Mitschrift vom 12.01.2001

25. Januar 2001

Korollar: Kleene'sches Normalformtheorem

Zu jeder Turing-berechenbaren oder RM-berechenbaren oder μ-rekursiven Funktion f, gibt es konstruktiv primitiv rekursive Funktionen g_1,g_2 und h, so daß $f(x) = g_1(g_2(x, \mu h(x)))$.

Man kommt zum Schluß,daß:

$$TM \quad \leftrightarrow \quad \text{partiell} - \text{rekursive Funkionen} \leftrightarrow \quad Ch - 0$$
$$\subsetneqq$$
$$\text{total} - \text{rekursive Funktionen}$$
$$\subsetneqq$$
$$\text{primitiv} - \text{rekursive Funktionen}$$

Die primitive Rekursion wird in den imperativen Programmiersprachen, wie etwa $PASCAL$, durch for-Schleifen umgesetzt. Die Anzahl der ineinander verschachtelten for-Schleifen wird als Schachteltiefe bezeichnet; dabei ist die Schleifenzahl auf einer Ebene unerheblich.

Alle primitiv-rekursive Funktionen, die ohne den Operator primitive Rekursion auskommt haben die Schachteltiefe 0. Dies sind die Konstante, Nachfolger und Projektion. Alle primitiv-rekursiven Funktionen, die das primitive Rekursionschema mit Funktionen der Schachteltiefe i zur Definition verwenden, haben die Schachteltiefe $i + 1$.

So benötigt man beispielsweise für der Umsetzung der Addition als primitiv-rekursives Programm eine for-Schleife. Somit hat die Addition die Schleifentiefe 1. Bei der primitiv-rekursiv definierten Multiplikation handelt es sich um zwei ineinander verschachtelte Additionen (for-Schleifen). Die Schachtlungstiefe beträgt also 2. Die nachfolgende Tabelle gibt die Schachtlungstiefe der bisher behandelten primitiv-rekursiven Funktionen:

Funktion	Schachteltiefe	
Addition	1	$x + y = (x + (y - 1) + 1)$
Sub 1	1	
Mult	2	$x * y = (x * (y - 1)) + x$
Sub	2	
Exp	3	fortgesetzte Mult

Durch die Schachteltiefe sind die primitiv-rekursiven Funktionen F_{prim} hierachisiert.

Bisher wurde gezeigt, daß $F_{prim} \subsetneqq F_\mu \subsetneqq R_\mu$. Im weiteren Verlauf wird nun die Beziehung zwischen den totalen Funktionen (F_{prim}) und den total-rekursiven Funktionen (R_μ) untersucht. Die primitiv-rekursiven Funktionen F_{prim} können durch for-Schleifen dargestellt werden. Schon beim Aufruf der Schleife steht die Anzahl der Durchläufe genau fest. Ein Änderung ist während des Schleifenlaufs nicht möglich. Die total-rekursiven Funktionen R_μ werden bei der Umsetzung als Programm in $while$-Schleifen geschrieben. Bei diesen Schleifen steht die Anzahl der Durchläufe am Anfang des Schleifenaufrufs noch nicht fest. Die Abbruchbedingung ergibt sich, wenn überhaupt, erst während der Laufzeit. Da in R_μ nur totale Funktionen sind, so terminieren auch alle $while$-Schleifen, die R_μ berechnen.

Satz: (über Ackermannfunktion)

$F_{prim} \subsetneqq R_\mu$; die primitiv-rekursiven Funktionen sind eine echte Teilmenge der beliebig μ-rekursiven Funktionen.

Beweisidee: „$\subset$“ Diese Richtung ist durch die Ausführungen in den zuvor gehaltenden Vorlesungen klar.

„$\neq$“ Um die „echte“ Teilmengenbeziehung zu zeigen, wird der Gedanke der festen Schachteltiefe von for- und der unbekannten Schachteltiefe von $while$-Schleifen, am Anfang des Schleifenaufrufes noch einmal aufgegriffen. Für den Nachweis wird die Ackermannfunktion $A(x, y)$ genutzt, die wie folgt definiert ist:

$A : N_0 \times N_0 \to N_0$ mit $A(x, y)$, wobei das Programm für A die Schachteltiefe y besitzt und für x ausgewertet wird. A kann nicht primitiv-rekursiv sein, wegen der erst am Ende der Laufzeit bekannten Schleifentiefe.

