Bemerkung: In einer Aufzählung einer Menge M $e(0), e(1), e(2), \ldots$ können Elemente beliebig oft vorkommen, aber jedes Element muss mindestens einmal auftreten.

Beispiele: (1) Die Menge der primitiv rekursiven Funktionen ist rekursiv aufzählbar aber nicht rekursiv. Bsp. Baum

(2) Die Menge der Argumente, für die eine μ-rekursive Funktion (gilt auch für TM, RM) terminiert, ist rekursiv aufzählbar, aber nicht rekursiv. □

μ-rek. Fkt. ist Abb. $f: \mathbb{N} \to \mathbb{N}$ (partiell)

Domain
$Dom(f) = \{x \in \mathbb{N} : f \text{ ist def für } x\}$

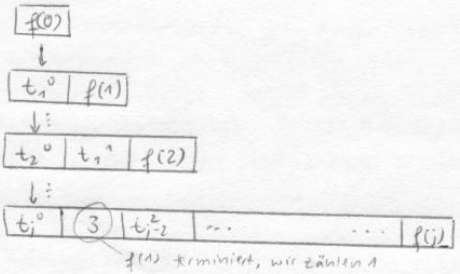
$f(0)$ $f(1)$ $f(2)$ $f(3)$ ... Zähle auf, für welche Argumente f terminiert.

↓ ↓ ↓ ↓

trm. nicht 3 trm. nicht 2

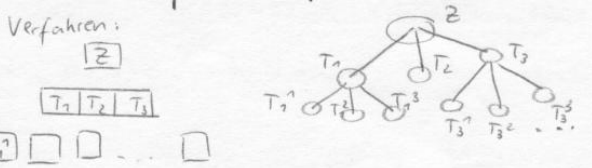
Verfahren durch parallele Berechnung:

$f(0) \to t_1^0 \to t_2^0 \to \ldots$ (Berechnung)



Satz: Jede Chomsky-Sprache ist rekursiv aufzählbar!

Beweis: Die Grammatik der Sprache definiert einen Baum von Ableitungen aus dem wir ein Aufzählverfahren



Satz: Die Menge der Argumente einer berechenbaren Funktion f, für die f nicht definiert ist (der Algorithmus nicht terminiert) ist i.A. nicht rekursiv aufzählbar.

Beweis: Folgt direkt aus dem Halteproblem!

Satz: Eine Menge $S \subseteq T^*$ ist genau dann rekursiv, wenn S und $T^* \setminus S$ rekursiv aufzählbar sind.

Beweis: Wir zählen S und $T^* \setminus S$ parallel auf. Irgendwann tritt jedes Element in einer der Aufzählungen auf. Dann liegt fest, ob es in S oder in $T^* \setminus S$ liegt. Dieses Verfahren terminiert für jedes Element aus T^*.

3. Komplexitätstheorie

nicht: wie "kompliziert" ein Problem ist
sondern: wie aufwendig der Algorithmus zur Lösung des Problems ist.

Jeder Algorithmus verbraucht bei der Ausführung gewisse Betriebsmittel, erfordert einen „Berechnungsaufwand".
Fragen:
(1) Bei einem gegebenen Berechnungsbegriff (TM, RM, μ-R) können wir nun für ein gegebenes Problem (Beispiel: Multiplikat fragen, welcher Algorithmus den geringsten Berechnungsaufwand hat.
(2) Wie hängt die Klassifizierung des Berechnungsaufwandes von der Wahl des Algorithmusbegriffs ab?

Wir werden die Aufwandsabschätzung für Algorithmen und Probleme an TM orientieren.

