Berechnungsmodelle

Mathias Hecht

April 29, 2010

1 Die Turingmaschine

1.1 Definition

Eine Turingmaschine wird durch ein Tupel (Γ, Q, δ) beschrieben.

- Γ ein endliches Alphabet
- Q : eine endliche Menge an Zuständen
- δ eine Zustandsüberführungsfunktion :
  $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$

1.2 Effizienz und Laufzeit

Definition 1.3

Seien $f : \{0,1\}^* \to \{0,1\}^*$ und $T : N \to N$ zwei Funktionen und M eine Turingmaschine gegeben. Wir sagen M berechnet f für jedes x aus $\{0,1\}^*$, ausgehend von der Startkonfiguration und mit dem Ergebnis $f(x)$ auf dem Ausgabeband sobald M hält. Wir sagen M berechnet $f(x)$ in $T(n)$-Zeit wenn für jeden beliebigen Eingabestring x, M nach maximal $T(|x|)$ Schritten hält.
Die Funktion $T : N \to N$ beschreibt die Laufzeit einer gegebenen Turingmaschine M wenn $T(|x|)$ mindestens n ist und M die Funktion $f(x)$ in $T(|x|)$ Schritten berechnet.

Claim 1.5

Für jede Funktion $f : \{0,1\}^* \to \{0,1\}^*$ die sich von einer Turingmaschine M mit dem Alphabet Γ in $T(n)$-Zeit berechnen lässt existiert eine Turingmaschine M' mit dem Alphabet (0,1,blank,start) das die selbe Funktion in $4log|\Gamma|T(n)$ berechnet.

Beweisskizze

Die Turingmaschine M' benötigt für jedes Symbol aus Γ insgesamt $log|\Gamma|$ Bits für die Kodierung. Die Turingmaschine M' verwendet die Überführungsfunktion von M und benötigt so zum Lesen und Schreiben der Symbole auf/von den Bänder jeweils $log|\Gamma|$ Schritte.

Claim 1.6

Sei M' eine 1-Band Turingmaschine, dann gilt für jede Funktion $f : \{0,1\}^* \to \{0,1\}$ die eine k-Band Turingmaschine M in $T(n)$-Zeit berechnet das sie von M' in $5kT(n)^2$-Zeit berechnet werden kann.

Beweisskizze

Die Turingmaschine M' kodiert die k-Bänder indem sie ihr Band in Partitionen der Größe k unterteilt. Zusätzlich wird das Alphabet Γ erweitert in der Form das für jedes Element x aus Γ nun auch noch das Element $\hat{x}$ hinzugefügt wird das die Position des Lesekopfes des jeweilgen Bandes markiert. Im ersten Schritt verteilt M' also die Eingabe der Größe n auf die Partitionen. Für jeden Schritt den nun M macht muss M' jeweils für das Lesen/Schreiben der Kopfposition einmal über das komplette Band fahren. Zusammen mit dem zusätzlichen Lese/Schreib Aufwand für die Kodierung lässt sich die Abschätzung von $5kT(n)^2$-Zeit treffen.

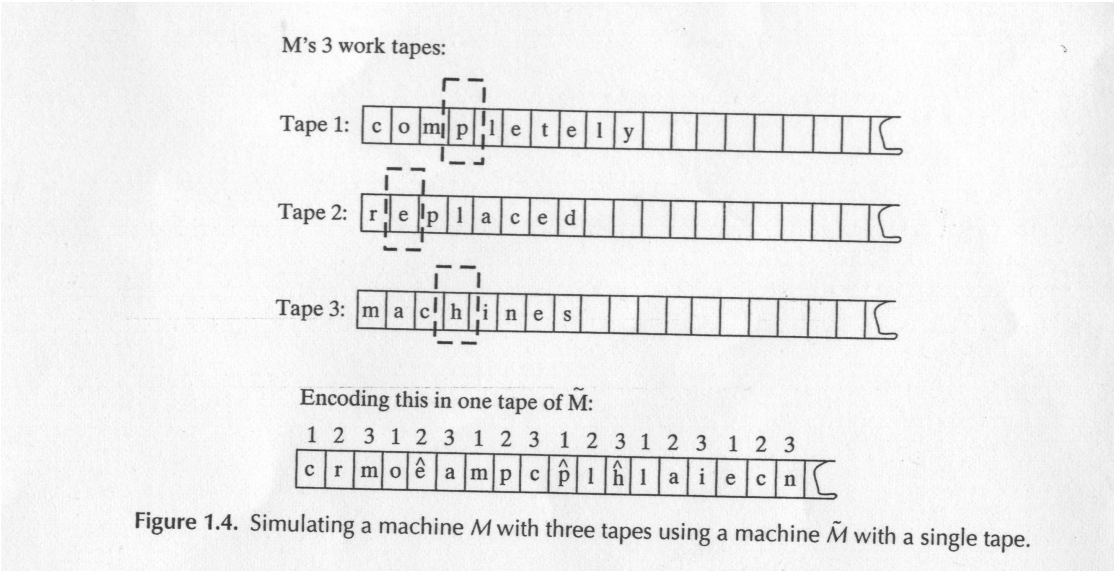


Figure 1.4. Simulating a machine M with three tapes using a machine $\tilde{M}$ with a single tape.