Die Ackermann-Funktion dient in der Informatik oft zur Analyse von Berechenbarkeitsproblemen.Sie ist durch die Regeln

$A(x, 0) = x + 1 ; \forall x \in N_0$

$A(0, y) = A(1, y - 1) ; \forall y \in N;$

$A(x + 1, y + 1) = A(A(x, y + 1), y) ; \forall x, y \in N_0$ definiert.

Es ist noch zu zeigen, daß zu jeder primitiv rekursiven Funktion $f : N_0 \to N_0$ eine Schranke $n_f \in N_0$ und ein $y_f \in N$, so daß für alle $n \geq n_f$ gilt: $f(n) < A(n, y_f)$. Dabei ist die Schachtelungstiefe y_f für die Ackermann-Funktion so zu wählen, daß diese größer ist, als die von f. Dies erreicht man z.B. durch die Wahl der Schachtelungstiefe von f zuzüglich 2. Die Ackermann-Funktion wächst dadurch wesentlich schneller an. Verdeutlicht werden soll dies durch ein Beispiel sowie einer Graphik:

Beispiel:

$A(x, 0) = x + 1;$

$A(x, 1) = x + 2, \Rightarrow A(1, 1) = 3;$

$A(x, 2) = 2x + 3, \Rightarrow A(2, 2) = 7;$

$A(x, 3) = 2^{x+3} - 3; \Rightarrow A(3, 3) = 61;$

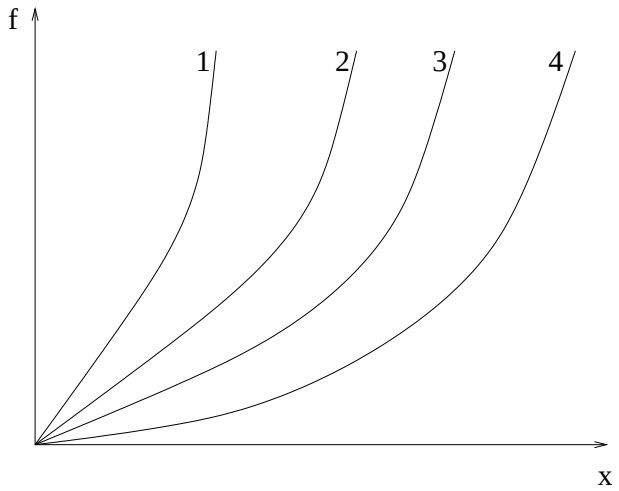
$A(2, 4) = 19729$

$A(4, 4) > 10^{10^{10^{21000}}}$

(Der Wert entspricht den Stellen.)

Die Erhöhung der Schachteltiefe führt zu einer rasanten Vergrößerung der Werte. Die Ackermanfunktion kann von keiner primitiv-rekursiven Funktion nach oben beschränkt werden.

Die folgende Graphik veranschaulicht den Wachstumsprozeß verschiedener Funktionsklassen.



Es wird deutlich, daß der Wachstum $f_4 \in F_{prim}(4) < f_3 \in A(3) < f_2 \in F_\mu(2) < f_1 \notin F_\mu(1)$ ist. Die nicht-berechenbaren Funktionen (1) wachsen am schnellsten. Ein Beispiel für solche nicht-berechenbare Funktionen ist die $Busy - beaver - Funktion.$

$Busy - beaver -$ **Funktion**

Anhand dieser nicht-berechenbaren Funktion soll die Undurchschaubarkeit der Arbeitsweise von Turingmaschinen, selbst bei einfach aussehenden Algorithmen verdeutlicht werden. Es wird versucht mit $s_n \in N_0$ Zuständen einer deterministisch arbeitenden TM möglichst viele | auf das Band zu schreiben, bis ein „Halt“ erreicht wird. Terminiert die TM aufgrund der Zustandsüberführungsfunktionen nicht, so läßt sich dies praktische nicht prüfen. Für n-Zustände sind durch den $Busy - beaver$ nur bis n=4 exakte Werte zu berechnen.

n	0	1	2	3	4	5
$bb(n)$	0	1	4	6	13	$\geq$4901
Schritte (ca.)	0	1		16	96	

Die folgende Turingmaschine $\tau = (\{s_0, s_1, s_2\}, \{\emptyset\}, \{|, b\}, \delta, s_0, b\})$ mit 3 Zuständen terminiert nach 16 Schritten und hat dann 6 Striche auf das Band geschrieben.

