbol.
  - **Scan:** Eingabe konsumieren

21

# Top-Down Parsing: Predict

- **Konfiguration:**
  - $\langle [A_1, A_2, \dots, A_k], [w_i, \dots, w_n] \rangle$
- **Regel:**
  - $A_1 \rightarrow B_1 \dots B_m$
- **Neue Konfiguration:**
  - $\langle [B_1, \dots, B_m, A_2, \dots, A_k], [w_i, \dots, w_n] \rangle$

22

# Top-Down Parsing: Scan

- **Konfiguration:**
  - $\langle [w_i, A_1, \dots, A_k], [w_i, w_{i+1}, \dots, w_n] \rangle$
- **Neue Konfiguration:**
  - $\langle [A_1, \dots, A_k], [w_{i+1}, \dots, w_n] \rangle$

23

# Problematische Regeln (Top-Down)

- Links- bzw. Rechtsrekursive Regeln – abhängig davon, ob die Eingabe von rechts nach links oder umgekehrt abgearbeitet wird.
  - $NP \rightarrow DET N, DET \rightarrow NP 's$
  - $S \rightarrow S CONJ S$

24

# Übungsaufgabe 1

- Implementiere eine Klasse für Parsebäume ...
- und erweitere den Beispielcode des Shift-Reduce Erkenners zu einem Parser.
- Hinweis: Speicher Parsebäume statt Nichtterminalsymbole auf dem Stack.

25

# Übungsaufgabe 2

- Implementiere den Shift-Reduce Parser als Iterator, der alle Ableitungsbäume aufzählt
- ```
for tree in parse(['der', 'student', ...]):  
    dosomething(tree)
```

26

## Übungsaufgabe 3

- Implementiere einen Top-Down Erkenner.
- Beachte: Es ist sinnvoll, zwischen „eentlichen“ Regeln und Lexikoneinträgen ( $A \rightarrow a$ ) zu unterscheiden. Warum?
  - Der Algorithmus muss dann leicht modifiziert werden.
- Bonus: Parser statt Erkenner (knifflig)