

Regulated Array Grammars of Finite Index

C. Gruber, J. Reiter

TU Wien

30.April, 2010

Table of contents

- 1 Basics
- 2 Kontrollmechanismen
- 3 Endliche Index Restriktion
- 4 Syntaktisches Pattern Recognition
 - Aspekte der syntaktischen Character Recognition
 - k-head finite array automata

n-dimensional array

Ein n-dimensionales Array A über ein Alphabet V (Menge aller non-terminal und terminal Symbole) ist eine Funktion

$$A : Z^n \rightarrow V \cup \{\#\} \quad n \in N = \{1, 2, \dots\}$$

wobei

$$\text{shape}(A) = \{v \in Z^n \mid A(v) \neq \#\}$$

endlich ist und $\# \notin V$ als *background* oder *blank Symbol* bezeichnet wird. Das Array A kann nun so definiert werden

$$A = \{(v, A(v)) \mid v \in \text{shape}(A)\}.$$

Der Vektor $(0, \dots, 0) \in Z^n$ wird als Ω_n bezeichnet.

n-dimensional array production and grammar

Eine n-dimensionale Array Produktion p über dem Alphabet V ist ein Tripel (W, A_1, A_2) wobei $W \subseteq Z^n$ eine endliche Menge von Koordinaten ist und A_1 und A_2 Abbildungen von W auf $V \cup \{\#\}$ sind.
 p kann als Λ -frei bezeichnet werden, falls $\text{shape}(A_2) \neq 0$ ist.

$$\text{shape}(A_i) = \{v \in W \mid A_i(v) \neq \#\}, \quad 1 \leq i \leq 2$$

Eine n-dimensionale Array Grammatik kann nun als Sechstupel

$$G = (n, V_N, V_T, \#, P, \{(v_0, S)\})$$

definiert werden. $\{(v_0, S)\}$ wird als Startarray (Axiom), v_0 als Startvektor und S als das Startsymbol bezeichnet.

matrix grammar

Eine Matrixgrammatik G_M ist ein 4-Tupel

$$G_M = (V_N, V_T, (M, F), S),$$

M ist eine endliche Menge von Matrizen, $M = \{m_i \mid 1 \leq i \leq n\}$. Die Matrizen m_i sind Sequenzen von der Form

$$m_i = (m_{i,1}, \dots, m_{i,n_i}), \quad n_i \geq 1, \quad 1 \leq i \leq n.$$

F kann auch als Fehlermenge bezeichnet werden. Ist $F = \emptyset$, dann kann G_M als Matrixgrammatik ohne appearance checking bezeichnet werden.

graph controlled grammar

Eine graph-controlled Grammatik G_P ist ein 4-Tupel

$$G_M = (V_N, V_T, (R, L_{in}, L_{fin}), S),$$

R ist eine endliche Menge von Regeln r der Form

$$(l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r))), l(r) \in Lab(G_P).$$

Falls $\varphi(l(r))$ leer ist für alle $r \in R$, dann kann G_P als graph-controlled Grammatik ohne appearance checking bezeichnet werden.

Falls $\forall r \in R \varphi(l(r)) = \sigma(l(r))$, dann kann G_P als graph-controlled Grammatik mit unconditional transfer bezeichnet werden.

Matrix- und graph-controlled Grammatiken können in Arraygrammatiken direkt übergeführt werden, indem ihre Produktionen durch Arrayproduktionen ersetzt werden.

graph controlled grammar

Eine graph-controlled Grammatik G_P ist ein 4-Tupel

$$G_M = (V_N, V_T, (R, L_{in}, L_{fin}), S),$$

R ist eine endliche Menge von Regeln r der Form

$$(l(r) : p(l(r)), \sigma(l(r)), \varphi(l(r))), l(r) \in Lab(G_P).$$

Falls $\varphi(l(r))$ leer ist für alle $r \in R$, dann kann G_P als graph-controlled Grammatik ohne appearance checking bezeichnet werden.