δ	Einblick	Aktion
1	(s_0, b)	$(s_1, \|, R)$
2	$(s_0, \|)$	$(s_2, \|, L)$
3	(s_1, b)	$(s_2, \|, R)$
4	$(s_1, \|)$	$(s_0, \|, H)$
5	(s_2, b)	$(s_0, \|, L)$
6	$(s_2, \|)$	$(s_1\|, L)$

6.4. Grammatiken

In diesem Abschnitt werden die vorgestellten Berechnungsmodelle den bekannten Grammatiken zugeordnet. Um eine Beziehung zwischen den durch Grammatiken erzeugten Sprachen und Turingmaschinen bzw. Registermaschinen bzw. partiell-rekursiven Funktionen (die Funktionen berechnen),herstellen zu können, wird der Ausgabewert der Maschine reduziert. Diese charakteristischen Funktionen haben als Bilder (Ausgabe der Maschine) nur noch zwei Symbole. Sie geben aus „JA“,wenn $w \in L(G)$ bzw. „NEIN“, wenn $w \notin L(G)$. Die Ja/Nein-Aussage könnte bei technischen Geräten z.B. über eine leuchtende bzw. nichtleuchtende Lampe erfolgen.

Definition P:

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

(i) Die (totale) Abbildung $\chi_L : T^* \to T^*$ mit $\chi_L(w) = \begin{cases} t, \, falls\, w \notin L \\ tt, \, falls\, w \in L \end{cases}$
heißt charakteristische Funktion von L.

(ii) Die (partielle) Abbildung $\psi_L : T^* \to T^*$ mit $\psi_L(w) = \begin{cases} \perp, \, falls\, w \notin L \\ tt, \, falls\, w \in L \end{cases}$
heißt partiell-charakteristische Funktion von L.

Für jede Sprache läßt sich eine Turingmaschine konstruieren, die $\psi_L(w)$ erkennt, nicht aber $\chi_L(w)$. Bisher wurde die Sprachklassen den Maschinenmodellen zugeordnet:

Sprachklasse		Maschine
Reg	$\leftrightarrow$	DA
CF	$\leftrightarrow$	NPDA
Ch-0	$\leftrightsquigarrow$	TM

Für die Turingmaschinen, die Ch-0 erkennen, bedeutet dies nach der obigen Definition, daß nur eine einseitige Akzeptanz besteht. Es existiert eine TM zu jeder beliebigen Sprache des Chomsky-Typs 0, die

die Funktion ψ_L nicht aber χ_L berechnet werden. Die Maschine beendet ihre Arbeit nur mit Sicherheit, wenn $w \in L$ ist. Im Fall $w \notin L$ terminiert die TM nicht mit Sicherheit. Aus diesem Grund finden auch keine Sprachen des Typ-0 Anwendung in Programmiersprachen, da die Syntax eines Programmes nicht eindeutig als falsch erkannt werden kann. Ein möglicher,aber sehr komplexer Lösungsansatz ist die Bildung einer Komplement-TM.

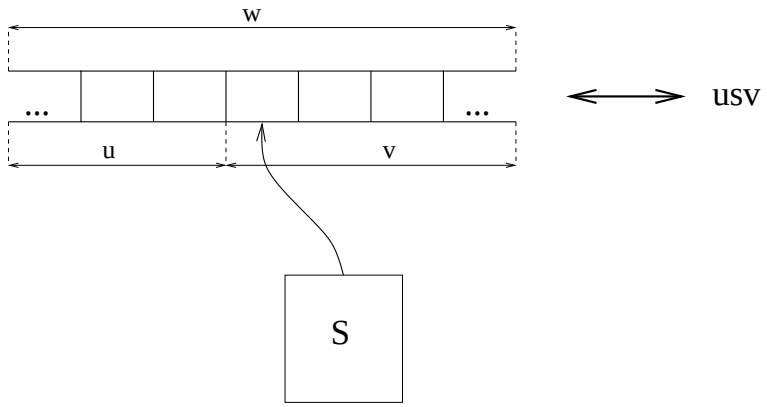
Satz Q:

Eine Sprache $L \subseteq T^*$ kann genau dann von einer Grammatik erzeugt werden, wenn $\psi_L : T^* \to T^*$ Turing-berechenbar ist.

