Vorlesungsskript Theoretische Informatik

vom 12.01.2001

Kleene'sche Normalformtheorem −

TM ⇔ partiell rekursive Funktionen ⇔ Ch−0
|
total rekursive Funktionen
|
primitiv rekursive Funktionen

Motivation / Erläuterung bezüglich Programmiersprachen:

- primitive Rekursion entspricht in Programmiersprachen der for−Schleife
- Alle primitiv rekursiven Funktionen, die ohne Operator »primitive Rekursion« auskommen haben die Schachtelungstiefe 0
- jede primitiv rekursive Funktion, die das primitive Rekursionsschema mit Funktionen der Schachtelungstiefe i zur Definition verwendet hat die Schachtelungstiefe i+1

Bsp.:

Fkt		**Schachtelungstiefe**
Add	x+y = (x+(y−1))+1	
Sub 1		1
Mult	x*y=(x*(y−1))+x	2
Sub		
Exp		3

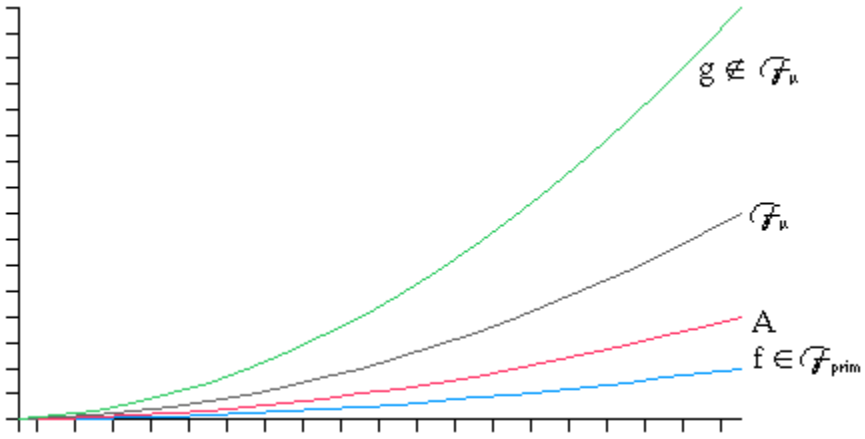
Aus Sicht von Pascal bedeutet dies:

Wenn man von den Grundfunktionen ausgeht, so muß zur Realisierung der Addition eine for−Schleife, zur Realisierung der Multiplikation zwei ineinander verschachtelte for−Schleifen usw. verwendet werden.
(~~~> $\mathcal{F}_{prim}$ wird dadurch hirarchisiert.)

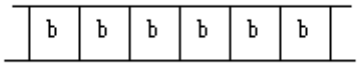
~~~> $\mathcal{F}_{prim}$ ⊂ $\mathcal{F}_{\mu}$ ⊃ $\mathcal{R}_{\mu}$
totale Funktionen <~~~> total rekursive Funktionen ????

Satz: $\mathcal{F}_{prim} \subset \mathcal{R}_{\mu}$

Beweisidee: »⊆« ist klar

um »≠« zu zeigen, geht man über die Schachtelungstiefe.
Man definiert eine Funktion, die die Shachtelungstiefe als Eingabe erhält :
A(x,y) mit A: $\mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$,
wobei das Programm von A die Schachtelungstiefe y besitzt und für x ausgewertet wird.
A kann nicht primitiv rekursiv sein weil ein for−Schleifenprogramm eine feste Schachtelungstiefe hat.

Bekannteste Fassung für A: Ackermannfunktion

A(x, 0) = x+1
A(0, y) = A(1, y−1) |∀ y ∈ ℕ
A(x+1, y+1) = A(A(x, y+1), y) |∀ x,y ∈ ℕ

A(x, 0) = x+1
A(x, 1) = x+2
A(x, 2) = 2x+3

$A(x, 3) = 2^{x+3} - 3$
$A(4, 4) = 10^{10^{10^{21000}}}$
~~~

Zeige: Zu jeder primitiv rekursiven Funktion f: $\mathbb{N}_0 \to \mathbb{N}_0$ gibt es ein $n_f \in \mathbb{N}_0$ und ein $y_f \in \mathbb{N}$, so daß für alle $n \geq n_f$ gilt $f(n) < A(n, y_f)$.
Wähle für y_f die Schachtelungstiefe von f zuzüglich 2.

