
3. Zeit- und Platzkomplexität

Die wichtigsten Kostenfaktoren einer Maschinenrechnung sind deren Länge (Rechenzeit) und deren Speicherbedarf. Wir führen hier diese Kostenfunktionen für Turingmaschinen ein und benutzen diese dann, um Probleme und (Funktionen) nach ihrer (Berechnungs-)Komplexität zu klassifizieren. Hierbei werden wir die worst-case-Komplexität betrachten, d.h. die maximalen Kosten, die bei Eingaben einer gegebenen Länge entstehen. Im folgenden sei $M = (\Sigma, m, T, k, \Gamma, Z, z_0, \delta)$ ein deterministischer Transduktor bzw. $M = (\Sigma, m, k, \Gamma, Z, z_0, E, \delta)$ ein (deterministischer oder nichtdeterministischer) Akzeptor beliebigen Typs.

Wir betrachten zunächst die Zeitkomplexität, das wohl bedeutendste Kostenmaß.

3.1 DEFINITION. Die *Rechenzeit* time_M von M ist eine partielle Funktion

$$\text{time}_M : (\Sigma^*)^m \rightarrow \mathbb{N}.$$

Für deterministisches M ordnet time_M jeder Eingabe $\vec{w} \in (\Sigma^*)^m$ die Länge der Rechnung von M bei dieser Eingabe zu, falls diese endlich ist, und ist sonst undefiniert. Für einen nd. Akzeptor M und eine von M akzeptierte Eingabe \vec{w} ist $\text{time}_M(\vec{w})$ die Länge der kürzesten akzeptierenden Rechnung von M bei dieser Eingabe; und für eine nicht akzeptierte Eingabe \vec{w} ist $\text{time}_M(\vec{w})$ die Tiefe des Rechenbaums von M bei dieser Eingabe, falls dieser Baum endlich ist, und andernfalls undefiniert.

Ist $\text{time}_M(\vec{w})$ undefiniert, so bedeutet dies, dass das Ergebnis von M bei Eingabe \vec{w} nach endlich vielen Rechenschritten noch nicht vorliegt, weshalb wir manchmal $\text{time}_M(\vec{w}) \uparrow$ mit $\text{time}_M(\vec{w}) = \infty$ identifizieren.

3.2 LEMMA. Die *Rechenzeit* $\text{time}_M : (\Sigma^*)^m \rightarrow \mathbb{N}$ ist eine *partiell rekursive Funktion*, die genau für die Eingaben \vec{w} definiert ist, für die M konvergiert. Die Funktion time_M ist also genau dann *total*, wenn M total ist. Weiter gilt, dass der Graph von time_M rekursiv ist¹.

BEWEISIDEE. Ist M deterministisch so simuliert man zur Berechnung von $\text{time}_M(\vec{w})$ die Rechnung von M bei Eingabe \vec{w} und zählt dabei die Rechenschritte. Terminiert die Rechnung, gibt man die Gesamtschrittzahl aus. Andernfalls ist $\text{time}_M(\vec{w})$ undefiniert. Um " $\text{time}_M(\vec{w}) = l$ " zu testen, verfährt man entsprechend, bricht die Simulation aber nach maximal $l + 1$ Schritten ab und akzeptiert genau dann, wenn M im Schritt l stoppt.

¹Ist time_M total, so gilt letzteres trivialerweise, da der Graph jeder total rekursiven Funktion rekursiv ist. Der Graph einer partiell rekursiven Funktion ist aber i.a. nur rekursiv aufzählbar. (S. Vorlesung "Einführung in die Theoretische Informatik".)

Im allgemeinen Fall, d.h. bei nd. M , erzeugt man zur Berechnung von $\text{time}_M(\vec{w})$ zunächst durch einen Breitendurchlauf² den Rechenbaum $\text{RB}_M(\vec{w})$ bis man die erste Endkonfiguration findet oder bis der Baum vollständig erzeugt ist. Die Rechenzeit ist dann die Tiefe des mit der gefundenen Endkonfiguration markierten Knotens bzw. die Tiefe des Rechenbaums. (Wird die Eingabe verworfen und ist der Baum unendlich, so bricht dieses Verfahren nicht ab und $\text{time}_M(\vec{w})$ ist undefiniert.)

Um " $\text{time}_M(\vec{w}) = l$ " zu entscheiden, geht man entsprechend vor, kann aber die Erzeugung des Rechenbaums bei Erreichen der Tiefe $l + 1$ abbrechen, da der Teilbaum $\text{RB}_{M(\vec{w}),l}$ von $\text{RB}_M(\vec{w})$ der Tiefe l ausreicht, um die Frage " $\text{time}_M(\vec{w}) = l$ " zu beantworten.

Die beschriebenen Verfahren sind effektiv, also nach Churchscher These durch geeignete Formalisierung in Turingmaschinenprogramme übersetzbar. \square

Als nächstes definieren wir (asymptotische) obere Schranken für die Rechenzeit einer totalen Maschine. Hierbei fassen wir alle Eingaben einer Länge zusammen und betrachten den ungünstigsten Fall also die *worst-case-Komplexität*. (Die Länge einer mehrstelligen Eingabe legen wir hierbei als die Summe der Längen der einzelnen Komponenten plus der Anzahl der erforderlichen Trennzeichen fest.) Da die Rechenzeit stets eine rekursive Funktion ist, genügt es hierbei rekursive Schranken zu betrachten.

3.3 DEFINITION. Sei $t : \mathbb{N} \rightarrow \mathbb{N}$ eine total rekursive Funktion. Die totale TM M ist $t(n)$ -zeitbeschränkt, falls

$$\forall \vec{w} = (w_1, \dots, w_m) \in (\Sigma^*)^m (\text{time}_M(\vec{w}) \leq t(|w_1| + \dots + |w_m| + m - 1)). \quad (3.1)$$

Für 1-stellige Eingaben w fordert man also, daß $\text{time}_M(w) \leq t(|w|)$ fast überall gilt.

