4. Entscheidbarkeit — Unentscheidbarkeit

4.1. Akzeptierungsproblem und Halteproblem für Turing–Maschinen

Unter einem *(Entscheidungs–) Problem* verstehen wir eine Frage der Art *"Ist die natürliche Zahl n ein Quadrat?"* Für eine Festlegung des Parameters n ist die Antwort entweder *"ja"* oder *"nein"*. Eine *Instanz* eines Problems ist eine Liste von Argumenten, ein Argument für jeden Parameter des Problems. So ist etwa (15) eine Instanz des obigen Problems.

Einem Problem P mit Parametern $p_1, \ldots, p_k$ kann auf offensichtliche Weise eine Sprache L_P wie folgt zugeordnet werden:

$$L_P = \{(n_1, \ldots, n_k) \;\mid \text{die Antwort auf die Instanz } (n_1, \ldots, n_k) \text{ des Problems } P \text{ ist "ja"}\}.$$

Definition 4.1.1: Ein Problem P, dessen Sprache L_P rekursiv ist, heißt *entscheidbar*. Andernfalls heißt P *unentscheidbar*. Ist L_P r.a., so sagt man auch P wäre *semi–entscheidbar*. ■

Definition 4.1.2: Zunächst bringen wir jede Turing–Maschine in eine gewisse kanonische Form, man spricht dabei von der *Kodierung* einer Turing–Maschine. Sei M eine Turing–Maschine mit Zustandsmenge $\{q_1, \ldots, q_n\}$, Eingabealphabet $\{0, 1\}$, Bandalphabet $\{0, 1, \sqcup\}$, Anfangszustand q_1, Endzustandsmenge $F = \{q_2\}$ und Überführungsfunktion δ. Das stellt keine Einschränkung dar, da jedes Alphabet Σ in $\{0, 1\}$ kodiert werden kann. Wir bezeichnen die Bewegungsrichtungen "links", "rechts" mit D_1, D_2 (der Einfachheit halber lassen wir nicht zu, daß der LS–Kopf stationär bleibt; diese Situation kann aber immer durch eine Folge von rechts–links Bewegungen simuliert werden), und die Symbole 0,1,$\sqcup$ mit X_1, X_2, X_3. Eine allgemeine Operation von M hat also die Gestalt

$$\delta(q_i, X_j) = (q_k, X_l, D_m).$$

Diese Operation wird kodiert als

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m. \tag{4.1.1}$$

Die Turing–Maschine M wird kodiert als

$$111\ code_1\ 11\ code_2\ 11\ \ldots\ 11\ code_r\ 111, \tag{4.1.2}$$

wobei jeder $code_i$ die Form (4.1.1) hat und jede Operation von M von einem $code_i$ kodiert ist. Die (bis auf Umordnung der $code_i$'s eindeutig bestimmte) Kodierung von M wird mit $\langle M\rangle$ bezeichnet. ■

Beispiel 4.1.1: Als Beispiel einer solchen Kodierung betrachten wir die Turing–Maschine M mit $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$, $F = \{q_2\}$, und $\delta(q_1, 1) = (q_3, 0, R)$, $\delta(q_3, 0) = (q_1, 1, R)$, $\delta(q_3, 1) = (q_2, 0, R)$, $\delta(q_3, \sqcup) = (q_3, 1, L)$. Eine mögliche Kodierung von M (Version von $\langle M\rangle$) ist

1110100100010100110001010100100110001001001010011000100010001001 0111. ■

Es ist also klar, daß die (Kodes von) Turing–Maschinen in einer Reihe $M_1, M_2, M_3, \ldots$ angeordnet werden können (etwa der Länge nach und alphabetisch bei gleicher Länge). Ebenso können die Wörter in $\{0,1\}^*$ in einer Reihe $w_1, w_2, w_3, \ldots$ angeordnet werden. Für $i, j \in \mathbf{N}$ definieren wir nun

$$a_{ij} = \begin{cases} 0, & \text{falls } w_i \in L(M_j), \\ 1, & \text{sonst.} \end{cases}$$

Auf diese Weise erhalten wir eine nach unten und rechts unendliche Matrix

Wörter	Turing-Maschine			
	M_1	M_2	M_3	$\cdots$
w_1	$\mathbf{a_{11}}$	a_{12}	a_{13}	$\cdots$
w_2	a_{21}	$\mathbf{a_{22}}$	a_{23}	$\cdots$
w_3	a_{31}	a_{32}	$\mathbf{a_{33}}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Definition 4.1.3: Wir benutzen die Diagonalelemente in dieser Matrix zur Definition der *Diagonalsprache*

$$L_d = \{\ w_j \ \mid\ a_{jj} = 1\ \}.$$

Also $w_j \in L_d \Longleftrightarrow w_j \notin L(M_j)$. ■

Satz 4.1.1: *Die Diagonalsprache L_d ist nicht r.a.*

Beweis: Angenommen es gäbe eine Turing–Maschine M_j, welche L_d akzeptiert. Daraus würde sich folgende Kontradiktion ergeben: falls $w_j \in L_d$, dann $a_{jj} = 1$, also $w_j \notin$

$L(M_j)$. Andererseits, falls $w_j \notin L_d$, dann $a_{jj} = 0$, also $w_j \in L(M_j)$. $L(M_j)$ wäre also verschieden von L_d. ■

Definition 4.1.4: Das *Akzeptierungsproblem für Turing–Maschinen* ist das Problem, für eine gegebene Turing–Maschine M (mit Bandalphabet $\{0, 1, \sqcup\}$) und ein gegebenes Wort $w \in \{0, 1\}^*$ zu entscheiden, ob M das Wort w akzeptiert. Die zum Akzeptierungsproblem gehörige Sprache heißt die *universelle Sprache*. Sie ist definiert als

$$L_u = \{ \ \langle M \rangle w \ \mid \ M \text{ akzeptiert } w \ \}.$$ ■

L_u ist universell in dem Sinn, daß die Frage ob eine Folge w in $\{0, 1\}^*$ von einer Turing–Maschine M (o.B.d.A. mit Bandalphabet $\{0, 1, \sqcup\}$) akzeptiert wird äquivalent ist zur Frage ob $\langle M \rangle w \in L_u$.

Satz 4.1.2: *L_u ist r.a.*

Beweis: Wir konstruieren eine Turing–Maschine mit 3 Bändern M_u^3, welche L_u akzeptiert. Das erste Band von M_u^3 ist das Eingabeband und enthält also eine Instanz $\langle M \rangle w$ des Akzeptierungsproblems. Die Operationen der Turing–Maschine M sind zwischen den ersten zwei Blöcken von drei 1en auf dem ersten Band kodiert. Das zweite Band von M_u^3 simuliert das Band von M. Auf dem dritten Band von M_u^3 wird der Zustand q_i von M als 0^i abgespeichert. Die Turing–Maschine M_u^3 funktioniert folgendermaßen:

(1) Überprüfe den Inhalt von Band 1 darauf hin, ob er mit einem Wort der Form (4.1.2) beginnt und es keine zwei Operationskodes gibt, welche mit 0^i10^j1 für dasselbe i und j beginnen. Weiters prüfe für jeden Operationskode $0^i10^j10^k10^l10^m$ ob $1 \leq j \leq 3$, $1 \leq l \leq 3$ und $1 \leq m \leq 2$. Bei diesen Berechnungen kann Band 3 als Arbeitsband benutzt werden.

(2) Schreibe w, den Teil der Eingabe nach dem zweiten Block von drei 1en, auf Band 2. Schreibe 0 (die Kodierung für den Zustand q_1) auf Band 3. Alle LS–Köpfe werden am linken Bandende positioniert. Diese Symbole können markiert werden, sodaß sie leicht wieder aufgefunden werden können.

(3) Enthält Band 3 das Wort 00, den Kode für den Endzustand q_2, so halte und akzeptiere.