Falls $\forall r \in R \varphi(l(r)) = \sigma(l(r))$, dann kann G_P als graph-controlled Grammatik mit unconditional transfer bezeichnet werden.

Matrix- und graph-controlled Grammatiken können in Arraygrammatiken direkt übergeführt werden, indem ihre Produktionen durch Arrayproduktionen ersetzt werden.

bounded derivations

Index einer Ableitung

Der Index einer Ableitung D eines Terminalobjekts w (String oder Array) in einer Grammatik G ist mit der maximalen Anzahl von non-terminal Symbolen, die in einem Zwischenableitungsschritt vorkommen, definiert und wird mit $ind_{G,D}(w)$ bezeichnet.

Weiters bezeichnet $ind_{G,min}(w)$ bzw. $ind_{G,max}(w)$ das Minimum bzw. das Maximum aus der Menge

$$\{ind_{G,D}(w) \mid w \text{ is generated by } G\}$$

Entsprechend gibt es nun die Definition für die Grammatik

$$ind_Y(G) = sup\{ind_{G,Y}(w) \mid w \text{ is generated by } G\} \quad Y \in \{min, max\}$$

Andere Art von endlicher Index Restriktion nur Objekte w beachtet, die von einer Grammatik G mit einer Ableitung $ind_{G,Y}(w) \leq k$ für $Y \in \{min, max\}$ erzeugt werden.

bounded derivations

Index einer Ableitung

Der Index einer Ableitung D eines Terminalobjekts w (String oder Array) in einer Grammatik G ist mit der maximalen Anzahl von non-terminal Symbolen, die in einem Zwischenableitungsschritt vorkommen, definiert und wird mit $ind_{G,D}(w)$ bezeichnet.

Weiters bezeichnet $ind_{G,min}(w)$ bzw. $ind_{G,max}(w)$ das Minimum bzw. das Maximum aus der Menge

$$\{ind_{G,D}(w) \mid w \text{ is generated by } G\}$$

Entsprechend gibt es nun die Definition für die Grammatik

$$ind_Y(G) = \sup\{ind_{G,Y}(w) \mid w \text{ is generated by } G\} \quad Y \in \{min, max\}$$

Andere Art von endlicher Index Restriktion nur Objekte w beachtet, die von einer Grammatik G mit einer Ableitung $ind_{G,Y}(w) \leq k$ für $Y \in \{min, max\}$ erzeugt werden.

grammar with prescribed teams

Eine Grammatik mit prescribed teams G_t ist ein 4-Tupel

$$G_t = (V_N, V_T, (P, R, F), S),$$

$G = (V_N, V_T, P, S)$ ist eine kontextfreie Grammatik, R ist eine endliche Menge von Teams aus P und F ist die Menge von Produktionen, die bei dem appearance checking übersprungen werden können. Ist $F = \emptyset$, dann kann G_t als Grammatik mit prescribed teams ohne appearance checking bezeichnet werden.

Wiederum beschränken wir die Anzahl von non-terminal Symbolen, die in einem Zwischenableitungsschritt vorkommen dürfen, indem nur Teams angewendet werden dürfen, die alle non-terminale im Zwischenschritt betreffen \Rightarrow durch max. Anzahl an kontextfreien Produktionen in einem Team beschränkt.

$L^{\lceil k \rceil}(G)$ bezeichnet nun die Sprache, die durch die Grammatik G mit endlicher Index Restriktion von k erzeugt wird.

