Automaten und Formale Sprachen

SoSe 2013 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

11. Juli 2013

Automaten und Formale Sprachen — Gesamtübersicht

- Organisatorisches
- Einführung
- Endliche Automaten und reguläre Sprachen
- Kontextfreie Grammatiken und kontrextfreie Sprachen
- Chomsky-Hierarchie

Noam Chomsky

Mehr Infos unter `www.chomsky.info`

Eine **(Phrasenstruktur-)Grammatik** ist ein Quadrupel $G = (\Sigma, N, R, S)$ mit:

- Σ ist das *Terminalalphabet*,
- N ist das *Nonterminalalphabet* (die *Variablenmenge*),
- $R \subset (\Sigma \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ ist das Alphabet der *Regeln*; übliche Schreibweise: $xAy \to v$ anstelle von $(xAy, v) \in R$, wobei $A \in N$ und $x, y, v \in (\Sigma \cup N)^*$ auch *linke Seite* bzw. *rechte Seite* der Regel heißen.
- $S \in N$ ist das *Startsymbol* oder *Anfangszeichen*.

Ein Wort über dem *Gesamtalphabet* $(\Sigma \cup N)$ heißt auch *Satzform*.

Der **Ableitungsmechanismus** einer Grammatik:

1-Schritt-Ableitungsrelation $\Rightarrow_G$
zwischen zwei Satzformen u, v einer Grammatik G:
$u \Rightarrow_G v$ (manchmal kurz $u \Rightarrow v$) gdw.
es gibt Regel $\alpha \to y$, sodass u und v wie folgt zerlegt werden können: $u = x\alpha z$
und $v = xyz$; hierbei sind x und z wiederum Satzformen.

Etwas formaler, ggb. $G = (\Sigma, N, R, S)$:

$$\forall u, v \in (\Sigma \cup N)^* : u \Rightarrow_G v \iff (\exists x, z \in (\Sigma \cup N)^* \exists (\alpha \to y) \in R : u = x\alpha z \wedge v = xyz)$$

$\Rightarrow^n$: n-Schritt-Ableitungsrelation.
$\overset{*}{\Rightarrow}$: Ableitung mit beliebig vielen Schritten.

Die von einer Grammatik G *erzeugte* oder *abgeleitete* Sprache ist gegeben durch:

$$L(G) := \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}.$$

Bequeme Schreibweise einer *Ableitung(sfolge)*:

$$u_0 \Rightarrow u_1 \Rightarrow u_2 \Rightarrow u_3$$

Gilt $u \overset{*}{\Rightarrow} v$, so gilt für ein k: $u \Rightarrow^k v$, bezeugt durch die Ableitungsfolge $u = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_{k-1} \Rightarrow u_k = v$. Dieses k heißt auch *Länge der Ableitung*.

RA: Familie der *(rekursiv) aufzählbaren Sprachen*, engl. *recursively enumerable*, (der durch Grammatiken erzeugbaren Sprachen).

Spezialfälle: MON und **KS**.

Eine Grammatik heißt *monoton* oder *nichtverkürzend* gdw. für alle Regeln gilt, dass die rechte Regelseite nicht kürzer als die linke ist.

Eine monotone Grammatik heißt *kontextsensitiv*, wenn alle Regeln von der Form $\alpha A \beta \to \alpha w \beta$ sind für irgendein $A \in N$.

Hinweis: Bei *kontextfreien* Regeln gilt $\alpha = \beta = \lambda$.

Sonderfall λ-Regeln: Um die Ableitung des leeren Wortes zu ermöglichen, kann eine nichtmonotone Regel $S \to \lambda$ für das Startzeichen S bei **MON** und bei **KS** zugelassen werden. Dann darf aber S auf keiner rechten Regelseite vorkommen.

