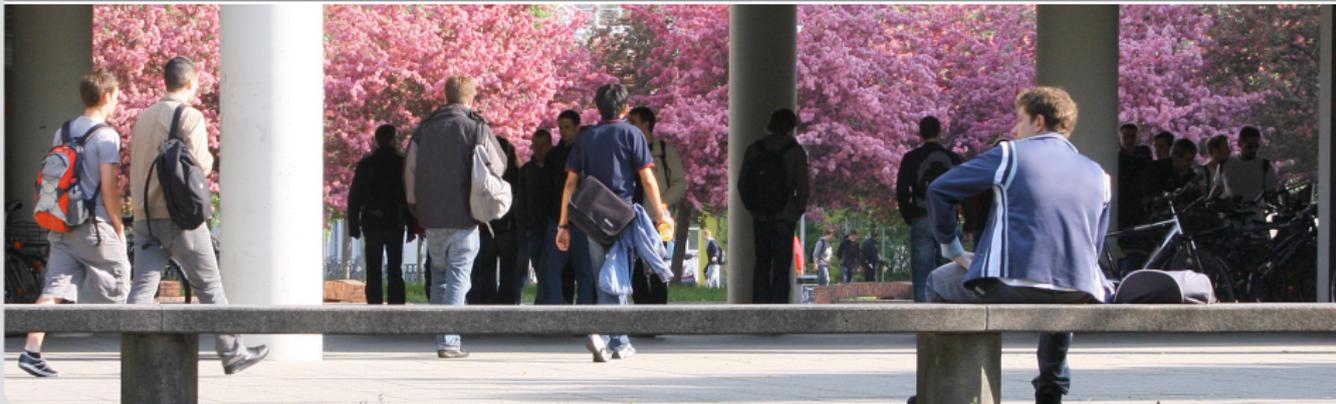


# Theoretische Grundlagen der Informatik

Vorlesung am 11.01.2011

INSTITUT FÜR THEORETISCHE INFORMATIK



# Thema dieses Kapitels

- Grammatiken sind Regelsysteme, mit denen sich die Wörter einer vorgegebenen Sprache erzeugen lassen

# Beispiel 1

Die Sprache aller Graphen  $G = (V, E)$ , die eine Clique der Größe  $\frac{|V|}{2}$  enthalten, lässt sich aufbauen durch:

- 1 Wahl der Zahl  $n$  für  $|V|$
- 2 Wahl einer Teilmenge der Größe  $\frac{|V|}{2}$
- 3 Zugabe aller Kanten zwischen Knoten aus dieser Teilmenge
- 4 Zugabe weiterer Kanten

Dieses Regelsystem ist an drei Stellen nichtdeterministisch: 1, 2 und 4.

## Beispiel 2 - Arithmetische Ausdrücke

- $a$ ,  $a + a$  und  $a \cdot a$  sind arithmetische Ausdrücke ( $a$  Symbol aus Alphabet).
- falls  $A_1$  und  $A_2$  arithmetische Ausdrücke sind, so sind auch  $(A_1) + (A_2)$  und  $(A_1) \cdot (A_2)$  arithmetische Ausdrücke.

Die Klammern sind teilweise überflüssig.

Eine **Grammatik**  $G = (\Sigma, V, S, R)$  besteht aus vier Komponenten:

- Endliches **Alphabet**  $\Sigma$  (auch Terminalalphabet genannt);
- Endliche Menge  $V$  mit  $V \cap \Sigma = \emptyset$  von **Variablen** (Nichtterminale)
- **Startsymbol**  $S \in V$ ;
- Endliche Menge von **Ableitungsregeln**  $R$  (Produktionen).

Dabei ist eine Ableitungsregel ein Paar  $(\ell, r)$ , wobei

- $\ell \in (V \cup \Sigma)^+$
- $r \in (V \cup \Sigma)^*$ .