Beweis:

Anmerkung: Es wird ein Simulationsbeweis geführt. Gezeigt wird, daß $TM \leftrightarrow Ch-0-Grammatiken$ ist. Es muß gezeigt werden:

1.) Konfiguration einer TM $\leftrightarrow$Wort der Sprache



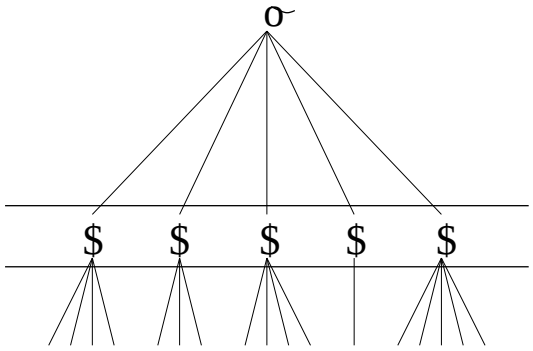
2.) Übergänge der TM $\leftrightarrow$ Grammatikregeln
$\delta(s,x) = (s', x', \{R, L, H\}) \leftrightarrow usxv' \to ux's'v'$.
Im folgenden werden 1. und 2. für Satz Q gezeigt.

„$\Rightarrow$“ Sei $G = (N, T, P, \sigma)$ eine Grammatik, die L erzeugt, also $L = L(G)$. Man betrachtet die Mengen der in i Ableitungsschritten abgeleiteten Wörter für $i = 0, 1, 2, ...$: $M_0 = \{\sigma\}$; $M_{i+1} = \{u \in (N \cup T)^* | \exists w \in M_i$,mit $w \to_G u\}$.

Die Turingmaschine erzeugt M_0,M_1,M_2,... in Form eines Breitendurchlaufs durch den Ableitungsbaum. Es werden jeweils alle möglichen zu erzeugenden Wörter der TM mit der Eingabe w verglichen. Die Vergleiche erfolgen stufenweise, da die astweise Abarbeitung in Endlosschleifen führen kann. Ist $w \notin L$ so wird das Wort auch nicht von der TM erzeugt und sie kann nicht terminieren. Es gilt:

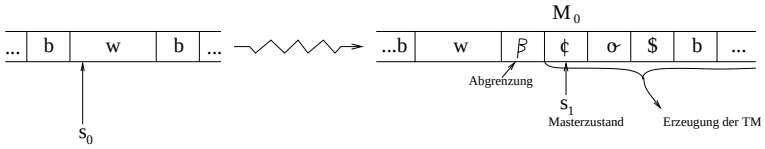
$$L(G) = (\bigcup_{i=0}^{\infty} M_i) \cap T^*$$

Eine solche TM hat eine riesige Zustandsmenge, da in ihr das gesamte Wissen über die Produktionen $p \in P$ enthalten sind. Entsprechend gigantisch fällt auch die Zahl der Zustandsüberführungen für jede Stufe des Breitendurchlaufs aus.

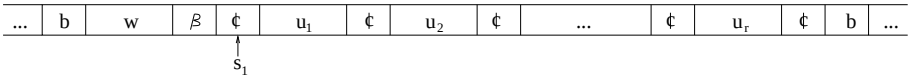


Eine TM zur Berechnung von ψ_L arbeitet folgendermaßen:

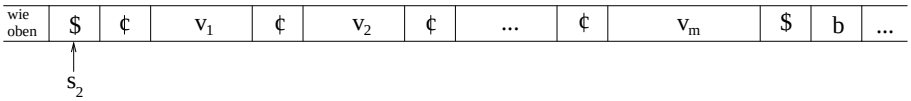
(1) Die TM schreibt die Menge M_0 hinter das eingegebene Wort. Als Trennsymbol wird $\beta \notin N \cup T$ verwendet. Für die Abgrenzung der Wörter M_i und M_{i+1} untereinander werden \$ und ¢ benutzt.



(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, bis die Menge M_{i+1} rechts von \$steht:

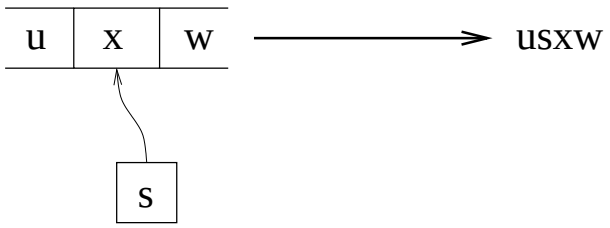


$M_{i+1} = \{v_1, ..., v_m\}$
Danach werden die $v_1, ..., v_m$ nach links verschoben und treten an die Stelle der $u_1, ..., u_r$.