Beispiel für eine Grammatik mit prescribed teams (1)

2-dimensionale Array-Grammatik mit prescribed teams:

$$G = (n, \{D, E, L, Q, R, S, U\}, \{a\}, \#, (P, R, F), \{((0, 0), S)\})$$

$$P = \left\{ \begin{array}{l} \# \\ S\# \end{array} \rightarrow \begin{array}{l} L \\ aD \end{array}, \begin{array}{l} \# \\ L \end{array} \rightarrow \begin{array}{l} L \\ a \end{array}, D\# \rightarrow aD, \begin{array}{l} \#\# \\ L \end{array} \rightarrow \begin{array}{l} aU \\ a \end{array}, \right.$$

$$D\# \rightarrow aR, \begin{array}{l} \# \\ R \end{array} \rightarrow \begin{array}{l} R \\ a \end{array}, U\# \rightarrow aU,$$

$$U\# \rightarrow aU, U \rightarrow E, \begin{array}{l} \# \\ R \end{array} \rightarrow \begin{array}{l} Q \\ a \end{array}, R \rightarrow a, E \rightarrow a \}$$

Beispiel für eine Grammatik mit prescribed teams (2)

$$R = \left\{ \left\langle \begin{array}{c} \# \\ S\# \end{array} \rightarrow \begin{array}{c} L \\ aD \end{array} \right\rangle, \left\langle \begin{array}{c} \# \\ L \end{array} \rightarrow \begin{array}{c} L \\ a \end{array}, D\# \rightarrow aD \right\rangle, \right. \\ \left. \left\langle \begin{array}{c} \#\# \\ L \end{array} \rightarrow \begin{array}{c} aU \\ a \end{array}, D\# \rightarrow aR \right\rangle, \left\langle \begin{array}{c} \# \\ R \end{array} \rightarrow \begin{array}{c} R \\ a \end{array}, U\# \rightarrow aU \right\rangle, \right. \\ \left. \left\langle U \rightarrow E, \begin{array}{c} \# \\ R \end{array} \rightarrow \begin{array}{c} Q \\ a \end{array} \right\rangle, \langle R \rightarrow a, E \rightarrow a \rangle \right\}$$

$$F = \left\{ \begin{array}{c} \# \\ R \end{array} \rightarrow \begin{array}{c} Q \\ a \end{array} \right\}$$

Ableitungssequenz:

$$S \Rightarrow_G \begin{array}{c} L \\ aD \end{array} \Rightarrow_G \begin{array}{c} aU \\ a \\ a a R \end{array} \Rightarrow_G \begin{array}{c} a a U \\ a \\ a a R \end{array} \Rightarrow_G \begin{array}{c} a a E \\ a R \\ a a a \end{array} \Rightarrow_G \begin{array}{c} a a a \\ a a \\ a a a \end{array}$$

Gefolgerte Resultate

$PT_{ac}^{[k]}(cf)$ bezeichnet die Sprachfamilie, die von String-Grammatiken mit prescribed teams aus kontextfreien Produktionen mit endlicher Index Restriktion erzeugt werden.

$PT_{ac}^{[k]}(X)$ bezeichnet die Familie aller Arraysprachen, die von Array-Grammatiken mit prescribed teams aus Array-Produktionen des Types X , mit $X \in \{n\text{-}\#\text{-}cf, n\text{-}cf, n\text{-}scf \mid n \geq 1\}$ erzeugt werden.

Theorem 1

Für jedes $k \in \{fin\} \cup \{j \mid j \geq 1\}$, gilt

$$PT^{[k]}(cf) = PT_{ac}^{[k]}(cf) = Z^{Y, [k]}(cf) = Z^{min, \cap [k]}(cf)$$

für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$.

Gefolgte Resultate

$PT_{ac}^{[k]}(cf)$ bezeichnet die Sprachfamilie, die von String-Grammatiken mit prescribed teams aus kontextfreien Produktionen mit endlicher Index Restriktion erzeugt werden.

$PT_{ac}^{[k]}(X)$ bezeichnet die Familie aller Arraysprachen, die von Array-Grammatiken mit prescribed teams aus Array-Produktionen des Types X , mit $X \in \{n\text{-}\#\text{-}cf, n\text{-}cf, n\text{-}scf \mid n \geq 1\}$ erzeugt werden.

Theorem 1

Für jedes $k \in \{fin\} \cup \{j \mid j \geq 1\}$, gilt

$$PT^{[k]}(cf) = PT_{ac}^{[k]}(cf) = Z^{Y, [k]}(cf) = Z^{min, \cap [k]}(cf)$$

für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$.