Wir schreiben oft auch  $\ell \rightarrow r$ .

## Bedeutung

- Wenn in einem Wort  $z$  das Wort  $\ell$  Teilwort von  $z$  ist, so darf  $\ell$  durch  $r$  in  $z$  ersetzt werden.

## Notation

- Wir schreiben  $w \rightarrow z$ , wenn  $w$  durch Anwendung einer Ableitungsregel in  $z$  verwandelt wird
- Wir schreiben  $w \xrightarrow{*} z$ , wenn  $w$  durch eine Anwendung von mehreren Ableitungsregeln in  $z$  verwandelt wird.

## Erzeugte Sprache

- Die von einer Grammatik  $G$  **erzeugte Sprache**  $L(G)$  ist die Menge aller Wörter  $z \in \Sigma^*$ , für die  $S \xrightarrow{*} z$  gilt.

# Beispiel

Grammatik für die Menge aller arithmetischen Ausdrücke über  $a$ .

$$\Sigma = \{ (, ), a, +, \cdot \}$$

$$V = \{ S \}$$

$$R : S \rightarrow (S) + (S)$$

$$S \rightarrow (S) \cdot (S)$$

$$S \rightarrow a$$

$$S \rightarrow a + a$$

$$S \rightarrow a \cdot a$$

- **Typ-0:** Grammatiken ohne weitere Einschränkungen.
- **Typ-1/kontextsensitiv:** Grammatiken, ausschließlich mit Ableitungsregeln der Form
  - $u \rightarrow v$  mit  $u \in V^+$ ,  $v \in ((V \cup \Sigma) \setminus \{S\})^+$  und  $|u| \leq |v|$ , oder
  - $S \rightarrow \varepsilon$
- **Typ-2/kontextfrei:** Grammatiken, ausschließlich mit Ableitungsregeln der Form

$$A \rightarrow v \quad \text{mit } A \in V \text{ und } v \in (V \cup \Sigma)^*$$

- **Typ-1/kontextsensitiv:** Grammatiken, ausschließlich mit Ableitungsregeln der Form
  - $u \rightarrow v$  mit  $u \in V^+$ ,  $v \in ((V \cup \Sigma) \setminus \{S\})^+$  und  $|u| \leq |v|$ , oder
  - $S \rightarrow \varepsilon$
- **Typ-2/kontextfrei:** Grammatiken, ausschließlich mit Ableitungsregeln der Form

$$A \rightarrow v \quad \text{mit } A \in V \text{ und } v \in (V \cup \Sigma)^*$$

**Bemerkung:** Bei kontextsensitiven Grammatiken kann die Ableitung

$$ABC \rightarrow AXYC$$

erlaubt, aber

$$DBC \rightarrow DXYC$$

verboten sein.

# Chomsky-0 Grammatiken und Semientscheidbarkeit

- **Typ-0:** Grammatiken ohne weitere Einschränkungen.

## **Satz:**

Falls  $L$  rekursiv aufzählbar (semi-entscheidbar) ist, so gibt es eine Chomsky-0-Grammatik mit  $L(G) = L$ .

## Satz:

Falls  $L$  rekursiv aufzählbar (semi-entscheidbar) ist, so gibt es eine Chomsky-0-Grammatik mit  $L(G) = L$ .

## Beweis:

- $L$  rekursiv aufzählbar  $\Rightarrow \exists$  DTM  $\mathcal{M}$ , die  $L$  akzeptiert.
- O.B.d.A. habe  $\mathcal{M}$  genau einen akzeptierenden Endzustand  $q_f$
- O.B.d.A. stehen auf dem Band nur Blanks wenn  $q_f$  erreicht wird
- $q_0$  komme nur in der Anfangskonfiguration vor

Wir konstruieren eine Grammatik  $G = (\Sigma, V = \Gamma \cup Q \cup \{S\}, S, R)$ , die die Berechnung von  $\mathcal{M} = (Q, \Sigma, \Gamma, s, \delta, F)$  rückwärts durchläuft.

