Der Satz von COOK (1971)

Voraussetzung: Das Konzept der k-Band-Turing-Maschine (TM)

1.) Notationen:

Eine *momentane Beschreibung (mB)* einer *Konfiguration* einer TM ist ein k-Tupel $(\alpha_1, \alpha_2, ..., \alpha_k)$ mit $\alpha_i = x_i q y_i$, falls die TM im Zustand q ist, auf dem i-ten Band das erste Zeichen des Wortes y_i liest und $x_i y_i BBBB...$ die i-te Bandinschrift ist.

Eine mB heisst *akzeptierend*, wenn q gleich dem akzeptierenden Zustand q_a ist.

Wir schreiben $\beta_1 \therefore_M \beta_2$, gewenn β_2 mB der auf die Konfiguration mit mB β_1 folgenden Konfiguration ist. Und wir schreiben $\beta_1 \overset{*}{\therefore}_M \beta_n$, gewenn $\exists \beta_2, ..., \beta_{n-1}$, so dass $\beta_1 \therefore_M \beta_2 \therefore_M \beta_3 \therefore_M ..... \therefore_M \beta_{n-1} \therefore_M \beta_n$.

Ein *Alphabet* ist eine endliche Menge A von Zeichen (Symbolen). Ein *Wort* w über A ist eine endliche Folge von Zeichen aus A, inkl. dem leeren Wort ε.

Eine *Sprache* ist eine Menge L von Wörtern. $L = A^*$ bezeichnet die Sprache aller Wörter über dem Alphabet A.

Die *Anfangskonfiguration zum Input x* (ein Wort über A) ist diejenige mit der mB $(q_0 x, q_0, q_0, ..., q_0)$.

Die von der TM M *akzeptierte Sprache* ist die Menge aller Wörter (Inputs) x, für die eine mB β' existiert, so dass $\beta \overset{*}{\therefore}_M \beta'$, wobei β die mB der Anfangskonfiguration zum Input x und β' akzeptierend ist.

Falls die TM M, in der Anfangskonfiguration zum Input x gestartet, schliesslich anhält, so sei $f(x)$ das erste Wort auf dem letzten Band bis vor das erste Blank-Zeichen. f heisst *die von M berechnete partielle Funktion*. Falls M mit Input x nie anhält, so ist f an der Stelle x nicht definiert.

Eine TM M heisst *T-Zeit-beschränkt* für eine Funktion $T: \mathbb{N} \to \mathbb{N}$, falls für jeden Input x der Länge n die Berechnung (wenn sie überhaupt terminiert) höchstens $T(n)$ Schritte dauert. (Hält die TM nach $T(n)$ Schritten noch nicht an, so hält sie also nie an!)
Es ist also $T(n) = \max_{x \in A_1^*, |x|=n} \{\text{Anzahl Schritte zur Berechnung}\}$.

2.) Nicht-deterministische TMen:

Alle bisherigen TMen waren deterministisch: Zu jedem Zeitpunkt war der nächste Schritt eindeutig gegeben. Nun können wir uns aber auch Maschinen vorstellen (natürlich nicht technisch verwirklicht), die zu jedem Zeitpunkt mehrere mögliche Fortsetzungen haben und aus diesen „irgendwie“ auswählen.

Eine *nicht-deterministische* TM ist ein 7-Tupel $(Q, I, A, \delta, B, q_0, q_a)$, wobei

- Q eine endliche Menge von Zuständen
- I das Inputalphabet
- A das Bandalphabet mit $I \subseteq A$
- $B \in A \setminus I$ das Leerzeichen (Blank)
- q_0 der Startzustand
- q_a der akzeptierende Zustand
- $\delta : Q \times A^k \to P\left(Q \times \left(A \times \{L, R, S\}\right)^k\right)$ die Programm-Funktion ist, die dem aktuellen Zustand und den aktuell gesehenen Zeichen den neuen Zustand und die neuen Zeichen samt den Bewegungen (für jedes Band) zuordnet. (L=Links, R=Rechts, S=Stop) Ein Funktionswert von δ ist aber ein Element der Potenzmenge von $Q \times \left(A \times \{L, R, S\}\right)^k$, d.h. es stehen der Maschine zu jedem Zeitpunkt mehrere mögliche Fortsetzungen offen! *Ein* möglicher Berechnungspfad der nicht-det. TM ergibt sich aus der Auswahl von jeweils einer der Aktionen aus $\delta\left(q, (a_1, ..., a_k)\right)$.

Eine nicht-det. TM *akzeptiert ein Wort (einen Input)*, falls es, von der Startkonfiguration ausgehend, mindestens einen akzeptierenden Berechnungspfad gibt, an dessen Ende die Maschine also im akzeptierenden Zustand q_a ist. Achtung: Wir sagen nicht, dass die Maschine diesen einen Pfad auch wirklich wählt. Wir sagen nur, dass die Maschine, sollte sie zufälligerweise den „richtigen“ Pfad (oder einen der richtigen Pfade) wählen, schliesslich das Input-Wort akzeptieren wird.