3.1. Komplexitätsmaße

Wir wählen zwei einfache Maßzahlen für den Berechnungsaufwand eines Algorithmus
- Anzahl der Schritte in der Berechnung (→ Zeitkomplexität)
- Anzahl der benötigten („Hilfs-") Speicherzellen (→ Bandkomplexität)

3.1.1. Zeitkomplexität

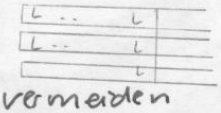
Wir betrachten k-Band-Turingmaschinen. Für jedes Band existiert ein Lese-Schreibe-Kopf. Für ein gegebenes Eingabewort (etwa auf Band 1) führt die TM eine endliche oder eine unendliche Anzahl von Schritten aus. Terminiert die Turing-Maschine für ein Wort w, so bezeichne $b(w)$ die Anzahl der Schritte.
Gilt für

$$T: \mathbb{N} \to \mathbb{N}$$

für alle Wörter $w \in \Sigma^*$ der Länge $n = |w|$: $b(w) \leq T(n)$
so heißt die TM (der Algorithmus) $T(n)$-zeitbeschränkt.

Achtung:
- Diese Def. schließt nichtdet. TM ein. Dann müssen alle Berechnungen für w durch $T(n)$ beschränkt sein.

#L0...L1L00L...L# } kompliziert, stattdessen:
§ Addition
#L0...0#

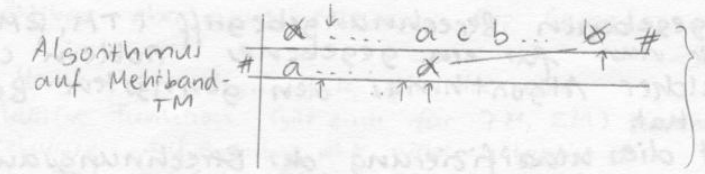


TM mit mehreren Bändern: einfacher und realistischer

- Mehrband-Turing-Maschinen vermeiden den Aufwand von Kopfbewegungen um zwischen Argumenten (siehe Bsp. oben - Addition) hin und her zu fahren und liefern „realistischere" Abschätzungen.

Die Definition von Zeitbeschränktheit lässt sich von Algorithmen auf Probleme übertragen:
Ein Problem*) heißt $T(n)$-zeitbeschränkt, wenn es einen Algorithmus gibt, der das Problem berechnet und $T(n)$-zeitbeschränkt ist.
*) Problem entspricht immer der Aufgabe eine Funktion / Prädikat zu berechnen.

Beispiel:
Das Problem, zu entscheiden, ob ein Wort in der Sprache $L = \{a^k c b^k : k \in \mathbb{N}\}$ ist, ist $(n+1)$-zeitbeschränkt. Dafür ist zu zeigen, dass ein Algorithmus (TM) existiert, der in $n+1$ Schritten für ein Wort w der Länge n entscheidet, ob $w \in L$.



Wir sagen die TM ist linear beschränkt.

Addition: Zahlen $a, b \in \mathbb{N}$ dargestellt durch Binärzahlen.
Eine TM für die Addition benötigt ~~$\log_2(b+a)$~~
$\log_2(b+a) + 2 + 2 \cdot \log_2(b) + 1 + 1$ Schritte.
Sprechweise: Das Problem ist logarithmisch zeitbeschränkt.

Annahme: Wir betrachten nur Probleme mit $T(n) \geq n+1$
d.h. wir betrachten nur Problemstellungen, wo die gesamte Eingabe für den Algorithmus relevant ist

6.6.2008 (Fr)

3.1.2. Bandkomplexität

Bandkomplexität bewertet den Speicheraufwand eines Algorithmus.
Wir messen die Bandkomplexität wieder durch TM über die Anzahl der im Laufe einer Berechnung beschriebenen Speicherzellen.
Wir TMen mit folgender Charakteristik:
- ein Eingabeband, das nicht beschrieben wird und eine Endemarkierung enthält
- k einseitig unendl. Bänder

Für ein Eingabewort $w \in \Sigma^*$ verwendet die TM $b_i(w) \in \mathbb{N}$ Speicherzellen auf ihrem i-ten Band $(1 \leq i \leq k)$.
Gilt für eine Abb. $S: \mathbb{N} \to \mathbb{N}$
für alle Wörter w mit $n = |w|$ für alle i, $1 \leq i \leq k$, $b_i(w) \leq S(n)$
so heißt die TM $S(n)$-band beschränkt.