Ein Beispiel für eine monotone Grammatik:

$S \to aY_abc$, $S \to abc$, $S \to \lambda$,
$aY_a \to aabY_c$, $bY_a \to Y_ab$,
$Y_cb \to bY_c$, $Y_cc \to Y_acc$, $Y_cc \to cc$

Eine Beispielableitung:
$S \Rightarrow aY_abc \Rightarrow aabY_cbc \Rightarrow aabbY_cc \Rightarrow aabbY_acc \Rightarrow aabY_abcc \Rightarrow aaY_abbcc \Rightarrow$
$aaabY_cbbcc \Rightarrow aaabbY_cbcc \Rightarrow aaabbbY_ccc \Rightarrow aaabbbccc$

Erzeugte Sprache: $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$.

Wie beweisen wir Sprachgleichheit ?

Standard, getrennte Induktionen für $L(G) \subseteq L$ und $L \subseteq L(G)$.
Alternativ / Hilfsmittel: Kennzeichnung der ableitbaren Satzformen.

Lemma: Für $L_k := \{w \mid a^k Y_a b^k c^k \Rightarrow^{k+2} w\}$ gilt:
$L_k = \{a^{k+1}b^{k+1}Y_a c^{k+1}, a^{k+1}b^{k+1}c^{k+1}\}$ für jedes $k \geq 1$.

Beweis: Für $k = 1$ durch Inspektion (keine anderen Regeln anwendbar!):

$$aY_abc \Rightarrow aabY_cbc \Rightarrow aabbY_cc \Rightarrow \begin{cases} a^2b^2c^2 \\ a^2b^2Y_acc \end{cases}$$

Als Zwischenbehauptung kann man per Induktion zeigen für alle u, v zeigen:
$uY_cb^\ell cv \Rightarrow^{\ell+1} ub^\ell ccv$ oder $uY_cb^\ell cv \Rightarrow^{\ell+1} ub^\ell Y_accv$ (und keine anderen Wörter sind in $\ell + 1$ Schritten ableitbar).

Der Induktionsanfang ergibt sich wieder durch Inspektion der Regeln.

Für den Induktionsschritt beobachte:

$uY_cb^{\ell+1}cv \Rightarrow ubY_cb^{\ell}cv$; mit $u' = ub$ können wir die Induktionsannahme anwenden und erhalten so: $u'Y_cb^{\ell}cv \Rightarrow^{\ell+1} ub^{\ell}ccv$ oder $u'Y_cb^{\ell}cv \Rightarrow^{\ell+1} ub^{\ell}Y_accv$ (und keine anderen Wörter). ✓

Inspektion liefert nun: $a^kY_ab^kc^k \Rightarrow a^{k+1}bY_cb^kc^k \Rightarrow^{k+1} w$ mit $w \in L_k$ (und nichts sonst) mit Hilfe der Zwischenbehauptung. □

Einfacher sogar kann man per Induktion nachweisen:

Lemma: Aus $a^kb^kY_ac^k$ ergibt sich nach k Ableitungsschritten $a^kY_ab^kc^k$ (und keine andere Satzform).

Ein nochmaliges Nachvollziehen der Beweise beider Lemmata liefert:

Lemma: Die einzigen in den beiden Lemmata beobachteten Satzformen, die nur aus Terminalzeichen bestehen, sind explizit im ersten Lemma angegeben.

Beweis: (der eigentlichen Sprachgleichheit)

Wegen $S \to \lambda$ und $S \to abc$ sind diese zwei Wörter aus L in G ableitbar.

$S \Rightarrow aY_abc$ gestattet die Anwendung der Lemmata, die beweisen:

Die einzige in genau $2k+2$ *Schritten aus* $a^kY_ab^kc^k$ *ableitbare Satzform ist* $a^{k+1}Y_ab^{k+1}c^{k+1}$.

Daher sind mit dem ersten Lemma alle Wörter aus L (und keine anderen wegen des dritten Lemmas) in G ableitbar. □

Eine Normalform für monotone Grammatiken (ähnlich bei Typ-0)

Satz: Zu jeder monotonen Grammatik gibt es eine äquivalente, bei der alle Regeln mit Terminalzeichen a von der From $A \to a$ sind.

