ZEIT- UND PLATZHIERARCHIEN

OLIVER THOMAS

1. Formale Sprachen

Definition 1 (Alphabet). Eine nicht-leere endliche Menge heißt Alphabet, die Elemente eines Alphabets heißen Buchstaben. Wir werden Alphabete meist mit Σ bezeichnen.

Beispiel 2. Klassisch sind die Alphabete $\{0,1\}$, $\{0,\dots,9\}$ oder die 255 ASCII-Zeichen.

Definition 3 (Wort, Länge, Formale Sprachen). Ist Σ ein Alphabet, so bezeichnen wir mit Σ^n die Menge aller n-Tupel in Σ (mit der Konvention $\Sigma^0 = \{\lambda\}$ für ein $\lambda \notin \Sigma$) und mit $\Sigma^* = \bigcup_{n\in\mathbb{N}_0} \Sigma^n$ die Menge aller abbrechenden Folgen in Σ. Ein Element aus Σ^* nennen wir Wort und λ nennen wir auch das leere Wort. Ist $w \in \Sigma^n$, so sagen wir auch, w habe die Länge n und schreiben $|w| = n$. Mit $w[i]$ bezeichnen wir die i-te Komponente von w. Eine Teilmenge $A \subseteq \Sigma^*$ nennen wir formale Sprache über Σ.

Wir können die natürlichen Zahlen offenbar als Teilmenge von $\{0,\dots,9\}^*$ interpretieren (und analog auch als Sprache über beliebigen Alphabeten), umgekehrt gilt aber auch:

Lemma 4. *Ist Σ ein Alphabet, so ist Σ^* abzählbar.*

Beweis. Bezeichne mit p_i die i-te Primzahl, dann ist für $\Sigma = \{\alpha_1,\dots,\alpha_n\}$ die Abbildung $w = (\alpha_{k_1},\dots,\alpha_{k_m}) \mapsto p_1^{k_1}\cdots p_m^{k_m}$ offenbar injektiv. ∎

Formale Sprachen sind ein sehr allgemeines Setting, in dem wir Probleme formulieren können. Bezeichnen wir mit U zum Beispiel die Menge der Unicode-Symbole, so gibt es in U^* die Menge ISO-C derjenigen Worte mit Buchstaben aus U, die gültige C-Programme nach ISO-Standard sind, die Menge GCC derjenigen C-Programme, die gegen gcc kompilieren, die Menge PRIME-P derjenigen C-Programme, die genau dann mit Ausgabe „t“ beenden, wenn ihre Eingabe eine Primzahl ist und mit „nil“ sonst, die Menge TERM derjenigen C-Programme, die bei jeder Eingabe terminieren – dann sind diese Mengen allesamt Sprachen über U. Interessant ist immer die Frage, ob ein gegebenes Wort in einer bestimmten Sprache ist („Implementiert gcc den gesamten Standard?“, „Stirbt mein Programm in einer Endlosschleife?“, …), bzw. formaler die charakteristische Funktion einer gegebenen Sprache. Wir können uns vorstellen, dass diese Frage zu beantworten unterschiedlich schwierig sein wird abhängig von der Sprache. Wichtig ist, dass wir anscheinend alle sich uns stellenden Probleme in dieser Form formulieren können (wir müssen uns bloß auf abzählbare Mengen beschränken).

2. (Deterministische) Turingmaschinen

Letztlich wollen wir uns überlegen, wie wir diese angesprochene Schwierigkeit genauer verstehen können. Wir haben gesehen, dass wir Probleme sehr gut umformulieren können, sodass es eigentlich nur um die Frage geht, ob ein bestimmtes Wort in einer bestimmten Sprache ist. Wenn das Problem formuliert ist, müssen wir uns überlegen, was wir als Lösung gelten lassen.

Definition 5 (Deterministische Turingmaschine). Eine deterministische k-Band-Turingmaschine (kurz k-DTM) ist ein Tupel bestehend aus:

(1) einer endlichen Menge Q, Zustände genannt

(2) einer Menge $Q_a \subseteq Q$, akzeptierende Zustände genannt
(3) einem Element $q_0 \in Q$, Anfangszustand genannt
(4) einem Alphabet Σ, Eingabealphabet genannt
(5) einem Alphabet $\Gamma \supsetneq \Sigma$, Bandalphabet genannt
(6) einem ausgezeichneten Element $\square \in \Gamma \setminus \Sigma$, Blank genannt
(7) und einer (nicht zwingend überall definierten, also sog. partiellen) Funktion

$$\delta : Q \times \Gamma^k \to Q \times \{-1, 0, +1\}^k \times \Gamma^k$$

Wenn wir ab sofort von einer Turingmaschine sprechen, dann werden wir mit Q, Q_a, q_0, Σ, Γ, $\square$ und δ die offensichtlichen Dinge meinen.
Hiermit haben wir erfolgreich formalisiert, was eine Turingmaschine in einem Schritt tut: Auf k Bändern (die unendlich lang gedacht sind, also formal Abbildungen $\mathbb{Z} \to \Gamma$) an der aktuellen Position die Daten einlesen, abhängig vom Zustand und diesen Daten pro Band etwas Neues auf das Band schreiben und (pro Band) in eine Richtung gehen oder stehen bleibe, sowie einen Zustandsübergang durchführen. Dieser Prozess lässt sich selbstverständlich auch formalisieren, was wir in Definition 8 tun werden, dafür brauchen wir aber noch etwas Vorarbeit.

Anmerkung 6. Ersetzen wir die Funktion δ in obiger Definition durch eine Relation, ist das die Definition einer nichtdeterministischen Turingmaschine.

Definition 7 (Konfiguration, (akzeptierende) Endkonfiguration). Eine Konfiguration einer k-DTM besteht aus einem Zustand $q \in Q$, Positionen $(x_i)_i \in \mathbb{Z}^k$ und Bändern $(t_i)_i \in \mathrm{Map}(\mathbb{Z}, \Gamma)^k$. Eine Konfiguration $(q, (x_i)_i, (t_i)_i)$ heißt Endkonfiguration, falls $\delta(q, (t_i(x_i))_i)$ nicht definiert ist, ist ferner $q \in Q_a$, so heißt diese Endkonfiguration akzeptierend.

Definition 8 (Betrieb einer DTM). Für eine k-DTM M und $w \in \Sigma^*$ definiere den Startzustand $C_0 = (q_0, 0, (t_i)_i)$ mit

$$t_i(j) = \begin{cases} w[j] & i = 0, j \in \{1, \dots, |w|\} \\ \square & \text{sonst.} \end{cases}$$

Für eine Konfiguration $C_i = (q, (x_i)_i, (t_i)_i)$, die keine Endkonfiguration ist, betrachte

$$\delta(q, (t_i(x_i))_i) = (q', (p_i)_i, (y_i)_i)$$

mit $q' \in Q, p_i \in \{-1, 0, +1\}, y_i \in \Gamma$ und definiere $C_{i+1} = (q', (x_i + p_i)_i, (t'_i)_i)$ mit

$$t'_i(j) = \begin{cases} y_i & j = x_i \\ t_i(j) & \text{sonst.} \end{cases}$$

Die Folge $(C_0, C_1, \dots)$ ist dann der Betrieb der Turingmaschine M bei Eingabe w.

Definition 9 (Halten, Akzeptierte Sprachen, Total, Entscheidbar). Landet ein DTM M nach endlich vielen Schritten für ein Eingabewort $w \in \Sigma^*$ in einer Endkonfiguration, so sagen wir, M hält an der Stelle w. Landet sie in einer akzeptierenden Endkonfiguration, so sagen wir, M akzeptiert w. Die akzeptierte Sprache ist dann die Menge alle akzeptierten Wörter. Hält M für alle $w \in \Sigma^*$, so heißt M total. Eine Sprache A heißt entscheidbar, falls es ein $k \in \mathbb{N}$ und eine totale k-DTM M gibt, sodass M genau die Wörter aus A akzeptiert.

Ohne Beweis akzeptieren wir folgenden

Satz 10. *Eine Sprache A ist genau dann entscheidbar, wenn es eine totale 1-DTM M gibt, sodass M genau die Wörter aus A akzeptiert, d. h. die Anzahl der Bänder ist irrelevant.*

3. Definition des Platz- und Zeitbedarfs

Definition 11 (Landau-Symbole). Ist $\limsup_{n\to\infty} |\frac{f(n)}{g(n)}| = 0$, so schreiben wir $f \in o(g)$, ist $\limsup_{n\to\infty} |\frac{f(n)}{g(n)}| < \infty$, so schreiben wir $f \in O(g)$. Häufig schreiben wir auch $O(g)$ bzw. $o(g)$, falls wir nur eine nicht näher bezeichnete Funktion aus der entsprechenden Menge meinen, also etwa $f = O(g)$ statt $f \in O(g)$. Gelegentlich werden wir auch Notationen wie $2^{O(g)}$ verwenden und meinen damit $\bigcup_{f\in O(g)} O(2^{f(n)})$.

