$\mathcal{L}_0 = \mathcal{RE}$

Theorem Jede von einer Turing-Maschine akzeptierte Sprache kann auch von einer Typ-0 Grammatik generiert werden und umgekehrt, kurz: $\mathcal{L}_0 = \mathcal{RE}$.

Der Beweis erfolgt in zwei Schritten:

1. $\mathcal{L}_0 \subseteq \mathcal{RE}$
2. $\mathcal{RE} \subseteq \mathcal{L}_0$

Beweisidee: Simulation der Ableitungen einer Grammatik durch eine TM, bzw. Simulation der Rechnung einer TM durch eine Grammatik.

$\mathcal{L}_0 \subseteq \mathcal{RE}$

- konstruiere zu Typ-0 Grammatik G eine NTM:
  - kopiere das Eingabewort w auf eine Spur des Arbeitsbandes,
  - schreibe auf zweite Spur das Startsymbol S von G;
- NTM führt die einzelnen Ableitungsschritte auf der zweiten Spur aus;
- NTM prüft nach jeder Ersetzung nach, ob das Ergebnis mit der Eingabe übereinstimmt und akzeptiert bei Gleichheit.

$\mathcal{RE} \subseteq \mathcal{L}_0$

- O.B.d.A. $L \in \mathcal{RE}$ ist $L(M)$ für eine DTM $M := (Z, \Sigma, \Gamma, \delta, q_0, Z_{\text{end}})$;
- Erzeuge aus Startsymbol S beliebige Anfangskonfiguration $q_0 w \in Z\Sigma^*$ und kopiere w dahinter;
  - wie eine Grammatik für $\{ww \mid w \in \Sigma^*\}$;
  - Ableitbarkeit von $\{\$Q_0 w \cent w \mid w \in \Sigma^*\}$ aus S;
- Gemäß δ zu $q_i \in Z$ das Nonterminal Q_i mit linkem und rechtem Nachbarsymbol ersetzen;
- Einer Endkonfiguration entsprechendes Anfangsstück löschen und in das Terminalwort w überführen.

$\mathcal{RE} \subseteq \mathcal{L}_0$ (Forts.)

$$\begin{array}{lrcl}
\forall y \in \Gamma: & yQ_ix & \longrightarrow & Q_jyz,\ \text{falls } \delta(q_i,x) = (q_j,z,L) \\
 & \$Q_ix & \longrightarrow & \$Q_j\#z,\ \text{falls } \delta(q_i,x) = (q_j,z,L) \\
 & Q_ix & \longrightarrow & zQ_j,\ \text{falls } \delta(q_i,x) = (q_j,z,R) \\
 & & & \text{und } x \neq ¢ \\
 & Q_i¢ & \longrightarrow & zQ_j¢,\ \text{falls } \delta(q_i,\#) = (q_j,z,R)
\end{array}$$

Löschen der Konfigurationsinformation:

$$\begin{array}{lrcl}
 & Q_i & \longrightarrow & F,\ \text{falls } q_i \in Z_{\text{end}} \\
\forall y \in \Gamma: & Fy & \longrightarrow & F, \\
\forall y \in \Gamma: & yF & \longrightarrow & F, \\
 & \$F¢ & \longrightarrow & \lambda
\end{array}$$

monotone Grammatik

- Was sind monotone Grammatiken?