Beweis: Es sei $G = (\Sigma, N, R, S)$ eine monotone Grammatik.
Für jedes Terminalzeichen a führe ein neues Nichtterminalzeichen X_a ein.
$M := \{X_a \mid a \in \Sigma\}$; $N' = N \cup M$.
Definiere Morphismus $h : (\Sigma \cup N)^* \to (M \cup N)^*$: $h(a) = X_a$ für $a \in \Sigma$ und $h(A) = A$ für $A \in N$.
$R' := \{A \to h(\alpha) \mid A \to \alpha \in R\} \cup \{X_a \to a \mid a \in \Sigma\}$.
In $G' := (\Sigma, N', R', S)$ kann eine Ableitung von G simuliert werden:
1. Phase: Simulation auf "Nichtterminalebene"
2. Phase: Verwandlung der X_a in entsprechende a-Terminale.
Da keine kontextabhängigen Regeln mit Terminalen auf der linken Seite existieren, kann jede terminierende Ableitung, in der $X_a \to a$-Regeln vor eigentlichen Simulationsregeln angewendet werden, umgeordnet werden in 1. und 2. Phase. □

Eine monotone NF-Grammatik für dieselbe Sprache

$S \to X_a Y_a X_b X_c$, $S \to X_a X_b X_c$, $S \to \lambda$,
$X_a Y_a \to X_a X_a X_b Y_c$, $X_b Y_a \to Y_a X_b$,
$Y_c X_b \to X_b Y_c$, $Y_c X_c \to Y_a X_c X_c$, $Y_c X_c \to X_c X_c$
$X_a \to a$, $X_b \to b$, $X_c \to c$.

Eine Beispielableitung:

$S \Rightarrow X_a Y_a X_b X_c \Rightarrow X_a X_a X_b Y_c X_b X_c \Rightarrow X_a X_a X_b X_b Y_c X_c \Rightarrow X_a X_a X_b X_b Y_a X_c X_c$
$\Rightarrow X_a X_a X_b Y_a X_b X_c X_c \Rightarrow X_a X_a Y_a X_b X_b X_c X_c$
$\Rightarrow X_a X_a X_a X_b Y_c X_b X_b X_c X_c \Rightarrow X_a X_a X_a X_b X_b Y_c X_b X_c X_c \Rightarrow X_a X_a X_a X_b X_b X_b Y_c X_c X_c$
$\Rightarrow X_a X_a X_a X_b X_b X_b X_c X_c X_c \overset{*}{\Rightarrow} aaabbbccc$

Eine Automatensicht auf monotone Grammatiken (skizzenhaft)

Grammatiken mit “Lesekopf” und Links- / Rechtsbegrenzung.
Der Lesekopf wandert wie ein Weberschiffchen hin und her.

Neue Regeln: $S' \to \$_L K_R S \$_R$ (beginnt Rechtsbewegung)
$AK_L \to K_L A$, $K_R A \to A K_R$ für alle Zeichen A.
$K_R \alpha \to K_R^{r:1} \alpha$ für “ursprüngliche” Regel $r = \alpha \to w$.
Es sei $\alpha = A_1 \dots A_n$ und $w = B_1 \dots B_m$ mit $m \geq n$.
$K_R^{r:i} A_i \to B_i K_R^{r:i+1}$ für $1 \leq i < n$; $K_R^{r:n} A_n \to K_L B_n \dots B_m$
$\$_L K_L \to \$ K_R$ sowie $K_R \$_R \to K \$_R$.

Man kann zeigen: $S \overset{*}{\Rightarrow} w,\ w \in \Sigma^*$ gdw. $S' \overset{*}{\Rightarrow} \$_L w K \$_R$.
Eine Ersetzung wird nur simuliert während einer “Rechtsbewegung.”
Man kann die Eigenschaften (Lesekopf, Begrenzungen) dieser Simulation sogar “retten” für eine Grammatik mit $S \to w,\ w \in \Sigma^*$ gdw. $S' \overset{*}{\Rightarrow} w$.