(4) Sei X_j das aktuelle Symbol (auf welches der LS–Kopf zeigt) von Band 2 und sei 0^i der Inhalt von Band 3. Suche auf Band 1 von links bis zum zweiten Block

von drei 1en nach einem Unterwort mit dem Präfix 110^i10^j1. Wird ein solches Wort nicht gefunden, so halte und weise die Eingabe zurück; M kann keinen nächsten Schritt ausführen und hat nicht akzeptiert. Wird jedoch ein Kode der Form $0^i10^j10^k10^l10^m$ gefunden, so schreibe 0^k auf Band 3, schreibe X_l auf die aktuelle Bandzelle in Band 2 und bewege den LS–Kopf 2 in die Richtung D_m. Fahre fort mit Schritt (3).

Offensichtlich akzeptiert M_u^3 das Eingabewort $\langle M\rangle w$ genau dann, wenn M das Wort w akzeptiert. Hält M nicht auf w oder hält es ohne zu akzeptieren, so tut M_u^3 dasselbe auf $\langle M\rangle w$. ■

Definition 4.1.5: Da L_u r.a. ist, gibt es eine Turing–Maschine M_u (mit einem Band), welche L_u akzeptiert. Diese (allerdings nicht eindeutig bestimmte) Turing–Maschine M_u heißt die *universelle Turing–Maschine*, da sie die Arbeit jeder Turing–Maschine verrichtet. ■

Satz 4.1.3: *L_u ist nicht rekursiv, das Akzeptierungsproblem für Turing–Maschinen ist also unentscheidbar.*

Beweis: Wir nehmen an, es gäbe eine Turing–Maschine M, welche L_u entscheidet (d.h. M stoppt auf jeder Eingabe und akzeptiert L_u). Dann könnten wir L_d wie folgt entscheiden: für eine gegebene Eingabe w bestimmen wir zunächst den Index i, für welchen gilt $w = w_i$. Aus i berechnen wir die Turing–Maschine M_i. Die Folge $\langle M_i\rangle w_i$ geben wir nun als Eingabe an M und akzeptieren w genau dann, wenn M die Folge $\langle M_i\rangle w_i$ nicht akzeptiert, also M_i das Wort w_i nicht akzeptiert. Da wir aber aus (1) wissen, daß L_d nicht rekursiv ist, kann es keinen solchen Entscheidungsalgorithmus geben, also auch keine Turing–Maschine M, die L_u entscheidet. ■

Ein scheinbar einfacher zu behandelndes Problem ist das *Halteproblem "stoppt (hält) die Berechnung der Turing–Maschine M auf dem Eingabewort w?"* Man sieht aber leicht, daß auch das Halteproblem unentscheidbar sein muß, da man andernfalls sofort einen Algorithmus zur Entscheidung des Akzeptierungsproblems konstruieren könnte. Bei genauerer Betrachtung stellt sich sogar heraus, daß es nicht einmal einen Algorithmus gibt, der von einer Turing–Maschine entscheiden könnte, ob sie auf der leeren Eingabe stoppt.

Def. 4.1.6: Mit der selben Bezeichnung wie in Def. 4.1.4 seien

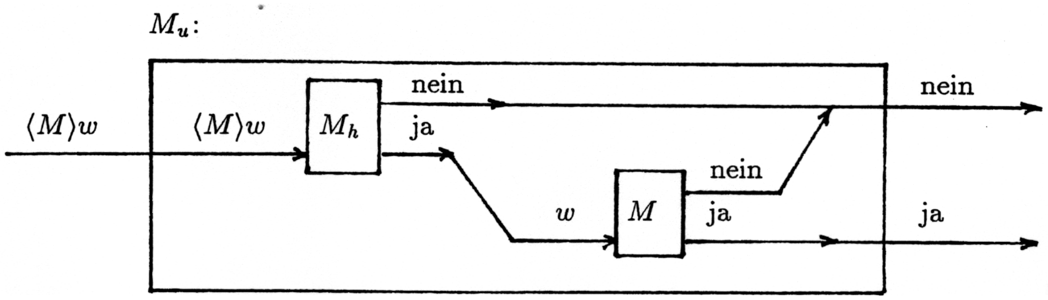
$$L_h = \{ \langle M \rangle w \ \mid \ M \text{ stoppt bei Eingabe } w \},$$

$$L_{h,\epsilon} = \{ \langle M \rangle \ \mid \ M \text{ stoppt bei Eingabe } \epsilon \}.$$

L_h ist die zum *Halteproblem* gehörige Sprache, $L_{h,\epsilon}$ die zum *eingeschränkten Halteproblem* gehörige Sprache. ■

Satz 4.1.4: *L_h ist nicht rekursiv, also das Halteproblem für Turing–Maschinen ist unentscheidbar.*

Beweis: Wäre L_h rekursiv, so wäre auch L_u rekursiv. Wäre nämlich M_h eine immer terminierende Turingmaschine, welche L_h akzeptiert, so wäre M_u eine immer terminierende universelle Turingmaschine:



■

Auch das eingeschränkte Halteproblem $L_{h,\epsilon}$ ist unentscheidbar, wir können das aber erst später beweisen.

Die üblichen Programmiersprachen, wie FORTRAN, Pascal, etc. sind universell, d.h. sie sind in der Lage, jede Turing–Maschine zu simulieren und umgekehrt. Es kann also z.B. auch kein Pascal–Programm geben, welches von einem beliebig vorgegebenen Pascal–Programm entscheiden könnte, ob dieses bei der Eingabe 0 stoppt.

4.2. Der Satz von Rice

Wir haben festgestellt, daß es unmöglich ist von einer Turing–Maschine M und einem Wort w zu entscheiden, ob M das Wort w akzeptiert, ja es ist nicht einmal möglich zu entscheiden, ob M auf der leeren Eingabe stoppt! Es erhebt sich also die Frage: Welche Eigenschaften von Turing–Maschinen sind denn nun entscheidbar? Oder mit anderen Worten: Welche Eigenschaften von r.a. Sprachen sind entscheidbar?

Zunächst definieren wir, was wir unter einer Eigenschaft r.a. Sprachen verstehen wollen.

Definition 4.2.1: Wir nennen jede Menge $\mathcal{S}$ rekursiv aufzählbarer Sprachen (über $\{0,1\}$) eine *Eigenschaft* r.a. Sprachen. $\mathcal{S}$ ist eine *triviale* Eigenschaft, wenn $\mathcal{S}$ leer ist oder alle r.a. Sprachen enthält. Die Eigenschaft $\mathcal{S}$ heißt *entscheidbar*, wenn die Sprache

$$L_{\mathcal{S}} = \{\ \langle M\rangle \ \mid\ L(M) \in \mathcal{S}\ \}$$

rekursiv ist, wenn es also möglich ist von jeder Turing–Maschine zu entscheiden, ob die von ihr akzeptierte Sprache in $\mathcal{S}$ ist. ■

Satz 4.2.1 (Satz von Rice [1]): *Jede nichttriviale Eigenschaft r.a. Sprachen ist unentscheidbar.*

Beweis: Sei $\mathcal{S}$ eine nichttriviale Eigenschaft r.a. Sprachen. Wir können annehmen, daß $\emptyset \notin \mathcal{S}$, denn andernfalls könnten wir anstatt $\mathcal{S}$ sein Komplement $\overline{\mathcal{S}}$ betrachten. Sei nun L eine Sprache in $\mathcal{S}$ und M_L eine Turing–Maschine , die L akzeptiert. Wir konstruieren nun eine Turing–Maschine A, welche den Kode $\langle M\rangle$ einer Turing–Maschine und ein Wort w als Eingabe nimmt und den Kode $\langle M'\rangle$ einer anderen Turing–Maschine produziert, sodaß

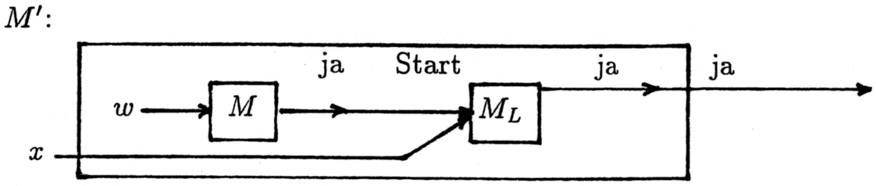
$$L(M') \in \mathcal{S} \iff \langle M\rangle w \in L_u \quad (M \text{ akzeptiert } w).$$

Konstruktion von A:

> A produziert den Kode einer Turing–Maschine M' mit folgender Eigenschaft: zuerst ignoriert M' seine Eingabe und simuliert M auf w. Wird w von M nicht

[1] H.G. Rice: *Classes of recursively enumerable sets and their decision problems*, *Trans. Amer. Math. Soc.* 89, 25–59 (1953).

akzeptiert, so akzeptiert auch M' sein Eingabewort x nicht. Andernfalls simuliert M' das Verhalten von M_L auf x und akzeptiert x genau dann, wenn M_L x akzeptiert.



