1 Kontextsensitive Sprachen

Eine Grammatik heißt *kontextsensitiv* ($CH1$-Grammatik) gdw.

$$\forall\, p \to q \in P\ \exists\, A \in N, u, v, w \in (N \cup T)^*, w \neq \varepsilon \wedge p = uAv \wedge q = uwv$$

(d.h. ein Nichtterminalsymbol $A \in N$ wird „im Kontext“ $u \ldots v$ ersetzt durch das nichtleere Wort w). Zusätzlich gibt es wieder die Sonderregel für $S \to \varepsilon$.

Man sieht, dass die Länge des aktuellen Wortes niemals abnimmt (außer für ε). Damit ist das Wortproblem entscheidbar, z.B. durch einen Algorithmus, der für w alle möglichen Ableitungen ausprobiert und jeweils abbricht, sobald das aktuelle Wort länger als $|w|$ ist. Schleifen können erkannt werden.
Der Algorithmus kommt mit polynomiellem Platz aus, aber benötigt exponentielle Zeit - tatsächlich ist er PSPACE-vollständig (s. nächstes Kapitel).

Automatenmodell: Linear beschränkter Automat (LBA) = NTM mit Band der Länge $|w|$. Der Name kommt daher, dass man das Band faktisch um Faktor k vergrößern kann, indem die Symbole k-Tupel kodieren.

Offene Frage: Gilt DLBA = NLBA?

Realistisches Computermodell (Speicher ist endlich in der Länge der Eingabe)?

Satz Es gibt Turing-entscheidbare Sprachen, die nicht kontextsensitiv sind.

Bew.: Betrachte $\Sigma = \{0, 1\}$. Wir konstruieren eine Sprache $L \subseteq \Sigma^*$, die von einer Turingmaschine τ entschieden wird.
Kontextsensitive Grammatiken sind in Σ kodierbar und aufzählbar. Wir können weiterhin eine Ordnung auf Wörtern definieren, d.h. jedem Wort $w \in \Sigma^*$ bijektiv eine eindeutige Zahl $z(w)$ zuweisen.
Konstruiere akzeptierende 3-Band-TM τ:
Band 1: Eingabewort
Band 2: Bestimme $z(w)$ und zähle die entsprechende Grammatik $G_{z(w)}$ auf.
Band 3: Entscheide die Frage $w \overset{?}{\in} L(G)$. Gib als Ausgabe genau dann 1 aus, wenn die Antwort „nein“ lautet (sonst 0).
Die Sprache lautet also $L = \{w \mid w \notin G_{z(w)}\}$ und ist entscheidbar durch τ.
Damit gibt es keine kontextsensitive Grammatik G mit $L(G) = L$, denn eine solche Grammatik hätte eine Nummer n in unserer Aufzählung, $G = G_n$. Wir wissen, dass $n = z(w)$ für ein w. Dann finden wir einen Widerspruch:

$$w \in L(G_n) \Leftrightarrow \tau \text{ schreibt eine 0 angesetzt auf } w \Leftrightarrow w \notin L = L(G_n)$$ □

Allgemein sind alle Probleme nicht kontextsensitiv, die nicht mit linearem Platz auskommen, z.B. Äquivalenz regulärer Ausdrücke (EXPSPACE-vollst.).

2 Rekursivität

Def. 2.13 Eine Funktion $f : A^* \rightharpoonup B^*$ heißt *rekursiv*, falls sie Turing-berechenbar ist.
Eine Menge $L \subseteq A^*$ heißt *rekursiv*, falls sie Turing-entscheidbar ist.

Church-Turing-These (1936): Die intuitiv durch Algorithmen berechenbaren Funktionen sind genau die rekursiven Funktionen.
Anders formuliert: Die TM ist ein mathematisches Modell für den Begriff „Algorithmus“.

