Der Cocke-Younger-Kasami-Algorithmus zum effizienten Entscheiden des Wortproblems für CFLs

Prof. Dr. David Sabel

LFE Theoretische Informatik

Letzte Änderung der Folien: 9. Juni 2022

Motivation für Kontextfreie Sprachen

- Kontextfreie Sprachen sind insbesondere nützlich um Sprachen mit Klammerungen zu beschreiben
- Die Syntax von Programmiersprachen wird meist mit einer kontextfreien Grammatik angegeben

Beispiele:

- $G = (\{E, M, Z\}, \{+, *, (,)\} \cup \{0, \ldots, 9\}, P, E)$ mit

$$\begin{aligned} P = \{ & E \to M \mid E + M, \\ & M \to Z \mid M * Z, \\ & Z \to N \mid (E), \\ & N \to 1D \mid \ \ldots \ \mid 9D, \\ & D \to 0D \mid \ \ldots \ \mid 9D \mid \varepsilon \} \end{aligned}$$

- $L = \{a^j b^j \mid j \in \mathbb{N}\}$ ist kontextfrei: Produktionen $\{S \to \varepsilon \mid T, T \to aTb \mid ab\}$ mit S als Startsymbol erzeugen L

Wiederholung: Nichtkontextfreie Sprache

Satz

Die Sprache $L = \{a^l b^l c^l \mid l \in \mathbb{N}\}$ ist nicht kontextfrei.

Beweis haben wir gesehen (nur FSK)

Einleitung zum Wortproblem

- Wir wissen bereits: Das Wortproblem für Typ 1-Sprachen ist entscheidbar
- Daher ist auch das Wortproblem für Typ 2-Sprachen entscheidbar
- Aber: Der Algorithmus für Typ 1-Sprachen benötigt exponentiell viele Schritte
- Nun sehen wir einen Polynomialzeitalgorithmus
- Die wesentliche Entwurfsmethode dabei ist dynamische Programmierung

Effizientes Lösen des Wortproblems für CFLs

- Algorithmus für Typ 1-Sprachen hat exponentielle Laufzeit
- Algorithmus von Cocke, Younger und Kasami für CFLs
- Veröffentlicht in den 1960er Jahren
- Kurz: CYK-Algorithmus

Idee des CYK-Algorithmus

- Eingabe:
  - CFG $G = (V, \Sigma, P, S)$ in Chomsky-Normalform:

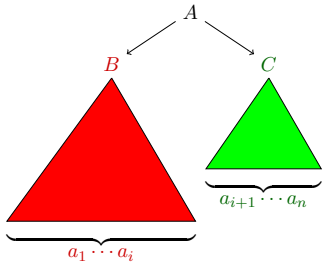
    Wiederholung / Bemerkung: G (mit $\varepsilon \notin L(G)$) ist in **Chomsky-Normalform**, wenn für jede Produktion $A \to w \in P$ gilt: $w = a \in \Sigma$ oder $w = BC$ mit $B, C \in V$.

    Satz: Für jede CFG G mit $\varepsilon \notin L(G)$ kann sprachäquivalente Chomsky-NF berechnet werden.
  - Wort $w \in \Sigma^*$
- Ausgabe:
  - ja, wenn $w \in L(G)$
  - nein, wenn $w \notin L(G)$
- Grundidee des Algorithmus:
  (Rekursiver) Test, ob Variable A ein Wort u erzeugt.
- Verwende Test zum Prüfen, ob S das Wort w erzeugt.

Idee des CYK-Algorithmus (2)

Prüfe, ob $A \in V$ ein Wort $u = a_1 \cdots a_n$ erzeugt:

- Wenn $u = a \in \Sigma$, dann prüfe ob $A \to a \in P$
- Anderenfalls ($|u| > 1$) kann u nur erzeugt werden, wenn:
  - Es gibt Produktion $A \to BC \in P$
  - Es gibt Index $1 \leq i < n$, sodass:
    - B erzeugt $a_1 \cdots a_i$ und
    - C erzeugt $a_{i+1} \cdots a_n$



- Daher prüfe für alle $A \to BC \in P$ und alle i mit $1 \leq i < n$ rekursiv, ob B das Wort $w = a_1 \cdots a_i$ und C das Wort $a_{i+1} \cdots a_n$ erzeugt.

Beispiel

Betrachte die Grammatik mit den Produktionen:

$S \to AB \mid BA,$
$A \to AA \mid AB \mid a,$
$B \to BB \mid b$

und das Wort $bbbaab$.

Bevor der rekursive Algorithmus diesen richtigen Pfad findet, sucht er einige erfolglose ab, z.B. für $S \to AB$

- Prüfe, ob A erzeugt b und B erzeugt $bbab$ gilt.
- Prüfe, ob A erzeugt bb und B erzeugt bab gilt.
- Prüfe, ob A erzeugt bbb und B erzeugt ab gilt.
- Prüfe, ob A erzeugt $bbba$ und B erzeugt b gilt.

Naives Suchen wiederholt dieselben Tests, z.B. ob „A erzeugt b“ gilt.