Also

$$L(M') = \begin{cases} \emptyset, & \text{falls } w \notin L(M), \\ L, & \text{sonst.} \end{cases}$$

Wäre nun $\mathcal{S}$ entscheidbar, so gäbe es eine Turing–Maschine $M_{\mathcal{S}}$, welche $L_{\mathcal{S}}$ entscheidet. Das aber hätte zur Folge, daß wir einen Algorithmus zur Entscheidung von L_u konstruieren könnten:

1. Wende A auf $\langle M \rangle w$ an und erzeuge $\langle M' \rangle$.

2. Wende $M_{\mathcal{S}}$ auf $\langle M' \rangle$ an. Ist die Antwort "ja", so ist $\langle M \rangle w \in L_u$, andernfalls nicht.

L_u ist aber nicht rekursiv, also kann auch $\mathcal{S}$ nicht entscheidbar sein. ■

Nun sind wir in der Lage, relativ einfach einzusehen, daß auch das eingeschränkte Halteproblem unentscheidbar ist. Dazu betrachten wir auch noch das eingeschränkte Akzeptierungsproblem.

Definition 4.2.2: Mit der selben Bezeichnung wie in Def. 4.1.4 sei

$$L_{u,\epsilon} = \{ \langle M \rangle \mid M \text{ akzeptiert } \epsilon \}.$$

$L_{u,\epsilon}$ ist die zum *eingeschränkten Akzeptierungsproblem* gehörige Sprache. ■

Satz 4.2.2: *$L_{u,\epsilon}$ ist nicht rekursiv.*

Beweis: Sei $\mathcal{S}_a$ die Eigenschaft r.a. Sprachen, das leere Wort zu enthalten, also

$$L_{\mathcal{S}_a} = \{ \langle M \rangle \mid \epsilon \in L(M) \}.$$

$\mathcal{S}_a$ ist eine nichttriviale Eigenschaft r.a. Sprachen, und $L_{\mathcal{S}_a} = L_{u,\epsilon}$. Wegen des Satzes von Rice ist $\mathcal{S}_a$ unentscheidbar. ■

Satz 4.2.3: *$L_{h,\epsilon}$ ist nicht rekursiv.*

Beweis: Wäre das eingeschränkte Halteproblem entscheidbar, so wäre auch das eingeschränkte Akzeptierungsproblem entscheidbar. Für gegebenes $\langle M \rangle$ müßten wir einfach entscheiden, ob M auf ϵ hält, und gegebenenfalls ob M das leere Wort ϵ akzeptiert. Wegen Satz 4.2.2 kann das aber nicht möglich sein. ∎

4.3. Die Unentscheidbarkeit eines Pflasterungsproblems

Wir geben uns eine endliche Menge von Typen von Pflastersteinen vor, von denen jeder die Form des Einheitsquadrats hat. Das Problem besteht darin, den ersten Quadranten der Ebene mit Pflastersteinen der gegebenen Typen zu pflastern. Die einzigen Einschränkungen bestehen darin, daß ein spezieller "Anfangs"Stein an der linken unteren Ecke plaziert werden muß, und daß nur gewisse Typen horizontal und vertikal benachbart sein dürfen. Es ist auch nicht gestattet, die Pflastersteine zu drehen.

Definition 4.3.1: Etwas formaler verstehen wir unter einer *Instanz des Pflasterungsproblems* ein Quadrupel $\mathcal{D} = (D, d_0, H, V)$, wobei D eine endliche Menge (von Pflastersteintypen) ist, $d_0 \in D$ (Anfangsstein), und $H, V \subseteq D \times D$ (erlaubte Paare benachbarter Typen). Eine *Pflasterung* durch $\mathcal{D}$ ist eine Funktion $f : \mathbf{N} \times \mathbf{N} \to D$ sodaß

$$f(0,0) = d_0,$$

$$(f(n,m), f(n+1,m)) \in H \quad \text{und} \quad (f(n,m), f(n,m+1)) \in V \qquad \text{für alle } n, m \in \mathbf{N}.$$

Das *Pflasterungsproblem* lautet also wie folgt: *"Gibt es zum Pflasterungssystem $\mathcal{D}$ eine Pflasterung durch $\mathcal{D}$?"* ■

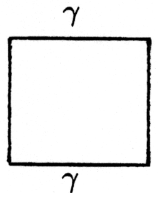
Satz 4.3.1: Das Pflasterungsproblem ist unentscheidbar.

Beweis: Daß das Pflasterungsproblem unentscheidbar ist, zeigen wir dadurch, daß wir das eingeschränkte Halteproblem darauf reduzieren. Wäre also das Pflasterungsproblem entscheidbar, so wäre auch das Halteproblem entscheidbar. Das ist aber laut Satz 4.2.3 nicht der Fall.

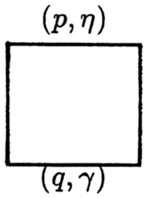
Dazu ist es notwendig für jede Turing–Maschine M ein Pflasterungssystem $\mathcal{D}_M$ zu konstruieren, sodaß es genau dann eine Pflasterung durch $\mathcal{D}_M$ gibt, wenn die Berechnung von M auf der leeren Eingabe nicht stoppt. Um die Betrachtung zu vereinfachen, schränken wir Turing–Maschinen dahingehend ein, daß sie nur entweder schreiben können, oder den LS–Kopf bewegen, aber nicht beides gleichzeitig. Man kann sich leicht überlegen, daß auf diese Weise die Berechnungspotenz einer Turing–Maschine nicht geschmälert wird.

Ist nun M eine Turing–Maschine mit Zustandsmenge Q, Anfangszustand q_0, Bandalphabet Γ und Überführungsfunktion δ, so ist $\mathcal{D}_M = (D, d_0, H, V)$, wobei die Paare in H, V an den benachbarten Seiten übereinstimmen müssen und D die folgenden Typen von Pflastersteinen enthält:

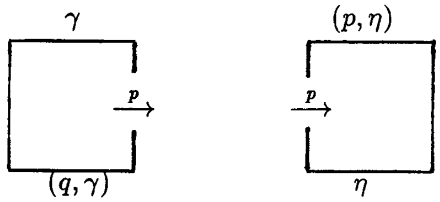
(a) Für jedes Symbol $\gamma \in \Gamma$ die Type



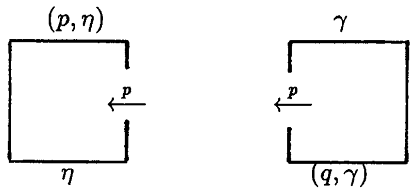
(b) Für jedes $\gamma \in \Gamma$ und $q \in Q$, sodaß $\delta(q, \gamma) = (p, \eta)$ für gewisse $p \in Q$, $\eta \in \Gamma$ die Type



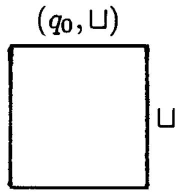
(c) Für jedes $\gamma \in \Gamma$ und $q \in Q$, sodaß $\delta(q, \gamma) = (p, R)$ für ein gewisses $p \in Q$, und für jedes $\eta \in \Gamma$ die Typen