Etwas genauer: Für alle Satzformen w mit $S \overset{*}{\Rightarrow} w$, führe explizite Regeln (1) $S' \to w'$ ein, w NT-Wort und $\ell(w) = 4$ sowie (2) $S' \to w$ für $w \in L$ mit $\ell(w) \leq 3$ (Starte mit NF-Grammatik). w' entsteht aus der Satzform w durch Markieren der ersten und der beiden letzten Zeichen.
Im zweiten und drittletzten Zeichen werden die ursprüngliche Information der ersten beiden bzw. letzten drei Zeichen gespeichert. Dann Terminierung mit Regeln der Bauart $\$L(a, b) \to ab$ am linken Rand (Bsp.)

Satz: KS $=$ MON

Beweis: $\subseteq$ trivial.

Für $\supseteq$ betrachte monotone NF-Grammatik $G = (\Sigma, N, R, S)$ mit Lesekopf und Begrenzungen (siehe vorige Folie).

Eine Regel $r = K_R A \to K_L w$ kann simuliert werden durch:

$K_R A \to rA$, $rA \to rw$, $r \to K_L$. (Hierbei ist r ein neues NT.)

Regeln der Form $AK_L \to K_L A$ werden simuliert durch:

$AK_L \to (K_L, A)K_L$, $(K_L, A)K_L \to (K_L, A)A$, $(K_L, A)A \to K_L A$.

Entsprechende Regeln (und neue Nichtterminale) benötigt man für die Rechtsbewegung.

Etwas schwieriger: Regeln der Form $K'_R A' \to B' K''_R$ werden wie folgt simuliert:

$K'_R A' \Rightarrow K'_R(B', K''_R) \Rightarrow B'(B', K''_R) \Rightarrow B' K''_R$.

Wie sehen also die Regeln genau aus? $\square$

Eine kontextsensitive Grammatik für dieselbe Sprache (nicht “durchkonstruiert”)

$S \to X_a Y_a X_b X_c$, $S \to X_a X_b X_c$, $S \to \lambda$,
$X_a Y_a \to X_a X_a X_b Y_c$, $X_b Y_a \to (a,b) Y_a$, $(a,b) Y_a \to (a,b) X_b$, $(a,b) X_b \to Y_a X_b$,
$Y_c X_b \to [b,c] X_b$, $[b,c] X_b \to [b,c] Y_c$, $[b,c] Y_c \to X_b Y_c$, $Y_c X_c \to Y_a X_c X_c$, $Y_c X_c \to X_c X_c$,
$X_a \to a$, $X_b \to b$, $X_c \to c$.

Eine Beispielableitung:

$S \Rightarrow X_a Y_a X_b X_c \Rightarrow X_a X_a X_b Y_c X_b X_c \Rightarrow X_a X_a X_b [b,c] X_b X_c \Rightarrow X_a X_a X_b [b,c] Y_c X_c$
$\Rightarrow X_a X_a X_b X_b Y_c X_c \Rightarrow X_a X_a X_b X_b Y_a X_c X_c \Rightarrow X_a X_a X_b (a,b) Y_a X_c X_c \Rightarrow X_a X_a X_b (a,b) X_b X_c X_c$
$\Rightarrow X_a X_a X_b Y_a X_b X_c X_c \overset{*}{\Rightarrow} X_a X_a Y_a X_b X_b X_c X_c \Rightarrow X_a X_a X_a X_b Y_c X_b X_b X_c X_c \overset{*}{\Rightarrow} X_a X_a X_a X_b X_b Y_c X_b X_c X_c$
$\overset{*}{\Rightarrow} X_a X_a X_a X_b X_b X_b Y_c X_c X_c \Rightarrow X_a X_a X_a X_b X_b X_b X_c X_c X_c \overset{*}{\Rightarrow} aaabbbccc$

Satz: **KF** $\subsetneq$ **KS**

Beweis: Jede kontextfreie Grammatik in erweiterter Chomky-Normalform ist kontextsensitiv. Dies zeigt die Inklusion **KF** $\subseteq$ **KS**.

