Bsp. LL(1) – Gr $G=(\{Z\},\{a,+,-\},\{a\to Z, -Z\to Z, +ZZ\to Z\},Z)$

Reduktion von $\langle +-a+aa\rangle$ durch KA:

LL-Technik (top-down)		LR-Technik (bottom-up)	
Keller	Resteingabe	Keller	Resteingabe
Z	$+-a+aa$	ε	$+-a+aa$
$ZZ+$	$+-a+aa$	$+$	$-a+aa$
ZZ	$-a+aa$	$+-$	$a+aa$
$ZZ-$	$-a+aa$	$+-a$	$+aa$
ZZ	$a+aa$	$+-Z$	$+aa$
$\vdots$	$\vdots$	$+Z$	$+aa$
a	a	$\vdots$	$\vdots$
ε	ε	Z	ε

jede LL(k)-Gr. ist auch eine LR(k)-Gr., d.h. u.a. jede LL(k)-Gr. ist eindeutig.

1.4.7. Rekursiver Abstieg

klassisches Verfahren der Programmierung eines top-down KA zur Zerteilung von Wörtern einer kfs. Orientiert an BNF:

- Für jedes Nichtterminal $X_i \in N$ der linken Seiten der Regeln $X_i ::= E_1 | \dots | E_n$ eine Prozedur, die (rekursiv) eine vollständige Fallunterscheidung der rechten Seite der zu X_i gehörenden Regel durchführt.

In der Regel ebenfalls Steuerung über Präfix des Restworts.

1.4.8. Das Pumping-Lemma für kfS

KfG $G=(T,N,\to,Z)$ ist in Chomsky-Normalform (CNF), falls alle Regeln folgende Gestalt haben: $a\to A$ oder $BC\to A$, für $a\in T$; $A,B,C\in N$

Satz: Zu jeder Kfg G mit $\varepsilon\notin L(G)$ gibt es eine äquivalente Kfg in Chomsky-Normalform. (ohne Beweis)

Satz: Pumping-Lemma für kfS

Zu jeder kfS L existiert ein $n\in\mathbb{N}$, so dass sich alle $z\in L$, mit $|z|\geq n$, in $u,v,w,x,y\in T^*$ zerlegen lassen:

$z=uovowoxoy$ mit 1. $|vox|\geq 1$
2. $|vowox|\leq n$
3. für alle $i\geq 0$: $uov^iwox^iy\in L$

Beweis: Sei $G=(T,N,\to,N_0)$ CNF-Gr. mit $L(G)=L$, $k=|N|$, $n=2^k$, $z\in L$ mit $|z|\geq n$.

Ableitungsbaum T für z ist Binärbaum der Höhe $h+1$ mit $|z|\geq n$ Blättern

- Knoten mit genau einem Sohn ($a\to A$) oder genau zwei Söhnen ($BC\to A$)
- $h\geq \log_2|z|\geq \log_2 n = k$

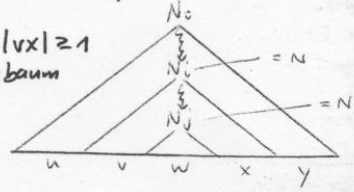
Im Pfad $p_0,\dots,p_{n+1}$ in T von der Wurzel p_0 zu Blatt p_{n+1} ist die Markierung $N_0,N_1,\dots,N_h$ der Knoten $p_0,\dots,p_n$ eine Folge von $h+1\geq k+1$ Nichtterminalen. D.h. es gibt $i,j\in\{0,\dots,h\}$ so dass:

$N_i=N_j=N$, $i<j$ und $N_{i+1},\dots,N_h$ paarweise verschieden

N_i und N_j sind gleiche Nichtterminale

Zerlegung von $z=uovowoxoy$

1. Da G in CNF, ist $v\neq\varepsilon$ oder $x\neq\varepsilon$; d.h. $|vx|\geq 1$
2. Teilbaum T' ab Knoten p_i ist Ableitungsbaum für $N_i \overset{+}{\to} vowox$
   Aus $N_{i+1},\dots,N_h$ verschieden folgt:
   T' hat Höhe $h-i\leq k$
   D.h. $|vowox|\leq 2^{n-i}\leq 2^k=n$



