

Erkennung der kontextfreien Grammatiken mittels Boolescher Matrixmultiplikation

Valiant-Erkenner

Referent: Fedor Uvarov

Seminar „Algorithmen zu kontextfreien Grammatiken“

Dozenten: Prof Dr. Hofmann, Dr. Leiß

Ludwig-Maximilians-Universität München, WiSe 2015/16

2. Dezember 2015

Erinnerung: CYK-Erkenner (Bsp.)

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

X					S
	X			S	Z
		X	S	Z	
			Y		
				Y	
					Y

(vgl. Vortrag von Iljazi)

Erinnerung: CYK-Erkenner

- Arbeitet mit einer dreieckigen Hälfte der Tabelle $n \times n$ (n = Eingabelänge)
- Jede Zelle $[i, k]$ mit $k = j$ enthält alle Nichtterminale, die w_i erzeugen
- Jede andere Zelle $[i, k]$ enthält alle Nichtterminale, den String $w_k \dots w_{i+k}$ in der Eingabe in einem Parsebaum erzeugen
- Die Eingabe ist erfolgreich geparst, wenn die Zelle $[0, n]$ das Startsymbol enthält
- Erwartet Grammatik in CNF (Expansion zum Startsymbol möglich)
- Komplexität: $O(n^3)$
- **Frage:** kann man es schneller machen?

Hauptoperation des CYK-Erkennters

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

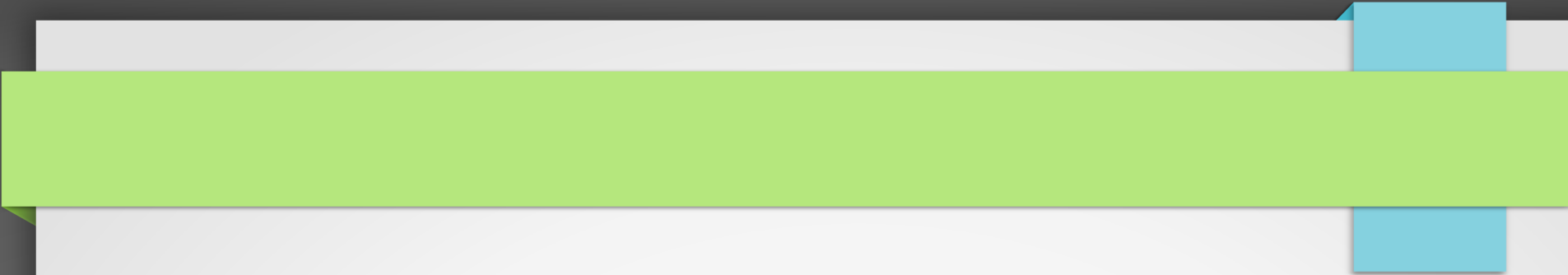
$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