Definition 12 (Zeitbedarf einer DTM). Ist M eine DTM mit Eingabealphabet Σ und geht die Turingmaschine bei Eingabe von $x \in \Sigma^*$ nach n Konfigurationsübergängen in eine Endkonfiguration über, so schreiben wir $t_M(x) = n$. Terminiert M bei Eingabe x nicht, so schreiben wir $t_M(x) = \infty$. Ferner betrachten wir die Abbildung $T_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}, n \mapsto \sup_{x\in\Sigma^n} t_M(x)$. Diese misst offenbar den (maximalen) Zeitbedarf einer DTM bei gegebener Eingabelänge.

Definition 13 (Zeitklasse). Für $\mathcal{F} \subseteq \mathrm{Abb}(\mathbb{N}, \mathbb{N})$ und Σ Alphabet setze $\mathrm{DTIME}(\mathcal{F})$ die Menge derjenigen Sprachen A über Σ, für die es eine totale (Mehrband-)DTM M und eine Zeitschranke $f \in \mathcal{F}$ gibt, sodass M genau die Worte in A akzeptiert und für alle $n \in \mathbb{N}$ die Ungleichung $T_M(n) \leq f(n)$ gilt.

Definition 14 (Platzbedarf einer DTM, Platzklasse). Die Definition des Platzbedarfs einer DTM M ist (technisch) etwas involvierter. Wir machen zuerst folgende Modellannahme: Unsere DTM hat $k + 2$ Bänder. Auf dem vorletzten Band steht die Eingabe. Dieses Band wird nur benutzt, um die Eingabe zu lesen, kein Schreibprozess findet hier statt. Auf das letzte Band wird die etwaige Ausgabe geschrieben, ein Leseprozess findet nicht statt. Auf den verbleibenden Bändern wird regulär gearbeitet. Wir definieren nun

$$S_M(n) = \sup_{y\in\Sigma^n} \left\{ \sum_{j=1}^{k} \left(1 + \sup_{i\in\mathbb{N}_0} x_{i,j}^{(y)} - \inf_{i\in\mathbb{N}_0} x_{i,j}^{(y)} \right) \right\},$$

wobei mit $x_{i,j}^{(y)}$ die Position des Kopfes des j-ten Bandes im i-ten Schritt bei Eingabe y bezeichnet werde. Wir definieren dann Platzklassen $\mathrm{DSPACE}(\mathcal{F})$ ganz analog zu Zeitklassen.

Wir können uns an dieser Stelle fragen, wie sehr Platz- und Zeitklasse von der Anzahl der Bänder abhängt. Der Beweis, dass Einband- mit Mehrband-DTMs äquivalent sind, liefert direkt den folgenden

Satz 15. *Zu jeder durch f zeit- und durch g platzbeschränkten Mehrband-DTM gibt es eine äquivalente durch $O(f(n)^2)$ zeit- und durch $O(g)$ platzbeschränkte 1-DTM.*

4. Wichtige Komplexitätsklassen

Definition 16. Einige Klassen bekommen eigene Namen[1], darunter insbesondere:

(1) $\mathrm{REG} = \mathrm{DSPACE}(O(1))$
(2) $\mathrm{L} = \mathrm{DSPACE}(\log n)$
(3) $\mathrm{P} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n^k)$
(4) $\mathrm{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(n^k)$
(5) $\mathrm{ESPACE} = \mathrm{DSPACE}(2^{O(n)})$
(6) $\mathrm{EXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(2^{n^k})$
(7) $k\mathrm{EXP} = \mathrm{DTIME}(f^k(n^{O(1)}) = (f \circ \cdots \circ f)(n^{O(1)})), f(n) = 2^n$
(8) $\mathrm{ELEMENTARY} = \bigcup_{k\in\mathbb{N}} k\mathrm{EXP}$

[1] vgl. `http://qwiki.stanford.edu/wiki/Complexity_Zoo` für eine ... umfassendere Liste

4.1. **Elementare Inklusionen**.