## ■ Rückwärtsrechnung

Falls  $\delta(q, a) = (q', a', R)$ , dann enthält  $G$  die Ableitungsregel

$$a'q' \rightarrow qa$$

Falls  $\delta(q, a) = (q', a', L)$ , dann enthält  $G$  die Ableitungsregel

$$q'ba' \rightarrow bqa \quad \text{für alle } b \in \Gamma$$

Falls  $\delta(q, a) = (q', a', N)$ , dann enthält  $G$  die Ableitungsregel

$$q'a' \rightarrow qa$$

- $G = (\Sigma, V, S, R)$  mit  $\Sigma = \Gamma$  und  $V = \{q_J\}$
- **Erzeugung der akzeptierenden Schlusskonfiguration**

$$S \rightarrow q_J, \quad q_J \rightarrow \sqcup q_J, \quad q_J \rightarrow q_J \sqcup$$

- **Rückwärtsrechnung**

Falls  $\delta(q, a) = (q', a', R)$ , dann enthält  $G$  die Ableitungsregel

$$a'q' \rightarrow qa$$

Falls  $\delta(q, a) = (q', a', L)$ , dann enthält  $G$  die Ableitungsregel

$$q'ba' \rightarrow bqa \quad \text{für alle } b \in \Gamma$$

Falls  $\delta(q, a) = (q', a', N)$ , dann enthält  $G$  die Ableitungsregel

$$q'a' \rightarrow qa$$

## ■ Rückwärtsrechnung

Falls  $\delta(q, a) = (q', a', R)$ , dann enthält  $G$  die Ableitungsregel

$$a'q' \rightarrow qa$$

Falls  $\delta(q, a) = (q', a', L)$ , dann enthält  $G$  die Ableitungsregel

$$q'ba' \rightarrow bqa \quad \text{für alle } b \in \Gamma$$

Falls  $\delta(q, a) = (q', a', N)$ , dann enthält  $G$  die Ableitungsregel

$$q'a' \rightarrow qa$$

## ■ Schlussregeln

Transformiere  $\sqcup \sqcup \dots \sqcup q_0 w_1 \dots w_n \sqcup \dots \sqcup$  zu  $w_1 \dots w_n$

$$\sqcup q_0 \rightarrow q_0, \quad q_0 a \rightarrow a q_0, \quad q_0 \sqcup \rightarrow q_0, \quad q_0 \rightarrow \varepsilon$$

- Alle Wörter  $w \in L$  können durch  $G$  erzeugt werden, indem die Berechnung von  $\mathcal{M}$  rückwärts durchlaufen wird.
- Umgekehrt kann  $G$  nur Berechnungen von  $\mathcal{M}$  rückwärts erzeugen.
- Daher ist  $L(G) = L$ .

# Chomsky-0 Grammatiken und Semientscheidbarkeit

- **Typ-0:** Grammatiken ohne weitere Einschränkungen.

**Satz:**

Die von Typ-0-Grammatiken  $G$  erzeugten Sprachen sind rekursiv aufzählbar.

**Satz:**

Die von Typ-0-Grammatiken  $G$  erzeugten Sprachen sind rekursiv aufzählbar.

**Beweis:** Wir konstruieren zunächst eine NTM  $\mathcal{M}$ , die  $L(G)$  akzeptiert:

- 1 Schreibe  $S$  auf das Band
- 2 Wähle eine beliebige anwendbare Ableitungsregel aus
- 3 Vergleiche das erzeugte Wort mit der Eingabe  $w$
- 4 Bei Gleichheit wird  $w$  akzeptiert
- 5 Ansonsten springe zu Punkt 2

Falls  $w \in L(G)$ , so gibt es eine akzeptierende Berechnung.