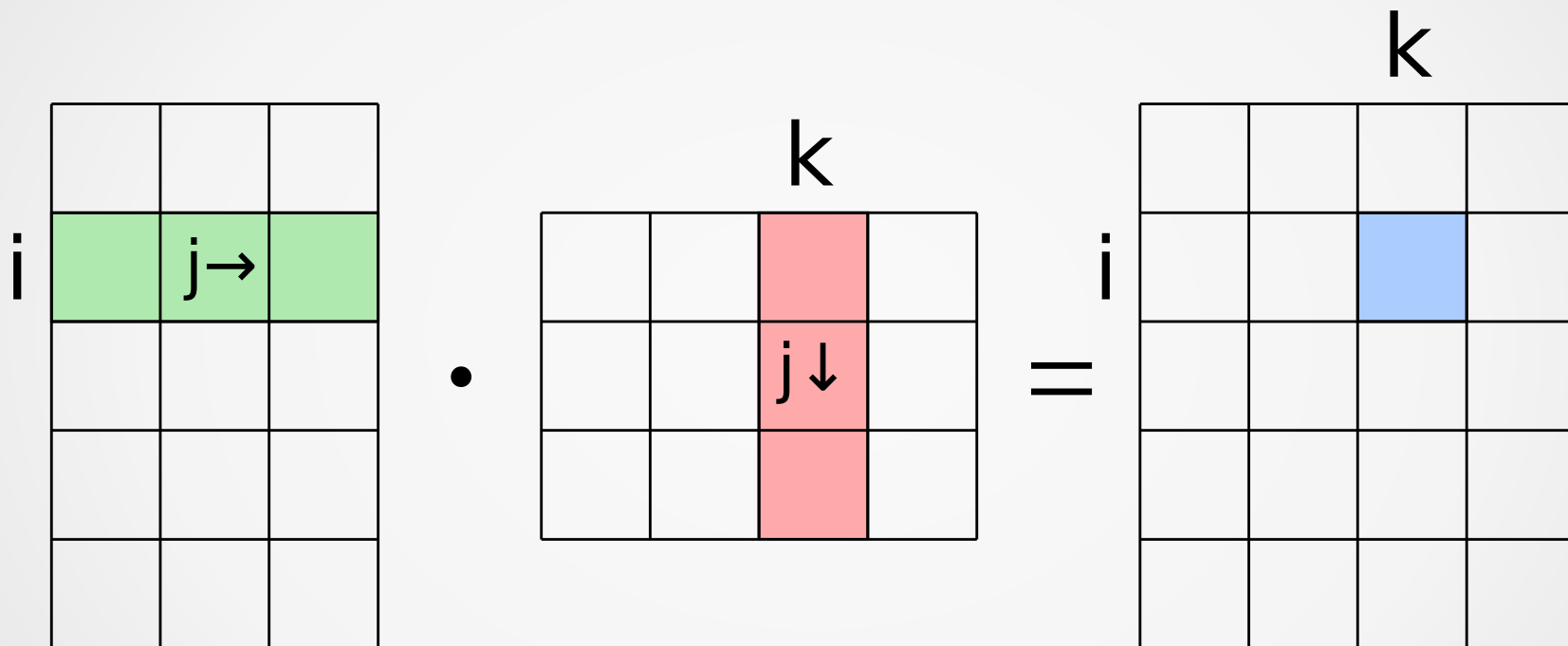
X					S
	X			S	Z
		X	S	Z	
			Y		
				Y	
					Y

Hauptoperation: laufe durch die Zeile nach links und durch die Spalte nach unten, finde die ersten zwei nichtleere Zellen, und für alle Nichtterminale A aus der ersten, B aus der zweiten, finde alle Regel $X \rightarrow A B$



Matrizen-Multiplikation?

Matrizenmultiplikation



Quelle:

https://commons.wikimedia.org/wiki/File:Matrix_multiplication_qtl2.svg (Benutzer Quartl)

CFG und Matrizenmultiplikation

- Matrizenmultiplikation ist schneller, als kubisch
 - Strassen-Algorithmus: $O(n^{\log_2 7}) \approx O(n^{2,81})$
 - Coppersmith-Winograd (mit Erweiterungen): $O(n^{2,37})$
 - Und andere
- ***Also:*** können wir CFG-Parsing als Matrizenmultiplikation darstellen?

Nichtterminale als Algebra

- Es gebe eine Grammatik $G = (N, \Sigma, P, A_1)$, wo N = Nichtterminale, Σ = Alphabet, P = Produktionen, $A_1 \in N$ = Startsymbol
- Sei in CNF, also mit nur drei Arten der Produktionen:

$$S \rightarrow A B$$

$$A \rightarrow a \text{ (Terminal)}$$

$$S \rightarrow \varepsilon \text{ (nur für das Startsymbol)}$$

- Sei $N_1 \cdot N_2 \stackrel{\text{def}}{=} \{A_i \mid \exists A_j \in N_1, A_k \in N_2: A_i \rightarrow A_k A_j \in P\}$, wo $N_1, N_2 \subseteq N$, also *alle möglichen Reduktionen*
- Sei $N_1 + N_2 \stackrel{\text{def}}{=} N_1 \cup N_2$
- $0 \stackrel{\text{def}}{=} \emptyset$
- So haben wir eine Algebra auf $\mathcal{P}(N)$ definiert