3. Ableitungen für uov^iwox^iy ergeben sich aus der Kombination der Möglichkeiten: $N_0\overset{+}{\to} uoN_oy$, $N\overset{+}{\to} w$, $N\overset{+}{\to} vNx$ o

1.5. Kontextsensitive Grammatiken (KsG)

es gilt: CFL $\subsetneq$ CSL
KsG sind definitionsgemäß wortlängenmonoton (wlm),
d.h. für jeden Schritt $x \Rightarrow y : |y| \geq |x|$.
KsG sind sehr allgemein; jede wortlängenmonotone Grammatik ist strukturäquivalent zu einer KsG! Es ist möglich für eine KsS L einen (ineffizienten) Algorithmus anzugeben, der für $w \in T^*$ entscheidet, ob $w \in L$
Begründung: Bei einer wortlängenmonotonen Gr. ist für jedes $w \in T^*$ die Menge der durch Reduktionen von w entstehenden Wörter endlich, mögliche unendliche Reduktionen können aufgrund der Wortlängen-Monotonie erkannt werden.
Für eine CH-0-G G_0 existiert i.a. kein solcher Algorithmus!
Für G_0 existiert lediglich ein Alg., der für jedes $w \in L(G_0)$ mit Resultat true terminiert. Für $w \notin L(G_0)$ kann die Terminierung nicht garantiert werden.

2 BERECHENBARKEIT

Es gibt mathematisch exakt beschreibbare Probleme, für die es keine Lösungsalgorithmen gibt!
typische Probleme: Aufgabe: Berechenbarkeit einer ges. Funktion f
Lösung: Algorithmus, der f berechnet
Def. Eine Funktion f heißt berechenbar, wenn es einen Algorithmus gibt, der für jedes Argument x (Eingabe) den Wert f(x) (Ausgabe) berechnet.

Nicht berechenbare Fkt. können nicht durch Programme einer Rechenanlage beschrieben werden!
Wir betrachten o.B.d.A. n-stellige Funktionen $f: \mathbb{N}^n \to \mathbb{N}$, wobei lediglich Repräsentationen von $n \in \mathbb{N}$ zur Verfügung stehen. (siehe Einführung in die Informatik I). Wir verwenden die Zeichenmenge T und die injektive, umkehrbare Abbildung $rep: \mathbb{N} \to T^*$ (Repräsentations-Abbildung).
Demnach existiert auch $abs: \{t \in T^* : \exists\, n \in \mathbb{N} : rep(n) = t\} \to \mathbb{N}$ (Abstraktions-Abbildung),
so dass $abs \circ rep = id$ und
statt abstraktem $f: \mathbb{N}^n \to \mathbb{N}$ betrachten wir ein konkretes
$\bar{f}: (T^*)^n \to T^*$, mit $f(x_1, \ldots, x_n) = abs(\bar{f}(rep(x_1), \ldots, rep(x_n)))$.
Konkrete Algorithmen arbeiten auf Repräsentationen (von $\mathbb{N}$)
Für einen realitischen Berechenbarkeitsbegriff müssen diese Rep. handhabbar sein. Wir setzen deshalb u.a. voraus, dass T endlich ist.

2.1. Hypothetische Maschinen

Wir suchen eine Möglichkeit zur Präzisierung des Algorithmen-Begriffs:
Hypothetische Maschinen:
Mathematische Nachbildung des Zustandsraums und der Übergangsfunktion realer Maschinen mit dem Ziel der Präzisierung und Vereinfachung.

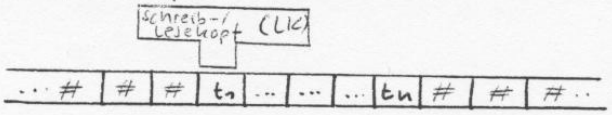
- Endliche Automaten (EA) und
- Keller-Automaten (KA)

genügen nicht; u.a. können KA für Kfs aber nicht für Kss L berechnen, ob $w \in L$. Wir betrachten allgemeinere Modelle als KA mit unbeschränkter Menge an Zuständen.

mögliche Kritik: unrealistisch, weil technisch nicht möglich ...?
(mit der Frage beschäftigen wir uns später)

2.1.1. Turing-Maschine (A. M. Turing, 1936)

Beidseitiges unendliches Band von Zellen zur Speicherung von Zeichen, ein Schreib-/Lesekopf, Steuereinheit.