Gefolgte Resultate (2)

Theorem 2

Für $X \in \{n\text{-}\#\text{-}cf, n\text{-}cf \mid n \geq 1\}$, gilt

- 1 $PT^{\lceil fin \rceil}(X) = Z^{Y, \lceil fin \rceil}(X) = Z^{min, \cap \lceil fin \rceil}(X)$ und auch
- 2 $PT_{ac}^{\lceil fin \rceil}(X) = Z_{ac}^{Y, \lceil fin \rceil}(X) = Z_{ac}^{min, \cap \lceil fin \rceil}(X)$

für alle $Z \in \{M, P\}$ und $Y \in \{min, max\}$.

Theorem 3

Für alle $k \in \{fin\} \cup \{j \mid j \geq 1\}$ gilt

- 1 $PT^{\lceil k \rceil}(X) = Z^{Y, \lceil k \rceil}(X) = Z^{min, \cap \lceil k \rceil}(X)$ *genau so wie*
- 2 $PT_{ac}^{\lceil k \rceil}(X) = Z_{ac}^{Y, \lceil k \rceil}(X) = Z_{ac}^{min, \cap \lceil k \rceil}(X)$

Für alle $X \in \{n\text{-}scf, n\text{-}scf_1 \mid n \geq 1\}$, $Z \in \{M, P\}$, und $Y \in \{min, max\}$.

Gefolgerte Resultate (2)

Theorem 2

Für $X \in \{n\text{-}\#\text{-cf}, n\text{-cf} \mid n \geq 1\}$, gilt

- ① $PT^{\lceil fin \rceil}(X) = Z^{Y, \lceil fin \rceil}(X) = Z^{min, \cap \lceil fin \rceil}(X)$ und auch
- ② $PT_{ac}^{\lceil fin \rceil}(X) = Z_{ac}^{Y, \lceil fin \rceil}(X) = Z_{ac}^{min, \cap \lceil fin \rceil}(X)$

für alle $Z \in \{M, P\}$ und $Y \in \{min, max\}$.

Theorem 3

Für alle $k \in \{fin\} \cup \{j \mid j \geq 1\}$ gilt

- ① $PT^{\lceil k \rceil}(X) = Z^{Y, \lceil k \rceil}(X) = Z^{min, \cap \lceil k \rceil}(X)$ *genau so wie*
- ② $PT_{ac}^{\lceil k \rceil}(X) = Z_{ac}^{Y, \lceil k \rceil}(X) = Z_{ac}^{min, \cap \lceil k \rceil}(X)$

Für alle $X \in \{n\text{-scf}, n\text{-scf}_1 \mid n \geq 1\}$, $Z \in \{M, P\}$, und $Y \in \{min, max\}$.

Gefolgte Resultate (3) - 1-dimensionaler Fall

Theorem 4

Für alle $X \in \{1 - cf_1, 1 - scf_1\}$ gilt

$$PT^{[1]}(X) = PT_{ac}^{[1]}(X) = Z^{Y, [1]}(X) = Z^{min, \cap [1]}(X) = L(1 - reg)$$

Für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$.

Theorem 5

Für alle $X \in \{1 - cf_1, 1 - scf_1\}$ und alle $k \geq 2$ gilt

$$PT^{[k]}(X) = PT_{ac}^{[k]}(X) = Z^{Y, [k]}(X) = Z^{min, \cap [k]}(X) = PT^{[2]}(X)$$

Für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$. $PT^{[2]}(X)$ repräsentiert eine Familie von String-Sprachen $L(lin)$.

Gefolgerte Resultate (3) - 1-dimensionaler Fall

Theorem 4

Für alle $X \in \{1 - cf_1, 1 - scf_1\}$ gilt

$$PT^{[1]}(X) = PT_{ac}^{[1]}(X) = Z^{Y, [1]}(X) = Z^{min, \cap [1]}(X) = L(1 - reg)$$

Für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$.