Die nicht-det. TM M heisst *T-Zeit-beschränkt* für eine Funktion $T: \mathbb{N} \to \mathbb{N}$, falls

$$T(n) = \max_{|x|=n} \min_{\substack{\text{akzept. Berechn.pfade} \\ \text{zum Input } x \in L}} \left\{ \begin{array}{l} \text{Länge des Berechn.pfades bis} \\ \text{zum akzeptierenden Zustand} \end{array} \right\}.$$

3.) Das P-NP-Problem:

Wir definieren nun als *P-Probleme* alle Sprachen, die von einer deterministischen TM in polynomialer Zeit akzeptiert werden, und als *NP-Probleme* alle Sprachen, die von einer nicht-deterministischen TM in polynomialer Zeit akzeptiert werden!

P:= Menge aller Sprachen, die von einer det. TM in polynom. Zeit akzeptiert werden.

NP := Menge aller Sprachen, die von einer nicht-det. TM in polynom. Zeit akzeptiert werden.

(NP = **N**on deterministic **p**olynomial)

Klar ist, dass $P \subset NP$ gilt, denn eine det. TM ist insbesondere auch eine nicht-det. mit nur einer möglichen Aktion pro Schritt. Wird also eine Sprache von einer det. TM in polynomialer Zeit akzeptiert, so auch von einer nicht-det. TM, nämlich derselben.

Die Frage ist aber, ob die Inklusion echt ist. Man vermutet dies natürlich stark: Sei nämlich eine Sprache von einer nicht-det. TM in polynomialer Zeit akzeptierbar, so hat die Maschine zu jeder Zeit mehrere mögliche Aktionen, die wir uns in einem Baumdiagramm von polynomialer Tiefe angeordnet vorstellen können. Und mindestens ein Pfad dieses Baumes führt (in polynomialer Zeit) zum akzeptierenden Zustand. Selbstverständlich kann dieselbe Sprache auch von einer det. TM akzeptiert werden: sie müsste ja einfach jeden möglichen Pfad des Baumes prüfen. Es ist aber sehr unwahrscheinlich, dass diese Baumsuche auch in polynomialer Zeit geschehen kann, denn wir können uns nur vorstellen, dass die det. TM eben jeden möglichen Pfad einzeln und nacheinander prüft, und dies dauert eben länger als polynomiale Zeit! Es spricht somit einiges dafür, dass eine NP-Sprache, also eine Sprache, die von einer nicht-det. TM in polynomialer Zeit $p(n)$ akzeptiert wird, von einer det. TM in einer Zeit $c^{p(n)}$ akzeptiert wird.

Wir sind daher geneigt zu glauben, dass $P \subsetneq NP$ gilt, aber streng bewiesen ist dies bis zum heutigen Tag nicht! Wir fragen deshalb auch heute noch:

Ist $P \subsetneq NP$ oder ist $P = NP$?	Das P-NP-Problem

4.) NP-vollständige Probleme:

Eine Sprache L_1 heisst *polynomial transformierbar* in eine Sprache L_2, gewenn es eine det. polynomial Zeit-beschränkte TM M gibt, welche eine Funktion f berechnet mit der Eigenschaft: $x \in L_1 \Leftrightarrow f(x) \in L_2$. Bezeichnung: $L_1 \leq_p L_2$.

Die Bedeutung dieser Definition ist, dass L_1 „nicht schwieriger“ ist als L_2: Kann nämlich L_2 in polynomialer Zeit entschieden werden, so auch L_1! Nehmen wir zur Begründung an, es soll $x \overset{?}{\in} L_1$ getestet werden. Dann berechnen wir mit der TM M in polynomialer Zeit $f(x)$ und entscheiden nun ebenfalls in polynomialer Zeit, ob $f(x) \in L_2$ oder nicht. Wegen der obigen Äquivalenz ist dann auch in polynomialer Zeit entschieden, ob $x \in L_1$ oder nicht ist.

Eine Sprache L heisst *NP-hart*, gewenn $L_1 \leq_p L \quad \forall L_1 \in NP$.

Die Bedeutung dieser Definition ist, Sprachen klassifizieren zu können, die mindestens so schwierig sind wie alle NP-Probleme.

Eine Sprache L heisst *NP-vollständig*, gewenn $L \in NP$ und NP-hart ist.

Die Bedeutung dieser Definition ist, Sprachen klassifizieren zu können, die zu den schwierigsten unter den NP-Problemen gehören. Ist ein Problem NP-vollständig, so ist jedes andere NP-Problem höchstens so schwierig: Mehr noch: Kann von einem einzigen NP-vollständigen

Problem gezeigt werden, dass es zu P gehört, so gehört automatisch jedes andere NP-Problem zu P und es ist $P = NP$. Damit wäre das $P - NP$-Problem ein für allemal entschieden. (Könnte nämlich von dem NP-vollständigen Problem L gezeigt werden, dass es zu P gehört, so gilt für jedes andere NP-Problem L_1, dass $L_1 \leq_p L$, und damit zählt dann eben auch L_1 zu P!)

