Satz (Kuroda)
Die von LBAs akzeptierte Sprache sind genau die Typ 1-Sprache.

$\Leftarrow$ ✓

$\Rightarrow$ ✓

Satz
Die durch allgemeine Turingmaschinen akzeptierte Sprachen sind genau die Typ 0-Sprachen.

Bew:

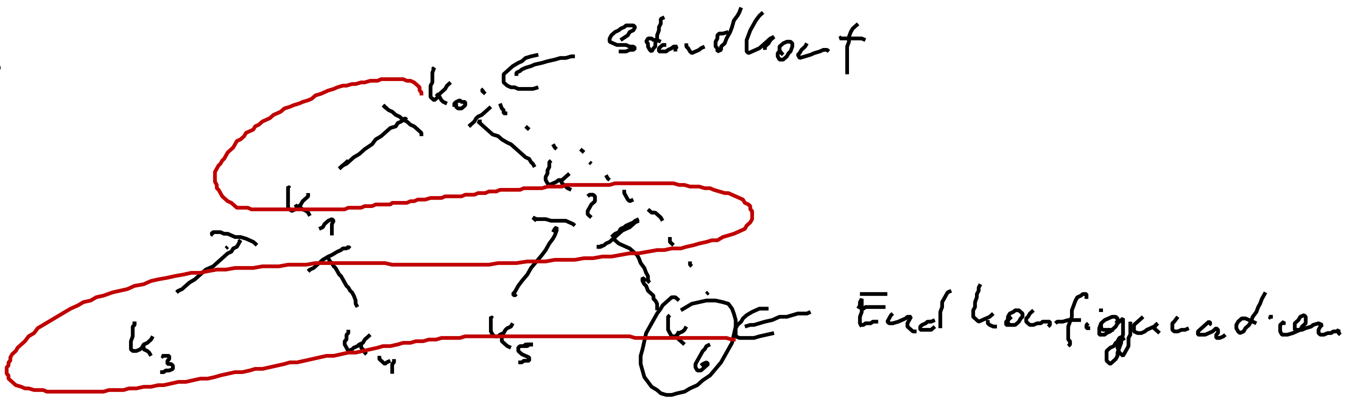
$\Leftarrow$: wie in dem Beweis für den Satz für Kuroda – bloß ohne Längenbeschränkung. U.U. brauche mir auf dem Band mehr Platz als durch das Eingabewort belegt wurde.

$\Rightarrow$: wie in dem Bew. des obigen Satzes, bloß man muss Zusatzregeln für das $\square$-Symbol einfügen.

$\square$

Bem: Nicht-determinismus ist essentiell.

Bem: Den Berechnungsbaum einer nicht-det. TM kann man systematisch ablaufen und nach einer Endkonfiguration suchen



Es folgt: Det. TMs und Nicht-det. TMs akz. die gleichen Sprachen.

Es ist offen ob dies auch für linear beschränkte Automaten gilt:

LBA-Problem

Zusammenfassung

Beschreibungsmittel

Ebene in der Chomsky-Hierarchie	Beschreibungsmittel	Automatentyp
Typ 3	reguläre Gram. reguläre Ausdrücke	DFA, NFA
det. kf. Spr.	LR(k) - Gram.	DPDA
Typ 2	kontextfreie Gram.	PDA
Typ 1	kontextsensitive Gram.	LBA
Typ 0	allg. Gram.	TM

Determinismus vs. Nichtdeterminismus

nicht.-det.	det.	äquivalent?
NFA	DFA	ja
PDA	DPDA	nicht
LBA	DLBA	?
TM	DTM	ja

Abschlusseigenschaften

	$\cap$	$\cup$	$\overline{}$	$\cdot$	$*$
Typ3	ja	ja	ja	ja	ja
det. kf.	nein	nein	ja	nein	nein
Typ2	nein	ja	nein	ja	ja
Typ1	ja	ja	ja	ja	ja
Typ0	ja	ja	nein	ja	ja

Entscheidbarkeit

	Wort-problem	Leerheitsproblem	Äquivalenzproblem	Schnittproblem
Typ 3	ja	ja	ja	ja
det. kf.	ja	ja	ja	nein
Typ 2	ja	ja	nein	nein
Typ 1	ja	nein	nein	nein
Typ 0	nein	nein	nein	nein

2. Berechenbarkeitstheorie

→ Wir haben alle eine Idee, was es heißt, dass etwas ($\hat{=}$ eine math. Funktion) berechenbar ist.