Theorem 5

Für alle $X \in \{1 - cf_1, 1 - scf_1\}$ und alle $k \geq 2$ gilt

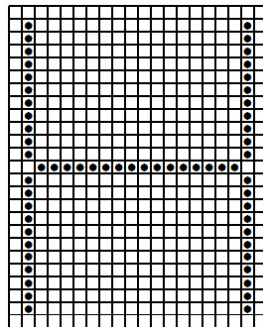
$$PT^{[k]}(X) = PT_{ac}^{[k]}(X) = Z^{Y, [k]}(X) = Z^{min, \cap [k]}(X) = PT^{[2]}(X)$$

Für alle $Z \in \{M, M_{ac}, P, P_{ut}, P_{ac}\}$ und $Y \in \{min, max\}$. $PT^{[2]}(X)$ repräsentiert eine Familie von String-Sprachen $L(lin)$.

Syntaktisches Pattern Recognition

Stufen in Character Recognition

- 1 Anlegen von Daten
- 2 Preprocessing
 - Normalisierung und Rauscheliminierung: anschließend abbilden auf eine 20 x 25 Gitter
 - Ausdünnung (Skelletierung)
- 3 Syntaktische Analyse



Beispiel zur Erkennung eines H

2-dimensionale Array-Grammatik mit prescribed teams:

$$G = (n, \{S, L, R, D_L, U_L, D_R, U_R\}, \{a\}, \#, (P, R, \emptyset), \{((0, 0), S)\})$$

$$P = \left\{ \begin{array}{l} S\# \rightarrow LR, \#L \rightarrow La, R\# \rightarrow aR, \begin{array}{c} \# \\ L \end{array} \rightarrow \begin{array}{c} U_L \\ a \\ D_L \end{array}, \\ \begin{array}{c} \# \\ R \end{array} \rightarrow \begin{array}{c} U_R \\ a \\ D_R \end{array}, \# \rightarrow U_L, \# \rightarrow U_R, D_L \rightarrow \begin{array}{c} a \\ D_L \end{array}, \\ D_R \rightarrow \begin{array}{c} a \\ D_R \end{array}, U_L \rightarrow a, U_R \rightarrow a, D_L \rightarrow a, D_R \rightarrow a \end{array} \right\}$$

Beispiel zur Erkennung eines H (2)

$$R = \{ \langle S\# \rightarrow LR \rangle, \langle \#L \rightarrow La, R\# \rightarrow aR \rangle, \}$$

$$\left\langle \begin{array}{c} \# \\ L \end{array} \rightarrow \begin{array}{c} U_L \\ a \\ D_L \end{array}, \begin{array}{c} \# \\ R \end{array} \rightarrow \begin{array}{c} U_R \\ a \\ D_R \end{array} \right\rangle,$$

$$\left\langle \begin{array}{c} \# \\ U_L \end{array} \rightarrow \begin{array}{c} U_L \\ a \end{array}, \begin{array}{c} \# \\ U_R \end{array} \rightarrow \begin{array}{c} U_R \\ a \end{array}, \begin{array}{c} D_L \\ \# \end{array} \rightarrow \begin{array}{c} a \\ D_L \end{array}, \begin{array}{c} D_R \\ \# \end{array} \rightarrow \begin{array}{c} a \\ D_R \end{array} \right\rangle,$$

$$\langle U_L \rightarrow a, U_R \rightarrow a, D_L \rightarrow a, D_R \rightarrow a \rangle \}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a a a a & \\ D_L & D_R \end{array} \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a & a \\ a a a a & \\ a & a \\ D_L & D_R \end{array} \Rightarrow_G$$

$$\begin{array}{cccc} U_L & U_R & a & a \\ a & a & a & a \\ a & a & a & a \\ a a a a & \Rightarrow_G & a a a a & \\ a & a & a & a \\ a & a & a & a \\ D_L & D_R & a & a \end{array}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a a a a & \\ D_L & D_R \end{array} \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a & a \\ a a a a & \\ a & a \\ D_L & D_R \end{array} \Rightarrow_G$$