Für die Striktheit betrachte die als fortlaufendes Beispiel betrachtete Grammatik für

$\{a^k b^k c^k \mid k \in \mathbb{N}\}$.

Abschlusseigenschaften bei Typ-0/-1 Sprachen (Auswahl)

Vereinigung / Konkatenation: “Standardkonstruktion” wie Typ-2

Durchschnitt: NF-Grammatiken G_1 und G_2 werden simuliert auf “Produktalphabet” $N_1 \times N_2$. Nur für Paare (X_a, X_a) gibt es terminierende Regeln $(X_a, X_a) \to a$.

Komplement: Typ-1 Sprachen sind abgeschlossen; die entsprechende Technik des induktiven Zählens ist Gegenstand der Komplexitätstheorie.
Typ-0 Sprachen sind nicht abgeschlossen (Satz von Post in der nächsten Theorie-Vorlesung).

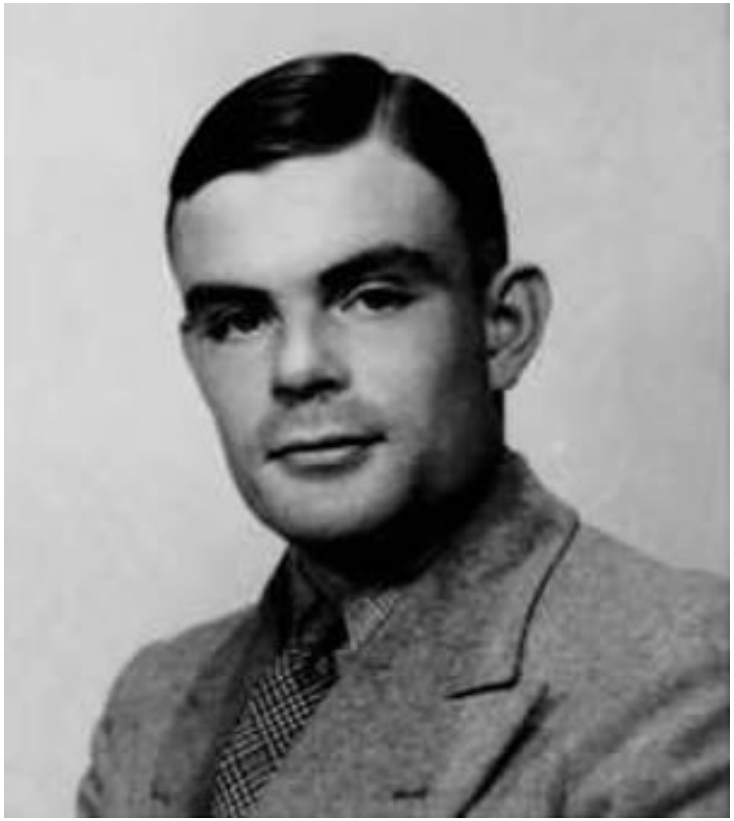
Die Chomsky-Hierarchy in Grammatik-Form:

Typ-0: Phrasenstrukturgrammatiken, also **RA**. (RE: recursively enumerable)
Typ-1: kontextsensitive Grammatiken, also **KS**. (CS: context-sensitive)
Typ-2: kontextfreie Grammatiken, also **KF**. (CF: context-free)
Typ-3: rechtslineare Grammatiken, also **REG**.