→ Wie können wir <u>beweisen</u>, dass etwas nicht berechenbar ist?

→ Formalisierung des Begriffs "Berechenbarkeit"

2.1 Intuitive Berechenbarkeit und Church-Turing-These

Eine Fkt $f: \mathbb{N}^k \to \mathbb{N}$ soll berechenbar heißen, falls es einen Algorithmus ($\hat{=}$ Rechenvorschrift), der für bel. $n_1, \dots, n_k$ mit endlichen vielen Schritten $f(n_1, \dots, n_k)$ berechnet. Falls f partiell ist und $f(n_1, \dots, n_k)$ undefiniert, dann soll der Alg. in eine Endlosschleife gehen.

Bsp. $\Omega: \mathbb{N} \to \mathbb{N}$ mit leerem Def.-Bereich.

Input(n); while (true);

Bsp.:

$$f(n) = \begin{cases} 1 & \text{falls } n \text{ ist Anfangsabschnitt der Dezimalbruchentwicklung von } \pi \\ 0 & \text{sonst} \end{cases}$$

$f(3) = 1$
$f(5) = 0$
$f(314) = 1$
$\rightarrow$ offensichtlich berechenbar

$$g(n) = \begin{cases} 1 & \text{falls } n \text{ irgendwo in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

$g(314) = 1$

$g(14) = 1$

$g(358976159) = ?$

Wir wissen nicht, ob das berechenbar ist oder nicht.

$$h(n) = \begin{cases} 1 & \text{falls in der Dezimalbruchentwicklung von } \pi \text{ irgendwo } n\text{-mal hintereinander eine } 7 \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

1. Fall: Es gibt beliebig lange Ketten von 7'en $\Rightarrow$ $h(n) = 1 \quad \forall n \in \mathbb{N}$

2. Fall: Es gibt $n_0 \in \mathbb{N}$, und π enthält maximal n_0 7'en hintereinander

$$h(n) = \begin{cases} 1 & n \leq n_0 \\ 0 & \text{sonst} \end{cases}$$

Berechenbar, ob wir wissen nicht wie?

Bsp: Gibt es für jede Zahl $r \in \mathbb{R}$ einen Alg., der r beliebig genau approximiert?
Die Menge $\mathbb{R}$ ist überabzählbar.
Die Menge der Alg. ist nur abzählbar.
$\Rightarrow$ Also nein

Church-Turing-These

Die durch die formale Def der Turingberechenbarkeit erfasste Klasse von Funktionen stimmt genau mit der Klasse von Fkt. überein, die durch die im intuitiven Sinne berechenbare Funktionen gegeben ist.

→ nicht beweisbar

→ stimmt für alle bisherigen Berechnungsmodelle

→ wir schauen uns versch. Modelle an

7.2 Turing-Berechenbarkeit

Def. Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt Turing-berechenbar (Tb), falls es eine (det.) Turingmaschine M gibt, so dass für alle $n_1, .., n_k, m \in \mathbb{N}$ gilt

$$\boxed{f(n_1, .., n_k) = m}$$

gdw.

$$z_0\, bin(n_1) \# bin(n_2) \# \ldots \# bin(n_k) \vdash^* \underline{\square\square \ldots \square\square z_e\, bin(m) \square\square \ldots \square\square}$$

wobei $bin(n_i)$ die Binärdarstellung von n_i ist und $z_e \in E$.

Bem: Wichtig: Was passiert, wenn wir mehrere Endkonf. erreichen können?