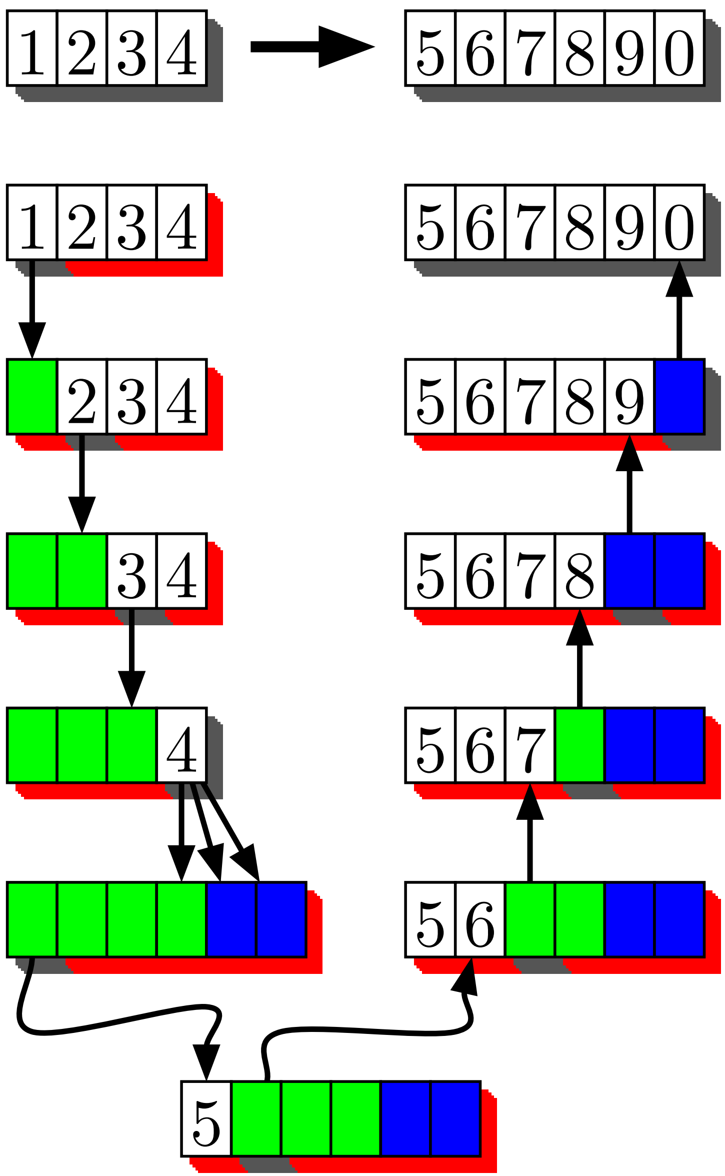
  Eine Typ-0 Grammatik $G = (V_N, V_T, P, S)$ heißt **monoton**, falls $\forall u \longrightarrow v \in P : (|u| \leq |v|)$.
  Einzige erlaubte Ausnahme: $S \longrightarrow \lambda$ und S kommt in keiner Produktion rechts vor.

- Kontextsensitive Grammatiken sind monoton!
- Wir können monotone Regel kontextsensitiv simulieren, mit neuen Nonterminalen:

$$V'_N := \left\{ \binom{A}{k} \mid A \in V \text{ und } 1 \leq k \leq |P| \right\} \cup \widetilde{V}$$

$$\widetilde{V} := \{\tilde{A} \mid A \in V_T\} \cup V_N \text{ (o.B.d.A. } \tilde{A} \notin V)$$

monotone Regel



beliebige monotone Regel

grüne und blaue Felder sind neue Nonterminale

rot unterlegt ist der jeweilige, unverändert bleibende Kontext.

Simulation monotoner Regel

Sei $A_1A_2\ldots A_n \longrightarrow B_1B_2\ldots B_m$ die k-te Regel aus P.

Simulation:

$$\hat{A} := \begin{cases} A & \text{falls } A \in V_N \\ \tilde{A} & \text{falls } A \in V_T \end{cases}$$

$$\begin{array}{rcl}
\hat{A}_1\hat{A}_2\ldots\hat{A}_n & \longrightarrow & \binom{A_1}{k}\hat{A}_2\ldots\hat{A}_n \\
\binom{A_1}{k}\hat{A}_2\ldots\hat{A}_n & \longrightarrow & \binom{A_1}{k}\binom{A_2}{k}\ldots\hat{A}_n \\
& \vdots & \\
\binom{A_1}{k}\binom{A_2}{k}\ldots\binom{A_{n-1}}{k}\hat{A}_n & \longrightarrow & \binom{A_1}{k}\ldots\binom{A_n}{k}\binom{B_{n+1}}{k}\ldots\binom{B_m}{k} \\
\binom{A_1}{k}\ldots\binom{A_n}{k}\binom{B_{n+1}}{k}\ldots\binom{B_m}{k} & \longrightarrow & \hat{B}_1\binom{A_2}{k}\ldots\binom{B_m}{k} \\
& \vdots & \\
\hat{B}_1\hat{B}_2\ldots\binom{B_m}{k} & \longrightarrow & \hat{B}_1\hat{B}_2\ldots\hat{B}_m
\end{array}$$

Simulation monotoner Regel (2)

…zusätzlich werden benötigt: $\tilde{A} \longrightarrow A$
für alle $A \in V_T$.

