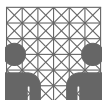


Äquivalenzen

Eine Menge M ist genau dann aufzählbar, wenn sie von einem Automaten der folgenden Art erkannt werden kann:

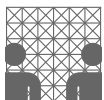
1. Einer deterministischen Turing-Maschine (DTM).
2. Einer nichtdeterministischen Turing-Maschine (NTM).
3. Einem 2-Keller-Automaten.
4. Einem 2-Zähler-Automaten.
5. M ist durch eine RAM akzeptierbar.



Äquivalenz DTM/NTM

1. Jede DTM ist auch NTM.
2. Simulation einer NTM auf einer DTM.
 - DTM hat keine Platzbeschränkung;
 - Merken des Konfigurationsbaumes auf dem Band;
 - systematische Suche nach Erfolgsrechnung.

Details im Skript und in der Literatur!

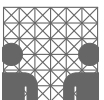


Die partiell rekursiven Funktionen

Theorem: Für eine n -stellige Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$ sind folgende Aussagen äquivalent:

1. f ist durch eine 1-DTM berechenbar,
2. f ist durch eine k -DTM berechenbar,
3. f ist μ -rekursiv (partiell-rekursiv),
4. f ist durch eine RAM berechenbar.

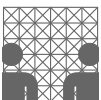
⋮



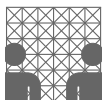
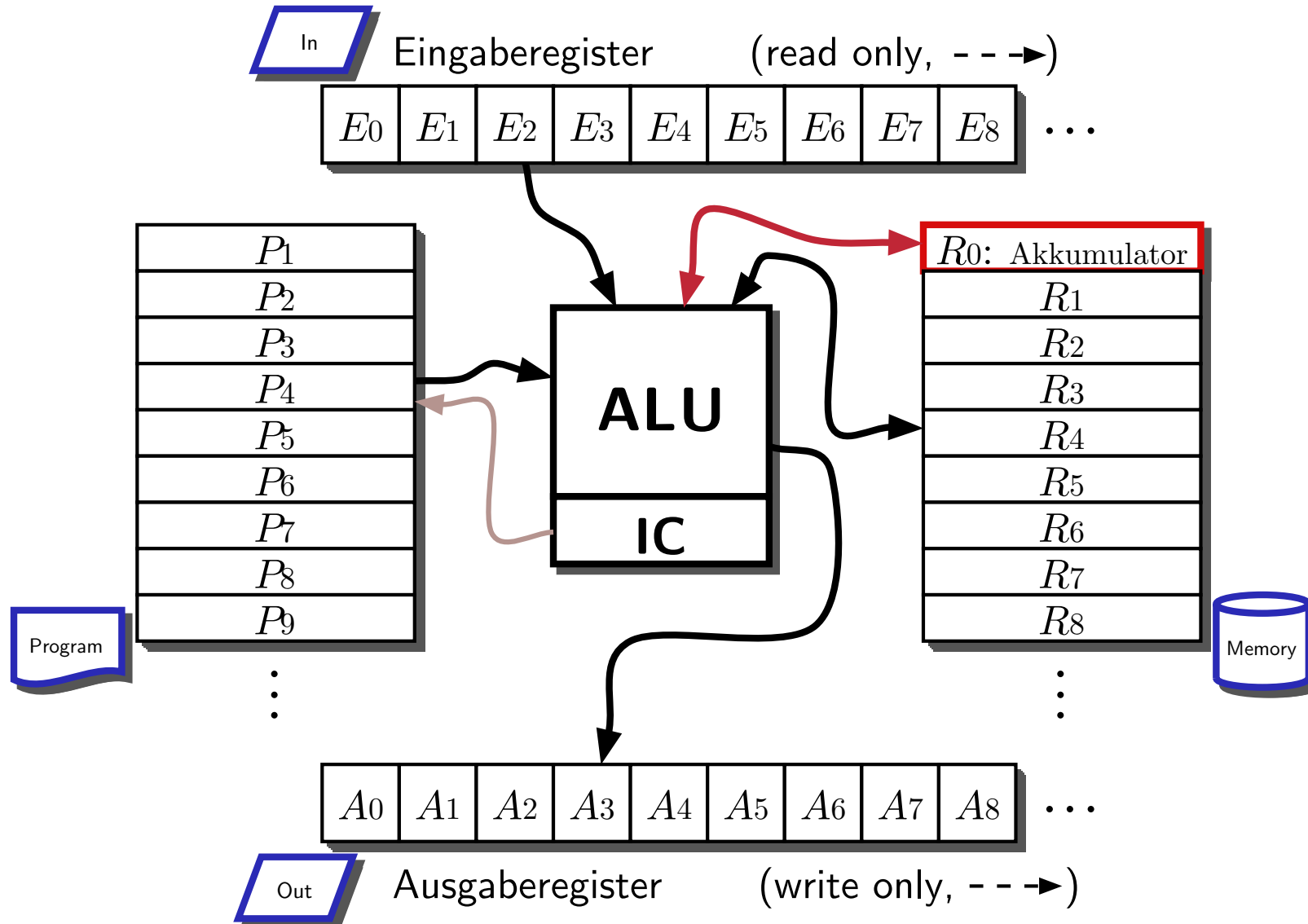
Random Access Machine (RAM)

