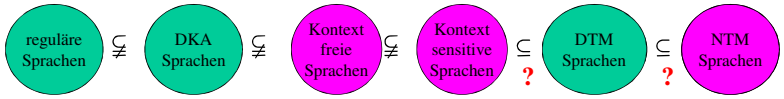


DEA / NEA	DKA	NKA	**LBA**	DTM	NTM
linkslineare Grammatik (rechtslin. Gr.)	LR(k) Gr.	kontext freie Gr.	kontext sensitive Gr.	?	Typ 0 Grammatik

reguläre Ausdrücke
Myhill-Nerode

Satz:
Die DTM-Sprachen sind genau das Gleiche wie die NTM-Sprachen.

Beweis: Es reicht zu zeigen:

Für jede einfache NTM $M=(\Sigma,\Gamma,\#,Q,S,F,\Delta)$ existiert
eine 2-Band DTM M', sodass $x\in L(M)$ genau dann wenn $x\in L(M')$

Idee: M' simuliert hintereinander "alle" möglichen Abläufe von M bei Eingabe x, bis ein akzeptierender Ablauf gefunden wird. (Wird keiner gefunden, wird x auch nie akzeptiert.)

Verwende einen "Leitstring", der bei jedem Ablauf ermöglicht, die nicht-deterministischen Wahlmöglichkeiten deterministisch zu treffen.

Für "Situation" $S=(q,A) \in Q\times\Gamma$
sei W_S die Anzahl der in S anwendbaren Regeln in Δ.

$W = \max\{ W_S \mid S \in Q\times\Gamma \}$

Für jede Situation S, gib den in S anwendbaren Regeln eine feste Ordnung.

Sei $LS = \{ 1,\cdots,W \}^*$ die Menge der "Leitstrings".

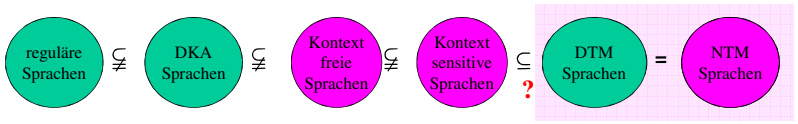
Verwendung von Leitstring $I = (i_1,i_2,\cdots,i_k) \in LS$ bedeutet:

Simuliere M mit Eingabe x für k Schritte.
Verwende im j-ten Schritt die i_j-te anwendbare Regel
(falls so eine Regel nicht existiere, stoppe diese Simulation)

Die deterministische TM M' verwendet
Arbeitsband 1 für die Simulation des Bandes des Maschine M
Arbeitsband 2 für den aktuellen Leitstring

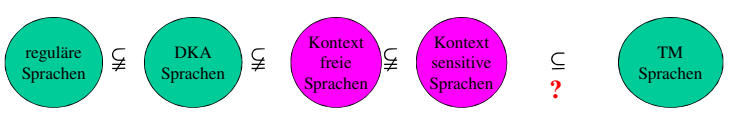
M' generiert einen Leitstring nach dem anderen auf Arbeitsband 2.

Für jeden Leitstring I:
Kopiere Inhalt des Eingabebandes auf Arbeitsband 1.
Verwende Leitstring I für Simulation von M auf Arbeitsband 1.
Wenn dabei M Eingabe x akzeptiert,
dann soll M' ebenfalls x akzeptieren.
Sonst löscht M' den Inhalt von Arbeitsband 1.



DEA / NEA	DKA	NKA	**LBA**	DTM	NTM
linkslineare Grammatik (rechtslin. Gr.)	LR(k) Gr.	kontext freie Gr.	kontext sensitive Gr.	?	Typ 0 Grammatik

reguläre Ausdrücke
Myhill-Nerode



DEA / NEA	DKA	NKA	**LBA**	TM
linkslineare Grammatik (rechtslin. Gr.)	LR(k) Gr.	kontext freie Gr.	kontext sensitive Gr.	Typ 0 Grammatik

reguläre Ausdrücke
Myhill-Nerode

Ein alternativer Rechenmechanismus: Register Maschinen

k-Register Maschine:

Speicher: $x_1, \cdots, x_k$ k "Register", jeder speichert eine natürliche Zahl

Operationen: $x_i = x_j$ op x_k mit op $\in \{ +, \dot{-}, *$ div , mod $\}$

$x_i = x_j$ op c

$x_i = c$

(tunix)

$a \dot{-} b = \max\{a-b,0\}$

a div $b = \lfloor a/b \rfloor$

c Konstante

Prädikate: $x_i =?\ 0$

mit der üblichen Semantik

Ein alternativer Rechenmechanismus: Register Maschinen

Programm: endliche Menge von Befehlen folgender Form

Label: Operation **goto** Label

Label: **if** Prädikat **then** Operation **goto** Label
else Operation **goto** Label

mit der üblichen Semantik

Es gibt zwei besondere Label: *Start*, *Halt*

Programm: endliche Menge von Befehlen folgender Form

Label: Operation **goto** Label

Label: **if** Prädikat **then** Operation **goto** Label
else Operation **goto** Label

mit der üblichen Semantik

Es gibt zwei besondere Label: *Start*, *Halt*

Ein Programm Π berechnet Funktion $f_\Pi: \mathbb{N}^m \to \mathbb{N}^n$

Registerinhalte
vor der Durchführung von Π: $x_1=a_1, \cdots x_m=a_m, x_{m+1}=0 \cdots x_k=0$
nach der Durchführung von Π: $x_1=b_1, \cdots x_m=b_n$

$$f_\Pi(a_1,\cdots,a_m) = (b_1,\cdots,b_n)$$

Beispielprogramm für $f(x) = \begin{cases} 1 & \text{falls x eine Primzahl} \\ 0 & \text{sonst} \end{cases}$

Start: $x_1 = x_1 \div 1$
L1: **if** $x_1 =? 0$ **then** $x_1 = 0$ **goto** *Halt*
else $x_1 = x_1 + 1$ **goto** *L2*
L2: $x_2 = x_1 \div 1$ **goto** *Loop*
Loop: $x_3 = x_2 \div 1$ **goto** *L3*
L3: **if** $x_3 =? 0$ **then** $x_1 = 1$ **goto** *Halt*
else goto *L4*
L4: $x_4 = x_1$ mod x_2 **goto** *L5*
L5: **if** $x_4 =? 0$ **then** $x_1 = 0$ **goto** *Halt*
else $x_2 = x_2 \div 1$ **goto** *Loop*
Halt:

```
if x < 2 then return 0
d = x – 1
while d >1 do
      if d teilt x then return 0
                   else d=d-1
return 1
```

x_1 .... x (oder Ausgabe)
x_2 .... d
x_3 .... d-1