- Die einmal begonnene Simulation einer Regel kann nur durch Abarbeiten der simulierenden Regeln beendet werden!
- Zu jeder monotonen Regel existiert ein Satz simulierender Regeln.

Damit haben wir gezeigt, daß $\mathcal{L}_1 = \mathcal{MON}$

Übung

Seien $G_i := (\{S_i, A, B, C\}, \{a, b, c\}, P_i, S_i)$ Grammatiken mit:

$$P_1 :$$

$$\begin{aligned} S_1 &\longrightarrow ABS_1 \mid a \\ aB &\longrightarrow aC \\ A &\longrightarrow a \\ C &\longrightarrow c \end{aligned}$$

$$P_2 :$$

$$\begin{aligned} S_2 &\longrightarrow AS_2B \mid \lambda \\ A &\longrightarrow a \\ B &\longrightarrow b \end{aligned}$$

- Sind G_1 und G_2 kontextsensitiv?
- Was sind $L(G_1)$ und $L(G_2)$?

Entscheidbarkeitsresultate

. . . zur Wiederholung:

- Wann ist eine Menge entscheidbar?

Wortproblem

Jede kontextsensitive Sprache ist entscheidbar, d.h. $\mathcal{L}_1 \subseteq \mathcal{REC}$.

- Für jedes Wort w kann entschieden werden, ob $w \in L \in \mathcal{L}_1$.
  - Es gibt endlich viele Wörter v mit $|v| \leq |w|$.
  - Es gibt eine kontextsensitive Grammatik $G = (V_N, V_T, P, S)$ mit $L(G) = L$.
  - Keine Regel verkürzt die Satzform!
  - Ohne Satzformwiederholungen gibt es nur endlich viele Ableitungsschritte, bis die Satzform zu lang ist!!

$\mathcal{L}_1 \neq \mathcal{REC} \subsetneq \mathcal{RE}$

Diagonalbeweis: (ähnlich L_d)

- Die Menge der kontextsensitiven Grammatiken ist aufzählbar: $G_1, G_2, \ldots$
- Sei $f : \{0,1\}^* \rightarrow I\!N$ eine berechenbare Bijektion, mit: $f(w) = i$ gdw. w ist i-tes Wort in der lexikalischen Ordnung auf $\{0,1\}^*$.
- $L_e := \{w \mid w \notin L(G_{f(w)})\}$ ist entscheidbar!
- ABER: L_e ist nicht kontextsensitiv!

$\mathcal{L}_1 \neq \mathcal{REC}$ (Fortsetzung)

Annahme: $\exists n$ mit $L(G_n) = L_e$. Für das n-te Wort u der lexikalischen Aufzählung von $\{0,1\}^*$ mit $f(u) = n$ ergibt sich ein Widerspruch für beide möglichen Fälle $u \in L_e$ und $u \notin L_e$:

$$u \in L_e \xrightarrow{\text{Def. } L_e} u \notin L(G_n) \xrightarrow{\text{Annahme}} u \notin L_e$$

Andererseits ergibt sich auch:

$$u \notin L_e \xrightarrow{\text{Def. } L_e} u \in L(G_n) \xrightarrow{\text{Annahme}} u \in L_e$$

linear beschränkte Automaten

- Gibt es eine Automatenmodell für kontextsensitive Sprachen?
  - Ein **linear beschränkter Automat** (**LBA**) ist eine NTM, die bei beliebiger Eingabe w auf dem Arbeitsband höchstens $c \cdot |w|$ Felder bis zur Akzeptierung besucht.
  - $c \in I\!R^+$ ist eine Konstante, die nicht von w abhängt.
  - Arbeitet die TM bei gleicher Beschränkung ihres Arbeitsbandes deterministisch, so wird der linear beschränkte Automat mit **DLBA** abgekürzt.