Die Komplexität einer rekursiven Sprache L oder Funktion f definieren wir durch die Komplexität der sie erkennenden bzw. berechnenden Maschinen. Da es i.a. keine besten Lösungen gibt, ist die Komplexität hier nicht durch *eine* Kostenfunktion, sondern durch eine Klasse von Kostenfunktionen bestimmt. Hieraus erhält man obere und untere Komplexitätsschranken für die Sprache L (und entsprechend für die Funktion f):

- Eine rekursive Funktion $o : \mathbb{N} \rightarrow \mathbb{N}$ ist eine *obere (Kosten-) Schranke* für L , wenn es eine $o(n)$ -(kosten)beschränkte totale Maschine gibt, die L erkennt.
- Eine rekursive Funktion $u : \mathbb{N} \rightarrow \mathbb{N}$ ist eine *untere (Kosten-) Schranke* für L , wenn für jede $k(n)$ -(kosten)beschränkte totale Maschine, die L erkennt, $u \leq_{\text{f.ü.}} k$ gilt.

Offensichtlich gilt $u \leq_{\text{f.ü.}} o$ für jede untere Schranke u und obere Schranke o für L . Will man die Komplexität von L möglichst genau beschreiben, so muß man also den "Abstand" zwischen den nachweisbaren unteren und oberen Schranken möglichst klein machen. Eine gute obere Schranke erhält man hierbei durch die Angabe einer möglichst effizienten Maschine zur Erkennung von L (und der Komplexitätsanalyse der Maschine). Gute untere Schranken sind meist sehr viel schwieriger zu bekommen, da

²Beim *Breitendurchlauf* wird der Rechenbaum beginnend bei der Wurzel Tiefenstufe für Tiefenstufe von links nach rechts durchlaufen. Beim *Tiefendurchlauf* wird zunächst der am weitesten links liegende Pfad durchlaufen, dann der nächste Pfad u.s.w. Während der Breitendurchlauf auch bei unendlichen Rechenbäumen das vollständige Durchlaufen des Baumes garantiert (d.h. jeder Knoten wird nach endlich vielen Schritten erreicht), erreicht man beim Tiefendurchlauf den Teilbaum rechts der ersten unendlichen Rechnung nicht. Wir sagen anschaulich auch, daß wir die Rechnungen *parallel* (Breitendurchlauf) bzw. *sequentiell* (Tiefendurchlauf) verfolgen.

hier Aussagen über die Komplexität *aller* Maschinen, die L erkennen, gemacht werden müssen. In den Fällen, in denen man nichttriviale untere Schranken beweisen konnte, gelang dies in der Regel mit (z.T. extrem schweren) kombinatorischen Argumenten. Für viele interessante Sprachen scheinen jedoch die Methoden noch zu fehlen, um die vermuteten optimalen unteren Schranken auch nur annäherungsweise zu beweisen.

Sprachen und Funktionen, die einer oberen Komplexitätsschranke genügen, fassen wir in *Komplexitätsklassen* zusammen. Hierbei werden wir uns auf den 1-dimensionalen Fall beschränken und nur Funktionen über einem Alphabet Σ (d.h. mit Definitionsbereich und Wertebereich in Σ^*) betrachten. Im Falle der *Zeitkomplexitätsklassen* betrachten wir als Rechnermodell on-line Vollband-Turingmaschinen. Die darüberhinaus möglichen Typenunterschiede werden in der Definition reflektiert.

3.4 DEFINITION. Sei Σ_m das m -äre Alphabet und sei $t : \mathbb{N} \rightarrow \mathbb{N}$ eine total rekursive Funktion. Die *deterministische* bzw. *nichtdeterministische Zeitklasse von Sprachen über Σ_m mit Schranke $t(n)$* ist durch

$$\text{DTIME}^{(m)}(t(n)) = \{A \subseteq \Sigma_m^* : \text{Es gibt eine deterministische, } t(n)\text{-zeitbeschränkte on-line Vollband-Turingmaschine } M, \text{ die } A \text{ erkennt}\}$$

bzw.

$$\text{NTIME}^{(m)}(t(n)) = \{A \subseteq \Sigma_m^* : \text{Es gibt eine nichtdeterministische, } t(n)\text{-zeitbeschränkte on-line Vollband-Turingmaschine } M, \text{ die } A \text{ erkennt}\}$$

definiert. Entsprechend ist die *deterministische Zeitklasse von Funktionen über Σ_m mit Schranke $t(n)$* durch

$$\text{FDTIME}^{(m)}(t(n)) = \{f : \Sigma_m^* \rightarrow \Sigma_m^* : \text{Es gibt eine deterministische, } t(n)\text{-zeitbeschränkte on-line Vollband-Turingmaschine } M, \text{ die } f \text{ erkennt}\}$$

definiert. Lassen wir in obigen Definitionen nur Turingmaschinen M mit fester Bandzahl k zu, so fügen wir den resultierenden Komplexitätsklassen den Index k hinzu ($\text{DTIME}_k^{(m)}(t(n)), \dots$).

Wir werden uns auf die Analyse der Sprachklassen beschränken und die Funktionsklassen nur als Hilfsmittel hierzu verwenden. Zur Vereinfachung der Notation beschränken wir uns hierbei in der Regel auf das binäre Alphabet ($m = 2$) und lassen dann den Parameter m in der Definition der Komplexitätsklassen weg. (Die Aussagen lassen sich alle auf $m \geq 2$ und in der Regel auch auf $m = 1$ übertragen.) Wir erhalten dann folgende triviale Beziehungen zwischen den Zeitklassen, (wobei (N,D)TIME für NTIME oder DTIME steht, $\text{TIME}_{(k)}$ für TIME_k ($k \geq 1$) oder TIME, und $t : \mathbb{N} \rightarrow \mathbb{N}$ und $t' : \mathbb{N} \rightarrow \mathbb{N}$ total rekursive Funktionen sind):

$$\text{DTIME}_{(k)}(t(n)) \subseteq \text{NTIME}_{(k)}(t(n)) \quad (3.2)$$

$$\begin{aligned} (\text{N,D})\text{TIME}_k(t(n)) &\subseteq (\text{N,D})\text{TIME}_{k+1}(t(n)) \\ &\subseteq \bigcup_{l \geq 1} (\text{N,D})\text{TIME}_l(t(n)) = (\text{N,D})\text{TIME}(t(n)) \end{aligned} \quad (3.3)$$

$$t \leq_{\text{f.ü.}} t' \Rightarrow (\text{N,D})\text{TIME}_{(k)}(t(n)) \subseteq (\text{N,D})\text{TIME}_{(k)}(t'(n)) \quad (3.4)$$

$$\text{REK} = \bigcup_{t \in \text{FREK}} \text{DTIME}(t(n)) = \bigcup_{t \in \text{FREK}} \text{NTIME}(t(n)) \quad (3.5)$$