Eigenschaften der Operationen

- „+“ ist assoziativ und kommutativ (da Vereinigung)

$$(x + y) + z = x + (y + z)$$

$$x + y = y + x$$

- „•“ ist **weder** kommutativ **noch** assoziativ

Bsp. bei $C \rightarrow B \ C$:

- $\{B\} \cdot (\{B\} \cdot \{C\}) = \{B\} \cdot \{C\} = \{C\}$

- $(\{B\} \cdot \{B\}) \cdot \{C\} = \emptyset$

- „•“ distributiv über „+“

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

- $0 \cdot x = 0, x \cdot 0 = 0, 0 + x = x$

Matrixmultiplikation

- Gegeben sei eine Matrix a der Dimension $m \times n$ und b der Dimension $n \times l$ (in der Praxis arbeiten wir nur mit Dreiecksmatrizen [also $n \times n$] gleicher Dimensionen)
- Dann definieren wir Multiplikation:

$$c = a \cdot b \Leftrightarrow c_{i,k} \stackrel{\text{def}}{=} \sum_{j=1}^n a_{i,j} b_{j,k}$$

- Sei $\sigma_k \stackrel{\text{def}}{=} \{A \mid (A, w_k) \in P\}$ (Reduktion der Terminale)
- Ausgangsmatrix $I(w)$: eine obere Dreiecksmatrix der Dimension $|w| + 1$:

$$I(w)_{k,k+1} \stackrel{\text{def}}{=} \sigma_k$$

$$I(w)_{k,j} \stackrel{\text{def}}{=} 0, \text{ wenn } j \neq k + 1$$

Addition und $W^{(n)}$

- Sei $a + b$ auf **Matrizen** so definiert:

$$c = a + b \Leftrightarrow c_{i,k} \stackrel{\text{def}}{=} a_{i,k} + b_{i,k} = a_{i,k} \cup b_{i,k}$$

- Sei $W^{(n+1)} \stackrel{\text{def}}{=} W^{(n)} \bullet W^{(n)} + W^{(n)}$

$$W^{(1)} \stackrel{\text{def}}{=} I_{(w)}$$

Bsp. Ausgangsmatrix $I(w)$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	0
0	0	{ X }	0	0	0	0
0	0	0	{ X }	0	0	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Berechnung $W^{(2)} = I(w) \cdot I(w) + I(w)$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	0
0	0	{ X }	0	0	0	0
0	0	0	{ X }	{ S }	0	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Berechnung $W^{(3)} = W^{(2)} \bullet W^{(2)} + W^{(2)}$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	0
0	0	{ X }	0	0	0	0
0	0	0	{ X }	{ S }	{ Z }	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Berechnung $W^{(4)} = W^{(3)} \bullet W^{(3)} + W^{(3)}$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	0
0	0	{ X }	0	0	{ S }	0
0	0	0	{ X }	{ S }	{ Z }	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Berechnung $W^{(5)} = W^{(4)} \bullet W^{(4)} + W^{(4)}$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	0
0	0	{ X }	0	0	{ S }	{ Z }
0	0	0	{ X }	{ S }	{ Z }	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Berechnung $W^{(6)} = W^{(5)} \bullet W^{(5)} + W^{(5)}$

Grammatik:

$S \rightarrow X Y$ (Start)

$S \rightarrow X Z$

$Z \rightarrow S Y$

$X \rightarrow a$ (Term.)

$Y \rightarrow b$ (Term.)