Andere Varianten, die zu Turingmaschinen (1936) äquivalent sind:
(1) Erfüllbarkeit in der Logik erster Stufe ($\rightarrow$ Logik)

(2) λ-Kalkül (1936)

(3) μ-rekursive Funktionen (1933):
primitiv rekursive Funktionen = Nullfunktion, Projektion, Nachfolgerfunktion; Komposition; Rekursion – Laufzeit vorher abschätzbar

Bsp. für nicht-primitiv-rekursive Funktion: Ackermann-Funktion (Péter):

$$\begin{aligned} a(0,n) &= n+1 \\ a(m+1,0) &= a(m,1) \\ a(m+1,n+1) &= a(m,a(m+1,n)) \end{aligned}$$

$a(3,4) = 125; a(4,0) = 13, a(4,1) = 65533, a(4,2) = 2^{65536} - 3$

wird benutzt zum Compilertest; Umkehrfunktion praktisch konstant

Erweiterung um μ-Operator:

$$\mu f(x_1,\ldots,x_k) := \begin{cases} \min M(f,x_1,\ldots,x_k) & M(f,x_1,\ldots,x_k) \neq \emptyset \\ \text{undef.} & \text{sonst} \end{cases}$$

Für $f : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ ist $M(f,x_1,\ldots,x_k)$ folgendermaßen definiert:

$$\{n \in \mathbb{N} \mid f(x_1,\ldots,x_k,n) = 0 \wedge \forall 0 \leq m \leq n : f(x_1,\ldots,x_k,m) \neq \text{undef.}\}$$

(4) Registermaschinen mit mind. 2 Registern („2-counter machine“) (1963):
n Register $R_1,\ldots,R_n$ mit je einer natürlichen Zahl, Inkrementierung, modifizierte Dekrementierung, Null-Test; Eingabe: R_1 bis R_{n-1}, Ausgabe: R_n

(5) GOTO-Programme: Entsprechen faktisch der Maschinensprache; Backus-Naur-Form (BNF) für k Konstanten, m Variablen und n Programmzeilen

$$
\begin{aligned}
P &::= \ell_1 : A; \ldots; \ell_{n-1} : A; \ell_n : HALT \\
A &::= X := X \pm C \mid \text{GOTO } L \mid \text{IF } X = C \text{ THEN GOTO } L \\
\pm &::= + \mid - \\
X &::= x_1 \mid \ldots \mid x_m \\
C &::= c_1 \mid \ldots \mid c_k \\
L &::= \ell_1 \mid \ldots \mid \ell_n
\end{aligned}
$$

Vorstufe: LOOP-Programme (Mächtigkeit: prim. rek. Funktionen):

$$P ::= P; P \mid X := X \pm C \mid \text{LOOP } X \text{ DO } P \text{ END}$$

(6) WHILE-Programme:

$$P ::= P; P \mid X := X \pm C \mid \text{IF } X = C \text{ THEN } P \mid \text{WHILE } X \neq 0 \text{ DO } P \text{ END}$$

Umwandlung eines n-GOTO-Programms $\ell_1 : P_1; \ldots; \ell_n : P_n$: Sei x' neu.

$x' := 1\,;$
WHILE $x' \neq 0$ DO
  IF $x' = 1$ THEN $A_1\,;$
  IF $x' = 2$ THEN $A_2\,;$
  $\vdots$
  IF $x' = (n-1)$ THEN $A_{n-1}\,;$
  IF $x' = n$ THEN $x' := 0\,;$
END

$$
A_i \equiv \begin{cases} x_j := x_k \pm c_l; x' := x' + 1 & P_i \equiv x_j := x_k \pm c_l \\ x' := k & P_i \equiv \text{GOTO } \ell_k \\ x' := x' + 1; \text{IF } x_j = c \text{ THEN } x' := k & P_i \equiv \text{IF } x_j = c \text{ THEN GOTO } \ell_k \end{cases}
$$

Das entstandene Programm benötigt sogar nur eine WHILE-Schleife.

Betrachte ein WHILE-Programm: $P_1; \text{WHILE } x' \neq 0 \text{ DO } P_2 \text{ END}; P_3$.

ℓ'_1: $P_1\,;$
ℓ_2: IF $x' = 0$ THEN GOTO $\ell'_5\,;$
ℓ'_3: $P_2\,;$
ℓ_4: GOTO $\ell_2\,;$
ℓ'_5: $P_3\,;$
ℓ_6: HALT