Hierbei folgen (3.2) und (3.3) aus der Tatsache, daß jeder deterministische Akzeptor auch ein nd. Akzeptor ist und jeder k -Band-Akzeptor als $(k+1)$ -Band-Akzeptor interpretiert werden kann, der das $(k+1)$ -te Band nicht benutzt. (3.4) folgt aus der Tatsache, daß für $t \leq_{f.ü.} t'$ jeder $t(n)$ -zeitbeschränkte Akzeptor auch $t'(n)$ -zeitbeschränkt ist. Zum Nachweis von (3.5) zeigt man

$$\text{REK} \subseteq \bigcup_{t \in \text{REK}} \text{DTIME}(t(n)) \subseteq \bigcup_{t \in \text{REK}} \text{NTIME}(t(n)) \subseteq \text{REK}$$

wobei diese Inklusionen leicht aus Lemma 3.2 bzw. (3.2) bzw. Lemma 2.7 und den entsprechenden Definitionen folgen.

Wir wenden uns nun der (Speicher-)Platzkomplexität zu. Grundsätzlich ist der Platzbedarf einer deterministischen Maschine M bei Eingabe \vec{w} die Anzahl der in der Rechnung bei dieser Eingabe benutzten Felder. Bei nichtdeterministischem M berücksichtigt man hierbei jedoch nur den Platzbedarf von für das Akzeptanzverhalten relevanten Rechnungen: Akzeptiert M die Eingabe, betrachtet man die akzeptierende Rechnung mit minimalem Platzbedarf. Wird die Eingabe verworfen, so maximiert man über den Platzbedarf aller Rechnungen. (D.h. schränkt man bei jeder möglichen Rechnung den zur Verfügung stehenden Platz entsprechend ein, so kann dies dazu führen, dass gewisse Rechnungen nun nicht mehr durchgeführt werden können. Bricht man diese bei Platzüberschreitung ab und rechnet sie als verwerfend, so ändert dies jedoch nichts am Akzeptanzverhalten der Maschine.)

Bei der Festlegung des Platzbedarfs einer Turingmaschine M haben sich – z.T. in Abhängigkeit vom Typ von M – gewisse Konventionen herausgebildet, die wir in der formalen Definition auch hier befolgen wollen. So werden bei off-line-Turingmaschinen das Ein- und Ausgabeband bei der Berechnung des Platzbedarfs nicht berücksichtigt. Weiter summiert man den Platzbedarf auf den einzelnen Bändern nicht auf, sondern betrachtet den maximalen Platzbedarf auf einem der Arbeitsbänder. Ähnlich zerlegt man ein Vollband in seine linke und rechte Hälfte und maximiert den Platzbedarf auf diesen Teilen.

Zur formalen Definition des Platzbedarfs von M ordnen wir zunächst jeder M -Konfiguration ihre Größe zu und bestimmen den Platzbedarf einer endlichen Konfigurationenfolge.

3.5 DEFINITION. Die *Größe* $g(\alpha)$ einer M -Konfiguration α ist die kleinste Zahl $g \in \mathbb{N}_+$, sodass für jedes Arbeitsband (f, p) von α gilt:

- (i) $\forall z \in \mathbb{Z} (|z| \geq g \Rightarrow f(z) = b)$
- (ii) $|p| < g$.

Für eine endliche M -Konfigurationenfolge $R = \alpha_1, \dots, \alpha_n$ ist der *Platzbedarf* $\text{space}(R)$ von R definiert als

$$\max\{g(\alpha_1), \dots, g(\alpha_n)\}.$$

3.6 DEFINITION. Der *Platzbedarf* space_M der Turingmaschine M ist eine partielle Funktion

$$\text{space}_M : (\Sigma^*)^m \rightarrow \mathbb{N}.$$

Für deterministisches M ordnet space_M jeder Eingabe $\vec{w} \in (\Sigma^*)^m$ den Platzbedarf der Rechnung R von M bei dieser Eingabe zu, falls R endlich ist, während für unendliches R $\text{space}_M(\vec{w})$ undefiniert ist. Für einen nd. Akzeptor M und eine von M akzeptierte Eingabe \vec{w} ist

$\text{space}_M(\vec{w}) = \inf\{\text{space}(R) : \text{Rakzeptierende Rechnung von } M \text{ bei Eingabe } \vec{w}\}$

während für eine nicht akzeptierte Eingabe \vec{w}

$$\text{space}_M(\vec{w}) = \sup\{g(\alpha) : \alpha \in \text{RB}_M(\vec{w})\},$$

falls $\text{RB}_M(\vec{w})$ endlich ist, und andernfalls $\text{space}_M(\vec{w})$ undefiniert ist.

Man beachte, dass der Platzbedarf einer on-line-Maschine durch die Eingabelänge nach unten beschränkt ist, dies für off-line Maschinen i.a. jedoch nicht gilt.

Statt $\text{space}_M(\vec{w}) \uparrow$ schreiben wir mitunter auch wieder $\text{space}_M(\vec{w}) = \infty$. D.h. wir interpretieren den Platzbedarf einer unendlichen Rechnung als unendlich, obwohl – im Falle einer Endlosschleife – die Konfigurationengröße in einer unendlichen Rechnung durchaus beschränkt sein kann. Eine *Schleife* tritt in einer Rechnung R auf, wenn sich eine Konfiguration in R wiederholt. (Für einen off-line-Transduktor M genügt das Vorkommen zweier Konfigurationen in der Rechnung, die sich nur im Ausgabeband unterscheiden.) Ist M deterministisch, so wird in diesem Fall die Schleife, d.h. die Konfigurationsfolge zwischen erstem und zweitem Auftreten der Konfiguration nach dem zweiten Auftreten in R , permanent wiederholt (*Endlosschleife*). Ist M nichtdeterministisch, so kann R selbst endlich sein, aber der Rechenbaum enthält die korrespondierende Rechnung mit Endlosschleife, hat also unendliche Tiefe. Mögliche Schleifen in Rechnungen müssen wir auch beim Beweis des folgenden Lemmas berücksichtigen, das das Analogon von Lemma 3.2 für die Platzkomplexität ist.