Eingabe:

aaabbb

0	{ X }	0	0	0	0	{ S }
0	0	{ X }	0	0	{ S }	{ Z }
0	0	0	{ X }	{ S }	{ Z }	0
0	0	0	0	{ Y }	0	0
0	0	0	0	0	{ Y }	0
0	0	0	0	0	0	{ Y }
0	0	0	0	0	0	0

Erkennung als transitive Hülle

- $W^{(n)}$ ist eine Berechnung der transitiven Hülle von $I(w)$
- Transitive Hülle W^+ der Matrix W ist die *kleinste* solche Matrix C , dass $C = C \cdot C + W$
- Bisher haben wir dasselbe gemacht, wie CYK
- Aber es gibt eine effizientere Methode, sie zu berechnen

Berechnung der transitiven Hülle

- Ist W eine 1×1 -Matrix, dann ist $W = \{0\}$ und $W^+ = \{0\}$, da sie eine obere Dreiecksmatrix ist
- Sonst ist sie in 4 Blöcke dividierbar (nicht unbedingt derselben Dimension):

$$W = \begin{bmatrix} A & X \\ 0 & B \end{bmatrix} \quad \text{und} \quad C = \begin{bmatrix} A' & X' \\ 0 & B' \end{bmatrix}$$

$$\text{wo } A' = A'A' + A$$

$$X' = A'X' + X'B' + X$$

$$B' = B'B' + B$$

- A' und B' können rekursiv berechnet werden. X' ist ein bisschen schwieriger.

Berechnung von X'

- Wir brauchen eine Funktion $V(A, B, X) = X'$
- Wir zerteilen A , B und X wieder (bei Matrizen 2×2 und 3×3 ist die Hülle einfach zu finden):

$$X' = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}$$

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}$$

- $X' = AX' + BX' + X$ kann jetzt durch normale Definition von „•“ und „+“ berechnet werden

Berechnung von X'

- Durch Multiplikation entdeckt man, dass:

$$X'_{11} = A_{11} X'_{11} + X_{11} + A_{12} X'_{21} + X'_{11} B_{11}$$

$$X'_{12} = A_{11} X'_{12} + X_{12} + A_{12} X'_{22} + X'_{11} B_{12} + X'_{12} B_{22}$$

$$X'_{21} = A_{22} X'_{21} + X_{21} + X'_{21} B_{11}$$

$$X'_{22} = A_{22} X'_{22} + X_{22} + X'_{21} B_{12} + X'_{22} B_{22}$$

- Es gibt keine zirkulären Abhängigkeiten, deswegen kann es Rekursiv mittels V berechnet werden

X' durch V

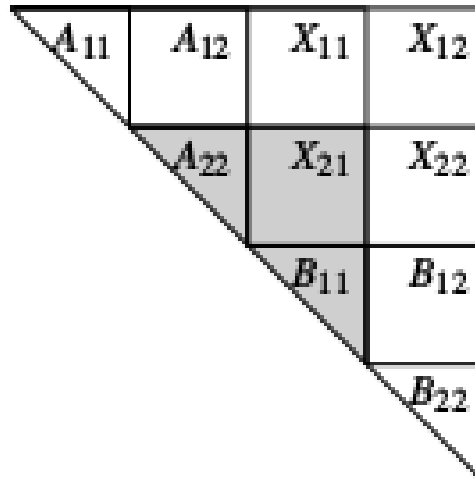
$$X'_{21} = V(A_{22}, X_{21}, B_{11})$$

$$X'_{11} = V(A_{11} + X_{11} + A_{12} X'_{21}, B_{11})$$

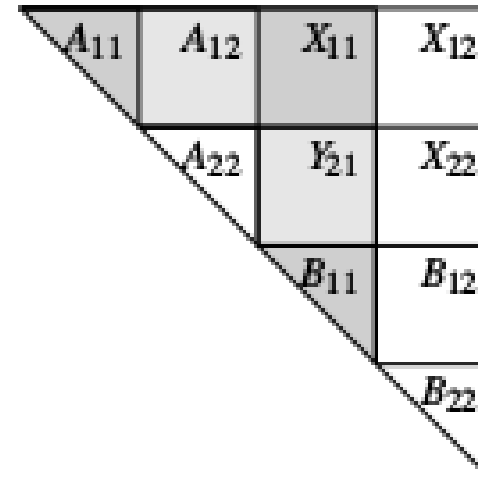
$$X'_{22} = V(A_{22}, X_{22} + X'_{21} B_{12}, B_{22})$$