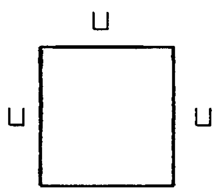
(d) Für jedes $\gamma \in \Gamma$ und $q \in Q$, sodaß $\delta(q, \gamma) = (p, L)$ für ein gewisses $p \in Q$, und für jedes $\eta \in \Gamma$ die Typen



(e) Darüberhinaus sei d_0 die Type

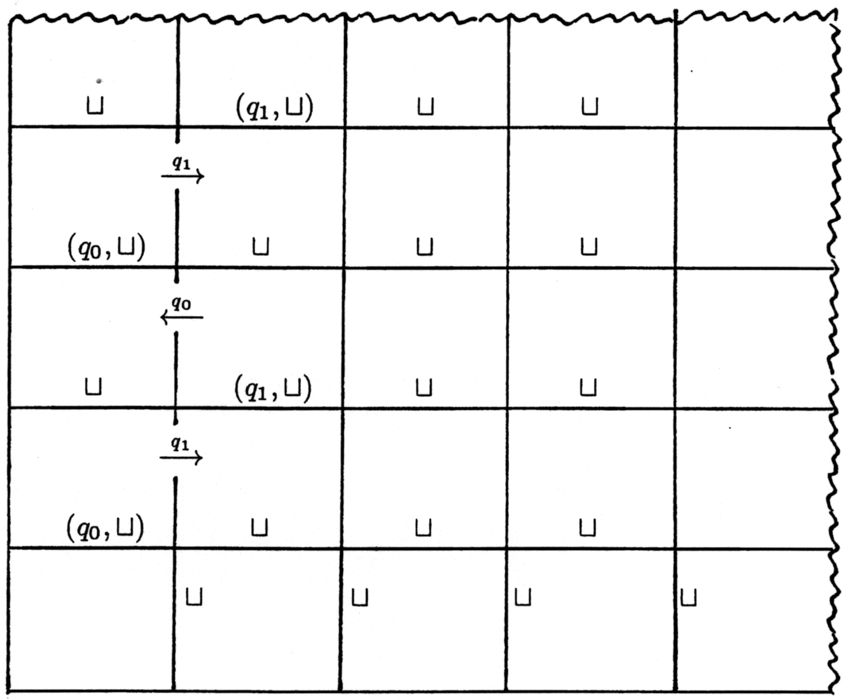


und wir haben auch die die folgende Type zur Verfügung.



Durch diese Festlegung von $\mathcal{D}_M$ erzwingen wir, daß die "Zeilen" einer durch $\mathcal{D}_M$ erzeugten Pflasterung den Konfigurationen in der Berechnung der Turing–Maschine M auf der leeren Eingabe entsprechen. Der erste Quadrant kann also genau dann durch $\mathcal{D}_M$ gepflastert werden, wenn die Berechnung von M auf der leeren Eingabe nicht stoppt. Somit ist die Reduktion des Halteproblems auf das Pflasterungsproblem durchgeführt. ■

Beispiel 4.3.1: Als Beispiel betrachten wir die Turing–Maschine M_4 mit $Q = \{q_0, q_1\}$, $\Gamma = \{\sqcup\}$, $\delta(q_0, \sqcup) = (q_1, R)$, $\delta(q_1, \sqcup) = (q_0, L)$. M_4 bewegt einfach den LS–Kopf hin und her und kommt dabei niemals über die zweite Bandzelle hinaus. Die Pflasterung des ersten Quadranten, die der unendlichen Berechnung von M_4 auf der leeren Eingabe entspricht, sieht folgendermaßen aus



4.4. Das Korrespondenzproblem von Post

Ein anderes unentscheidbares Problem ist das Korrespondenzproblem von Post. Die Unentscheidbarkeit des Korrespondenzproblems von Post hat zur Folge, daß auch viele andere interessante Problem als unentscheidbar erwiesen werden können, so z.B. die Mehrdeutigkeit kontextfreier Grammatiken.

Definition 4.4.1: (a) Eine *Instanz des Korrespondenzproblems von Post (Post's correspondence problem, PCP)* besteht aus zwei Listen von Wörtern

$$A = w_1, \dots, w_k \qquad \text{und} \qquad B = x_1, \dots, x_k$$

über einem Alphabet Σ. Diese Instanz des PCP *hat eine Lösung* genau dann, wenn es eine Folge von ganzen Zahlen $i_1, \dots, i_m \in \{1, \dots, k\}$, mit $m \geq 1$, gibt, sodaß

$$w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}.$$

Die Folge $i_1, \dots, i_m$ ist eine *Lösung* dieser Instanz des PCP.

(b) Ein *Instanz des modifizierten Korrespondenzproblems von Post (MPCP)* sieht aus wie eine Instanz des PCP. Für eine *Lösung* des MPCP muß aber zusätzlich gelten $i_1 = 1$, also der Lösungsstring muß jeweils mit dem ersten Element der Listen A, B beginnen. ■

Beispiel 4.4.1: Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

(a) Wir betrachten die Instanz des PCP, welche durch folgende Listen gegeben ist:

	Liste A	Liste B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Diese Instanz des PCP hat eine Lösung, nämlich

$$m = 4, i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3,$$

also

$$w_2 w_1 w_1 w_3 = 101111110 = x_2 x_1 x_1 x_3.$$

(b) Wir betrachten die Instanz des PCP, welche durch folgende Listen gegeben ist:

	Liste A	Liste B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Angenommen $i_1, \ldots, i_m$ ist eine Lösung dieser Instanz des PCP. Um die Übereinstimmung am Stringanfang zu gewährleisten muß gelten $i_1 = 1$. Also wir haben bisher als Teillösung

$$w_1 = 10$$
$$x_1 = 101$$

Wir müssen nun aus A ein Wort wählen, das mit 1 beginnt, also $i_2 \in \{1, 3\}$. $i_2 = 1$ funktioniert aber nicht, da wir sonst in der Teillösung eine Diskrepanz an der vierten Stelle hätten. Somit bleibt nur $i_2 = 3$, also die Teillösung

$$w_1 = 10101$$
$$x_1 = 101011$$

Nun sind wir aber wieder in der gleichen Situation wie vorhin, nämlich daß die Teillösung aus B um eine 1 länger ist als die Teillösung aus A. Mit derselben Argumentation wie oben ergibt sich also $i_3 = i_4 = \ldots = 3$. Eine andere Wahlmöglichkeit gibt es nicht. Die beiden Strings können also nie dieselbe Länge haben. Somit ist diese Instanz des PCP unlösbar. ■

Satz 4.4.1: *Wäre* PCP *entscheidbar, dann wäre auch* MPCP *entscheidbar.*

Beweis: Sei

$$A = w_1, \ldots, w_k, \qquad B = x_1, \ldots, x_k$$

eine Instanz von MPCP. Wir transformieren diese Instanz von MPCP in eine Instanz von PCP, welche genau dann lösbar ist, wenn die gegebene Instanz von MPCP lösbar ist. Damit ist der Satz dann bewiesen.

Sei Σ das kleinste Alphabet, welches alle Symbole in A und B enthält und zusätzlich die beiden Symbole $\emptyset$ und \$. Sei y_i das Wort, das aus w_i entsteht, indem man nach jedem Symbol in w_i das Symbol $\emptyset$ einfügt, und sei z_i das Wort, das aus x_i entsteht, indem man vor jedem Symbol in x_i das Symbol $\emptyset$ einfügt. Weiters betrachten wir neue Wörter

$$y_0 = \emptyset y_1, \qquad z_0 = z_1,$$

$$y_{k+1} = \$, \qquad z_{k+1} = \emptyset\$.$$

Seien nun

$$C = y_0, y_1, \ldots, y_{k+1}, \qquad D = z_0, z_1, \ldots, z_{k+1}.$$

Für die Listen aus Beispiel 4.4.1(a) ergibt sich etwa dadurch folgende Transformation:

MPCP:

	Liste A	Liste B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

PCP:

	Liste C	Liste D
i	y_i	z_i
0	∅1∅	∅1∅1∅1
1	1∅	∅1∅1∅1
2	1∅0∅1∅1∅1∅	∅1∅0
3	1∅0∅	∅0
4	\$	∅\$

Die Listen C und D spezifizieren eine Instanz des PCP, welche genau dann lösbar ist, wenn die gegebene Instanz des MPCP lösbar ist.