Was unterscheidet die RAM von der TM?

- Rechnet auf natürlichen Zahlen.
- Speichermodell: Register
 - nehmen beliebige natürliche Zahlen auf
 - direkter Zugriff auf Register über Adressen möglich
 - indirekte Adressierung möglich
- Programmmodell: festes Programm



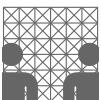
RAM-Schaubild



RAM-Varianten

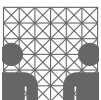
zwei verschiedene Varianten der RAM werden unterscheiden:

- Die *Bit-RAM*, bei der die Register beliebige natürliche (ganze) Zahlen enthalten dürfen
- Die *arithmetische RAM*, bei der die Speicherinhalte aus einem beliebigen Körper, wie z.B. \mathbb{R} oder \mathbb{Q} , stammen können.
- Weitere Modellvarianten entstehen bei Einschränkung des erlaubten Befehlssatzes.



RAM-Befehle

- Operationen: READ, WRITE, LOAD, STORE, ADD, SUB, JUMP, JZERO, JGTZ
- Operanden:
 - $= i$ [die Zahl i]
 - i [Inhalt des Registers R_i]
 - $*i$ [indirekte Adresse: Inhalt des Registers R_j , wenn j Inhalt des Registers R_i ist]
- Besonderheiten: zusätzliche Operationen
MULT und DIV bei RAM_{*}



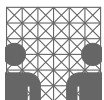
RAM-Befehle (2)

1. LOAD a $c(0) \leftarrow v(a)$
2. STORE i $c(i) \leftarrow c(0)$
 STORE $*i$ $c(c(i)) \leftarrow c(0)$
3. ADD a $c(0) \leftarrow c(0) + v(a)$
4. SUB a $c(0) \leftarrow c(0) - v(a)$
5. MULT a $c(0) \leftarrow c(0) \times v(a)$
6. DIV a $c(0) \leftarrow \lfloor c(0) \div v(a) \rfloor$
7. READ i $c(i) \leftarrow$ aktuelles Eingabesymbol
 READ $*i$ $c(c(i)) \leftarrow$ aktuelles Eingabesymbol.
8. WRITE a $v(a)$ wird an der aktuellen Position auf die Ausgabe geschrieben



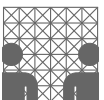
RAM-Befehle (3)

9. JUMP b Der *location counter* (Befehlsregister) wird auf die mit b markierte Anweisung gesetzt.
10. JGTZ b Der *location counter* wird auf die mit b markierte Anweisung gesetzt, wenn $c(0) > 0$ ist, sonst auf die nachfolgende Anweisung.
11. JZERO b Der *location counter* wird auf die mit b markierte Anweisung gesetzt, wenn $c(0) = 0$ ist, sonst auf die nachfolgende Anweisung.
12. HALT Programmausführung beenden.



Besonderheiten

- Felder auf dem Eingabe- und Ausgabeband enthalten Zahlen!
- Der Eingabekopf bewegt sich bei jedem READ um ein Feld nach rechts.
- Nach jedem WRITE bewegt sich der Kopf auf der Ausgabe um ein Feld nach rechts.
- Eine RAM ist immer **deterministisch!**

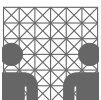


RAM-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}^l$ heißt *RAM-berechenbar* gdw. eine RAM existiert, die für die Eingabe (x_1, \dots, x_k) immer die Ausgabe $(y_1, \dots, y_l) = f(x_1, \dots, x_k)$ liefert, sofern $(x_1, \dots, x_k) \in \text{Def}(f)$.