3.7 LEMMA. *Der Platzbedarf $\text{space}_M : (\Sigma^*)^m \rightarrow \mathbb{N}$ von M ist eine partiell rekursive Funktion, die genau für die Eingaben \vec{w} definiert ist, für die M konvergiert. Die Funktion space_M ist also genau dann total, wenn M total ist. Weiter gilt, dass der Graph von space_M rekursiv ist.*

Zum Beweis des Lemmas benötigen wir die Beobachtungen, dass sowohl der Platzbedarf wie auch das Akzeptanzverhalten einer Maschine nur von deren schleifenfreien Rechnungen abhängt, und dass sich die Länge einer schleifenfreien Rechnung effektiv in deren Platzbedarf beschränken lässt. Hierbei heisst eine M -Konfigurationsfolge *schleifenfrei*, wenn die in ihr vorkommenden Konfigurationen paarweise verschieden sind, wobei im Falle von off-line Transduktoren sich der Unterschied nicht nur im Ausgabeband findet.

3.8 LEMMA. *Sei R eine endliche Rechnung von M bei Eingabe \vec{w} , die mit der Konfiguration α endet. Dann gibt es eine schleifenfreie Rechnung R' von M bei Eingabe \vec{w} , die ebenfalls mit der Konfiguration α endet und nur Konfigurationen aus R enthält.*

BEWEIS. Die Menge aller M -Rechnungen bei Eingabe \vec{w} , die mit α enden und nur Konfigurationen aus R enthalten, ist nicht leer, da R selbst zu dieser Menge gehört. Wählt man R' als Rechnung minimaler Länge aus dieser Menge, so ist R' wegen der Minimalität schleifenfrei und R' hat die gewünschten Eigenschaften. \square

3.9 LEMMA. *Sei M ein nichtdeterministischer k -Band-on(off)-line Turingakzeptor, sei \vec{w} eine Eingabe für M der Gesamtlänge n und sei $m \geq 1$. Dann gilt für die Anzahl $\text{kon}_M(\vec{w}, m)$ der M -Konfigurationen, die in einer M -Rechnung bei Eingabe \vec{w} vorkommen können und deren Größe durch m beschränkt ist,*

$$\text{kon}_M(\vec{w}, m) \leq (n + 2)2^{cm},$$

wobei sich die Konstante c effektiv aus M berechnen lässt.

BEWEIS. Sei z die Anzahl der Zustände von M , g die Größe des Bandalphabets, und k die Anzahl der Bänder. Dann lässt sich, unter der Annahme, dass M on-line arbeitet, die Anzahl der M -Konfigurationen α von M der Größe $\leq m$ durch das Produkt der folgenden Parameter beschränken:

$$kon_M(\vec{w}, m) \leq z(2m-1)^k (g^{2m-1})^k.$$

Hierbei ist z die Anzahl der möglichen Zustände von α , $2m+1$ die Anzahl der möglichen Positionen des Arbeitsfeldes auf einem der Bänder (dieses muss zwischen $-(m-1)$ und $+(m-1)$ liegen) und g^{2m-1} die Anzahl der möglichen Inschriften eines Bandes. Da sich die Konstanten z , g und k aus M ergeben, kann man aus dieser Abschätzung effektiv eine Konstante c mit

$$kon_M(\vec{w}, m) \leq 2^{cm}$$

berechnen. Im Falle einer off-line-Maschine M muss man diese obere Schranke noch mit den $n+2$ möglichen Positionen des Lesekopfes auf dem Eingabeband multiplizieren. \square

BEWEIS VON LEMMA 3.7. Ist M deterministisch, so können wir wie im Beweis der entsprechenden Aussage für die Zeitkomplexität verfahren. Im allgemeinen Fall, d.h. bei nd. M , muss man das dort verwendete Verfahren etwas modifizieren, um $\text{space}_M(\vec{w})$ für gegebenes \vec{w} zu berechnen: Zunächst durchläuft man wieder den Rechenbaum $\text{RB}_M(\vec{w})$ der Breite nach bis man die erste Endkonfiguration α findet oder bis der Baum vollständig erzeugt ist. (Bricht dieses Verfahren nicht ab, so ist $\text{space}_M(\vec{w})$ undefiniert.) Bricht das Verfahren ab, ohne dass eine Endkonfiguration gefunden wurde, so verwirft M die Eingabe \vec{w} und wir erhalten $\text{space}_M(\vec{w})$, indem wir die maximale Größe der in dem konstruierten Rechenbaum $\text{RB}_M(\vec{w})$ vorkommenden Konfigurationen bestimmen. Stoßen wir auf eine Endkonfiguration α , so betrachten wir die Rechnung R die zu α führt und bestimmen deren Platzbedarf m . Die Zahl m ist dann eine obere Schranke für $\text{space}_M(\vec{w})$, da – im Falle der Akzeptanz – $\text{space}_M(\vec{w})$ als der minimale Platzbedarf einer akzeptierenden Rechnung definiert ist. Um festzustellen, ob es eine akzeptierende Rechnung mit geringerem Platzbedarf als m gibt, benutzen wir die beiden gerade gezeigten Lemmata. Ist R' eine akzeptierende Rechnung mit minimalem Platzbedarf $m' \leq m$, so ist nach Lemma 3.8 R' schleifenfrei, weshalb die Länge von R' nach Lemma 3.9 durch $t = (n+2)2^{cm}$ beschränkt ist, wobei sich c aus der Beschreibung von M ergibt und n die Eingabengänge ist. Die Schranke t ist also berechenbar und wir können $\text{space}_M(\vec{w})$ berechnen, indem wir den Baum $\text{RB}_M(\vec{w})$ bis zur Tiefe t erzeugen und den minimalen Platzbedarf der in diesem Teilbaum vorkommenden akzeptierenden Rechnungen berechnen. Um " $\text{space}_M(\vec{w}) = l$ " zu entscheiden, geht man entsprechend vor. Hier genügt es den Teilbaum $\text{RB}_{M(\vec{w}),t'}$ von $\text{RB}_M(\vec{w})$ der Tiefe $t' = (n+2)2^{cl}$ zu erzeugen. \square

Platzbeschränkte Turingmaschinen und die Platzkomplexitätsklassen definiert man entsprechend wie bei der Rechenzeit. Bei den Platzkomplexitätsklassen gehen wir jedoch von off-line-Turingmaschinen aus, um auch nichttriviale sublineare Platzschranken zu erhalten.

