Seminar Mustererkennung mit syntaktischen und graphenbasierten Methoden

---

Thema:

Herleitung von Grammatiken aus Mustermengen

Thomas Böttcher
2204636
Informatik dipl.

Inhalt

I. Herleiten von regulären Grammatiken aus Mustermengen

II. Erzeugung von kontextfreien Grammatiken

III. Herleiten von Baumgrammatiken aus Mustermengen

I.
Reguläre Grammatiken

Erzeugung von reg. Grammatiken durch:

I.1 Die Hilfe eines „Teacher“

I.2 Generierung von kanonischen Grammatiken

I.3 Erzeugung von endl. Automaten

Erzeugung von reg. Grammatiken
durch einen „Teacher“

Vorab Definieren wir 2 Mengen

$R^+ \subseteq \{\alpha \mid \alpha \in L(G)\}$

$R^- \subseteq \{\alpha \mid \alpha \in \overline{L}(G)\}$

R ist meistens endliche Mustermenge!

Geg. Eine Menge R^+ einer unbekannten reg. Grammatik

Schritt 1:

1. Lösche für jeden String *x* aus R^+ mögliche Teilstrings s
   Mögl. Bedeutet hier nicht beliebige Teilstrings
   Für den reg. Fall nur zusammenhängende Strings

2. Teste mit Hilfe des „Teacher“ den Reststring x’ auf Gültigkeit

3. Ist *x*' eine gültige Folge so wird *s* k-mal eingefügt
   Test ob $(x'' = x_1..x_k(s)^j x_l..x_z ; j \leq k)$ gültige Sequenz ist

4. Erfolgreich -> Erkennung eines Zyklus

Erzeugung von reg. Grammatiken
durch einen „Teacher“

Ein einfaches Beispiel:

Sei x = abc ein gültiger String

Dann sind

x' = bc	-	s = a
x' = ac	-	s = b
x' = ab	...	
x' = a		
x' = c	-	s = ab

mögliche Kandidaten

Jetzt wird der „Teacher“ gefragt welche Teilstrings x' gültig sind

Sei bc der einzige gültige Teilstring

Test von $(a)^j bc,\ j \leq k$ (aabc,aaabc,....)
auf Gültigkeit

Erzeugung von reg. Grammatiken
durch einen „Teacher“

bis jetzt – jeden String aus R^+ auf Zyklen untersucht

Schritt 2:

- Def.: Sei $R^+ = R_c^+ \bigcup R_0^+$
  R_c^+ - Strings mit erkannten Zyklen
  R_0^+ - Strings ohne Zyklen

- Konstruktion einer Grammatik für R_c^+
  am Bsp. ab((c)(d))
  cd – Zyklus mit Unterzyklen c und d

  S -> aA1
  A1 -> bA2 | bA3 | b
  A2 -> cA2 | cA3 | c
  A3 -> dA3 | dA2 | d
  ==> Produktionen der Grammatik
  (1. nicht zyklen;2. Für Zyklen/Subzyklen)

- analog für komplexe Zyklen

Erzeugung von reg. Grammatiken
durch einen „Teacher“

Schritt 3:

- Abschließend Produktionen für R_0^+ (falls nötig)
- formale Grammatik + Vereinfachung

Algorithmus im Überblick

1. Suche nach Zyklen u. Teilzyklen mit Hilfe des Teachers -> bilden von R_0^+ R_c^+
2. Erstellen der Produktionen für R_c^+
3. Produktionen für R_0^+ (falls nötig)
4. Grammatik erstellen und vereinfachen

Erzeugung von reg. Grammatiken durch Generierung kanonischer Grammatiken

Strategie:

1. Erstellen einer kanonischen Grammatik

2. Mischen der Nonterminals um eine rekursive Grammatik zu erstellen

3. Minimierung d. Grammatik

Erzeugung von reg. Grammatiken durch Generierung kanonischer Grammatiken

Schritt 1:

- Ordnen der Strings von R^+ nach Länge

- Erzeugung von Produktionen jedes Strings (Abarbeitung bzgl. der Länge)

  => kanonische Grammatik die genau R^+ erzeugt

Schritt 2:

- Mischen von Nonterminals für iterative Grammatik (zur Erzeugung von Zyklen)

- Suchen von Regeln der Form
  A->ab u. B-> cA $a, b, c \in \Sigma,\ A, B \in N$

- Ersetze alle A´s durch B und lösche A->ab

- neue Regel(n):
  B-> cB
  (alle A´s ersetzen z.B.auch A->b durch B->b)

Schritt 3:

- Aufstellen u. minimieren der Grammatik

Ein Beispiel:

Sei $R^+ = \{caaab, bbaab, caab, bbab, cab, bbb, cb\}$

Erstellung der kanonischen Grammatik:

caaab:

S -> c A_1 | A_2 -> a A_3

A_1 -> a A_2 | A_3 -> ab (max. Länge=2)

bbaab:

S -> b A_4 | A_5 -> a A_6

A_4 -> b A_5 | A_6 -> ab

caab:

A_3 -> b

…

…

Erzeugung von reg. Grammatiken durch Generierung kanonischer Grammatiken

S -> c A_1 | b A_4
A_1 -> a A_2 | b
A_2 -> a A_3 | b
A_3 -> ab | b

A_4 -> b A_5
A_5 -> a A_6 | b
A_6 -> ab | b

Mischen der Endproduktionen(mit Länge2)

A_2 -> a A_3
A_3 -> ab
A_3 -> b

A_5 -> a A_6
A_6 -> ab
A_6 -> b

Ersetze alle Nonterminals A_3 und A_6
und löschen der Regeln A_3 -> ab u. A_6 -> ab

A_2 -> a A_2
A_2 -> b
A_5 -> b

A_5 -> a A_5

Minimierung

S -> c A | b B
A -> a A | b
B -> b A

Erzeugung von reg. Grammatiken
mittels endl. Automaten

- vorab Definitionen

Sei $R^+ \subseteq \Sigma^*$ und $z \in \Sigma^*$ s.d. $zw \in R^+, w \in \Sigma^*$

Weiterhin Sei $h(z, R^+, k) = \{w \mid zw \in R, \mid w \mid \leq k\}$

d.h. das k-Endstück von z ist eine Menge von Worten w mit den Eigenschaften $zw \in R$ und $\mid w \mid \leq k$

Beispiel(für k=1):

$R^+ = \{a, b, cde\}$

$h(\varepsilon, R^+, 1) = \{w \mid \varepsilon w \in R^+, \mid w \mid \leq 1\}$

$= \{a, b\} \quad - \quad (w = a \vee w = b) \wedge \mid w \mid \leq 1$

$h(cde, R^+, 1) = \{\varepsilon\} \quad - \quad w = \varepsilon \wedge \mid w \mid \leq 1$

Erzeugung von reg. Grammatiken
mittels endl. Automaten

Schritt 1: Finiter Automat $\mathrm{M} = (Q, \Sigma, \delta, q_o, F)$

- Q - Zustandsmenge: $Q = \{q \mid q = h(z, R^+, k), z \in \Sigma^*\}$

- Übergangsfunktion δ:
  $\delta(q,a) = \{\, q' \in Q \mid q' = h(za, R^+, k), mit\, q = h(z, R^+, k)\,\}$

- Startzustand $q_o = h(\varepsilon, R^+, k)$

- Endzustände F – Alle Zustände in deren
  Repräsentationsmenge
  $q = h(z, R^+, k) = \{w \mid zw \in R, \mid w \mid \leq k\}$ ε enthalten ist

Schritt 2: Aufstellen und minimieren von M

Schritt 3: Transformation in eine reg. Grammatik

Erzeugung von reg. Grammatiken
mittels endl. Automaten

Beispiel: $R^+ = \{a, ab, abb\}$ und sei k=1

Dann ist $Q = \{q_0, q_1, q_2, q_m\}$

$z = \varepsilon \quad h(\varepsilon, R^+, 1) = \{a\} = q_0$

$z = a \quad h(a, R^+, 1) = \{\varepsilon, b\} = q_1$

$z = ab \quad h(ab, R^+, 1) = \{\varepsilon, b\} = q_1$

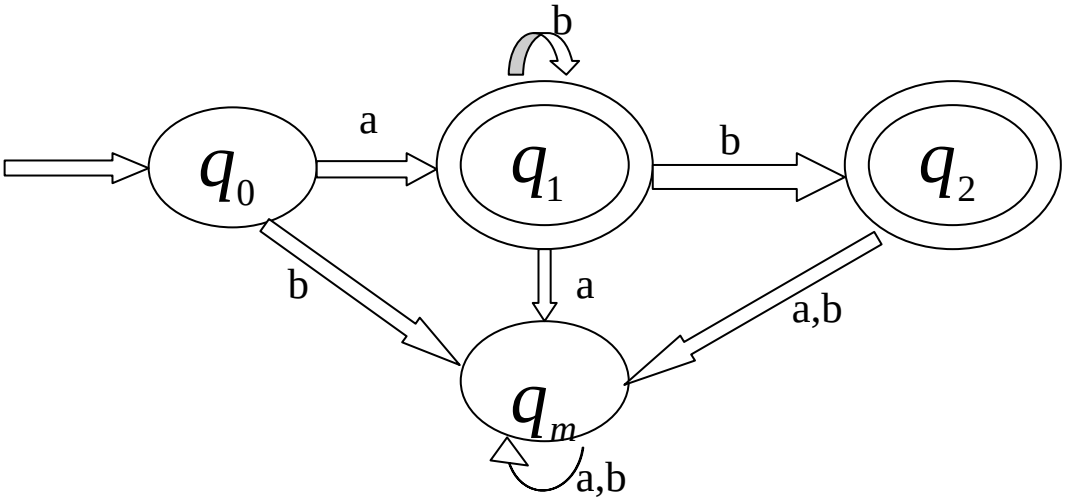
$z = abb \quad h(abb, R^+, 1) = \{\varepsilon\} = q_2$