Ein endlicher Abschnitt des Bands trägt relevante Information, alle anderen Zellen enthalten das Symbol # für die leere Information.

Prinzip: Turing-Maschine (TM) liest in jedem Schritt ein Zeichen $t \in T \cup \{\#\}$ unter dem Kopf. Abhängig vom Zustand wird das Zeichen überschrieben, ein neuer Zustand eingenommen und der Kopf um höchstens eine Position nach links oder rechts bewegt.

Anmerkung: Turing-Maschinen sind sehr mächtig! Jeder Algorithmus der berechenbar ist, kann durch eine TM dargestellt werden (Church These)

Eine Turing-Maschine $TM = (T, S, \delta, s_0)$ umfasst:

- endliche Menge T von Zeichen, mit denen das Band beschrieben wird (es sei $\# \notin T$)
- endliche Menge S von Zuständen
- (endliche) Übergangsfunktion / -relation
  $\delta : S \times (T \cup \{\#\}) \to \mathcal{P}(S \times (T \cup \{\#\}) \times \{<<, >>, \downarrow\})$
  „<<" bzw. „>>" bewirken eine Verschiebung des Kopfes um eine Zelle nach links bzw. rechts und „↓" das Beibehalten der Position.

Gilt für TM: $\forall t \in T \cup \{\#\}, s \in S : |\delta(s,t)| \leq 1$, so heißt TM deterministisch.

Beispiel Prüfung, ob Anzahl der „L" in $\langle w_1 \dots w_n \rangle$, $n \geq 1$, $w_i \in \{0,1\}$ gerade.
Zu Beginn sei $\dots \# w_1 \dots w_n \# \dots$ auf Band und der Kopf auf w_n positioniert.
TM hält mit Bandinhalt $\dots \# L \# \dots$ falls (die Anzahl L?) gerade bzw. mit $\dots \# 0 \# \dots$ sonst.

$TM = (\{0, L\}, \{s_0, s_1, s_2, s_3\}, \delta, s_0)$

Übergangsfunktion δ:

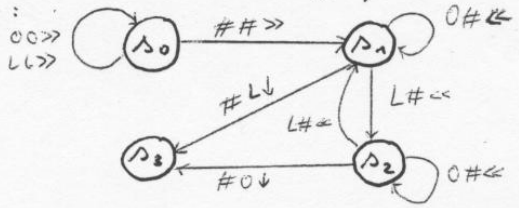
δ	0	L	#
s_0	$(s_0, 0, >>)$	$(s_0, L, >>)$	$(s_1, \#, <<)$
s_1	$(s_1, \#, <<)$	$(s_2, \#, <<)$	$(s_3, L, \downarrow)$
s_2	$(s_2, \#, <<)$	$(s_1, \#, <<)$	$(s_3, 0, \downarrow)$
s_3	$\emptyset$	$\emptyset$	$\emptyset$

Eingabezeichen (Lesekopf) zeichnet das geschriebene / Kopfbewegung

s_3 Endzustand ergebnis 0

16.5.2003 (Freitag)

Das Verhalten einer TM kann auch graphisch durch einen Automaten dargestellt werden:

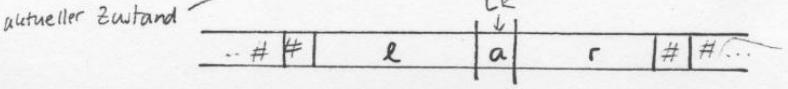


(graphische Darstellung des Automaten aus dem Bsp)

Die Markierung $0 \# <<$ für einen Zustandsübergang s nach s' bedeutet, dass dieser Übergang im Zustand s ausgeführt werden kann, falls 0 unter dem Lesekopf (LK) steht; dann wird # geschrieben und der LK bewegt sich nach links.

Ob ein Zustandsübergang möglich ist, ist abhängig von dem Zeichen, das unter dem Lesekopf steht → unendlicher Zustandsraum → unendlicher Automat. (! kein EA !)

Eine Konfiguration einer TM beschreibt den Berechnungszustand. Wir verwenden ein 4-Tupel: $(s, l, a, r) \in S \times (T \cup \{\#\})^* \times (T \cup \{\#\}) \times (T \cup \{\#\})^*$



Da das Band nach links und nach rechts unendlich fortgesetzt ist, sind folgende Konfigurationen äquivalent:

(s, l, a, r)
$(s, \langle\#\rangle \circ l, a, r)$
$(s, l, a, r \circ \langle\#\rangle)$

} d.h. l und r können um bel. viele #-Zeichen erweitert werden.