**Satz:**

Die von Typ-0-Grammatiken  $G$  erzeugten Sprachen sind rekursiv aufzählbar.

**Beweis:**

Gemäß Übungsaufgabe 2, Blatt 4, kann aus der NTM eine DTM konstruiert werden, die dasselbe leistet.

Die Klasse der rekursiv aufzählbaren Sprachen ist genau

- die Klasse der von DTMs akzeptierten Sprachen
- die Klasse der von NTMs akzeptierten Sprachen
- die Klasse der von Typ-0-Grammatiken erzeugten Sprachen

### Bemerkung

- Das Wortproblem für Typ-0-Grammatiken ist unentscheidbar.
- Als Grundlage für Programmiersprachen sind die Typ-0-Grammatiken also sicherlich zu allgemein.

# Chomsky–3–Grammatiken und reguläre Sprachen

- **Typ–3/rechtslinear:** Grammatiken, ausschließlich mit Ableitungsregeln der Form

$$A \rightarrow v \quad \text{mit } A \in V \text{ und } v = \varepsilon \text{ oder } v = aB \text{ mit } a \in \Sigma, B \in V$$

## **Satz:**

Die Klasse der von endlichen Automaten akzeptierten Sprachen ist genau die Klasse der von Chomsky–3–Grammatiken erzeugten Sprachen.

## Beweis

$\Rightarrow$  Zu einem DEA  $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ , gibt es eine Typ-3 Grammatik  $G_L$ , die  $L(\mathcal{A}_L)$  erzeugt.

$\Rightarrow$  Zu einem DEA  $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ , gibt es eine Typ-3 Grammatik  $G_L$ , die  $L(\mathcal{A}_L)$  erzeugt.

$G_L$  sei definiert durch:

- $V := Q$ ;
- $S := q_0$ ;
- $R$  enthält die Regel
  - $q \rightarrow \varepsilon$  für alle  $q \in F$
  - $q \rightarrow aq'$ , falls  $\delta(q, a) = q'$ .

$\Rightarrow$  Zu einem DEA  $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ , gibt es eine Typ-3 Grammatik  $G_L$ , die  $L(\mathcal{A}_L)$  erzeugt.

$G_L$  sei definiert durch:

- $V := Q$ ;
- $S := q_0$ ;
- $R$  enthält die Regel
  - $q \rightarrow \varepsilon$  für alle  $q \in F$
  - $q \rightarrow aq'$ , falls  $\delta(q, a) = q'$ .

Für  $w = w_1 \dots w_n$  durchlaufe  $\mathcal{A}_L$  die Zustände  $q_0, \dots, q_n \in Q$  mit  $q_n \in F$ .  
Dann gilt:

$$q_0 \rightarrow w_1 q_1 \rightarrow w_1 w_2 q_2 \rightarrow \dots \rightarrow w q_n \rightarrow w$$

⇒ Zu einem DEA  $\mathcal{A}_L = (Q, \Sigma, \delta, q_0, F)$ , gibt es eine Typ-3 Grammatik  $G_L$ , die  $L(\mathcal{A}_L)$  erzeugt.

$G_L$  sei definiert durch:

- $V := Q$ ;
- $S := q_0$ ;
- $R$  enthält die Regel
  - $q \rightarrow \varepsilon$  für alle  $q \in F$
  - $q \rightarrow aq'$ , falls  $\delta(q, a) = q'$ .

Außerdem gibt es für alle Ableitungen von  $G_L$  eine entsprechende akzeptierende Berechnung des endlichen Automaten  $\mathcal{A}_L$ .

## Beweis

⇐ Zu einer Typ-3 Grammatik  $G_L$  gibt es einen NEA  $\mathcal{A}_L := (Q, \Sigma, \delta, q_0, F)$ , der  $L(G_L)$  akzeptiert.

## Beweis

⇐ Zu einer Typ-3 Grammatik  $G_L$  gibt es einen NEA  $\mathcal{A}_L := (Q, \Sigma, \delta, q_0, F)$ , der  $L(G_L)$  akzeptiert.