Satz 17. *Für $f \leq g$ ist DTIME$(f) \subseteq$ DTIME(g) und DSPACE$(f) \subseteq$ DSPACE(g). Ferner ist DTIME$(f) \subseteq$ DSPACE$(O(f))$, und falls $f \geq \log$, gilt außerdem DSPACE$(O(f)) \subseteq$ DTIME$(2^{O(f)})$.*

Beweis. Die erste Inklusion ist trivial. In jedem Konfigurationsübergang werden maximal k neue Felder beschrieben, wobei k die Anzahl der Bänder ist, damit gilt die zweite Inklusion. Sei nun $f \geq \log$ und M eine totale $k+2$-DTM, platzbeschränkt durch f, die genau die Worte aus $A \subseteq \Sigma^*$ akzeptiert und sei $w \in \Sigma^n$ so, dass $t_M(x) = T_M(n)$. Nach Voraussetzung terminiert M bei Eingabe w, also ist für den Betrieb $(C_0, \ldots, C_m)$ von M offenbar $C_i \neq C_j$ für $i \neq j$. Folglich gilt

$$T_M(n) \leq \#\{C \text{ Konfig.} \mid \text{Eingabeband} = w, t_i(\mathbb{Z} \setminus (-f(n), f(n))) = \{\square\} \text{ für } i \leq k+1\}.$$

Die Kardinalität dieser Menge können wir aber abschätzen, offenbar ist

$$\#\{\ldots\} \leq (n+2)(\#\Gamma)^{2(k+1)f(n)}(2f(n))^{k+1}\#Q \leq 2^{cf(n)},$$

denn es gibt $n+2$ mögliche Positionen auf dem Eingabeband, $(\#\Gamma)^{2(k+1)f(n)}$ mögliche Beschriftungen der Arbeitsbänder und des Ausgabebandes, und $(2f(n))^{k+1}$ mögliche Positionen auf den Arbeitsbändern und dem Ausgabeband. Für die letzte Abschätzung benutzen wir $n \leq 2^{f(n)}$, was aber nach Voraussetzung gilt. ∎

Daraus folgt dann sofort:

Korollar 18. *REG $\subseteq$ L $\subseteq$ P $\subseteq$ PSPACE $\subseteq$ ESPACE $\subseteq$ 1EXP $\subseteq$ EXPSPACE $\subseteq$ ELEMENTARY*

4.2. **Beispiele**.

Beispiel 19. $\{0^n1^m \mid n, m \in \mathbb{N}\}, \{0^{2^n} \mid n \in \mathbb{N}\} \in$ REG

Beispiel 20. $\{0^n1^n \mid n \in \mathbb{N}\}, \{0^n1^n2^n \mid n \in \mathbb{N}\} \in$ L

Wir sehen (live im Vortrag, nicht hier), dass es recht einfach zu zeigen ist, dass gewisse Probleme in gewissen Komplexitätsklassen sind: Wir konstruieren einfach eine entsprechende Turing-Maschine. Umgekehrt ist es ungleich schwieriger zu zeigen, dass gewisse Probleme nicht in gewissen Komplexitätsklassen sind – noch haben wir keine Argumente gesehen, dass etwa REG $\neq$ ELEMENTARY gilt.

5. Hierarchiesätze

Satz 21 (Platzhierarchiesatz). *DSPACE$(f(n)) \subsetneq$ DSPACE$(f(n)\log n)$ für hinreichend gutartige Funktionen f – sog. „platzkonstruierbare" Funktionen, also Funktionen, die in Platz $O(f)$ berechenbar sind. (Wir haben die Komplexitätsklassen nur für Sprachen definiert, aber wir können uns gut vorstellen, was das für Funktionen heißen soll.) Darunter sind insbesondere die Funktionen* $\log n, n, n^k, 2^n$.

Beweis. Der Beweis ist aufwendiger als wir Zeit und Platz haben – aber immerhin konstruktiv:

$$A = \{\text{code}(M) \mid M \text{ akzeptiert nicht code}(M) \text{ in Platz} \leq f(|x|)\}$$

ist eine Sprache mit der gewünschten Eigenschaft. Hierbei ist code(M) eine Codierung der Turingmaschine. Zu sehen, dass $A \notin$ DSPACE$(f(n))$ gilt, ist einfach: Wäre A via M in DSPACE$(f(n))$, so führt die Betrachtung von $M(\text{code}(M))$ zu einem Widerspruch. Dass allerdings $A \in$ DSPACE$(f(n)\log n)$ gilt, ist ein ganzes Stück mehr Arbeit. ∎

Satz 22 (Zeithierarchiesatz). *Mit beinahe identischem Beweis gilt für hinreichend gutartige Funktionen f ferner DTIME$(f) \subsetneq$ DTIME$(f(n)^3)$.*

Korollar 23. *P $\subsetneq$ EXP $\subsetneq$ ELEMENTARY, L $\subsetneq$ PSPACE $\subsetneq$ EXPSPACE.*