$\Rightarrow$ In allen solchen erreichbaren Endkonfigurationen muss das gleiche Ergebnis berechnet werden, ansonsten berechnet die TM nicht eine Fktswert.

Def. Eine Fkt $f: \Sigma^* \to \Sigma^*$ heißt TB, falls es eine (det.) TM M gibt, so dass für alle $x, y \in \Sigma^*$:

$$f(x) = y$$

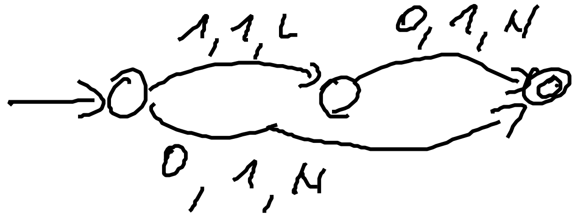
gdw.

$$\underline{z_0 x} \vdash^* \sqcup\sqcup \ldots \sqcup \underline{z_e y} \sqcup\sqcup \ldots \sqcup.$$

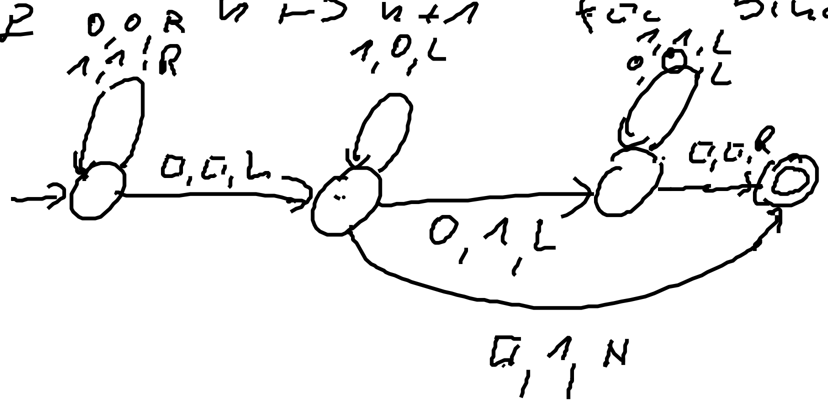
Bem: Modifikationen für Sonderfälle:

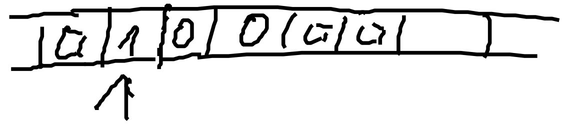
$\Sigma = \{1\}$. $\Rightarrow$ Unäre Darstellung der Zahlenwerte.

Bsp: Unäres Alphabet, wir wollen $n \mapsto n+1$ auf einer TM implementieren



Bsp $n \mapsto n+1$ für binäre Darstellung $\Sigma = \{1, 0\}$, $\Gamma = \Sigma \cup \{\square\}$





$M_1 \to M_2$

$M_1 \cdot M_2$ Hintereinanderschaltung =
Endzustand von M_1 = Anfangszustand von M_2

berechnet Komposition der Funktionen

M_1 berechnet f_1

M_2 berechnet f_2

$f_2(f_1(x))$

Konditionale Hintereinanderschaltung:

$M_1 \xrightarrow{=a} M_2$: weiter nur, wenn a auf dem Band steht

$M_1 \xrightarrow{\neq a} M_2$: " , wenn a nicht auf Band steht

$$\begin{tikzcd} M_1 \arrow[r, "=a"] \arrow[d, "\neq a"] & M_2 \\ M_3 & \end{tikzcd}$$: Verzweigung

$$\begin{tikzcd} \boxed{M_1} \arrow[r] & \boxed{M_2} \arrow[l, bend right, "=a"'] \arrow[r, "\neq a"] & \boxed{M_3} \end{tikzcd}$$: Schleifen

$\Rightarrow$ Flussdiagramme (ähnlich wie bei Programmierspr.)