3.10 DEFINITION. Sei $s : \mathbb{N} \rightarrow \mathbb{N}$ eine total rekursive Funktion. Die totale TM M ist $s(n)$ -platzbeschränkt, falls

$$\forall \vec{w} = (w_1, \dots, w_m) \in (\Sigma^*)^m (\text{space}_M(\vec{w}) \leq s(|w_1| + \dots + |w_m| + m - 1)). \quad (3.6)$$

3.11 DEFINITION. Sei Σ_m das m -äre Alphabet und sei $s : \mathbb{N} \rightarrow \mathbb{N}$ eine total rekursive Funktion. Die *deterministische* bzw. *nichtdeterministische Platzklasse von Sprachen über Σ_m mit Schranke $s(n)$* ist durch

$$\text{DSPACE}^{(m)}(s(n)) = \{A \subseteq \Sigma_m^* : \text{Es gibt eine deterministische, } s(n)\text{-platzbeschränkte off-line Vollband-Turingmaschine } M, \text{ die } A \text{ erkennt}\}$$

bzw.

$$\text{NSPACE}^{(m)}(s(n)) = \{A \subseteq \Sigma_m^* : \text{Es gibt eine nichtdeterministische, } s(n)\text{-platzbeschränkte off-line Vollband-Turingmaschine } M, \text{ die } A \text{ erkennt}\}$$

definiert. Entsprechend ist die *deterministische Platzklasse von Funktionen über Σ_m mit Schranke $s(n)$* durch

$$\text{FDSpace}^{(m)}(s(n)) = \{f : \Sigma_m^* \rightarrow \Sigma_m^* : \text{Es gibt eine deterministische, } s(n)\text{-platzbeschränkte off-line Vollband-Turingmaschine } M, \text{ die } f \text{ erkennt}\}$$

definiert. Lassen wir hierbei nur Turingmaschinen M mit fester Bandzahl k zu, so fügen wir den resultierenden Komplexitätsklassen den Index k hinzu ($\text{DSPACE}_k^{(m)}(t(n)), \dots$).

Die in (3.2) – (3.5) festgestellten Beziehungen zwischen den verschiedenen Zeitklassen gelten für die Platzklassen ebenfalls, d.h. bleiben gültig, wenn wir durchgehend TIME durch SPACE ersetzen.

Im Rest dieses Abschnittes zeigen wir noch, dass der gewählte Maschinentyp (off-line vs. on-line und Vollband vs. Halbband) bei der Definition der Zeit- und Platzkomplexitätsklassen im wesentlichen vernachlässigt werden kann. Nur für sublineare Platzschranken ist es entscheidend, dass man off-line Maschinen betrachtet, da für on-line Maschinen durch Einbeziehung der Eingabengänge in die Platzkosten solche Schranken nicht möglich sind. Die sonst auftretenden geringen Verschiebungen bei den Kosten können wir durch das Phänomen der linearen Komprimierbarkeit bzw. Beschleunigung abfangen.

3.12 LEMMA. (a) *Zu jedem $t(n)$ -zeit- und $s(n)$ -platzbeschränkten k -Halbband-Turingakzeptor M kann man effektiv einen äquivalenten k -Band-Turingakzeptor M' angeben, der ansonsten von demselben Typ wie M ist und der $(t(n) + 3)$ -zeit- und $\max(s(n), 1)$ -platzbeschränkt ist.*

(b) *Umgekehrt kann man zu jedem $t(n)$ -zeit- und $s(n)$ -platzbeschränkten k -Band-Turingakzeptor M effektiv einen äquivalenten k -Halbband-Turingakzeptor M' vom ansonsten gleichen Typ angeben, der wiederum $t(n)$ -zeit- und $s(n)$ -platzbeschränkt ist.*

BEWEISIDEE. Zum Beweis von (a) beobachtet man, dass man eine Halbband-TM M als Vollband-TM auffassen kann. Probleme gibt es dann nur, wenn M auf einem Band das Feld -1 besuchen möchte. Bei der Halbband-TM führt das zu einem Fehlerstopp ohne Ausführung des letzten Befehls, während die Vollband-TM den Befehl regulär ausführt. Dieses unterschiedliche Verhalten eliminiert man dadurch, dass M' bevor es mit der Simulation von M beginnt, auf jedem Arbeitsband das Feld -1 mit einem neuen Zeichen $*$ $\notin \Gamma$ beschriftet (was 2 zusätzliche Schritte kostet).

Zum Beweis von (b) konstruieren wir M' so, dass es mit dem gefalteten Band von M arbeitet. D.h. eine Bandinschrift

	-2	-1	0	1	2	
	a_{-2}	a_{-1}	a_0	a_1	a_2	

von M wird durch das Halbband

	a_0	a_1	a_2	
	*	a_{-1}	a_{-2}	

repräsentiert, das aus zwei *Spuren* besteht, wobei die obere Spur die rechte Hälfte, die untere Spur die gespiegelte linke Hälfte des ursprünglichen Bandes beschreibt. Formal bedeutet dies, dass man das Bandalphabet Γ' von M' als

$$\Gamma' = \Gamma \cup (\Gamma \times \Gamma) \cup (\Gamma \times \{*\})$$

wählt. Im Zustand merkt sich M' dann, ob das M -Arbeitsfeld auf der oberen bzw. unteren Spur des korrespondierenden M' -Arbeitsfeldes liegt. (D.h. $Z' = (Z \times \{o, u\}^k) \cup \{z_0\}$ wobei z_0 wiederum der Startzustand von M' ist.) Bei noch nicht in Spurdarstellung vorliegenden, mit a beschrifteten Feldern, verrät der Zustand, ob diese als (a, b) bzw. (b, a) zu lesen sind und die entsprechende Ersetzung wird im nächsten Schritt vorgenommen. Beim Start wird entsprechend die Randmarke $*$ in die untere Spur von Feld 0 geschrieben. Im Falle einer 1-Band-TM M bedeutet dies z.B., dass jede Instruktion mit einer Linksbewegung