Die Berechnung einer Turing-Maschine (ausgehend von einer Konfiguration; d.h. einem Anfangszustand, einer (endl.) Anfangsbandbelegung und Stellung des LK) besteht aus einer endlichen oder unendlichen Folge von Konfigurationen $K_0 \to K_1 \to K_2 \to \ldots$
wir definieren eine Relation auf Konfigurationen $(s,l,a,r) \to (s',l',a',r')$ wie folgt: Es gilt genau eine der folgenden Aussagen:

(0) $z = \downarrow \wedge l = l' \wedge r = r' \wedge a' = x$ — Kopf bleibt
(1) $z = \ll \wedge l = l' \circ \langle a' \rangle \wedge r' = \langle x \rangle \circ r$ — Kopf nach links
(2) $z = \gg \wedge l' = l \circ \langle x \rangle \wedge r = \langle a' \rangle \circ r'$ — Kopf nach rechts.

wobei $(s', x, z) \in \delta(s, a)$

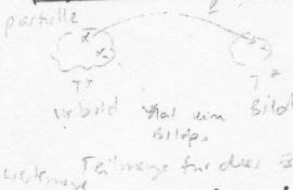
Eine Konfiguration K heißt terminal, wenn keine Nachfolgekonfiguration existiert. Die TM bleibt stehen, die Berechnung endet.
Eine Berechnung heißt vollständig, wenn sie unendlich ist oder endlich ist und die letzte Konfiguration terminal ist.

Bsp. $K_0 \to K_1 \to K_2 \to \ldots \to K_n$ ← terminal, keine Nachfolgekonfiguration

" auch eine Berechnung, aber nicht Terminal → unvollständige Berechnung!

Bemerkung: Jede TM lässt sich in einer der gebräuchlichen Programmiersprachen (Java, Pascal, C, etc) simulieren.
Wir stützten nun auf TM den Begriff der Berechenbarkeit ab (genauer Turing-Berechenbarkeit):
Eine partielle Funktion $f: T^* \to T^*$ (bildet Wörter auf Wörter ab) heißt Turing-berechenbar, wenn es eine deterministische TM gibt, so dass für jedes Wort $t \in T^*$ eine der folgenden Aussagen gilt:

(1) $f(t)$ hat einen Bildpunkt ($f(t)$ ist definiert, genauer der Wert von f für Argument t ist definiert) und es existiert eine vollständige Berechnung: $(s_0, t, \#, \varepsilon) \to \ldots \to (s_e, r, \#, \varepsilon)$, wobei $f(t) = r$.
(2) f ist für Argument t nicht definiert und es existiert eine unendliche Berechnung $(s_0, t, \#, \varepsilon) \to \ldots \to \ldots \to \ldots \to \ldots$



Dieser Berechenbarkeitsbegriff lässt sich auf partielle Abb. $f: \mathbb{N}^n \to \mathbb{N}$ übertragen: Dazu müssen wir nur eine Zahldarstellung festlegen.
Wir stellen natürliche Zahlen durch Strichzahlen dar und verwenden "␣" als Trennzeichen. f heißt berechenbar, wenn (crne Fkt) $g: \{|, ␣\}^* \to \{|, ␣\}^*$ existiert, die berechenbar ist und für alle $x_1, \ldots, x_n \in \mathbb{N}$ gilt:
$g(\langle ␣ \rangle \cdot |^{x_1} \cdot \langle ␣ \rangle \cdot \ldots \cdot \langle ␣ \rangle |^{x_n} \langle ␣ \rangle) = \langle ␣ \rangle \cdot | \langle ␣ \rangle$ g.d.w. $f(x_1, \ldots, x_n) = y$.
f heißt dann genauer Turing-Berechenbar.