- $Q := V$ ;
- $q_0 := S$ ;
- $F := \{A \in V \mid (A \rightarrow \varepsilon) \in R\}$
- $\delta(A, a) := \{B \mid (A \rightarrow aB) \in R\}$ .

⇐ Zu einer Typ-3 Grammatik  $G_L$  gibt es einen NEA  $\mathcal{A}_L := (Q, \Sigma, \delta, q_0, F)$ , der  $L(G_L)$  akzeptiert.

- $Q := V$ ;
- $q_0 := S$ ;
- $F := \{A \in V \mid (A \rightarrow \varepsilon) \in R\}$
- $\delta(A, a) := \{B \mid (A \rightarrow aB) \in R\}$ .

Für  $w = w_1 \dots w_n \in L$  hat die Ableitung von  $w$  mittels  $G_L$  das Aussehen:

$$S \rightarrow w_1 A_1 \rightarrow w_1 w_2 A_2 \rightarrow \dots \rightarrow w A_n \rightarrow w$$

$\mathcal{A}_L$  kann bei Eingabe  $w = w_1 \dots w_n$  folgende Abarbeitung durchlaufen:

$$S \xrightarrow{w_1} A_1 \xrightarrow{w_2} A_2 \xrightarrow{w_3} \dots \xrightarrow{w_{n-1}} A_{n-1} \xrightarrow{w_n} A_n,$$

wobei  $A_n \in F$ , also  $w$  akzeptiert wird.

## Beweis

⇐ Zu einer Typ-3 Grammatik  $G_L$  gibt es einen NEA  $\mathcal{A}_L := (Q, \Sigma, \delta, q_0, F)$ , der  $L(G_L)$  akzeptiert.

- $Q := V$ ;
- $q_0 := S$ ;
- $F := \{A \in V \mid (A \rightarrow \varepsilon) \in R\}$
- $\delta(A, a) := \{B \mid (A \rightarrow aB) \in R\}$ .

Außerdem gibt es für alle akzeptierenden Berechnungen von  $\mathcal{A}_L$  eine entsprechende Ableitung in  $G_L$ .

- Wir wissen, dass die Sprache der korrekten Klammersausdrücke nicht regulär ist.
- Typ-3-Grammatiken sind also zu einschränkend, um syntaktisch korrekte Programme zu beschreiben.

**Fazit:** Programmiersprachen sollten echt zwischen Typ-0 und Typ-3 angesiedelt sein.

# Chomsky-1-Grammatiken bzw. kontextsensitive Sprachen

## Annahme:

- Die Eingabe einer Turingmaschine steht auf einem separaten Read-Only-Band.
- Der Speicherbedarf einer Turingmaschine wird nur anhand des Arbeitsbandes gemessen.

- $DTAPE(s(n))$  ist die Klasse der Sprachen  $L$ , für die es eine **DTM** mit Platzbedarf  $s(n)$  (bei Eingabelänge  $n$ ) gibt, die  $L$  akzeptiert.
- $NTAPE(s(n))$  ist die Klasse der Sprachen  $L$ , für die es eine **NTM** mit Platzbedarf  $s(n)$  (bei Eingabelänge  $n$ ) gibt, die  $L$  akzeptiert.

# Chomsky-1-Grammatiken bzw. kontextsensitive Sprachen

- $DTAPE(s(n))$  ist die Klasse der Sprachen  $L$ , für die es eine **DTM** mit Platzbedarf  $s(n)$  (bei Eingabelänge  $n$ ) gibt, die  $L$  akzeptiert.
- $NTAPE(s(n))$  ist die Klasse der Sprachen  $L$ , für die es eine **NTM** mit Platzbedarf  $s(n)$  (bei Eingabelänge  $n$ ) gibt, die  $L$  akzeptiert.