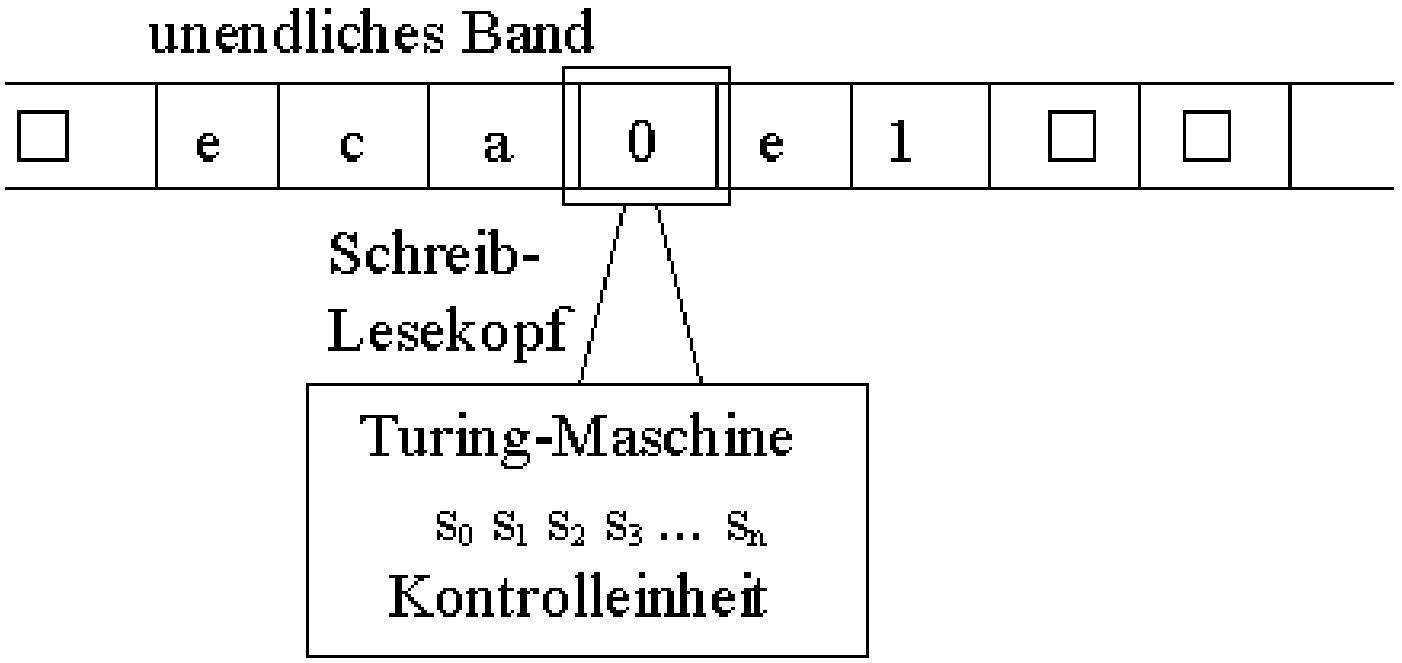
Satz: **REG** $\subsetneq$ **KF** $\subsetneq$ **KS** $\subsetneq$ **RA**.

Beweis: Alles bekannt bis auf Striktheit der letzten Inklusion, die erst in der zweiten Grundlagenvorlesung gezeigt werden wird: nicht für alle Typ-0-Sprachen L gibt es einen Algorithmus A_{L}, der entscheidet, ob $w \in \mathrm{L}$ liegt oder nicht. Für monotone Grammatiken gibt es hingegen solch einen ("teuren") Algorithmus (Ableiten aller Satzformen bis zu vorgegebener Länge möglich).

Alan Turing



Turingmaschinen



Grundidee Turings: Simulation eines menschlichen Computers.

Turingmaschinen: formaler

Eine *Turingmaschine* (TM) ist durch ein 7-Tupel beschrieben:

$$TM = (S, E, A, \delta, s_0, \square, F)$$

Dabei bedeuten

- $S = \{s_0, s_1, ..., s_n\}$ die Menge der *Zustände*,
- $E = \{e_1, e_2, ..., e_r\}$ das endliche *Eingabealphabet*,
- $A = \{a_0, a_1, ..., a_m\}$ das endliche *Arbeitsalphabet* (auch Bandalphabet genannt), es sei dabei $E \subset A$,

- s_0 der *Startzustand*,

- $a_0 = \square$ das *Blank-Symbol*, das zwar dem Arbeitsalphabet, aber nicht dem Eingabealphabet angehört,

- $F \subseteq S$ die Menge der *Endzustände*,

- δ sei die *Überführungsfunktion/-relation* mit (im deterministischen Fall)

$$\delta : (S \setminus F) \times A \to S \times A \times \{L, R, N\}$$

bzw. (im nichtdeterministischen Fall) $\delta \subseteq ((S \setminus F) \times A) \times (S \times A \times \{L, R, N\})$.
Hier bedeutet: L= links, R = rechts, N= neutral.