S erzeugt $bbbaab$, denn $S \to BA$ und

- B erzeugt bbb, denn $B \to BB$ und
  - B erzeugt bb, denn $B \to BB$ und $B \to b$ und $B \to b$
  - B erzeugt b, denn $B \to b$
- A erzeugt aab, denn $A \to AB$ und
  - A erzeugt aa, denn $A \to AA$ und
    - A erzeugt a, denn $A \to a$ und
    - A erzeugt a, denn $A \to a$
  - B erzeugt b, denn $B \to b$

Idee des CYK-Algorithmus (3)

- Effizienz: Statt Rekursion verwende dynamische Programmierung
- Algorithmus berechnet Menge $V(i,j) \subseteq V$, sodass

$$V(i,j) = \{A \in V \mid A \Rightarrow^* a_i \cdots a_{i+j-1}\}$$

„$V(i,j)$ enthält alle $A \in V$, die $a_i \cdots a_{i+j-1}$ (=Teilwort von u ab Position i mit Länge j) erzeugen“

- Berechnung der $V(i,j)$:
  - Starte mit $V(i,1) = \{A \mid A \to a_i \in P\}$.
  - Berechne $V(i,j)$ mit ansteigender Länge j.

    Für $j > 1$ gilt:

$$A \in V(i,j) \text{ g.d.w.}$$
$$A \to BC \in P \text{ und } B \in V(i,k),\ C \in V(i+k, j-k)$$

Berechnung: Für festes (i,j) betrachte alle k mit $k = 1, 2, \ldots, j-1$

- Finaler Schritt: Prüfe, ob $S \in V(1,n)$ ist.

Algorithmus 8: CYK-Algorithmus

Eingabe: CFG $G = (V, \Sigma, P, S)$ in Chomsky-Normalform und Wort $w = a_1 \cdots a_n \in \Sigma^*$
Ausgabe: `Ja`, wenn $w \in L(G)$ und `Nein`, wenn $w \notin L(G)$
Beginn

- **für** $i = 1$ *bis* n **tue**
  - $V(i,1) = \{A \in V \mid A \to a_i \in P\}$
- **für** $j = 2$ *bis* n **tue**
  - **für** $i = 1$ *bis* $n+1-j$ **tue**
    - $V(i,j) = \emptyset$;
    - **für** $k = 1$ *bis* $j-1$ **tue**
      - $V(i,j) = V(i,j) \cup \left\{ A \in V \;\middle|\; \begin{array}{l} A \to BC \in P, \\ B \in V(i,k), \\ C \in V(i+k, j-k) \end{array} \right\}$
- **wenn** $S \in V(1,n)$ **dann**
  - return `Ja`
- **sonst**
  - return `Nein`

Laufzeit des CYK-Algorithmus

- Drei geschachtelte Für-Schleifen
- Im Inneren wird noch über alle Produktionen aus P iteriert
- Mit $n = |w|$ und $|P|$ = Anzahl der Iterationen kann die Laufzeitkomplexität mit $\mathcal{O}(n^3 \cdot |P|)$ abgeschätzt werden.

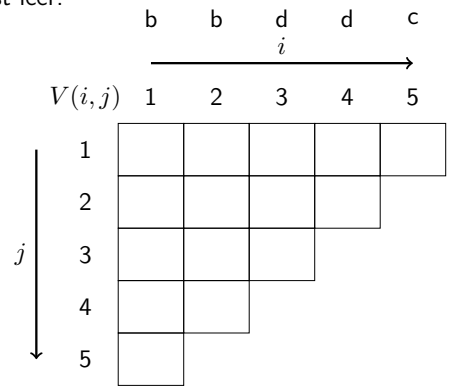
Theorem

Das Wortproblem für kontextfreie Sprachen kann in Polynomialzeit entschieden werden.

Beispiel

Sei $w = bbddc$ und $G = (\{S, A, B\}, \{b, c, d\}, P, S)$ mit
$P = \{S \to AC,\ A \to BE,\ A \to BD,\ E \to AD,\ C \to c,\ B \to b,\ D \to d\}$

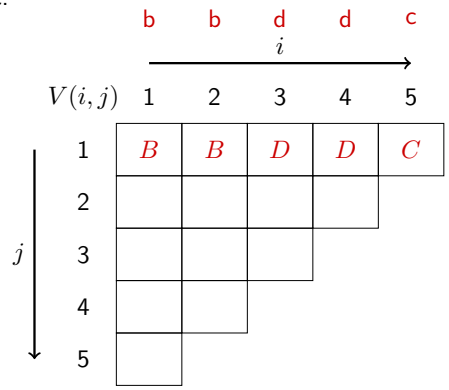
$V(i,j)$-Tabelle ist zunächst leer:



Beispiel (2)

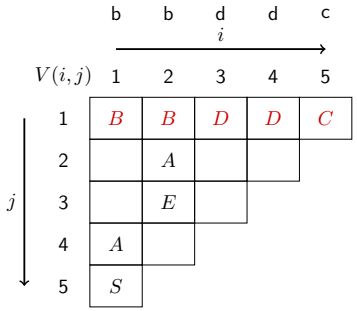
Sei $w = bbddc$ und $G = (\{S, A, B\}, \{b, c, d\}, P, S)$ mit
$P = \{S \to AC,\ A \to BE,\ A \to BD,\ E \to AD,\ C \to c,\ B \to b,\ D \to d\}$

Füllen der $V(i, 1)$-Einträge:



Beispiel (3)

Sei $w = bbddc$ und $G = (\{S, A, B\}, \{b, c, d\}, P, S)$ mit
$P = \{S \to AC,\ A \to BE,\ A \to BD,\ E \to AD,\ C \to c,\ B \to b,\ D \to d\}$



Da $S \in V(1, 5)$ gilt $w \in L(G)$

Web-Tool zum Üben / Anschauen

`http://www.cip.ifi.lmu.de/~lindebar/`