$$(z, a, \hat{a}, -1, \hat{z}) \in \delta$$

durch die Instruktionen

$$\begin{aligned} &((z, o), (a, *), (\hat{a}, *), +1, (\hat{z}, u)) \\ &((z, o), (a, \tilde{a}), (\hat{a}, \tilde{a}), -1, (\hat{z}, o)) \\ &((z, u), (\tilde{a}, a), (\tilde{a}, \hat{a}), +1, (\hat{z}, u)) \\ &((z, o), a, (\hat{a}, b), -1, (\hat{z}, o)) \\ &((z, u), a, (b, \hat{a}), +1, (\hat{z}, u)) \end{aligned}$$

für alle $\tilde{a} \in \Gamma$ ersetzt wird. Für $z = z_0$ (und $a = b$) kommt weiter noch die Instruktion

$$(z_0, a, (\hat{a}, *), +1, (\hat{z}, u))$$

hinzu. (Hiermit wird im ersten Rechenschritt die Randmarke $*$ in die untere Spur von Feld 0 gesetzt.) \square

- 3.13 LEMMA. (a) Zu jedem $t(n)$ -zeit- und $s(n)$ -platzbeschränkten k -(Halb-)Band on-line Turingakzeptor M kann man effektiv einen äquivalenten off-line Turingakzeptor M' vom ansonsten gleichen Typ angeben, der $(t(n) + 2n + 2)$ -zeit- und $\max(s(n), n + 1)$ -platzbeschränkt ist.
- (b) Umgekehrt kann man zu jedem $t(n)$ -zeit- und $s(n)$ -platzbeschränkten k -(Halb-)Band off-line Turingakzeptor M einen äquivalenten $(k + 1)$ -(Halb-)Band on-line Turingakzeptor M' vom ansonsten gleichen Typ angeben, der $t(n)$ -zeit- und $\max(s(n), n + 1)$ -platzbeschränkt ist.

BEWEISIDEE. Der off-line Turingakzeptor M' zur Simulation des on-line Turingakzeptors M kopiert zunächst das Eingabeband auf das erste Arbeitsband ($2n + 2$ Schritte) und arbeitet dann wie M . Der on-line Akzeptor M' zur Simulation des off-line Akzeptors M arbeitet wie M , wobei M' sein erstes Arbeitsband wie M sein Eingabeband behandelt und die weiteren Bänder $2, \dots, k + 1$ wie M' seine Arbeitsbänder $1, \dots, k$. \square

Zur Motivation der Wahl des off-line-Maschinenmodells bei der Platzkomplexität beobachten wir zunächst, dass sublineare Zeitschranken und im Falle von on-line-Maschinen geeignet zu definierende sublineare Platzschranken bereits zu konstanten Zeit- bzw. Platzschranken führen.

3.14 LEMMA. Sei $n \geq 0$ und sei M ein totaler Akzeptor, der eine 1-stellige Sprache akzeptiert und der keine Eingabe der Länge n vollständig einliest (d.h. das Feld direkt hinter der Eingabe wird nicht besucht). Dann ist M $O(1)$ -zeitbeschränkt und – falls wir dem Platzbedarf nur die Arbeitsfelder im Laufe der Rechnung zugrundelegen – $O(1)$ -platzbeschränkt. Weiter lässt sich die von M akzeptierte Sprache darstellen als

$$L(M) = F \cup \bigcup_{w \in I} w\Sigma^*, \quad (3.7)$$

wobei $F \subseteq \Sigma^{<n}$, $I \subseteq \Sigma^{=n}$ und $w\Sigma^* = \{wv : v \in \Sigma^*\}$.

BEWEIS. Für ein gegebenes Wort w der Länge n verhält sich M bei Eingabe wv für jedes $v \in \Sigma^*$ genauso wie bei Eingabe w . Rechenzeit und Speicherbedarf sind daher durch die endlich vielen Eingaben der Länge $\leq n$ bestimmt und damit durch eine Konstante beschränkt. Die Darstellung (3.7) von $L(M)$ ergibt sich, wenn man $F = \{w : |w| < n \text{ \& } w \in L(M)\}$ und $I = \{w : |w| = n \text{ \& } w \in L(M)\}$ wählt. \square

3.15 SATZ. Die folgenden Zeitkomplexitätsklassen fallen zusammen³

$$\text{DTIME}(O(1)) = \text{NTIME}(O(1)) = \text{DTIME}(n) = \text{NTIME}(n) \quad (3.8)$$

und sind echt in der Klasse REG der regulären Sprachen (über dem binären Alphabet) enthalten.

³Hierbei benutzen wir folgende Schreibweise. Für eine Klasse F von Schrankenfunktionen bezeichnet

$$\text{DTIME}(F) = \bigcup_{f \in F} \text{DTIME}(f(n))$$

(und entsprechend für die anderen Komplexitätsklassen).

BEWEISIDEE. Die Gleichheit der Sprachklassen in (3.8) ergibt sich wie folgt: Nach Definition sind diese Klassen bezüglich der Inklusion von links nach rechts geordnet. Da nach Lemma 3.14 die Sprachen in $\text{NTIME}(n)$ die Gestalt (3.7) haben, genügt es also zu zeigen, dass jede Sprachen von diesem Typ in $\text{DTIME}(O(1))$ liegt, d.h. von einer deterministischen Turingmaschine in konstanter Zeit erkannt wird. Da die in der Darstellung (3.7) vorkommenden Mengen F und I endlich sind, ist dies aber klar, da man für eine Eingabe w nur entscheiden muss, ob w in F oder ein Anfangsstück von w in I liegt. Zum Beweis des zweiten Teil des Satzes muss man dann noch zeigen, dass die Sprachen vom Typ (3.7) regulär sind, es aber andererseits reguläre Sprachen gibt, die nicht von diesem Typ sind. Ersteres ergibt sich aus der Tatsache, dass endliche Sprachen regulär sind, dass für jedes Wort w die Sprache $w\Sigma^*$ regulär ist und dass die Klasse der regulären Sprachen gegen Vereinigung abgeschlossen ist. Ein Beispiel für eine reguläre Sprache über dem binären Alphabet, die nicht vom Typ (3.7) ist, ist die Menge $\{0\}^*$ aller Unärwörter. \square

Betrachten wir off-line Turingmaschinen, so können wir – im Gegensatz zu den obigen Ergebnissen für on-line Maschinen – mit sublinearen Platzschranken nicht-reguläre Sprachen erkennen.