Äquivalenz: $\mathcal{LBA} = \mathcal{L}_1$

Die Familie der von LBA's bzw. DLBA's akzeptierten Sprachen wird mit $\mathcal{LBA}$ bzw. $\mathcal{DLBA}$ bezeichnet.

Theorem: Es gilt: $\mathcal{LBA} = \mathcal{MON} = \mathcal{L}_1$

Also: Die *kontextsensitiven Sprachen* sind genau die *durch LBA's akzeptierbaren Sprachen*!

Beweis: $\mathcal{MON} \subseteq \mathcal{LBA}$

Diese Inklusion läßt sich auf ähnliche Weise zeigen, wie im Falle von Typ-0 Grammatiken und TM:

- Simuliere Ableitung in *monotoner* Grammatik auf einer NTM.
- Brich ab, sobald die abgeleitete Satzform auf Spur 2 länger ist als das Eingabewort.
- Akzeptiere, falls Satzform auf Spur 2 exakt mit dem Eingabewort übereinstimmt.

Diese Maschine benötigt nur $c \cdot |w|$ Platz.

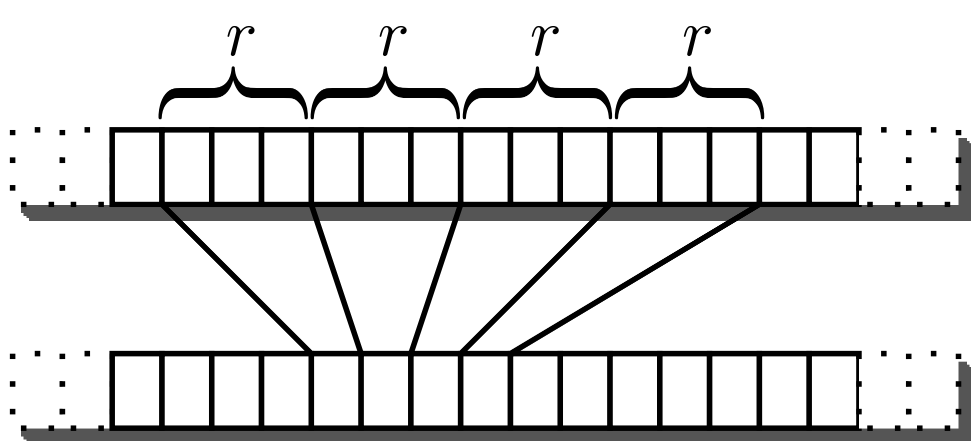
Beweis: $\mathcal{LBA} \subseteq \mathcal{MON}$

Gegeben ist ein LBA, d.h. eine TM, die mit $c \cdot |w|$ Platz auskommt.

- Die Konstruktion für TM und Typ-0-Grammatik liefert *keine* monotone Grammatik!
- Deshalb muß eine andere Simulation gewählt werden.
- Zunächst „normieren“ wir den LBA auf Platzbedarf max. $|w|$.

Bandkompression

- Für $0 \leq c \leq 1$ ist nichts zu tun.
- Ansonsten. Vergrößerung der Bandalphabets und Blockbildung.
  - Blocklänge $r \in \mathbb{N}$;
  - Bandbedarf nur noch $\frac{c}{r} \cdot |w|$;
  - Für $r := \min(n \in \mathbb{N} \mid n \geq c)$ sind das höchstens $|w|$ Felder.



Symbole für die Simulation

Erklärung der verwendeten Nonterminale:

1. Spur	x	Eingabesymbol
2. Spur	y	Bandsymbol
3. Spur	$\$$	Anfang-/Ende-Marker
4. Spur	q	Zustand/Kopfposition

Simulation des LBA

Die *Anfangs-Konfiguration* $q_0 w$ für ein beliebiges Wort $w \in \Sigma^*$ wird aus S mit folgenden Regeln *erzeugt*:

$$S \longrightarrow A \begin{bmatrix} x \\ x \\ ¢ \\ \# \end{bmatrix}, \; A \longrightarrow A \begin{bmatrix} x \\ x \\ \# \\ \# \end{bmatrix} \text{ und } A \longrightarrow \begin{bmatrix} x \\ x \\ \$ \\ q_0 \end{bmatrix},$$

für alle $x \in \Sigma$.

