Bemerkung

Für die Praxis (z.B. Syntaxanalyse von Programmen) sind polynomiale Algorithmen wie CYK noch zu langsam. Für Teilklassen von CFLs sind schnellere Algorithmen bekannt, z.B.

Jay Earley:
An Efficient Context-free Parsing Algorithm.
Communications of the ACM **13**(2), pp. 94–102, 1970

4.12 Earley's Algorithmus

Sei G eine CFG, die o.B.d.A. keine ϵ-Produktion enthält (die algorithmische Behandlung des Falles $\epsilon \in L(G)$ wurde bereits besprochen) und bei der die rechte Seite einer jeden Produktion aus

$$V^+ \cup \Sigma$$

ist.

Sei $x = x_1 \cdots x_n \in \Sigma^+$ gegeben.

Definition 104

Wir definieren

$$[iAj[\alpha_1 \cdots \alpha_k . \alpha_{k+1} \cdots \alpha_r \ ,$$

falls G die Produktion

$$A \to \alpha_1 \cdots \alpha_r$$

enthält und, falls $j > i$, dann $k > 0$ und

$$\alpha_1 \cdots \alpha_k \to^* x_i \cdots x_{j-1} \ .$$

Wir nennen Objekte der soeben definierten Art t-Ableitung. (t steht dabei für tree oder table oder top-down.)

Earley's Algorithmus

$S_1 := \{[1S1[.\alpha;\ \alpha$ ist rechte Seite einer S-Produktion$\}$

for $j := 1$ **to** n **do**

führe folgende Schritte so oft wie möglich aus:

if $[iAj[\alpha_1 \cdots \alpha_k.B\alpha_{k+2} \cdots \alpha_r \in S_j$ **then**

füge für jede B-Produktion $B \rightarrow \beta$ die t-Ableitung $[jBj[.\beta$ zu S_j hinzu (falls noch nicht dort)

if $[iAj[\alpha_1 \cdots \alpha_r. \in S_j$ **then**

füge für jede t-Ableitung $[lBi[\beta_1 \cdots \beta_k.A\beta_{k+2} \cdots \beta_r$ die t-Ableitung $[lBj[\beta_1 \cdots \beta_k A.\beta_{k+2} \cdots \beta_r$ zu S_j hinzu

if $[jAj[.a \in S_j$ **and** $x_j = a$ **then**

füge zu S_{j+1} die t-Ableitung $[jAj+1[a.$ hinzu

if $S_{j+1} = \emptyset$ **then return** $x \notin L$

od

if S_{n+1} enthält t-Ableitung der Form $[1Sn+1[\alpha.$, α rechte Seite einer S-Produktion

then return $x \in L$

Bemerkungen:

1. Die drei Schritte in der Laufschleife werden auch als

   predictor, completer und scanner

   bezeichnet.
2. Der Algorithmus ist eine Mischung aus einem top-down- und einem bottom-up-Ansatz.
3. Die Korrektheit des Algorithmus ergibt sich unmittelbar (per Induktion) aus der Definition der t-Ableitung.
4. Für eine feste CFG G und eine Eingabe x der Länge $|x| = n$ existieren höchstens $\mathcal{O}(n^2)$ t-Ableitungen.
5. Damit enthält jedes S_j höchstens $\mathcal{O}(n)$ t-Ableitungen.
6. Die erste und dritte if-Anweisung benötigen daher (pro Iteration der j-Schleife) Zeit $\mathcal{O}(n)$, die zweite if-Anweisung $\mathcal{O}(n^2)$.

Bemerkungen:

7. Will man statt der ja/nein-Antwort für das Wortproblem einen (oder alle) Ableitungsbäume, falls $x \in L(G)$, so kann der completer-Schritt dafür geeignete Informationen kompakt abspeichern (es kann exponentiell viele verschiedene Ableitungsbäume geben!).

Satz 105

Die Laufzeit des Earley-Algorithmus ist, für eine feste CFG und in Abhängigkeit von der Länge n des Testworts, $\mathcal{O}(n^3)$.