$$\begin{array}{cccc} U_L & U_R & a & a \\ a & a & a & a \\ a & a & a & a \\ a a a a & \Rightarrow_G & a a a a & \\ a & a & a & a \\ a & a & a & a \\ D_L & D_R & a & a \end{array}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a a a a & \\ D_L & D_R \end{array} \Rightarrow_G \begin{array}{cc} U_L & U_R \\ a & a \\ a a a a & \\ a & a \\ D_L & D_R \end{array} \Rightarrow_G$$

$$\begin{array}{cccc} U_L & U_R & a & a \\ a & a & a & a \\ a & a & a & a \\ a a a a & \Rightarrow_G & a a a a & \\ a & a & a & a \\ a & a & a & a \\ D_L & D_R & a & a \end{array}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$\begin{array}{ccccccc}
 & & & & U_L & & U_R \\
 & & & & a & & a \\
 S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G & a a a a & \Rightarrow_G & a a a a & \Rightarrow_G & & \\
 & D_L & & D_R & & D_L & D_R
 \end{array}$$

$$\begin{array}{cccc}
 U_L & U_R & a & a \\
 a & a & a & a \\
 a & a & a & a \\
 & a a a a & \Rightarrow_G & a a a a \\
 a & a & a & a \\
 a & a & a & a \\
 D_L & D_R & a & a
 \end{array}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$\begin{array}{ccccccc}
 & & & & U_L & & U_R \\
 & & & & a & & a \\
 S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G & a a a a & \Rightarrow_G & a a a a & \Rightarrow_G & & \\
 & D_L & & D_R & & D_L & D_R
 \end{array}$$

$$\begin{array}{cccc}
 U_L & U_R & a & a \\
 a & a & a & a \\
 a & a & a & a \\
 a a a a & \Rightarrow_G & a a a a & \\
 a & a & a & a \\
 a & a & a & a \\
 D_L & D_R & a & a
 \end{array}$$

Beispiel zur Erkennung eines H (3)

Ableitungssequenz:

$$\begin{array}{ccccccc}
 & & & & U_L & & U_R \\
 & & & & a & & a \\
 S \Rightarrow_G L R \Rightarrow_G L a a R \Rightarrow_G & a a a a & \Rightarrow_G & a a a a & \Rightarrow_G & & \\
 & D_L & & D_R & & D_L & D_R
 \end{array}$$

$$\begin{array}{cccc}
 U_L & U_R & a & a \\
 a & a & a & a \\
 a & a & a & a \\
 a a a a & \Rightarrow_G & a a a a & \\
 a & a & a & a \\
 a & a & a & a \\
 D_L & D_R & a & a
 \end{array}$$

Theorie und Realität (1)

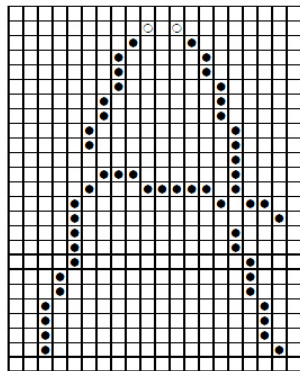
Zur Ableitung der horizontalen Linie muss
z.B. zusätzlich zu der Produktion

$$R\# \rightarrow aR$$

auch noch die Produktionen

$$R \xrightarrow{\#} aR, \quad R \xrightarrow{\#} aR$$

hinzugefügt werden.