Transitionen:

$\delta(q,a) = \{\, q' \in Q \mid q' = h(za, R^+, k), mit\, q = h(z, R^+, k)\,\}$

$\delta(q_0, a) = h(\varepsilon a, R^+, 1) = q_1$
$\delta(q_1, a) = q_m$
$\delta(q_2, a) = q_m$
$\delta(q_m, a) = q_m$

$\delta(q_0, b) = q_m$
$\delta(q_1, b) = \{q_1, q_2\}$
$\delta(q_2, b) = q_m$!!
$\delta(q_m, b) = q_m$



II.
Kontextfreie Grammatiken(CFG)

Erzeugung von CFG´s durch:

II.1 Die Hilfe eines „Teacher“

II.2 Generierung von kanonischen Grammatiken

Erzeugung von kontextfreien Grammatiken
durch einen „Teacher“

Analog zum Beispiel für reg Grammatiken

Algorithmus:

1. Suche nach Zyklen u. Teilzyklen mit Hilfe
   des Teachers (einfache u. paarweise Zyklen)
   Bilden von R_0^+ und R_c^+

2. Analyse der Worte aus R_c^+
   (Suche nach weiteren (Teil-) Zyklen)

3. Erstellen der Produktionen für R_c^+

4. Produktionen für R_0^+ (falls nötig)

5. Grammatik erstellen und vereinfachen

Erzeugung von kontextfreien Grammatiken durch Generierung kanonischer Grammatiken

Algorithmus:

1. Konstruktion einer kanonischen CFG für jedes Wort aus R^+

2. Konstruktion von 2 Mengen $L_t(\alpha)\ und\ R_t(\alpha)$
   Für jede Produktion $A_j -> \alpha_j$

3. für jedes Paar Nonterminals A_i, A_j vergleiche
   $L_t(\alpha_i)\ mit\ L_t(\alpha_j)\ und\ R_t(\alpha_i)\ mit\ R_t(\alpha_j)$
   Ersetze A_i durch A_j bei Gleichheit

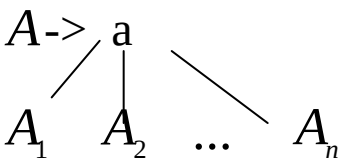
4. Erstelle CFG als Vereinigung aller Einzelgrammatiken
   Vereinfachung der CFG

III.
Baum-Grammatiken

Erzeugung von Baum-Grammatiken

Definitionen: $G = (V, r, P, S)$

Verwendung von Produktionen



$A, A_i \in N$

$a \in \Sigma$

Äquivalenz von Nonterminals:

$A_i \equiv A_j$ wenn $\mathrm{T}_i = \mathrm{T}_j$ T_i- die von A_i erzeugbaren Bäume

Entscheidbar wenn T_i *und* T_j endlich

Einführung eines Index(Gorn-Adressen)

Menge von Knoten $\{a_1, a_2, ..., a_n\}$ mit

Adresslänge k -> Ebene k

Seien $\mathrm{T}_i, \mathrm{T}_j$ 2 Bäume

$N_i(k)$ - Knoten auf Ebene k

$A(k)$ - Anz. d. Knoten auf Ebene k
Mit gleichem Index a. anderem Label

$B(k)$ - Distanz zwischen 2 Bäumen
$B(k) = (A(k) + | N_i(k) - N_j(k) |)$

Self-Embedding mit Tiefe D_S>0

2 Unterbäume $\mathrm{T}_i, \mathrm{T}_j$ von T mit den Wurzeln $\$_i, \$_j$ und $r(\$) > 1$ sind selbstenthaltend wenn

1) Index $\$_i$ ist Präfix des Index von $\$_j$ bzgl. T
2) $N_i(k) = N_j(k) \quad k = 0,1,...,D_S$
3) B(k)=0 $\quad k = 0,1,...,D_S$

Iterative Regelmäßigkeit mit Tiefe D_R>0

2 Unterbäume $\mathrm{T}_i, \mathrm{T}_j$ von T mit den Wurzeln $\$_i, \$_j$ und $r(\$) = 1$ sind iterativ Regulär wenn

1) Index $\$_i$ ist Präfix des Index von $\$_j$ bzgl. T
2) $N_i(k) = N_j(k) \quad k = 0,1,...,D_R$
3) B(k)=0 $\quad k = 0,1,...,D_R$
4) Keine Selbstenthaltung auf Ebene $k = 1,2,...,D_R$

Algorithmus:

Geg.: $\{T_1, T_2, ..., T_m\}$

1. Konstruktion von Produktionen für jeweils einen Baum der Mustermenge

   $A_{ij} -> a_{ij}$

   ... ...