Zum Beispiel ist die Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ mit $f(n) := n^2$ RAM-berechenbar (siehe Skript).

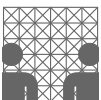
RAM-Programme für Bit-RAM's berechnen genau die partiell rekursiven Funktionen!



RAM-Sprachakzeptierung

Sei $\Sigma = \{a_1, \dots, a_m\}$ ein Alphabet. Das Symbol a_i wird durch die natürliche Zahl i kodiert (Schreibweise: $a_i^{(k)} = i$).

- Das Eingabewort $w = w_1 \dots w_k$ wird in den Feldern des Eingabebandes gespeichert.
 - w_i als die Zahl $w_i^{(k)}$ im i -ten Feld.
- im $(k + 1)$ -ten Feld wird 0 als Endmarkierung gespeichert.
- Ein RAM-Programm P akzeptiert w , gdw. P am Ende der Berechnung 1 in das erste Feld der Ausgabe geschrieben wurde.



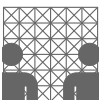
RAM-Sprachakzeptierung (2)

Die Sprache, welche von Programm P akzeptiert wird, ist die Menge der Wörter, die P akzeptiert.

RAM-Programme für Bit-RAM's akzeptieren genau die rekursiv aufzählbaren Sprachen.

Problem: Wie kann Komplexität definiert werden?

- Bei TM: Einfach durch Zählen der Schritte und Zählen der max. in einer Erfolgsrechnung genutzten Bandzellen.
- Bei der RAM: Zwei unterschiedliche Maße!



Komplexitätsmaße

- uniformes Maß
- logarithmisches Maß

Uniformes Zeitmaß:

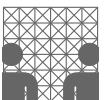
1 Schritt = 1 Zeiteinheit

Uniformes Platzmaß:

1 Register = 1 Platzeinheit

Logarithmisches Maß ist geeigneter, da es die Größe der Zahlen in Registern und Adressen berücksichtigt. Dazu definieren wir:

$$l(i) = \begin{cases} \lfloor \log i \rfloor + 1 & i \neq 0 \\ 1 & i = 0 \end{cases}$$

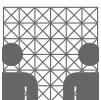


Logarithmisches Kostenmaß

- Kosten hängen von den verwendeten Operanden ab:

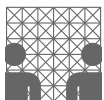
Operand a	Kosten $t(a)$
$= i$	$l(i)$
i	$l(i) + l(c(i))$
$*i$	$l(i) + l(c(i)) + l(c(c(i)))$

- Diese Kosten werden für die Berechnung der Komplexität des *Akzeptierens* bzw. der *Funktionsberechnung* auf die einzelnen Befehle angewandt.



konkrete Kosten

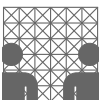
1. LOAD a $t(a)$
2. STORE i $l(c(0)) + l(i)$
STORE $*i$ $l(c(0)) + l(i) + l(c(i))$
3. ADD a $l(c(0)) + t(a)$
4. SUB a $l(c(0)) + t(a)$
5. MULT a $l(c(0)) + t(a)$
6. DIV a $l(c(0)) + t(a)$
7. READ i $l(\text{input}) + l(i)$
READ $*i$ $l(\text{input}) + l(i) + l(c(i))$
8. WRITE a $t(a)$



konkrete Kosten (2)

9.	JUMP b	1
10.	JGTZ b	$l(c(0))$
11.	JZERO b	$l(c(0))$
12.	HALT	1

-
- Das *logarithmische Platzmaß* eines Registers ist die maximale Länge $l(i)$ aller Zahlen i , die im Register gespeichert wurden.
 - Beim logarithmischen Kostenmaß für den Platz muss noch eine Korrekturgröße addiert werden!

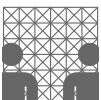


Simulation einer TM durch RAM

Wir skizzieren, wie TM und RAM sich gegenseitig simulieren können.