Theorie und Realität (2)

- Entscheidung anhand des Abweichungsfehlers zu idealen Buchstaben
- Effizienz
 - Verwendung von regulated array Grammatiken (z.B.: graph-controlled Grammatik)
 - Reduzierung der nicht deterministischen Auswahl von Regeln

Anforderungen

- Automat um Sprachen aus kontextfreien Array-Grammatiken mit endlicher Index Restriktion zu charakterisieren
- Array-Grammatiken verarbeiten nicht nur Symbol- sondern auch Positionsinformationen \Rightarrow Ein-Weg Automat nicht möglich
- selbe Information darf nur einmal gelesen werden

Ein Ableitungsschritt für ein selektiertes Team ist nur möglich, wenn

- Alle Non-Terminals im aktuellen Array werden parallel abgeleitet.
- Der *shape* des aktuellen Arrays muss nach der Ableitung Teil des *shape* des originalen Arrays sein. (nicht bei #-kontextfreien Array-Produktionen)
- Jede Position eines Terminalsymbols im aktuellen Array muss mit der Position des originalen Arrays übereinstimmen.

Anforderungen

- Automat um Sprachen aus kontextfreien Array-Grammatiken mit endlicher Index Restriktion zu charakterisieren
- Array-Grammatiken verarbeiten nicht nur Symbol- sondern auch Positionsinformationen \Rightarrow Ein-Weg Automat nicht möglich
- selbe Information darf nur einmal gelesen werden

Ein Ableitungsschritt für ein selektiertes Team ist nur möglich, wenn

- Alle Non-Terminals im aktuellen Array werden parallel abgeleitet.
- Der *shape* des aktuellen Arrays muss nach der Ableitung Teil des *shape* des originalen Arrays sein. (nicht bei #-kontextfreien Array-Produktionen)
- Jede Position eines Terminalsymbols im aktuellen Array muss mit der Position des originalen Arrays übereinstimmen.

Definition

Ein n -dimensionaler k -head finite Array-Automat vom Typ X ($X \in \{n\text{-}\#\text{-}cf, n\text{-}cf, n\text{-}scf, n\text{-}cf_1, n\text{-}scf_1 \mid n \geq 1\}$) ist folgendermaßen definiert

$$(n, V_N, V_T, \#, (P, R, F), \{v_0, S\}),$$

sodass dieser einer n -dimensionalen Array-Grammatik mit prescribed Teams G mit endlichem Index k entspricht, mit

- jedes Team in R hat maximal k Array-Produktionen des Typs X
 - für jede Array-Produktion $p \in P$, ist p von der Form
- $$p = (W, \{(\Omega_n, A)\} \cup \{(v, \#) \mid v \in W \setminus \{\Omega_n\}\}, \{(v, X_v) \mid v \in W\})$$

Funktionsweise

Der Automat M arbeitet auf einem gegebenen Array aus V_T^{*n} nun wie folgt:

M arbeitet auf Objekten von $\{(a, a), (a, X) \mid a \in V_T, X \in V_N \cup \{\#\}\}^{*n}$, wo die erste Komponente das gegebene Array beinhaltet und die zweite Komponente das bis jetzt durch G generierte Array.

Der aktuelle Zustand des Automaten ist durch die Menge der Non-Terminale Y aus den (a, Y) Paaren des aktuellen Arrays gegeben.

Definition der durch den Automaten M akzeptierten Array-Sprache

$$L(M) = \{A \mid A \in V_T^{*n}, \{(v, (A(v), \#)) \mid v \in \text{shape}(A)\} \Longrightarrow_M^* \{(v, (A(v), A(v))) \mid v \in \text{shape}(A)\}\}$$

Funktionsweise

Der Automat M arbeitet auf einem gegebenen Array aus V_T^{*n} nun wie folgt:

M arbeitet auf Objekten von $\{(a, a), (a, X) \mid a \in V_T, X \in V_N \cup \{\#\}\}^{*n}$, wo die erste Komponente das gegebene Array beinhaltet und die zweite Komponente das bis jetzt durch G generierte Array.

Der aktuelle Zustand des Automaten ist durch die Menge der Non-Terminale Y aus den (a, Y) Paaren des aktuellen Arrays gegeben.