Simulation des LBA-Befehle

Linksschritt: Für $x_1, x_2, x_3 \in \Sigma$, $y_1, y_2 \in \Gamma$ und für $(p, y, z, L, q) \in K$, folgende Regeln in G:

$$\begin{bmatrix} x_1 \\ y_1 \\ \# \\ \# \end{bmatrix} \begin{bmatrix} x_2 \\ y \\ \# \\ p \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ y_1 \\ \# \\ q \end{bmatrix} \begin{bmatrix} x_2 \\ z \\ \# \\ \# \end{bmatrix}, \quad \begin{bmatrix} x_1 \\ y_1 \\ \$ \\ \# \end{bmatrix} \begin{bmatrix} x_2 \\ y \\ \# \\ p \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ y_1 \\ \$ \\ q \end{bmatrix} \begin{bmatrix} x_2 \\ z \\ \# \\ \# \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ \$ \\ \# \end{bmatrix} \begin{bmatrix} x_2 \\ y \\ ¢ \\ p \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ y_1 \\ \$ \\ q \end{bmatrix} \begin{bmatrix} x_2 \\ z \\ ¢ \\ \# \end{bmatrix}$$

Simulation des LBA-Befehle (2)

Rechtsschritt: Für $x_1, x_2, x_3 \in \Sigma$, $y_1, y_2 \in \Gamma$ und für $(p, y, z, R, q) \in K$, folgende Regeln in G:

$$\begin{bmatrix} x_2 \\ y \\ \# \\ p \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \# \\ \# \end{bmatrix} \longrightarrow \begin{bmatrix} x_2 \\ z \\ \# \\ \# \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \# \\ q \end{bmatrix}, \quad \begin{bmatrix} x_2 \\ y \\ \# \\ p \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \cent \\ \# \end{bmatrix} \longrightarrow \begin{bmatrix} x_2 \\ z \\ \# \\ \# \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \cent \\ q \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y \\ \$ \\ p \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \cent \\ \# \end{bmatrix} \longrightarrow \begin{bmatrix} x_2 \\ z \\ \$ \\ \# \end{bmatrix} \begin{bmatrix} x_3 \\ y_2 \\ \cent \\ q \end{bmatrix}$$

Ende der Simulation

Bei Erreichen eines Endzustands des LBA müssen alle Nonterminale in entsprechende Terminalsymbol überführt werden.

Für alle $x, z \in \Sigma,\ y \in \Gamma$ und $\& \in \{\#, \$, \cent\}$ folgende Regeln:

$$\begin{bmatrix} x \\ y \\ \& \\ q \end{bmatrix} \longrightarrow x \text{ falls } q \in Z_{end}, \qquad \begin{bmatrix} x \\ y \\ \& \\ \# \end{bmatrix} z \longrightarrow xz, \qquad z \begin{bmatrix} x \\ y \\ \& \\ \# \end{bmatrix} \longrightarrow zx$$

Problem/Fehler dieser Simulation

…leider steckt ein Fehler in diesem Satz von Regeln!

- Es können nur Wörter w mit $|w| \geq 2$ generiert werden!
- Warum? $\quad$ ¢ und $\$$ sind keine echten Begrenzer!
- Abhlife: $\quad$ Einführung des Euro **:-)**

Zusätzliche Regel für Anfangskonfig.: $\quad S \longrightarrow \begin{bmatrix} x \\ x \\ € \\ q_0 \end{bmatrix}$

- …und Simulation der Kopf(-nicht-)bewegung **H**.

Vorteile der LBA-Darstellung

Mit den Produktionen kann ein terminales Wort erzeugt werden, gdw. es eine Erfolgsrechnung für w im LBA mit $|w|$ Speicherplatz gibt.

- Mit Hilfe der Charakterisierungen von Sprachfamilien durch Automaten, lassen sich häufig Abschlußeigenschaften leichter beweisen als mit Grammatiken.
- Die Familie $\mathcal{L}_1 = \mathcal{LBA}$ ist gegen Durchschnittsbildung abgeschlossen.
- Beweisidee: Spurenbildung und Kombination der beiden LBA’s