Eine $T(n)$ -zeitbeschränkte Mehrband DTM M kann durch eine RAM_+ simuliert werden, die im uniformen Maß $O(T(n))$ -zeitbeschränkt ist und die im logarithmischen Maß $O(T(n) \cdot \log T(n))$ -zeitbeschränkt ist.

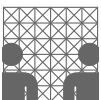
- pro Bandzelle ein Register
- log-Faktor wg. Speicherung der Kopfposition!



Simulation einer RAM durch TM

Eine im logarithmischen Maß $T(n)$ -zeitbeschränkte RAM_+ kann durch eine $O(T^2(n))$ -zeitbeschränkte 5-Band DTM M simuliert werden.

- Register hintereinander auf dem Band
- quadratischer Faktor wg. Speicherung der Register auf dem Band! $\Rightarrow S(n) \cdot T(n)$
abschätzen durch $T(n) \cdot T(n)$!

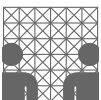


Komplexitätsklasse \mathcal{P}

\mathcal{P} ist die Klasse der Sprachen (algorithmischen Probleme), die man in Polynomialzeit erkennen (lösen) kann.

$\mathcal{P}Space$ ist die Klasse der Sprachen (algorithmischen Probleme), die man mit polynomiell viel Platz erkennen (lösen) kann.

- Obige Klassen sind unabhängig vom verwendeten Modell (RAM oder TM).

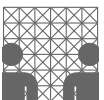


Literatur-Tip

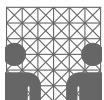


... eine unterhaltsame Exposition vieler interessanter Zusammenhänge zum Thema *Grenzen allen Wissens* (insbesondere aus der theoretischen Informatik und der Logik).

David Harel
Das Affenpuzzle
Springer-Verlag, 2001



Phrasenstrukturgrammatiken und kontextsensitive Grammatiken

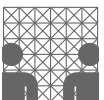


Sprachfamilie

Eine Menge \mathcal{L} heißt **Sprachfamilie** genau dann, wenn folgendes gilt:

1. Es existiert eine Sprache $L \in \mathcal{L}$ mit $L \neq \emptyset$, d.h. folglich gilt auch $\mathcal{L} \neq \emptyset$.
2. Für jede Sprache $L \in \mathcal{L}$ gibt es ein endliches Alphabet Σ_L mit $L \subseteq \Sigma_L^*$.

Bisher bekannt: $\mathcal{Cf} \cap \mathcal{Reg} = \mathcal{Reg}$, d.h. jede reguläre Sprache ist auch kontextfrei.



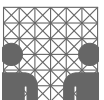
semi-Thue System (STS)

Ein **semi-Thue System** (STS) über dem Alphabet Σ ist eine (endliche oder unendliche) Teilmenge $S \subseteq \Sigma^* \times \Sigma^*$ ist ein Tupel (Σ, S) .

- Produktionen $(u, v) \in S$ werden oft als $u \longrightarrow v$ geschrieben.
- Die **Einschrittige Ableitungsrelation**

$\xRightarrow{(S)} \subseteq \Sigma^* \times \Sigma^*$, ist definiert durch: $w_1 \xRightarrow{(S)} w_2$,

wenn $w_1 = \alpha u \beta$, $w_2 = \alpha v \beta$ für $\alpha, \beta \in \Sigma^*$ und $u \longrightarrow v \in S$.



Ableitungsrelation

- Die reflexive, transitive Hülle von $\xRightarrow{(S)}$ wird mit

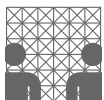
$\xRightarrow{*}_{(S)}$ bezeichnet, und ist die von S definierte

Ableitungsrelation.

- Weiterehin für $n \in \mathbb{N}$:

- $\xRightarrow{n+1}_{(S)} := \xRightarrow{n}_{(S)} \circ \xRightarrow{(S)}$

- $\xRightarrow{0}_{(S)} := Id_S$



Phrasenstrukturgrammatik

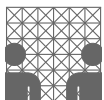
Eine **Typ-0** oder **Phrasenstruktur-Grammatik** wird durch ein Tupel $G := (V_N, V_T, P, S)$ spezifiziert. Hierbei gilt:

V_N ist ein endliches Alphabet von **Nonterminalen**,

V_T ist endliches Alphabet von **Terminalsymbolen** mit $V_N \cap V_T = \emptyset$,

$S \in V_N$ ist das **Startsymbol**.