! Abschätzungen, wie viele Bänder die TM höchstens braucht!

Wir sagen: "Die TM hat Bandkomplexität $S(n)$".

Beispiel:
(1) Sprache $L = \{a^n c b^n : n \in \mathbb{N}\}$
Wenn wir die Zahlen in Binärschreibweise notieren, dann benötigen wir $1 + \log_2 n$ Speicherzellen um die a^n Zeichen zu zählen.
Wir erhalten einen logarithmisch bandbeschränkten Algorithmus

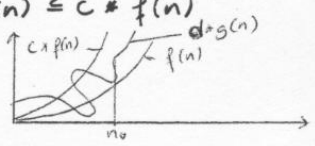
Def. Ein Problem heißt $S(n)$-bandbeschränkt, wenn es einen Algorithmus (eine TM) gibt, die das Problem löst und $S(n)$-bandbeschränkt ist.

Wenn wir von Bandkomplexität $S(n)$ sprechen, meinen wir stets $\max\{1, S(n)\}$.

Wir sind bei der Abschätzung von Band-/Zeitkomplexität i.d.R. nicht an exakten Zahlen interessiert, sondern an Größenordnungen.
Dazu verwenden wir Abbildungen $g, f: \mathbb{N} \to \mathbb{N}$
um das asymptotische Verhalten abzuschätzen. Wir sagen:
"Die Funktion g wächst mit der Ordnung $\Theta(f(n))$" wenn gilt

$$\exists\, c,d \in \mathbb{N}\setminus\{0\},\ n_0 \in \mathbb{N}:\ \forall n \in \mathbb{N},\ n \geq n_0 :\ f(n) \leq d * g(n) \leq c * f(n)$$

(d.h. bis auf Faktoren wächst $g(n)$ wie $f(n)$, muss ab n_0 gelten)



Folie: Zeit zur Lösung eines Problems der Größe n, wenn die Ausführung des Lösungsalgorithmus $O(n)$ Mikrosekunden benötigt. (S. 302)

n	$\log_2 n$	n	$n \log n$	n^2	2^n
10	0.000003 sec.	0.00001 sec.	0.00003 sec.	0.0001 sec.	0.001 sec.
10^2	0.000007 sec.	0.0001 sec.	0.0007 sec.	0.01 sec.	10^{16} Jahre
10^3	0.00010 sec.	0.001 sec.	0.01 sec.	1 sec.	astronomisch
10^4	0.00013 sec.	0.01 sec.	0.13 sec.	1,7 min.	astronomisch
10^5	0.00017 sec.	0.1 sec.	1.7 sec.	2.8 Std.	astronomisch

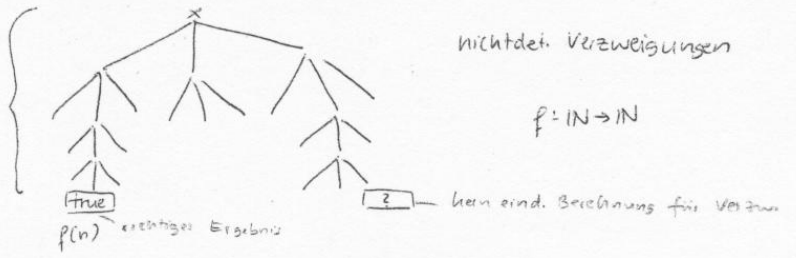
Diese Tabelle zeigt den Unterschied zwischen
- prinzipiell ("theoretisch") berechenbar
- praktisch berechenbar.

„Ein Algorithmus, der Jahre benötigt ist nicht besser als gar keinen Algorithmus zu haben"

Auch die Bandkomplexität lässt sich auf nichtdeterministische TM anwenden. Eine nichtdeterministische Turing-Maschine hat Bandkomplexität $S(n)$, falls für jedes Wort $w \in \Sigma^*$ mit $|w| = n$ alle Berechnungen der TM höchstens $S(n)$ Speicherzellen pro Band benötigen.