Nun bliebe das alles graue Theorie, wenn es gar keine NP-vollständigen Probleme gäbe. Die Definition allein erzeugt natürlich noch keine konkreten Probleme! Deshalb ist ein 1971 erschienener Satz von COOK, der zum ersten Mal in der Geschichte von einem Problem dessen NP-Vollständigkeit bewies, von enormer Bedeutung. Der Satz war so bahnbrechend, dass er heute neben den Unvollständigkeitssätzen von Gödel als wichtigster Satz der theoretischen Informatik gilt.

5.) Was ist „SAT“?

Um den Satz von COOK sinnvoll formulieren zu können, muss die Sprache erklärt werden, die sich dann als NP-vollständig herausstellt!

Die Aussagenlogik wird syntaktisch aufgebaut aus Aussagevariablen $A_0, A_1, A_2, ...$ und den Junktoren $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$.

Als *aussagenlogische Formeln* gelten per definitionem alle Aussagevariablen; sind ferner φ, ψ aussagenlogische Formeln, so auch $\varphi \wedge \psi, \varphi \vee \psi, \neg\varphi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi$. Nichts anderes ist eine aussagenlogische Formel.

Eine *Belegung der Aussagevariablen* ist eine Funktion b, die jeder Aussagevariablen einen der Wahrheitswerte T, F zuordnet.

Der Wahrheitswert f einer aussagenlogischen Formel bei einer Belegung b wird definiert durch:

- $f(A_i) = b(A_i)$
- $f(\neg\varphi) = \neg f(\varphi)$, wobei $\neg T = F$ und $\neg F = T$.
- $f(\phi \wedge \psi) = \wedge(f(\varphi), f(\psi))$, wobei $\wedge(T,F) = \wedge(F,T) = \wedge(F,F) = F$ und $\wedge(T,T) = T$.
- $f(\varphi \vee \psi) = \vee(f(\varphi), f(\psi))$, wobei $\vee(T,F) = \vee(F,T) = \vee(T,T) = T$ und $\vee(F,F) = F$
- usw.

Eine aussagenlogische Formel heisst *erfüllbar*, gewenn es eine Belegung gibt, so dass die Formel den Wahrheitswert T bekommt.

Eine aussagenlogische Formel ist in *konjunktiver Normalform* (KNF), gewenn sie die folgende Form hat:

$$(A_{11} \vee A_{12} \vee ... \vee A_{1i_1}) \wedge (A_{21} \vee A_{22} \vee ... \vee A_{2i_2}) \wedge ... \wedge (A_{j1} \vee A_{j2} \vee ... \vee A_{ji_j}),$$

wobei alle A_{rs} Aussagevariablen oder negierte Aussagevariablen sind. Die A_{rs} heissen *Literale*, die Klammerterme heissen *Klauseln*. Vertauscht man überall $\wedge$ und $\vee$, so wäre die Formel in *disjunktiver Normalform* (DNF).

Es ist zwar möglich, jede aussagenlogische Formel in die KNF umzuformen, dies ist aber nicht immer in polynomialer Zeit möglich. Ist die Formel etwa in DNF gegeben, so ist die Umformung sehr aufwändig! Man kann aber in polynomialer Zeit eine Formel in KNF finden, die zu einer gegebenen Formel in dem Sinne äquivalent ist, dass die eine erfüllbar ist, gewenn es die andere ist. Und weil also zu einer gegebenen Formel „schnell“ eine äquivalente in KNF gefunden werden kann, wäre es natürlich toll, man könnte eine Formel in KNF auch „schnell“ auf Erfüllbarkeit testen! (Es gibt übrigens bis heute keinen schnellen Algorithmus, der eine Formel in DNF bringt!) Dieser Wunsch ist aber leider nicht erfüllbar:

Benennen wir die Sprache der *erfüllbaren aussagenlogischen Formeln in KNF* mit dem Begriff „SAT“ (Satisfiability), so gilt nämlich:

6.) Der Satz von COOK (1971)

SAT ist NP-vollständig!

(Quelle: „The complexity of theorem-proving procedures“, Proceedings of the 3rd annual STOC (Symposium on Theory of Computer), 1971, pp. 151-158)

Nach 1971 konnte für viele Probleme, für die kein schneller Algorithmus gefunden werden konnte, nachgewiesen werden, dass sie NP-vollständig sind. Das hilft in einem negativen Sinne: Hat man nämlich ein Problem vor sich, von dem man weiss, dass es NP-vollständig ist, so weiss man, dass es ungeheuer schwierig ist. Man wird dann die Suche nach einem schnellen Algorithmus aufgeben und vielmehr mit einer guten Näherungslösung zufrieden sein!

7.) Beweis des Satzes von COOK:

am Dienstag, 18. März, 2003, als „amuse bouche“!