Denn ist etwa $1, i_1, \ldots, i_r$ eine Lösung des MPCP, dann ist $0, i_1, \ldots, i_r, k+1$ eine Lösung des PCP.

Andererseits, ist $i_1, \ldots, i_r$ eine Lösung des PCP, dann muß gelten $i_1 = 0$ und $i_r = k+1$. Sei j der kleinste Index sodaß $i_j = k+1$. Dann ist $i_1, \ldots, i_j$ auch eine Lösung, da das Symbol \$ nur als letztes Symbol von y_{k+1} und z_{k+1} vorkommt, und für kein l mit $1 \leq l < j$ gilt $i_l = k+1$. Somit ist $1, i_1, \ldots, i_{j-1}$ offenbar eine Lösung des MPCP.

Gibt es also einen Algorithmus um PCP zu entscheiden, so erhalten wir daraus auch einen Algorithmus um MPCP zu entscheiden. ∎

Mit dieser Hilfsüberlegung sind wir nun in der Lage, die Unentscheidbarkeit von PCP zu beweisen.

Satz 4.4.2: PCP *ist unentscheidbar.*

Beweis: Wir zeigen, daß das Akzeptierungsproblem für Turing–Maschinen auf MPCP reduzierbar ist, d.h. wäre MPCP entscheidbar, so wäre auch das Akzeptierungsproblem entscheidbar. Das ist aber laut Satz 4.1.3 nicht der Fall. Somit ist MPCP nicht entscheidbar, und wegen Satz 4.4.1 auch PCP nicht entscheidbar.

Für jede TM M und jedes Wort w konstruieren wir ein MPCP, welches lösbar ist genau dann, wenn es eine Lösung der Form

$$\#q_0w\#u_1q_1v_1\#\cdots\#u_kq_kv_k\#$$

hat, wobei Strings zwischen aufeinanderfolgenden # aufeinanderfolgende Konfigurationen von M sind bei Eingabe w, und q_k ein akzeptierender Zustand von M ist.

Wir geben nun die Listen A und B des zugehörigen MPCP an. Wir nehmen an, daß die TM M stoppt, sobald ein akzeptierender Zustand erreicht wird. Außer für das erste Paar vergeben wir keine Nummern, da diese für die Existenz einer Lösung irrelevant sind.

Gruppe 0: erstes Paar

Liste A	Liste B
$\#$	$\#q_0w\#$

Gruppe I: (Kopieren) für $\alpha \in \Gamma$:

Liste A	Liste B
α	α
$\#$	$\#$

Gruppe II: (Anwenden der Überführungsfunktion) für $q \in Q \setminus F$, $p \in Q$, $\alpha, \beta, \gamma \in \Gamma$:

Liste A	Liste B	
$q\alpha$	βp	falls $\delta(q,\alpha) = (p,\beta,R)$
$\gamma q\alpha$	$p\gamma\beta$	falls $\delta(q,\alpha) = (p,\beta,L)$
$q\#$	$\beta p\#$	falls $\delta(q,\sqcup) = (p,\beta,R)$
$\gamma q\#$	$p\gamma\beta\#$	falls $\delta(q,\sqcup) = (p,\beta,L)$

Gruppe III: (Endbehandlung) für $q \in F$, und $\alpha, \beta \in \Gamma$:

Liste A	Liste B
$\alpha q\beta$	q
αq	q
$q\beta$	q

Gruppe VI: (Liste A abschließen) für $q \in F$:

Liste A	Liste B
$q\#\#$	$\#$

Wir nennen (x, y) eine *Teillösung* von MPCP bzgl. der Listen A und B, wenn x ein Anfangsstück (Prefix) von y ist, und x bzw. y durch Verkettung korrespondierender Strings aus den Listen A bzw. B hervorgehen. Ist $xz = y$, so nennen wir z den *Rest* von (x, y).

Angenommen ausgehend von der Konfiguration q_0w gibt es eine gültige Folge von k weiteren Konfigurationen, also eine Berechnung dieser Länge (die aber nicht notwendigerweise hier zu Ende sein muß). Dann behaupten wir, daß es eine Teillösung der Form

$$(x, y) = (\#q_0w\#u_1q_1v_1\# \cdots \#u_{k-1}q_{k-1}v_{k-1}\#,\\ \#q_0w\#u_1q_1v_1\# \cdots \#u_{k-1}q_{k-1}v_{k-1}\#u_kq_kv_k\#)$$

gibt. Zudem ist das die einzige Teillösung, deren längerer String so lang ist wie $|y|$.

Wir beweisen diese Behauptung durch Induktion über k. Für $k = 0$ ist die Behauptung trivial, da das Paar $(\#, \#q_0w\#)$ als erstes gewählt werden muß.

Angenommen die Behauptung gilt für ein k, und daß q_k nicht in F ist. Der Rest des Paares (x, y) ist $z = u_kq_kv_k\#$. Die nächsten Paare müssen nun so gewählt werden, daß ihre Anteile aus A den String z ergeben. Unabhängig davon welche Symbole rechts und links von q_k stehen, gibt es höchstens ein Paar in Gruppe II, welches es erlaubt, die Teillösung über q_k hinaus fortzusetzen. Dieses Paar entspricht in natürlicher Weise einem Berechnungsschritt von M ausgehend von der Konfiguration $u_kq_kv_k$. Die anderen Symbole von z führen zwangsläufig zu Paaren aus der Gruppe I. Keine andere Auswahl von Paaren erlaubt es, z darzustellen als Verkettung von Elementen der Liste A. Somit erhalten wir eine neue Teillösung $(y,\, yu_{k+1}q_{k+1}v_{v+1}\#)$. Offensichtlich ist $u_{k+1}q_{k+1}v_{k+1}$ die einzige Konfiguration, welche M in einem Schritt von $u_kq_kv_k$ erreichen kann. Es gibt auch keine andere Teillösung, deren zweiter String so lang ist wie $|yu_{k+1}q_{k+1}v_{k+1}|$. Somit ist die Behauptung auch für $k + 1$ nachgewiesen.

Ist nun $q_k \in F$, so findet man leicht Paare aus den Gruppen I und III, welche es erlauben die Teillösung (x, y) zu einer Lösung des MPCP mit Listen A und B zu vervollständigen, nachdem als letztes Paar dasjenige aus Gruppe IV gewählt wurde.

Somit ist klar, daß MPCP mit Listen A und B eine Lösung besitzt, wenn M ausgehend von der Konfiguration q_0w einen akzeptierenden Zustand erreicht. Erreicht M keinen akzeptierenden Zustand, so können die Paare aus den Gruppen III und IV nicht angewendet werden. Somit wird in jeder Teillösung der zweite String länger sein als der erste, und es ist daher keine vollständige Lösung möglich.

Abschließend stellen wir also fest, daß die Instanz des MPCP genau dann eine Lösung hat, wenn M bei Eingabe w in einem akzeptierenden Zustand hält. Die obige Konstruktion kann für jede TM M und Eingabewort w ausgeführt werden. Gäbe es also einen Algorithmus zur Entscheidung von MPCP, dann gäbe es auch einen Algorithmus zur Entscheidung des Akzeptierungsproblems für Turing–Maschinen, im Widerspruch zu Satz 4.1.3. ■

Übung: Sei M die TM

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, \sqcup\}, q_0, \{q_2\}, \delta),$$

wobei δ wie folgt definiert ist:

	0	1	$\sqcup$
q_0	$(q_1, 1, R)$	$(q_1, 0, L)$	$(q_1, 1, L)$
q_1	$(q_2, 0, L)$	$(q_0, 0, R)$	$(q_1, 0, R)$
q_2	$-$	$-$	$-$

Wie sieht die zugehörige Instanz des MPCP aus? M akzeptiert das Wort $w = 01$ in 4 Berechnungsschritten. Wie sieht die zugehörige Lösung der Instanz des MPCP aus?