$$X'_{12} = V(A_{11}, X_{12} + A_{12} X_{22} + X'_{11} B_{12}, B_{22})$$

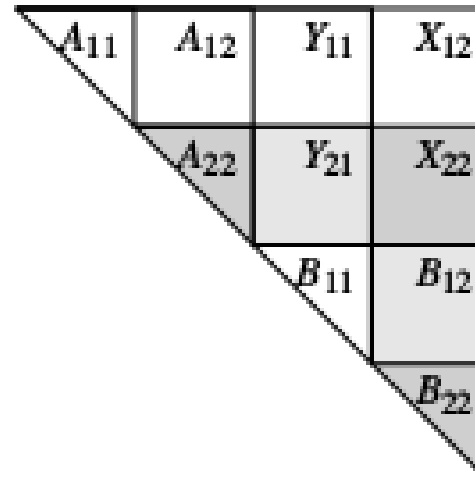
Step 1:



Step 2:



Step 3:



Step 4:

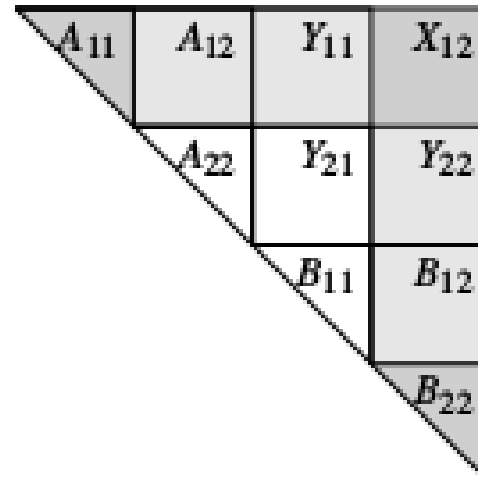


Fig. 2. The recursive step of function V . The charts A and B are already complete. To complete the matrix X , that is, compute $Y = V(A, X, B)$, one splits the matrices and performs 4 recursive calls. Each recursive call is depicted graphically. In each figure, to complete the dark-gray square, multiply the light-gray rectangles and add them to the dark-gray square, then do a recursive call on triangular matrix composed of the completed dark-gray square and the triangles.

Quelle: Bernardy, Claessen *Efficient Parallel and Incremental Parsing of Practical Context-Free Languages* (under consideration in *Journal of Functional Programming*)

Übergang zu Booleschen Matrizen

- Wir können die Multiplikation der Matrizen durch Matrizen von Booleschen Werten ($\{0, 1\}$) ausführen
- Das ist unsere Matrizenmultiplikation:

$$c = a \cdot b \Leftrightarrow c_{i,k} \stackrel{\text{def}}{=} \sum_{j=1}^n a_{i,j} b_{j,k}$$

- Wir wollen c finden

Übergang zu Booleschen Matrizen

- Wir definieren für jedes Nichtterminal A_l Boolesche Matrizen, der Form:

$$a[l]_{ik} \stackrel{\text{def}}{=} 1 \Leftrightarrow A_l \in a_{ik}$$

$$b[l]_{ik} \stackrel{\text{def}}{=} 1 \Leftrightarrow A_l \in b_{ik}$$

- Dann berechnen wir für jedes Paar der Matrizen eine Matrix:

$$c[l, m] = a[l] \cdot b[m]$$

- Und die gesuchte Matrix c kann dann konstruiert werden:

$$A_x \in c_{ik} \Leftrightarrow \exists_{l, m} : c[l, m]_{ik} = 1 \wedge A_x r \rightarrow A_l A_m \in P$$