$P \subseteq V^*V_NV^* \times V^*$ ist endliche Menge von **Produktionen** (oder **Regeln**), also ein spezielles STS über V .



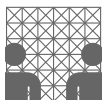
Generator: Typ-0-Grammatik

■ Eine Produktion $(u, v) \in P$ wird auch hier meist als $u \longrightarrow v$ geschrieben.

■ Die von G generierte oder erzeugte Sprache ist

$$L(G) := \{w \in V_T^* \mid S \xrightarrow[(P)]{*} w\}.$$

■ Mit \mathcal{L}_0 wird die Familie aller von Typ-0 Grammatiken erzeugbaren Sprachen bezeichnet. D.h., $\mathcal{L}_0 := \{L \mid L = L(G) \text{ für eine Typ-0 Grammatik } G\}$



Typ-0-Sprachen

Beispiel:

Sei G gegeben durch die Produktionen:

$$S \longrightarrow abc \mid aRbc$$

$$Rb \longrightarrow bR$$

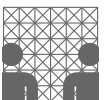
$$bL \longrightarrow Lb$$

$$Rc \longrightarrow Lbcc$$

$$aL \longrightarrow aaR$$

$$aL \longrightarrow aa$$

Dann gilt $L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

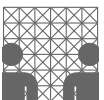


kontextsensitive Grammatik

Ein Spezialfall der Typ-0 Grammatik $G = (V_N, V_T, P, S)$ ist die **Typ-1** oder **kontextsensitive Grammatik**. Hierfür ist $u \longrightarrow v \in P$, wenn *entweder*

- $u = \alpha A \beta$, $v = \alpha w \beta$ mit $A \in V_N$, $w \in V^+$ und $\alpha, \beta \in V^*$ *oder*
- $u = S$, $v = \lambda$ und S kommt in keiner Produktion auf der rechten Seite vor.

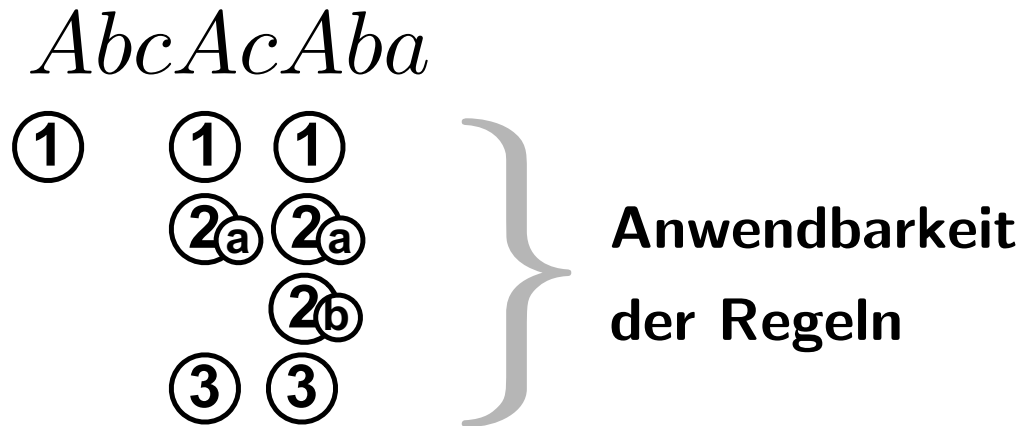
Die Familie der **kontextsensitiven Sprachen** wird mit \mathcal{C}_s oder \mathcal{L}_1 abgekürzt, und es ist $\mathcal{L}_1 := \{L \mid L = L(G) \text{ für eine Typ-1 Grammatik } G\}$.



Vergleich der Regeln

- ① *kontextfrei* $A \rightarrow aBccA$
- ② *kontextsensitiv* ① $cA \rightarrow caBccD$
- ③ *nicht kontextsensitiv* ② $cAb \rightarrow caBccDb$
- $\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
 $\alpha \quad \beta \quad \alpha \quad \beta$

Beispiel:



Ausblick

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

- Was sind monotone Grammatiken?
- Gibt es eine Automatenmodell für kontextsensitive Sprachen?
- Einige Entscheidbarkeitsresultate.