4.5. Es gibt auch interessante entscheidbare Problem

Lösbarkeit algebraischer Gleichungssysteme

Zu gegebenen Polynomen $f_1(x_1, \ldots, x_\nu), \ldots, f_m(x_1, \ldots, x_\nu)$ im Polynomring $\mathbb{Q}[x_1, \ldots, x_\nu]$ in den Variablen $x_1, \ldots, x_\nu$ über dem Körper $\mathbb{Q}$ entscheide man, ob es eine Lösung des Gleichungssystems

$$\begin{aligned} f_1(x_1, \ldots, x_\nu) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_\nu) &= 0 \end{aligned} \tag{4.5.1}$$

in $\mathbb{C}^n$ gibt. Dazu berechnet man eine reduzierte Gröbner–Basis $g(x_1, \ldots, x_\nu), \ldots, g_n(x_1, \ldots, x_\nu)$ für das von $f_1, \ldots, f_m$ aufgespannte Ideal [2]. Das Gleichungssystem (4.5.1) hat genau dann eine Lösung, wenn eines der Polynome g_i das Einspolynom ist.

Als Beispiel betrachten wir das algebraische Gleichungssystem

$$\begin{aligned} xz - xy^2 - 4x^2 - \frac{1}{4} &= 0 \\ y^2 z + 2x + \frac{1}{2} &= 0 \\ x^2 z + y^2 + \frac{1}{2}x &= 0 \end{aligned}$$

Durch den Übergang zur Gröbner–Basis erhalten wir das neue Gleichungsystem

$$\begin{aligned} z + \frac{64}{65}x^4 - \frac{432}{65}x^3 + \frac{168}{65}x^2 - \frac{354}{65}x + \frac{8}{5} &= 0 \\ y^2 - \frac{8}{13}x^4 + \frac{54}{13}x^3 - \frac{8}{13}x^2 + \frac{17}{26}x &= 0 \\ x^5 - \frac{27}{4}x^4 + 2x^3 - \frac{21}{16}x^2 + x + \frac{5}{32} &= 0 \end{aligned}$$

Das Gleichungssystem ist also lösbar, und die Approximation $(-0.128475,\ 0.321145,\ -2.356718)$ einer Lösung ist leicht ablesbar.

---

[2] B. Buchberger: *Gröbner bases: An algorithmic method in polynomial ideal theory*, in: *Mathematical Systems Theory*, N.K. Bose (ed.), D. Reidel Publ. Comp., Dordrecht/Boston/Lancaster (1985).

Integration transzendenter elementarer Funktionen

Für eine gegebene transzendente elementare Funktion $f(x)$ — aufgebaut aus den rationalen Funktionen $\mathbb{C}(x)$ über dem komplexen Zahlkörper durch Adjunktion von Logarithmen und Exponentialen — stelle man fest, ob $g(x) = \int f(x)dx$ ebenfalls eine transzendente elementare Funktion ist und berechne gegebenenfalls eine Darstellung von $g(x)$.

Dieses Problem ist mithilfe des Algorithmus von Risch [3)] lösbar, und wird in der Praxis auch von Programmsystemen der Computer Algebra gelöst. So erhält man etwa

$$\int 2x \cdot e^{x^2} \cdot \log(x) + \frac{e^{x^2}}{x} + \frac{\log(x) - 2}{(\log^2(x) + x)^2} + \frac{(2/x) \cdot \log(x) + 1/x + 1}{\log^2(x) + x}\, dx \quad =$$

$$e^{x^2} \cdot \log(x) - \frac{\log(x)}{\log^2(x) + x} + \log(\log^2(x) + x)$$

oder die Antwort, daß

$$\int e^{x^2} dx$$

nicht transzendent elementar ist.

Parametrisierung algebraischer Kurven

Eine (ebene) algebraische Kurve $\mathcal{C}$ besteht aus denjenigen Punkten $P = (x, y)$ in der Ebene, welche eine algebraische Gleichung der Form

$$f(x, y) = 0$$

lösen, wobei $f(x, y)$ ein Polynom in den Variablen x, y ist.

Ein Beispiel dafür ist die sogenannte "Tacnode"–Kurve $\mathcal{T}$, deren Punkte Lösungen der Gleichung

$$2x^4 - 3x^2y + y^4 - 2y^3 + y^2 = 0$$

sind. Ein Bild der Tacnode–Kurve befindet sich auf der nächsten Seite.

Diese Gleichungsdarstellung algebraischer Kurven ist manchmal vorteilhaft, etwa wenn man feststellen will, ob ein gegebener Punkt auf der Kurve liegt oder nicht. Sie eignet sich aber nicht besonders gut für andere Aufgabenstellungen, etwa wenn

---

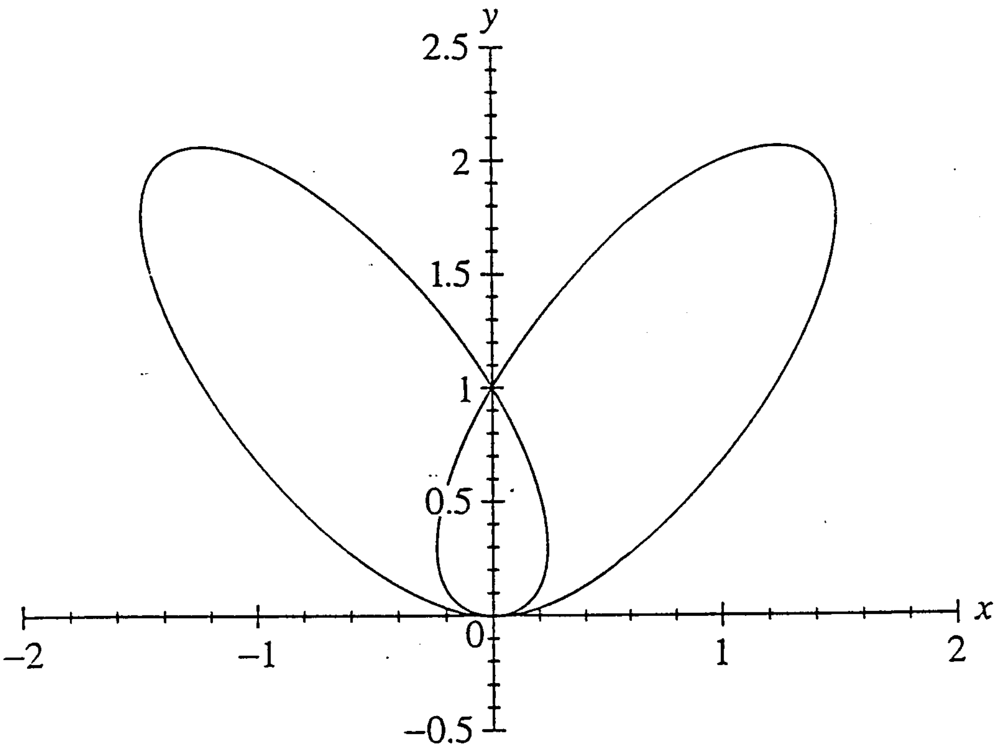
[3)] R.H. Risch: *The problem of integration in finite terms*, *Trans. Amer. Math. Soc.* 139, 167–189 (1969).

man Punkte auf der Kurve erzeugen will. Dazu wäre eine Parameterdarstellung, etwa mittels rationaler Funktionen in einem Parameter, vorteilhaft. Dabei drückt man die Koordinaten von Punkten auf der Kurve aus durch rationale Funktionen in einem unabhängigen Parameter:

$$x(t) = \frac{p_1(t)}{q_1(t)}, \quad y(t) = \frac{p_2(t)}{q_2(t)},$$

wobei p_1, p_2, q_1, q_2 Polynome in t sind. Allerdings ist nicht jede algebraische Kurve so darstellbar. Das Problem der rationalen Parametrisierung algebraischer Kurven besteht also darin, für eine gegebene Kurve in Gleichungsdarstellung zu entscheiden, ob sie parametrisierbar ist, und gegebenenfalls eine solche Parameterdarstellung zu berechnen. Eine symbolische algorithmische Lösung dieses Problems findet man etwa in den Arbeiten von Sendra und Winkler [4]. So kann die Tacnode–Kurve wie folgt rational parametrisiert werden:

$$x(t) = \frac{t^3 - 6t^2 + 9t - 2}{2t^4 - 16t^3 + 40t^2 - 32t + 9}, \quad y(t) = \frac{t^2 - 4t + 4}{2t^4 - 16t^3 + 40t^2 - 32t + 9}.$$



Tacnode–Kurve

[4] J.R. Sendra und F. Winkler: *Symbolic Parametrization of Curves*, *J. Symbolic Computation* 12, 607–631 (1991),
J.R. Sendra und F. Winkler: *Parametrization of Algebraic Curves over Optimal Field Extensions*, *J. Symbolic Computation* 23, 191–207 (1997).