Veranschaulichung: Wortproblem
Ist $w \in L \subseteq \Sigma^*$?
Wird durch nichtdet. TM gelöst:

Berechnungsbaum:



1.3. Zeit- und Bandkomplexitätsklassen

ne Problemstellung (Aufgabe: Berechne Funktion / Prädikat) heißt deterministisch
n)-bandbeschränkt* wenn es eine det. TM gibt, die das Problem löst und
(n)-zeitbeschränkt (bzw. bandbeschränkt) ist. *(bzw. det. T(n)-bandbeschränkt)
nalog definieren wir nicht-det. T(n)-zeit / bandbeschränkt.

Abkürzungen:

DTIME(T(n))	Klasse der Probleme, die det. T(n)-zeitbeschränkt sind
NTIME(T(n))	" nicht-det. "
DSPACE(T(n))	Klasse der Probleme, die det. T(n)-band beschränkt sind.
NSPACE(T(n))	" nicht-det. "

Ziel DTIME(n^2)

ichtig: Wir brauchen uns nur die Größenordnung (asymptotisches Verhalten)
zu interessieren, wie folgende Sätze zeigen:

tz: Ist ein Problem S(n)-bandbeschränkt (det. oder nicht-det.), so ist für
jede Konstante $c \in \mathbb{R}$ mit $c>0$ das Problem auch $c * S(n)$ bandbeschränkt.

eis: Konstruiere TM, die r Zeichen in ein Zeichen zusammenfasst.

ollar: Für alle $c \in \mathbb{R}$ mit $c>0$ gilt
$NSPACE(S(n)) = NSPACE(c \cdot S(n))$
$DSPACE(S(n)) = DSPACE(c \cdot S(n))$.

tz: (Reduktion der Bänderzahl):
Sei M eine TM, S(n)-bandbeschränkt mit k Bändern.
Wir können eine TM konstruieren, die die k Bänder von M auf
einem Band simuliert. (↳ Anzahl d. Bänder spielt keine Rolle")

z: (Linearer Speed-Up):
Wird ein Problem von einer T(n)-zeitbeschränkten k-Band TM gelöst,
so wird es für jede reelle Zahl $c \in \mathbb{R}$, $c>0$, auch durch eine
$c * T(n)$-zeitbeschränkte k-Band TM gelöst,
falls $k>1$: $\inf_{n \to \infty} \frac{T(n)}{n} = \infty$

ollar: Unter dieser Voraussetzung gilt:
$DTIME(T(n)) = DTIME(c \cdot T(n))$
$NTIME(T(n)) = NTIME(c \cdot T(n))$.

Satz (nach Savitch):
Für jedes Akzeptanzproblem A (erkennen, ob ein Wort in
einer Sprache liegt):
Ist A in $NSPACE(S(n))$ dann gilt auch
A ist in $DSPACE(S(n)^2)$
} gilt nicht für Zeitkomplexität
falls gilt (die Funktion S ist voll-bandkonstruierbar):
Es ex. eine TM so dass gilt:
Für alle Zahlen $n \in \mathbb{N}$ ex. ein Eingabewort w, mit $|w|=n$,
so dass die Turing-Maschine S(n) Speicherzellen benötigt.

3.1.4. Polynomiale und nicht polynomiale Zeitkomplexität

Ein Problem heißt (bzgl. Zeitkomplexität) polynomial det. lösbar, wenn es in

$$DTIME(n^k) \quad \text{mit} \quad k \in \mathbb{N}.$$

Analog definieren wir exponentielle Zeitkomplexität:

$$DTIME(2^n)$$

Besonders interessiert $DTIME(2^n) \setminus \bigcup DTIME(n^k)$, $k \in \mathbb{N}$

Dies sind die Probleme, die für größere n theoretisch, aber nicht praktisch berechenbar sind.
Wir diskutieren nun zwei besonders interessante Klassen:

$$P = \bigcup_{k \in \mathbb{N}} DTIME(n^k)$$ Klasse der mit det. TM polynomial Lösbaren Probleme

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

Es gilt $NP \subseteq DTIME(2^n)$

Frage: gilt $P \stackrel{?}{=} NP$?