Offensichtlich gilt

$$DTAPE(s(n)) \subseteq NTAPE(s(n))$$

Außerdem gilt

$$NTAPE(n) = NTAPE(f(n))$$

für alle  $f(n) \in \theta(n)$ .

**Satz:**

Die Klasse der von Chomsky-1-Grammatiken erzeugten Sprachen stimmt mit der Klasse  $\mathcal{N}\mathcal{T}\mathcal{A}\mathcal{P}\mathcal{E}(n)$  überein.

Ohne Beweis.

## Bemerkung

Es ist offen, ob  $\mathcal{N}TAP\mathcal{E}(n) = \mathcal{D}TAP\mathcal{E}(n)$  ist.

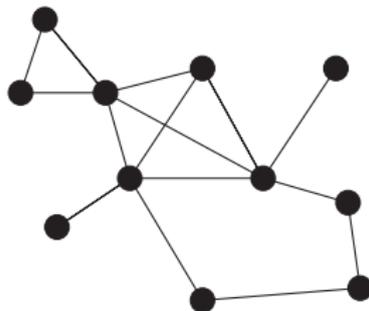
# Wiederholung: Das Problem CLIQUE

Eine **Clique** in einem Graphen  $G = (V, E)$  ist eine Menge  $V' \subseteq V$  so, dass für alle  $i, j \in V', i \neq j$ , gilt:  $\{i, j\} \in E$ .

## Problem CLIQUE

**Gegeben:** Graph  $G = (V, E)$  und ein Parameter  $K \leq |V|$

**Frage:** Gibt es in  $G$  eine Clique der Größe mindestens  $K$ ?



**Satz:**

Das Cliques-Problem gehört zu  $\mathcal{DTAPE}(n)$ .

- Gegeben sei  $G = (V, E)$  mit  $V = \{1, \dots, n\}$  und  $1 \leq K \leq n$ .
- Benutze  $n$ -Vektor  $c \in \{0, 1\}^n$  mit der Bedeutung für eine Menge  $C$

$$c_i = 1 \iff i \in C$$

- Teste für jeden Vektor  $c \in \{0, 1\}^n$ , ob die zugehörige Menge  $C \subseteq V$  eine Clique der Größe  $K$  in  $G$  ist:
  - Zähle die Einsen in  $c$ .
  - Überprüfe für jedes Paar  $c_i, c_j$  mit  $c_i = c_j = 1$  ob  $\{i, j\} \in E$ .

**Satz:**

Das Cliques-Problem gehört zu  $DTAPE(n)$ .

Alle Vektoren  $c \in \{0, 1\}^n$  können nacheinander, beginnend mit  $(0, 0, \dots, 0)$ , durchgetestet werden, wobei:

- nach einem positiven Test  $(G, K)$  akzeptiert wird;
- nach einem negativen Test der Vektor durch seinen lexikalischen Nachfolger überschrieben wird.

Dazu wird insgesamt nur linearer Speicherplatz benötigt.

## Bemerkung 1

Falls  $\mathcal{P} \neq \mathcal{NP}$ , kann das Wortproblem für kontextsensitive Grammatiken im allgemeinen nicht in polynomialer Zeit entschieden werden.

## Bemerkung 2

Für jede Chomsky-1-Grammatik gibt es eine äquivalente Chomsky-1-Grammatik, bei der alle Regeln folgende Form haben:

- $A \rightarrow C$
- $A \rightarrow CD$
- $AB \rightarrow CD$
- $A \rightarrow a$
- $S \rightarrow \varepsilon$

wobei jeweils  $A, B \in V$ ,  $C, D \in V \setminus \{S\}$  und  $a \in \Sigma$ .

## Notation

Statt Regeln

$$S \rightarrow \alpha \text{ und } S \rightarrow \beta$$

schreiben wir abkürzend

$$S \rightarrow \alpha \mid \beta$$