Einschub: Wachstumsprozesse sind oft zur Anschauung von Funktionsklassen geeignet.



Γ ={|, b}



f: $\mathbb{N} \to \mathbb{N}$
f(n) = maximale Zahl von Striche die BB−Maschine (Busy Beaver) mit n Zuständen schreibt.
f(5) ~ 16 Mio.

6.4. Grammatiken

Frage: Wie ordnen sich Grammatiken in die Berechnungsmodelle ein ?

Problem: Beziehung zwischen Sprachen (erzeugt durch Grammatiken) und Turingmaschinen, Registermaschinen, partiell rekursiven Funktionen (die Funktionen berechnen)herstellen.

Lösung: Übergang zu charakteristischen Funktionen, die als Bilder nur noch zwei Symbole haben (für $w \in L(G)$, $w \notin L(G)$).

Def. P: Sei T ein Alphabet und $t \in T$ ein beliebiges, aber fest gewähltes Zeichen.
Sei $L \subseteq T^*$ eine Sprache.

i) Die totale Abb. χ_L: $T^* \to T^*$ mit $\chi(w) = \begin{cases} t, \text{ falls } w \notin L \\ tt \text{ falls } w \in L \end{cases}$
heiß charakteristische Funktion von L.

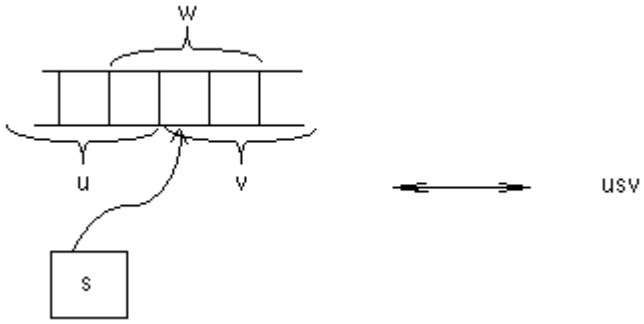
ii) Die partielle Abb. ψ_L: $T^* \to T^*$ mit $\psi(w) = \begin{cases} \text{nicht definiert, falls } w \notin L \\ tt \text{ falls } w \in L \end{cases}$
heiß partiell charakteristische Funktion von L.

Reg <—> DA
CF <—> NPDA
Ch−0 <· · ·> TM − einseitige Akzeptanz in dem Sinne, daß zu beliebiger Sprache L des Typs 0 die Funktion ψ_L aber nicht χ_L berechenbar ist.

Satz Q: eine Sprache L⊆ T* kann genau dann von einer Grammatik erzeugt werden, wenn ψ_L:T* → T* Turingberechenbar ist.

Beweis: »⇒«
Ein Simulationsbeweis: TM <—> Ch−0 Grammatiken

Konfiguration TM <—> Wort aus Sprache

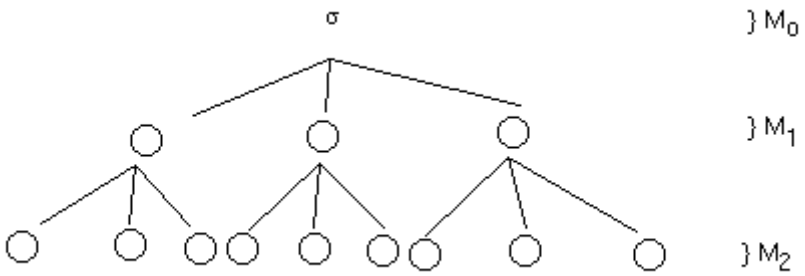


Übergang <—> Grammatikregeln

δ(s,x) = (s', x', RLH) <—> usv' → ux's'v'