(3) Man vergleiche jedes v_k mit w. Falls ein $w = v_k$ existiert, so lösche das gesamte Band und schreibe tt (nach der Definition von ψ_L). Stoppe und Akzeptiere.
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....

Man sieht, ist $w \in L$, so muß das Verfahren terminieren: $\sigma \to^s w$ und $w \in M_s$. Die TM hält an und liefert $tt \Rightarrow \psi_L(w) = tt$. Falls $w \notin L : \forall i \in N_0 : w \notin M_i$. Die TM stoppt nicht $\Rightarrow \psi_L(w) = \Rightarrow \perp$.
Durch die Turingmaschine wird ψ_L berechnet.

„$\Leftarrow$“ Sei $\tau = (S, X, \Gamma, \delta, s_0, b)$ eine beliebige Turingmaschine. Konstruiere eine Grammatik $G_\tau = (N, T, P, \sigma)$ mit $L(G_\tau) = \{w\#v | Res_\tau(w) = v\}$. Res_τ ist der Wertebereich der TM.
Die Arbeitsweise der Turingmaschine wird mithilfe einer Grammatik simuliert. Dabei soll der jeweilige Zustand der TM vor den Anfang des Wortes geschrieben werden, auf welchem sich der SLK gerade befindet.



Ein Konfigurationsübergang: $\delta(s,x) = (s', x',\ \begin{matrix} L \\ R \end{matrix}\ )\ ysx \rightarrow \begin{matrix} s'yx' & \text{(Linkseinschub)} \\ yx's' & \text{(Rechtseinschub)} \end{matrix}$ für alle $y \in \Gamma$. Die Grammatik G_τ ist wie folgt definiert:

$N = S \cup \{ ¢, \$, s_f, \sigma, \sigma_1, < \text{Nonterminals nach Bedarf} > \}$;

$T = \Gamma \cup \{\#\}$;

P besteht aus mehreren Gruppen von Produktionen folgender Funktionalität:

Gruppe 1: Erzeugung aller Wörter $w\#¢s_0w\$$ mit $w \in X^*$

Diese Grammatik ist kontextsensitiv. Die Produktionsregeln P_1 sind selbst auszuführen.

Gruppe 2: Simulation der TM durch Grammatikregeln auf $¢s_0w\$$

$P_2 = \{sac \rightarrow a's'c | \forall a \in \Gamma, s \in S \text{ mit } \delta(s,a) = s', s', R | \forall c \in \Gamma\}$

$\cup \{sa\$ \rightarrow a's'b\$ | \forall a \in \Gamma, s \in S \text{ mit } \delta(s,a) = (s', a, R)$ //rechts schieben

$\cup \{csa \rightarrow s'ca | \forall a \in \Gamma, s \in S \text{ mit } \delta(s,a) = (s', a', L | \forall c \in \Gamma\}$ //links schieben

$\cup \{ ¢sa \rightarrow ¢s'ba' | \forall a \in \Gamma, s \in S \text{ mit } \delta(s,a) = (s', a', h)\}$

$\cup \{sa \rightarrow s_f a' | \forall a \in \Gamma, s \in S \text{ mit } \delta(s,a) = (s', a', h)\}$

Auf dem Band findet man die Situation $w\#¢us_fv\$ | u, v \in \Gamma^*$.

Gruppe 3: Überflüssige Zeichen löschen

$P_3 = \{s_f \rightarrow ....\}$ auch diese Produktionen sind einfach selbst zu gestalten. Dabei sind die Symbole $¢, \$$ vom Band zu entfernen. Die Markierung s_f verschwindet aufgrund der Arbeitsweise der TM.

Als Ergebnis bleibt $w\#v$ stehen, wenn der Arbeitprozeß stoppt und die Turingmaschine eine Haltesituation erreicht hat. Alle anderen Situationen führen zu Veränderungen, die ohne weitere Folge für das Ergebnis sind. □

Literaturverzeichnis

- Hopcroft, J.E. und Ullman, J.D. (2000). Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. Oldenburg Verlag München
- Erk, K. Pries, L. (2000). Theoretische Informatik - Eine umfassende Einführung. Springer-Verlag
- Wegener, I. (1999). Theoretische Informatik - Eine algorithmische Einführung. B.G.Teubner
- Schwill, A. (1993). Duden Informatik. Bibliographisches Inst. Mannheim