Einige Daten zu Alan Turing

- Turing-Test:
  - Test „künstliche Intelligenz": wenn ein Mensch ein Programm für einen Menschen hält, ist dies künstlich intelligent. Bis jetzt hat noch kein Programm den Test bestanden
  - Enigma-Entschlüsselung im 2. Weltkrieg mit „eistem" halbelektr. Rechner Bomba
  - Turing Berechenbarkeit (1936)

geboren 1912, gestorben 1954

Einige Beispiel für TM-berechenbare Funktionen

(1) Konstante: $c : \mathbb{N}^n \to \mathbb{N}$ mit $c(x_1, \dots, x_n) = k$ für geg. $k \in \mathbb{N}$

(2) Nachfolgerfunktion: $succ : \mathbb{N} \to \mathbb{N}$ mit $succ(n) = n+1$

(3) Projektionen: $(1 \le i \le n)$ $\Pi_i^n : \mathbb{N}^n \to \mathbb{N}$ mit $\Pi_i^n(x_1, \dots, x_n) = x_i$

(4) Vorgänger: $pre : \mathbb{N} \to \mathbb{N}$ total

$pred : \mathbb{N} \to \mathbb{N}$ partiell

$$pre(x) = \begin{cases} x-1 & \text{falls } x > 0 \\ 0 & \text{sonst (falls } x = 0) \end{cases}$$

$$pred(x) = \begin{cases} x-1 & \text{falls } x > 0 \\ \text{undef.} & \text{sonst} \end{cases}$$

Komplexere Funktionen können wir durch Komposition (Funktionskomposition) gewinnen.

Beispiel $g, h : \mathbb{N} \to \mathbb{N}$ $f =$ falls h def. für x.

$f : \mathbb{N} \to \mathbb{N}$ f ht def. für x falls h nicht def. für x

f Komposition von g und h, g und h TM-berechenbar → f TM-berechenbar.

Wir definieren eine verallgemeinerte Komposition für partielle Funktionen:

gegeben: $g : \mathbb{N}^n \to \mathbb{N}$, $h_i : \mathbb{N}^m \to \mathbb{N}$ $1 \le i \le n$.

wir definieren $f : \mathbb{N}^m \to \mathbb{N}$ durch die Gleichung

$f(x_1, \dots, x_m) = g(h_1(x_1, \dots x_m), \dots, h_n(x_1, \dots, x_m))$ falls $h_1, \dots, h_n$ für $(x_1, \dots, x_m)$ definiert und g für $h_1(x_1, \dots, x_m), \dots, h_n(x_1, \dots, x_m)$ definiert, sonst ist f für $(x_1, \dots x_m)$ undefiniert.

Satz: f ist TM-berechenbar, wenn $g, h_1, \dots, h_n$ TM-berechenbar sind.

Notation: Wir schreiben dann für f auch $g \circ [h_1, \dots, h_n]$

Es gilt: Addition, Multiplikation, Division (d.h. die übliche Arithmetik) ist TM-berechenbar. Ebenso Fakultät und Fibonacci

Frage: Gibt es Funktionen, die nicht TM-berechenbar sind?
Ja, es sind auch solche Funktionen TM-berechenbar, die man nicht programmieren kann. → Begründung siehe später

„berechenbar" Methoden: TM-Berechenbarkeit, Termersetzung, Textersetzung, Programmiersprachen, rekursive Funktionen.
Berechenbarkeit? → Churchsche These

Bemerkung:
- Ob wir Turing-Maschinen mit mehreren Bändern oder nur einseitig unendlichen TMen verwenden ändert nichts am Begriff der Turing-Berechenbarkeit.
- Andere Berechenbarkeitsbegriffe haben sich als äquivalent erwiesen. Dazu folgen zwei Beispiele.
  (bzw. zu mächtigeren Begriffen ex. keine Maschine → schlecht!).

2.1.2. Register-Maschinen (RM)

Eine Register Maschine ist (wie TM) eine hypothetische Maschine (math. def.), die eher unseren gebräuchlichen Rechnern entspricht.
Eine Register-Maschine mit n Registern (n-RM) besitzt n Register (Speicherplätze) für natürliche Zahlen und ein Programm (ein n-RM-Programm).
Diese Programme haben eine extrem einfache Syntax:

(0) ε ist ein n-RM-Prog (leeres Programm)
(1) $succ_i$ mit $1 \le i \le n$ ist ein n-RM-Prog
(2) $pred_i$ " " " " " "
(3) sind M_1 und M_2 n-RM-Prog., so ist $M_1; M_2$ ein n-RM-Prog.
(4) ist M ein n-RM-Prog., so ist $while_i(M)$ mit $1 \le i \le n$ ein n-RM-Prog.