4.6. Unentscheidbarkeitshierarchie

Das *Emptiness–Problem* besteht darin, für eine gegebene Turingmaschine M zu entscheiden, ob $L(M) = \emptyset$. Die zugehörige Sprache ist

$$L_e = \{\langle M\rangle | L(M) = \emptyset\}.$$

Satz 4.6.1: *Seien*

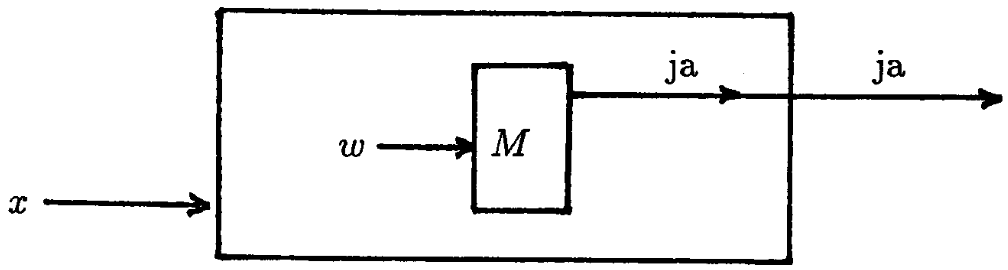
$$L_e = \{\langle M\rangle | L(M) = \emptyset\} \quad \textit{und} \quad L_{ne} = \{\langle M\rangle | L(M) \neq \emptyset\}.$$

L_{ne} ist r.a. aber nicht rekursiv, und L_e ist nicht r.a.

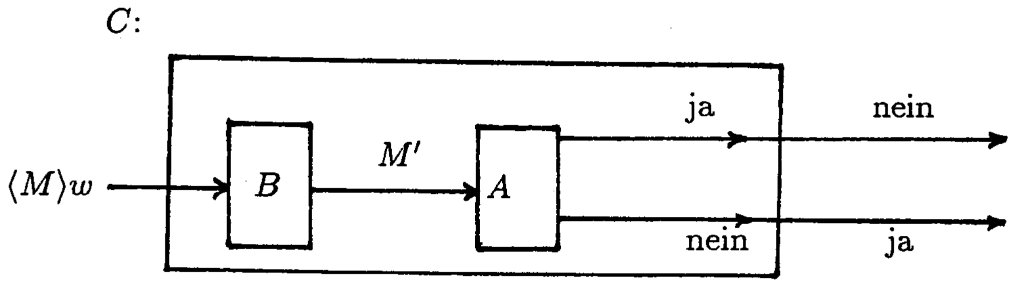
Beweis: Zunächst zeigen wir, daß L_{ne} r.a. ist, indem wir eine Turingmaschine M konstruieren, welche Kodes von Turingmaschinen erkennt, welche nichtleere Sprachen akzeptieren. M nimmt also als Eingabe einen Kode $\langle M_i\rangle$, und erzeugt systematisch Paare $(j, k) \in \mathbb{N}^2$. Für so ein Paar (j, k) simuliert M nun k Berechnungsschritte der Turingmaschine M_i auf dem j–ten Wort w_j (wie im Beweis zu Satz 1.4.1). Wird dabei w_j akzeptiert, so akzeptiert M den Kode $\langle M_i\rangle$.

Nun zeigen wir, daß L_e nicht rekursiv ist. Das folgt aus dem Satz von Rice. Wir wollen aber hier einen direkten Beweis geben, da wir die dabei notwendigen Konstruktionsschritte auch später noch brauchen werden. Es ist nicht schwer zu sehen, daß es einen Algorithmus (immer terminierende Turingmaschine) B gibt, welcher eine Eingabe der Form $\langle M\rangle w$ nimmt und eine Turingmaschine M' konstruiert, sodaß

$$L(M') = \begin{cases} \emptyset, & \text{falls } w \notin L(M), \\ (0+1)^*, & \text{falls } w \in L(M). \end{cases}$$



(Details siehe [HU], p. 185 ff). Angenommen L_e wäre rekursiv, und A wäre ein Algorithmus mit $L(A) = L_e$. Dann könnten wir einen Algorithmus C mit $L(C) = L_u$ (Akzeptierungsproblem, Def. 4.1.4) konstruieren wie in folgender Skizze:



So ein Algorithmus C kann aber nach Satz 4.1.3 nicht existieren. Also kann auch A nicht existieren. Somit ist L_e nicht rekursiv.

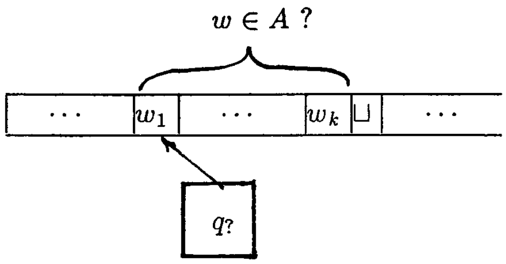
Wäre nun L_{ne} rekursiv, dann auch sein Komplement L_e. Somit ist L_{ne} nicht rekursiv.

Wäre L_e r.a., dann wären L_e und L_{ne} rekursiv (Bemerkung nach Def. 1.4.4). Also ist L_e nicht r.a. ■

Man könnte sich aber nun die Frage stellen, was wäre, wenn man das Emptiness–Problem (oder ein anderes unentscheidbares Problem) nun doch irgendwie entscheiden könnte. Wäre dann alles entscheidbar? Könnten wir dann alles berechnen?

Dabei muß man natürlich vorsichtig vorgehen. Wenn wir einfach die Entscheidbarkeit des Emptiness–Problems als zusätzliches Axiom annehmen würden, so wäre unsere Theorie sofort widersprüchlich, und wir könnten natürlich alles beweisen. Das vermeiden wir durch die Einführung von Orakel–Turingmaschinen.

Definition 4.6.1: Sei A eine Sprache über Σ, $A \subseteq \Sigma^*$. Eine *Orakel–Turingmaschine (OTM) mit Orakel* A ist eine (Ein–Band–) Turingmaschine M^A mit drei ausgezeichneten Zuständen, $q_?$, q_j und q_n. Wenn M^A den Zustand $q_?$ annimmt, so ersucht sie um eine Antwort auf die Frage: "ist das Wort rechts vom (und beginnend beim) LS–Kopf bis zum ersten Blank in A?" Je nachdem, ob die Antwort auf diese Frage "ja" oder "nein" ist, nimmt M im nächsten Schritt den Zustand q_j oder q_n an. ■



Eine so definierte Turingmaschine hat also ein Orakel zur Verfügung, an das sie Fragen betreffend die Sprache A stellen kann. Ist A eine rekursive Sprache, so kann dieses Orakel durch eine andere Turingmaschine simuliert werden, und die von der Orakel–Turingmaschine M^A akzeptierte Sprache ist daher r.a. Ist dagegen A nicht rekursiv, so akzeptiert M^A möglicherweise eine Sprache, welche nicht r.a. ist.