3.16 LEMMA. (i) $\text{REG} \subseteq \text{DSPACE}(1)$

(ii) $\{0^n 1^n 0^n : n \geq 0\} \in \text{DSPACE}(O(\log n))$

Da die Sprache $\{0^n 1^n 0^n : n \geq 0\}$ nicht kontextfrei ist, zeigt letzteres, dass

$$\text{DSPACE}(O(\log n)) \not\subseteq \text{KF},$$

also insbesondere mit (i) dass

$$\text{REG} \subset \text{DSPACE}(O(\log(n))).$$

BEWEISIDEE. Teil (i) folgt aus der Beobachtung, dass ein endlicher Automat gerade ein off-line Turingakzeptor ist, der die Eingabe einmal von links nach rechts liest (“read once”) und dann sofort akzeptiert oder verwirft ohne die Arbeitsbänder benutzt zu haben. Ein $O(\log(n))$ -platzbeschränkter off-line Akzeptor M für die Sprache $\{0^n 1^n 0^n : n \geq 0\}$ prüft zunächst, ob die Eingabe von der Form $0^n 1^m 0^l$ ist. (Hierzu werden die Arbeitsbänder nicht benötigt.) Dann schreibt M zunächst $\text{Bin}(n)$ auf sein 1. und 2. Arbeitsband. Hierzu inkrementiert M beim Einlesen von 0^n für jede gelesene Null das Binärwort. Da $|\text{Bin}(n)| \in O(\log(n))$ bleibt M hierbei innerhalb der zulässigen Platzschranke. Durch schrittweises Dekrementieren testet M dann, ob die folgenden Teile 1^m und 0^l der Eingabe jeweils Länge n haben. (Details s. Übungen.) \square

3.17 BEMERKUNG. Wie obiger Beweis zeigt, kann eine $O(\log(n))$ -platzbeschränkte off-line TM bis n zählen. Im folgenden wollen wir dies voraussetzen, werden also nur Platzschranken $s(n) \geq \log(n)$ betrachten. Für Ergebnisse über kleinere Platzschranken sei auf die Literatur verwiesen. Insbesondere kann man hier (in Entsprechung zu Lemma 3.14 für die Zeit) zeigen, dass $\log(\log(n))$ -platzbeschränkte off-line Akzeptoren bereits konstant beschränkt sind, also

$$\text{DSPACE}(\log(\log(n))) = \text{DSPACE}(O(1))$$

gilt (s. Literatur).

Zum Abschluss bemerken wir noch, dass es in der Regel genügt Zeit- und Platzschranken bis auf einen linearen Faktor zu bestimmen.

3.18 SATZ. (LINEARE KOMPRESSION) Sei $s : \mathbb{N} \rightarrow \mathbb{N}$ eine schwach monotone, unbeschränkte, total rekursive Funktion und sei $c \geq 1$. Dann kann man zu jedem $s(n)$ -platzbeschränkten off-line Turingakzeptor M effektiv einen äquivalenten $(\frac{1}{c}s(n))$ -platzbeschränkten Turingakzeptor M' des gleichen Typs angeben. Insbesondere gilt daher

$$(N,D)SPACE_{(k)}(s(n)) = (N,D)SPACE_{(k)}(O(s(n))).$$

BEWEISIDEE. Die Maschine M' fasst m -Blöcke ($m \geq 2$ fest) von Feldern von M auf einem Band, d.h. m benachbarte Felder, zu einem Feld zusammen. Formal bedeutet dies, dass das Bandalphabet Γ' von M' neben dem Blankzeichen b alle m -Tupel von Buchstaben aus Γ als Buchstaben enthält:

$$\Gamma' = \Gamma^m \cup \{b\}.$$

Im Zustand merkt sich M' dann die lokale Position p des M -Arbeitsfeldes innerhalb des M' -Arbeitsfeldes auf jedem Band ($Z' = Z \times \{1, \dots, m\}$, $z'_0 = (z_0, 1, \dots, 1)$). M' simuliert dann Schritt-für-Schritt M auf dieser geänderten Bandstruktur. Für eine 1-Band-Maschine M (und $m \geq 2$) wird dabei z.B. eine Instruktion

$$(z, a, \hat{a}, +1, \hat{z}) \in \delta$$

von M mit Rechtsbewegung durch folgende M' -Instruktionen ersetzt (für $1 \leq p < m$ und $a_j \in \Gamma$):

$$\begin{aligned} &((z, p), (a_1, \dots, a_{p-1}, a, a_{p+1}, \dots, a_m), (a_1, \dots, a_{p-1}, \hat{a}, a_{p+1}, \dots, a_m), 0, (\hat{z}, p+1)) \\ &((z, m), (a_1, \dots, a_{m-1}, a), (a_1, \dots, a_{m-1}, \hat{a}), +1, (\hat{z}, 1)) \end{aligned}$$

und zusätzlich für $a = b$

$$\begin{aligned} &((z, 1), b, (\hat{a}, b, \dots, b), 0, (\hat{z}, 2)) \\ &((z, m), b, (b, \dots, b, \hat{a}), +1, (\hat{z}, 1)). \end{aligned}$$

Da der Platzbedarf von M' durch

$$\frac{1}{m} \cdot s(n) + 1$$

beschränkt ist, erhält man die Behauptung für $m = c + 1$. □

3.19 SATZ. (LINEARE BESCHLEUNIGUNG) Sei $t : \mathbb{N} \rightarrow \mathbb{N}$ eine total rekursive Funktion mit $n \in o(t(n))$ (d.h. t ist hyperlinear) und sei $c \geq 1$. Dann kann man effektiv zu jedem $t(n)$ -zeitbeschränkten k -Band on-line Turingakzeptor M mit $k \geq 2$ einen äquivalenten Turingakzeptor M' des selben Typs angeben, der $\frac{1}{c} \cdot t(n)$ -zeitbeschränkt ist.