Bemerkung:

Man kann zeigen (siehe Earley's Arbeit):

Ist die Grammatik eindeutig, so benötigt der completer-Schritt nur Zeit $\mathcal{O}(n)$, der ganze Algorithmus also Zeit

$$\mathcal{O}(n^2)\,.$$

Beispiel 106

Wir betrachten wiederum unsere Grammatik für arithmetische Ausdrücke mit den Regeln:

$$
\begin{aligned}
S &\to A \\
A &\to E \mid A + E \mid A - E \\
E &\to P \mid E \times P \mid E/P \\
P &\to (A) \mid a
\end{aligned}
$$

sowie das Testwort

$$a + a \times a$$

Beispiel 106

Earley's Algorithmus liefert:

S_1 : $\boxed{\text{[1S1[.A}}$ $\boxed{\text{[1A1[.E}}$ $\boxed{\text{[1A1[.A+E}}$ $\boxed{\text{[1E1[.P}}$ $\boxed{\text{[1P1[.a}}$ $\cdots$

S_2 : $\boxed{\text{[1P2[a.}}$ $\boxed{\text{[1E2[P.}}$ $\boxed{\text{[1A2[E.}}$ $\boxed{\text{[1S2[A.}}$ $\boxed{\text{[1A2[A.+E}}$ $\cdots$

S_3 : $\boxed{\text{[1A3[A+.E}}$ $\boxed{\text{[3E3[.P}}$ $\boxed{\text{[3P3[.a}}$ $\boxed{\text{[3E3[.E}\times\text{P}}$ $\cdots$

S_4 : $\boxed{\text{[3P4[a.}}$ $\boxed{\text{[3E4[P.}}$ $\boxed{\text{[3E4[E.}\times\text{P}}$ $\boxed{\text{[1A4[A+E.}}$ $\boxed{\text{[1S4[A.}}$ $\cdots$

S_5 : $\boxed{\text{[3E5[E}\times\text{.P}}$

S_6 : $\boxed{\text{[3E6[E}\times\text{ P.}}$ $\boxed{\text{[1A6[A+E.}}$ $\boxed{\text{[1S6[A.}}$

5. Kontextsensitive und Typ-0-Sprachen

5.1 Turingmaschinen

Turingmaschinen sind das grundlegende Modell, das wir für Computer/Rechenmaschinen verwenden. Es geht auf Alan Turing (1912–1954) zurück.

Definition 107

Eine nichtdeterministische Turingmaschine (kurz TM oder NDTM) wird durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ beschrieben, das folgende Bedingungen erfüllt:

1. Q ist eine endliche Menge von Zuständen.
2. Σ ist eine endliche Menge, das Eingabealphabet.
3. Γ ist eine endliche Menge, das Bandalphabet, mit $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$ ist die Übergangsfunktion.
5. $q_0 \in Q$ ist der Startzustand.
6. $\Box \in \Gamma \setminus \Sigma$ ist das Leerzeichen.
7. $F \subseteq Q$ ist die Menge der (akzeptierenden) Endzustände.

Eine Turingmaschine heißt deterministisch, falls gilt

$$|\delta(q, a)| \leq 1 \qquad \text{für alle } q \in Q, a \in \Gamma.$$

Erläuterung:

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$ bzw. $\delta(q, a) \ni (q', b, d)$:

Wenn sich M im Zustand q befindet und unter dem Schreib-/Lesekopf das Zeichen a steht, so geht M im nächsten Schritt in den Zustand q' über, schreibt an die Stelle des a's das Zeichen b und bewegt danach den Schreib-/Lesekopf um eine Position nach rechts (falls $d = R$), links (falls $d = L$) bzw. lässt ihn unverändert (falls $d = N$).

Beispiel 108

Es soll eine TM angegeben werden, die eine gegebene Zeichenreihe aus $\{0,1\}^+$ als Binärzahl interpretiert und zu dieser Zahl 1 addiert. Folgende Vorgehensweise bietet sich an:

1. Gehe ganz nach rechts bis ans Ende der Zahl. Dieses Ende kann durch das erste Auftreten eines Leerzeichens gefunden werden.
2. Gehe wieder nach links bis zur ersten 0 und ändere diese zu einer 1. Ersetze dabei auf dem Weg alle 1en durch 0.

Also:

$$
\begin{aligned}
\delta(q_0, 0) &= (q_0, 0, R) & \qquad \delta(q_1, 1) &= (q_1, 0, L)\\
\delta(q_0, 1) &= (q_0, 1, R) & \qquad \delta(q_1, 0) &= (q_f, 1, N)\\
\delta(q_0, \Box) &= (q_1, \Box, L) & \qquad \delta(q_1, \Box) &= (q_f, 1, N)
\end{aligned}
$$

Damit ist $Q = \{q_0, q_1, q_f\}$ und $F = \{q_f\}$.