Turingmaschinen: Dynamik

$\delta(s, a) = (s', b, x)$ bzw. $((s, a), (s', b, x)) \in \delta$ (mit $x \in \{L, R, N\}$) habe folgende Bedeutung:

> Wenn sich der Automat im Zustand s befindet und unter dem Kopf das Zeichen a steht, so schreibt der Automat b und geht ein Feld nach rechts (R), bzw. links (L), bzw. bewegt sich nicht (N) und geht in den Zustand s' über.

Wie schon bei anderen Automatenmodellen gesehen, kann δ in Form einer Überführungstafel notiert werden. In jedem Kästchen stehen dann ein oder mehrere Tripel, bestehend aus neuem Zustand, geschriebenem Zeichen und Positionszeichen.

Eine **Konfiguration** einer Turingmaschine $TM = (S, E, A, \delta, s_0, \square, F)$ ist ein Tripel (u, s, v) aus $A^* \times S \times A^+$:

- uv ist aktuelle Bandinschrift
- s ist der aktuelle Zustand.
- Schreib-Lesekopf über erstem Zeichen von v, daher $v \neq \varepsilon$.
- Start der Maschine: $v \in E^* \cup \{\square\}$ (Eingabe), $s = s_0$, $u = \varepsilon$.
- O.B.d.A. $S \cap E = \emptyset$, daher Schreibweise usv statt (u, s, v)

Übergangsrelation $\vdash$

Zu einer (nicht)deterministischen Turingmaschine

$$TM = (S, E, A, \delta, s_0, \square, F)$$

wird die Übergangsrelation $\vdash$ aus $(A^* \times S \times A^+)^2$ folgendermaßen definiert:

- $a_1 ... a_m s b_1 ... b_n \vdash a_1 ... a_m s' c b_2 ... b_n$ mit $((s, b_1), (s', c, N)) \in \delta, m \geq 0, n \geq 1$
- $a_1 ... a_m s b_1 ... b_n \vdash a_1 ... a_m c s' b_2 ... b_n$ mit $((s, b_1), (s', c, R)) \in \delta, m \geq 0, n \geq 2$
- $a_1 ... a_m s b_1 ... b_n \vdash a_1 ... a_{m-1} s' a_m c b_2 ... b_n$ mit $((s, b_1), (s', c, L)) \in \delta, m \geq 1, n \geq 1$

Bem.: Die Konfigurationen fügen nach Bedarf Blanks an die Teilwörter v bzw. u an.

Inkrementieren einer Binärzahl

Beispiel:

$$TM = (\{s_0, s_1, s_2, s_f\}, \{0, 1\}, \{0, 1, \square\}, \delta, s_0, \square, \{s_f\})$$

mit

$\delta(s, a)$	0	1	$\square$
s_0	$(s_0, 0, R)$	$(s_0, 1, R)$	$(s_1, \square, L)$
s_1	$(s_2, 1, L)$	$(s_1, 0, L)$	$(s_f, 1, N)$
s_2	$(s_2, 0, L)$	$(s_2, 1, L)$	$(s_f, \square, R)$
s_f	$-$	$-$	$-$

Beispiel:

$$
\begin{aligned}
s_0111 &\vdash 1s_011 \vdash 11s_01 \vdash 111s_0\square \\
&\vdash 11s_11\square \vdash 1s_110\square \vdash s_1100\square \vdash s_1\square000\square \\
&\vdash s_f1000\square
\end{aligned}
$$