BEWEIS. Wir benutzen die Blocktechnik aus dem vorhergehenden Beweis. D.h. wir wählen $m \geq 2$ und definieren

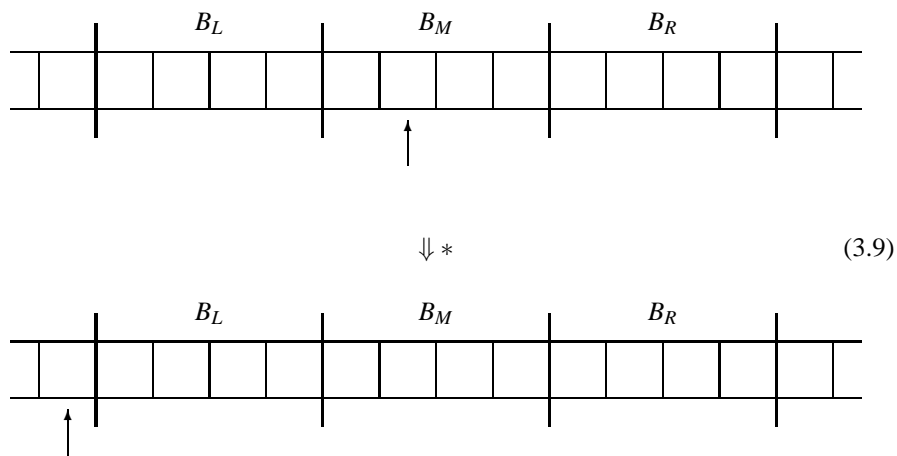
$$\Gamma' = \Gamma \cup \Gamma^m$$

Da die Eingabe nun auf dem Arbeitsband steht, benötigen wir hier $\Gamma \subseteq \Gamma'$ oder zumindest $\Sigma \cup \{b\} \subseteq \Gamma'$, und wir müssen die Eingabe zunächst in m -Blockform bringen, bevor wir mit der eigentlichen Simulation von M' beginnen können.

In dieser *Initialisierungsphase* kopiert M' die Eingabe von Band 1 in m -kompaktifizierter Form auf Band 2 unter gleichzeitigem Löschen der Bandinschrift 1. Dies erfordert bei einer Eingabe w der Länge n genau $n + 1$ Schritte. Danach wird der Kopf auf Band 2 noch vor die kompaktifizierte Eingabe gesetzt ($\frac{n}{m} + 2$ Schritte). (In der folgenden Simulationsphase interpretiert M' Band 1, 2, 3, ..., k als Band 2, 1, 3, ..., k von M .) Die Initialisierungsphase erfordert also insgesamt

$$n + \frac{n}{m} + 3 \text{ Schritte.}$$

In der Simulationsphase arbeitet M' in den im folgenden beschriebenen Zyklen (fester, von m unabhängiger Länge!), die eine Teilrechnung von M beschreiben, in deren Verlauf (auf einem Band) das Arbeitsfeld den aktuellen Block B_M und dessen linke und rechte Nachbarblöcke B_L bzw. B_R verlässt oder stoppt (im Bild für Blocklänge $m = 4$ und ein Band veranschaulicht, für den Fall, dass die Blöcke links verlassen werden):



Da (beim Nichtstoppen) M hierfür B_L oder B_R vollständig durchläuft, also zumindest m Schritte benötigt, können wir M hier beschleunigen, indem wir m größer als die Zyklenlänge von M' wählen. Da obige Teilrechnung von M nur von der Belegung der jeweiligen Blöcke B_L, B_M, B_R auf jedem Band abhängt und auch nur diese Blöcke verändert werden, kann M' diese Teilrechnung in maximal 8 Schritten simulieren:

1. Lese B_M und gehe nach rechts auf B_R .
2. Lese B_R und gehe nach links auf B_M .
3. Gehe nach links auf B_L .
4. Lese B_L , aktualisiere B_L und gehe nach rechts auf B_M .
5. Aktualisiere B_M und gehe nach rechts auf B_R .
- 6_r Aktualisiere B_R und gehe nach rechts, falls neues Arbeitsfeld rechts von B_R (oder falls M stoppt). Andernfalls:
- 6_l Aktualisiere B_R und gehe nach links auf B_M .
- 7_l Gehe nach links auf B_L .

8_l Gehe nach links.

Die Simulationsphase erfordert also insgesamt

$$\leq \frac{8}{m} \cdot t(n) + 8$$

Schritte, weshalb M' insgesamt $t'(n)$ -zeitbeschränkt ist für

$$t'(n) = n + \frac{n}{m} + 3 + \frac{8}{m} \cdot t(n) + 8 \leq (2n + 11) + \frac{8}{m} \cdot t(n)$$

Da nach Annahme $n \in o(t(n))$, also $dn \leq_{f.ü.} t(n)$ für alle $d \geq 1$ gilt, gilt insbesondere (für $d = 6c$)

$$2n + 11 \leq_{f.ü.} 3n <_{f.ü.} \frac{1}{2c} t(n).$$

Wählen wir also die Blocklänge m als $m = 16 \cdot c$, so erhalten wir insgesamt

$$t'(n) \leq_{f.ü.} \frac{1}{2c} t(n) + \frac{8}{16c} t(n) = \frac{1}{c} t(n),$$

so dass M' die gewünschte Beschleunigung aufweist. □

3.20 BEMERKUNG. Lineare Kompression und Beschleunigung werden oft als Indiz gesehen, dass die Turingmaschinenkomplexität nicht realistisch ist, da für Programme auf einem realen Rechner solche linearen speed-ups nicht möglich sind. Hierbei muss man jedoch berücksichtigen, dass das Turingmaschinenkonzept uns nicht einen Rechner sondern eine Familie von Rechnern gibt, die sich in der Mächtigkeit der einzelnen Schritte wesentlich unterscheiden. Vergrößern wir das Bandalphabet, so können wir in einem Feld mehr Information speichern, die wir gleichzeitig (parallel) in einem Schritt verarbeiten können. Die in den obigen Beweisen vorgenommene Vergrößerung des Bandalphabets stellt daher den Übergang zu einer mächtigeren, schnelleren Hardware dar.

Anders ausgedrückt: Würden wir die Kosten mit der Größe des Bandalphabets multiplizieren, wäre ein linearer speed-up wie oben nicht mehr möglich.