2. Suche nach Selbstenthaltende bzw. iterativ Reguläre Tiefe $D_S(a_{ij})\ und\ D_R(a_{ij})$
   - erstes Vorkommen - A_{ij} -> Grad 1
   - nä. Vorkommen - A_{ij} -> Grad 2
   - kein Vorkommen - A_{ij} -> Grad 0

3. Teste $A_{ij} \equiv A_{ik}$, $A_{ij} \wedge A_{ik} mit\ Grad\ 1$
   Ggf. Mischen bei Gleichheit

4. Kombination von
   $A_{ij}\ mit\ A_{ik}$, $A_{ij}\ mit\ Grad\ 1 \wedge A_{ik} mit\ Grad\ 2$

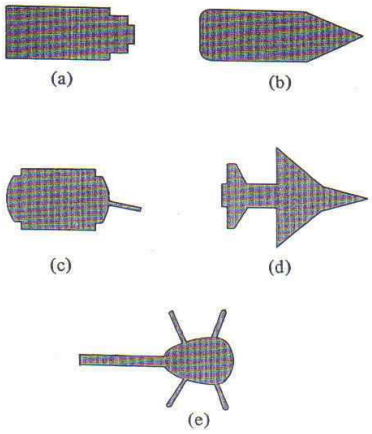
5. Teste $A_{ij} \equiv A_{ik}$, $A_{ij} \wedge A_{ik} mit\ Grad\ 0$
   Ggf. Mischen bei Gleichheit
   => Reduzierung der Grammatik

6. Ersetze das linke Nonterminal jeder Startproduktion A_i durch S

7. Aufstellen der Grammatik

Baumgrammatiken aus 2D – binär Bilder

(Verwendung des Algorithmus zur Minimierung von Baumgrammatiken)



Darstellung der Bilder als
128x128 binär Matrix

Verfahren: (direkt am Beispiel)

Schritt 1:

Scannen der Matrix -> Suche nach 1. Eins
-> Feld a_{22} -> Liste

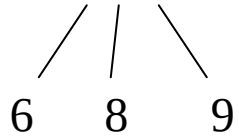
0	0	0	0	0	0
0	1	1	0	1	0
0	1	1	0	1	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	0	0	0	0

Schritt 2:

Suche nach Nachbarn mit
Wert 1 in Matrix(gemäß Liste)

1	2	3
4	X	6
7	8	9

a_{22} einziges Element d. Liste
Nachbarn 6,8,9
=> $



nä. Listenelement bearbeiten

a_{23} und a_{32} haben nur Nachbarn aus Liste

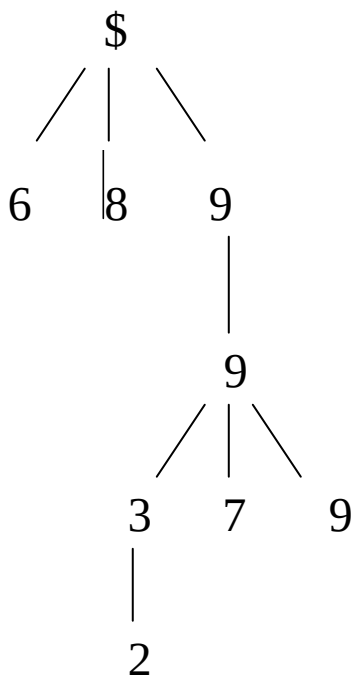
a_{33} enthält neue Nachbarn

…

…

0	0	0	0	0	0
0	1	1	0	1	0
0	1	1	0	1	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	0	0	0	0

Endgültiger Baum:



Schritt 3:

Erzeugung der Grammatik gemäß des vorgestellten Algorithmus

Quellenverzeichnis:

„Grammatikal Inference“ -
Rafael C. Gonzalez, Michael G. Thomason: Syntactic Pattern Recognition. An Introduction. Addison-Wesley, Reading etc. 1978. S. 216-258

Rafael C. Gonzalez, Michael G. Thomason: Syntactic Pattern Recognition. An Introduction. Addison-Wesley, Reading etc. 1978. S. 64-91