Definition 4.6.2: Eine Sprache B ist *rekursiv aufzählbar bezüglich* A, wenn es eine Orakel–Turingmaschine M^A gibt mit $B = L(M^A)$. Eine Sprache B ist *rekursiv bezüglich* A, wenn es eine Orakel–Turingmaschine M^A gibt, welche immer hält, mit $B = L(M^A)$. Zwei Sprachen A und B sind *äquivalent*, wenn jede rekursiv ist bezüglich der anderen. ■

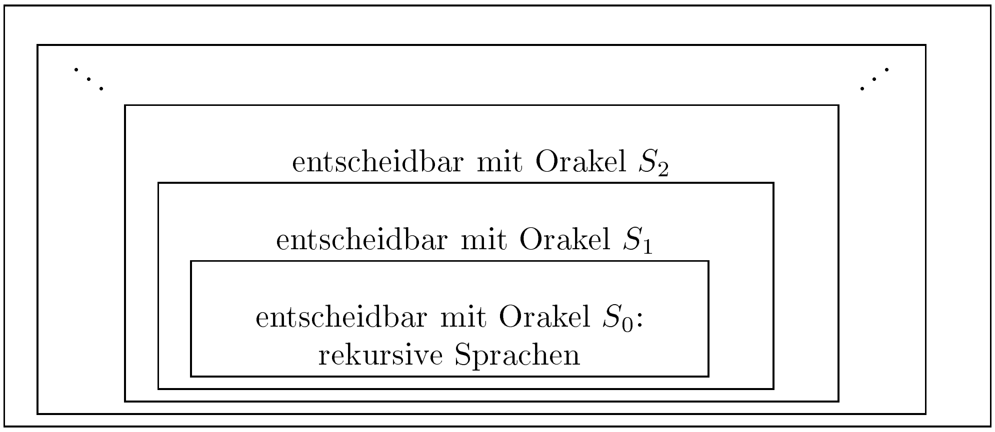
Natürlich können nicht alle Sprachen r.a. bzgl. L_e sein, da es überabzählbar viele Sprachen gibt, aber nur abzählbar viele Turingmaschinen. Es gibt also eine Sprache, welche nicht r.a. bzgl. L_e ist. Diese könnten wir wiederum als Orakel verwenden, und so fort. Auf diese Weise erhalten wir eine ganze Hierarchie unentscheidbarer Sprachen bzw. Probleme.

Definition 4.6.3: Wir definieren eine Folge kanonischer Orakelmengen:

$$\begin{aligned} S_0 &= \emptyset, \\ S_1 &= \{\langle M\rangle | L(M^{S_0}) = \emptyset\}, \\ &\vdots \\ S_{i+1} &= \{\langle M\rangle | L(M^{S_i}) = \emptyset\}, \\ &\vdots \end{aligned}$$

S_{i+1} ist also ein Orakel, um das Emptiness–Problem für Berechnungen bzgl. S_i zu lösen. ■

Eine TM mit Orakel S_0 ist also offenbar eine TM, welche kein Orakel benutzt. S_1 ist nichts anderes als die Menge L_e aus Satz 4.6.1. Wir erhalten folgende Einteilung der Klassen von Sprachen über $\{0, 1\}$:



Nun kann man einige (aber nicht alle) unentscheidbare Probleme nach ihrer Äquivalenz zu Elementen der Folge $(S_i)_{i\in\mathbb{N}}$ klassifizieren.

Satz 4.6.2: *Das Akzeptierungsproblem für Turingmaschinen ohne Orakel ist äquivalent zu* S_1.

Beweis: L_u rekursiv bzgl. S_1: Wir konstruieren eine OTM M^{S_1}, welche zunächst bei Eingabe $\langle M\rangle w$ den Kode der TM M' wie im Beweis zu Satz 4.6.1 konstruiert, also

$$L(M') = \begin{cases} \emptyset, & \text{falls } w \notin L(M), \\ (0+1)^*, & \text{falls } w \in L(M). \end{cases}$$

Hierauf nimmt M^{S_1} den Zustand $q_?$ an, mit dem Kode von M' rechts vom LS–Kopf und akzeptiert genau dann, wenn im nächsten Schritt der Zustand q_n angenommen wird. Das Akzeptierungsproblem für TM ohne Orakel ist also rekursiv bzgl. S_1.

S_1 rekursiv bzgl. L_u: Umgekehrt zeigen wir, daß es eine OTM $M_2^{L_u}$ mit Orakel L_u gibt, welche S_1 erkennt. Wir konstruieren eine TM M_2, welche bei Eingabe $\langle M\rangle$ eine neue TM M' erzeugt. M' operiert folgendermaßen: zunächst ignoriert M' seine Eingabe, und erzeugt systematisch alle Paare $(i,j)\in\mathbb{N}^2$. Für jedes (i,j) simuliert M' i Berechnungsschritte von M auf dem j–ten Wort. Akzeptiert M, so akzeptiert M' seine Eingabe.

$$L(M') = \begin{cases} \emptyset, & \text{falls } L(M) = \emptyset, \\ (0+1)^*, & \text{falls } L(M) \neq \emptyset. \end{cases}$$

$M_2^{L_u}$ befragt nun sein Orakel, ob M' das leere Wort ϵ akzeptiert, also ob $\langle M'\rangle\epsilon \in L_u$. Ist das der Fall, so weist $M_2^{L_u}$ die Eingabe M zurück, andernfalls akzeptiert es M. S_1 ist also rekursiv bzgl. L_u. ∎

Nun wollen wir das Problem "$L(M) = \Sigma^*$?" betrachten. Dabei sei $\Sigma = \{0,1\}$ das Eingabealphabet der Turingmaschine M. Wir werden sehen, daß dieses Problem äquivalent ist zu S_2, also in gewisser Weise schwieriger ist als das Emptiness–Problem oder das Akzeptierungsproblem. In einem praktischen Sinn hat das natürlich keinerlei Bedeutung, da alle diese Probleme ohnedies unentscheidbar sind. Wenn wir aber diese Probleme einschränken, etwa auf reguläre Ausdrücke oder kontextfreie Grammatiken, so können wir tatsächlich feststellen, daß das "$= \Sigma^*$"–Problem schwieriger ist als das Akzeptierungsproblem.

Satz 4.6.3: *Das Problem "$L(M) = \Sigma^*?$" ist äquivalent zu S_2.*

Beweis: Zuerst zeigen wir, daß das "$= \Sigma^*$"–Problem rekursiv bzgl. S_2 ist. Wir konstruieren eine OTM $M_3^{S_2}$, welche genau jene Turingmaschinen M erkennt, die alle Eingabewörter akzeptieren. Zunächst erzeugt $M_3^{S_2}$ eine OTM $\hat{M}^{S_1}$ mit Orakel S_1, welche folgendermaßen operiert: $\hat{M}^{S_1}$ zählt Wörter $x \in \Sigma^*$ auf, und für jedes x wird das Orakel S_1 befragt, ob M das Wort x akzeptiert (siehe Satz 4.6.2). $\hat{M}^{S_1}$ akzeptiert seine Eingabe, wenn es ein x gibt, welches von M nicht akzeptiert wird, also

$$L(\hat{M}^{S_1}) = \begin{cases} \emptyset, & \text{falls } L(M) = \Sigma^*, \\ \Sigma^*, & \text{sonst.} \end{cases}$$

Nach der Konstruktion von $\hat{M}^{S_1}$ befragt nun $M_3^{S_2}$ sein Orakel S_2, ob $L(\hat{M}^{S_1}) = \emptyset$. Ist das der Fall, so akzeptiert $M_3^{S_2}$ seine Eingabe M. Andernfalls wird M zurückgewiesen. Also

$$L(M_3^{S_2}) = \{\langle M \rangle \mid L(M) = \Sigma^*\}.$$

Um zu zeigen, daß S_2 rekursiv ist bzgl. "$= \Sigma^*$", konstruiert man eine OTM $M_4^{L_*}$, welche S_2 akzeptiert, wobei

$$L_* = \{\langle M \rangle \mid L(M) = \Sigma^*\}.$$

Wir führen diesen Teil des Beweises aber nicht mehr aus, sondern verweisen auf [HU], Theorem 8.20. ■