Definition der durch den Automaten M akzeptierten Array-Sprache

$$L(M) = \{A \mid A \in V_T^{*n}, \{(v, (A(v), \#)) \mid v \in \text{shape}(A)\} \Longrightarrow_M^* \{(v, (A(v), A(v))) \mid v \in \text{shape}(A)\}\}$$

Beispiel eines Automaten

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) (a, S) (a, \#) (a, \#) & \Rightarrow_M & (a, \#) (a, L) (a, R) (a, \#) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, U_L) & (a, U_R) \\
 (a, L) (a, a) (a, a) (a, R) & \Rightarrow_M & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, D_L) & (a, D_R) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

Beispiel eines Automaten

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) (a, S) (a, \#) (a, \#) & \Rightarrow_M & (a, \#) (a, L) (a, R) (a, \#) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, U_L) & (a, U_R) \\
 (a, L) (a, a) (a, a) (a, R) & \Rightarrow_M & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, D_L) & (a, D_R) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

Beispiel eines Automaten

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) (a, S) (a, \#) (a, \#) & \Rightarrow_M & (a, \#) (a, L) (a, R) (a, \#) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, U_L) & (a, U_R) \\
 (a, L) (a, a) (a, a) (a, R) & \Rightarrow_M & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, D_L) & (a, D_R) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

Beispiel eines Automaten

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) (a, S) (a, \#) (a, \#) & \Rightarrow_M & (a, \#) (a, L) (a, R) (a, \#) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

$$\begin{array}{cccc}
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, U_L) & (a, U_R) \\
 (a, L) (a, a) (a, a) (a, R) & \Rightarrow_M & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M \\
 (a, \#) & (a, \#) & (a, D_L) & (a, D_R) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#) \\
 (a, \#) & (a, \#) & (a, \#) & (a, \#)
 \end{array}$$

Beispiel eines Automaten (2)

$$\begin{array}{ccccccc}
 (a, \#) & & (a, \#) & & (a, U_L) & & (a, U_R) \\
 (a, U_L) & & (a, U_R) & & (a, a) & & (a, a) \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 (a, D_L) & & (a, D_R) & & (a, a) & & (a, a) \\
 (a, \#) & & (a, \#) & & (a, D_L) & & (a, D_R)
 \end{array}$$

$$\begin{array}{ccc}
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a)
 \end{array}$$

Beispiel eines Automaten (2)

$$\begin{array}{ccccccc}
 (a, \#) & & (a, \#) & & (a, U_L) & & (a, U_R) \\
 (a, U_L) & & (a, U_R) & & (a, a) & & (a, a) \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 (a, D_L) & & (a, D_R) & & (a, a) & & (a, a) \\
 (a, \#) & & (a, \#) & & (a, D_L) & & (a, D_R)
 \end{array}$$

$$\begin{array}{ccc}
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a)
 \end{array}$$

Beispiel eines Automaten (2)

$$\begin{array}{ccccccc}
 (a, \#) & & (a, \#) & & (a, U_L) & & (a, U_R) \\
 (a, U_L) & & (a, U_R) & & (a, a) & & (a, a) \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & & (a, a) (a, a) (a, a) (a, a) & \Rightarrow_M & \\
 (a, a) & & (a, a) & & (a, a) & & (a, a) \\
 (a, D_L) & & (a, D_R) & & (a, a) & & (a, a) \\
 (a, \#) & & (a, \#) & & (a, D_L) & & (a, D_R)
 \end{array}$$

$$\begin{array}{ccc}
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 & (a, a) (a, a) (a, a) (a, a) & \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a) \\
 (a, a) & & (a, a)
 \end{array}$$

Conclusions

- Regulated Array-Grammatiken mit endlichen Index ist geeignete Methode für syntaktisches Pattern Recognition
- Mögliche Erweiterung zur Erkennung von 3-dimensionalen Objekten

Danke für die Aufmerksamkeit!
Fragen?

Conclusions

- Regulated Array-Grammatiken mit endlichen Index ist geeignete Methode für syntaktisches Pattern Recognition
- Mögliche Erweiterung zur Erkennung von 3-dimensionalen Objekten

Danke für die Aufmerksamkeit!
Fragen?