Arbeitsweise:
(1) Die Maschine läuft solange nach rechts läuft, bis sie das Ende der Zahl (Ziffer mit kleinstem Gewicht) erreicht hat.
(2) Ein weiterer Schritt führt auf ein Blank (wenn zu Beginn nichts auf dem Band steht, steht der Lese-Schreibkopf schon auf einem Blank).
(3) Dann erfolgt wieder ein Schritt nach links und Übergang nach s_1. $\leadsto$ Arbeitsposition erreicht.
(4A) Trifft sie auf eine Null, so invertiert sie sie (Addition von 1) und geht nach s_2.
(4B) Trifft sie vorher auf Einsen, müssen diese invertiert werden.
(4C) Spezialfall:keine 0 in der Zahldarstellung; dann wächst die Stellenzahl um 1 und die Maschine schreibt eine 1 anstelle des ersten linken Blanks.
(5) Ist die führende Stelle erreicht, geht die Maschine geht in den Endzustand über und der Kopf bleibt hier stehen.

Initialkonfiguration, Finalkonfiguration, akzeptierte Sprache

- *Initialkonfiguration* beim Start der Turingmaschine mit Eingabe $w \in E^*$ ist $s_0 w$ (bzw. $s_0 \Box$, falls $w = \varepsilon$).

- *Finalkonfigurationen* sind alle Konfigurationen $u s_f v$ mit $s_f \in F$. Hier kann die Berechnung nicht mehr fortgesetzt werden.

- Weiter ist

$$L(TM) := \{w \in E^* \mid s_0 w \vdash^* u s_f v, s_f \in F, u, v \in A^*\}$$

  die *von der Turingmaschine akzeptierte Sprache* L.

Hinweis: TM-Simulatoren

Einige Beobachtungen

Simulation endlicher Automat durch Turingmaschine z.B. wie folgt:

- Schreib-Lesekopf hat ausschließlich lesende Funktion
- Lesekopf zeichenweise lesend nach rechts
- Zustandsübergänge des endlichen Automaten übernommen
- Akzeptieren bei Erreichen von $\square$ (d.h. Ende der Eingabe) im 'Automaten-Endzustand'

Simulation Kellerautomat durch Turingmaschine z.B. wie folgt:

- Turing-Band zweigeteilt: Rechts Eingabewort, links Keller
- Übergangsfunktion des NKA durch mehrere Schritte der TM
- Gelesene Zeichen der Eingabe mit Spezialzeichen markieren
- Bestimmung des Überganges im NKA durch Inspektion der noch nicht markierten Eingabe und des simulierten Kellers

$\Longrightarrow$ Turingmaschinen mindestens so mächtig wie Kellerautomaten

These von Church / Turing

Turingmaschinen können “alles”, was überhaupt jemals von “Computern” gemacht werden kann.

Diese These sieht man durch Übung ein (Bsp.: Typ-0 Sprachen können von TM akzeptiert werden), sie lässt sich aber nicht beweisen.

Möchten Sie aber Ihre Programme in Zukunft in TM-Notation schreiben ?!

Typ-0 Sprachen lassen sich durch TM-Akzeptanz kennzeichnen

Eine Phrasenstrukturgrammatik, die eine TM simulieren soll, arbeitet wie folgt:

(1) Es wird die Sprache "$s_0 w \$ w$" generiert für beliebige Eingabewörter w der Turingmaschine.
(2) Die Arbeitsweise der Turingmaschine (Konfigurationsübergänge) wird nun auf dem ersten Wortteil simuliert.
(3) Wird eine Finalkonfiguration erreicht, so besteht die Möglichkeit, den ersten Wortteil und den Trenner $\$$ zu löschen und so w zu generieren.

Linear beschränkte Automaten

Eine TM heißt *linear beschränkter Automat* (LBA), wenn sie keine Blankzeichen überschreiben darf.

Satz: L ist Typ-1 gdw. L wird von LBA akzeptiert.

Beweis: . . . ähnlich wie auf der letzten Folie . . .

Achtung: Bandaufteilung in drei “Spuren”, um Löschen zu vermeiden. □