Sei G = (N, T, P, σ) eine Grammatik, die L erzeugt, also L = L(G). Man betrachte die Mengen der in i Ableitungsschritten ableitbaren Wörter für i = 0, 1, 2 ...
$M_0 = \{\sigma\}$,
$M_{i+1} = \{u \in (N \cup T)^* | \; w \in M_i \text{ mit } w \rightarrow u \}$

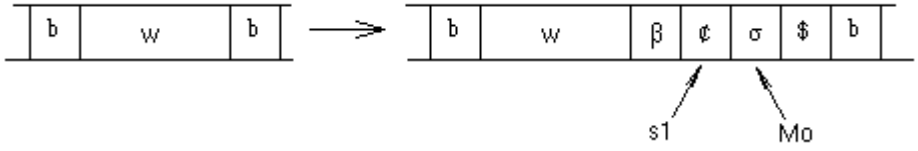
Idee: TM erzeugt schrittweise M_0, M_1, M_2 in Form eines Breitendurchlauf durch den Ableitungsbaum.



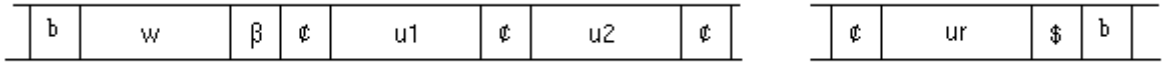
Gilt: $L(G) = \left(\bigcup_{i=0}^{\infty} M_i \right) \cap T^*$

Die TM zur Berechnung von ψ_L arbeitet folgendermaßen:
(Skizze, Zustandsmenge riesig, Zustandsmenge enthält Wissen über Produktionen $p \in P$):

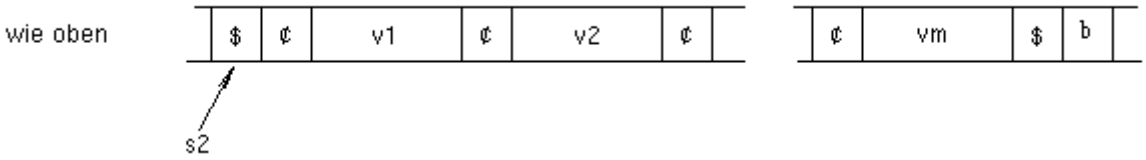
1.) Die TM schreibt die Menge M_0 hinter das eingegebene Wort, als Trennsymbol verwende $\beta \notin (N \cup T)^*$, verwende ¢ und \$ zur Abgrenzung der Wörter aus M_i und M_{i+1} und untereinander.



2.) Wenn die Situation



mit $M_i = \{u_1, ..., u_r\}$ vorliegt, dann schreibt die TM alle aus u_1 ableitbaren Wörter hinter \$, dahinter alle aus u_2 ableitbaren Wörter, bis die Menge M_{i+1} rechts von \$ steht.



$M_{i+1} = \{v_1, ..., v_{rm}\}$
Danach werden die $v_1, ..., v_m$ nach links verschoben und treten an die Stelle der $u_1, ..., u_r$.

3.) Nun vergleiche jedes v_k mit w Falls ein $w = v_k$, so lösche Band und schreibe tt (siehe Def. ψ_L) Stop mit Akzeptieren.
Falls alle $v_k \neq w$, so gehe in Konfiguration mit dem Zustand s_1 über und prüfe nun M_{i+2} gemäß 2.) durch usw.
$w \in L$, so muß Verfahren terminieren:
$\sigma \rightarrow^s w$ und $w \in M_s$. TM hält an und liefert $tt \rightarrow \psi_L(w) = tt$.
Falls $w \notin L, \forall i \in \mathbb{N}_0 : w \notin M_i$. TM stoppt nicht $\rightarrow \psi_L(w) = \perp$.
Also wird ψ_L berechnet.