Claim 1.8

Sei M eine TM deren Bänder unendlich in beide Richtungen sind. Für jede Funktion $f : \{0,1\}^* \to \{0,1\}^*$ die M in $T(n)$-Zeit berechnet. Lässt sich eine TM M' mit einem Standardband konstruieren die f in $4T(n)$ berechnet.

Beweisskizze

Um das bidirektionale Endlosband zu simulieren wird ein neues Alphabet konstruiert. Das nun für alle Symbole x,y aus Γ das Symbol xy enthält. Dieser Fall lässt sich nun als Spezialfall von Claim 1.5 betrachten, nach der Vereinfachung bleibt dann nur noch ein konstanter Faktor.

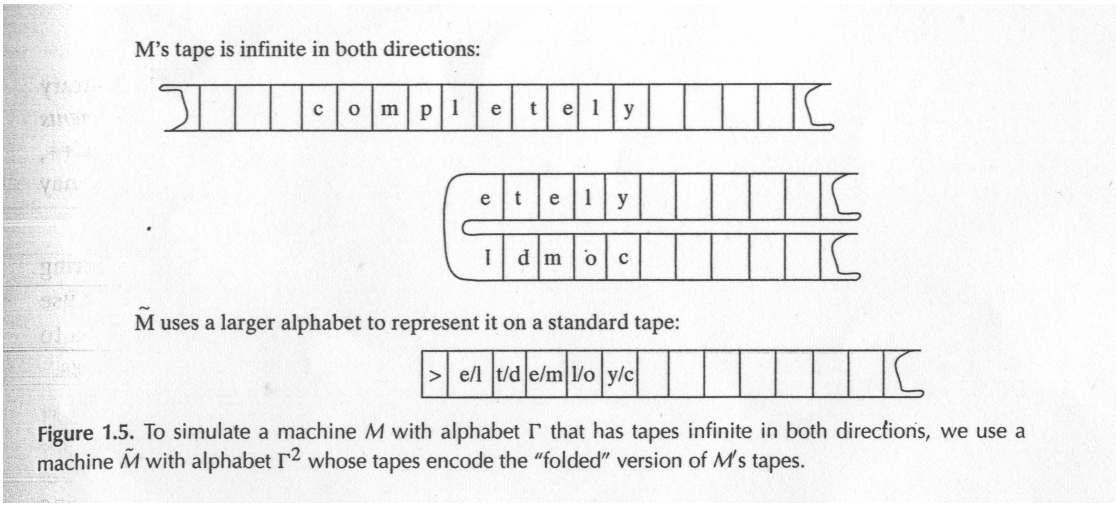
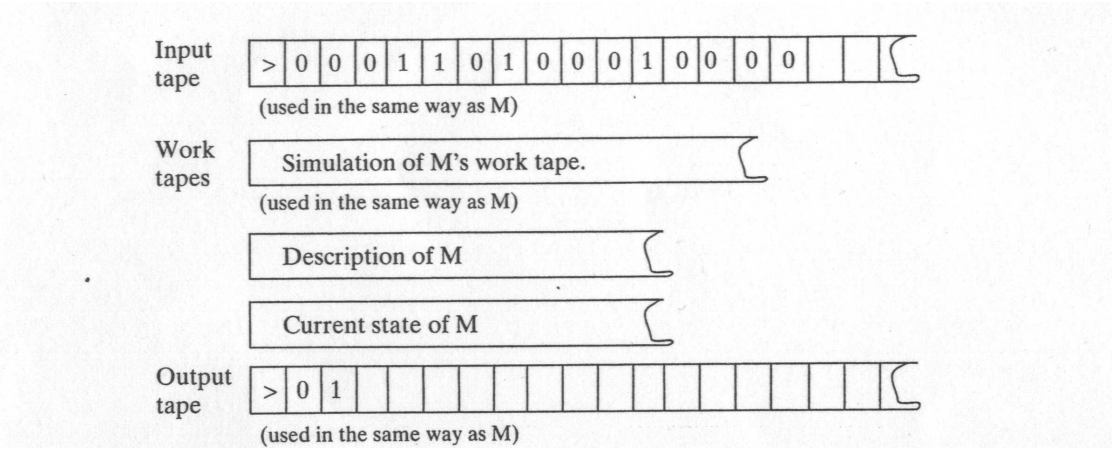


Figure 1.5. To simulate a machine M with alphabet Γ that has tapes infinite in both directions, we use a machine $\tilde{M}$ with alphabet Γ^2 whose tapes encode the "folded" version of M's tapes.

2 Die Universelle Turingmaschine



2.1 Machinen als Strings

- Jeder String aus $\{0,1\}^*$ repräsentiert eine Turingmaschine. Alle Strings die die Kodierung verletzen werden als die Standard Maschine übersetzt die hält und 0 ausgibt.
- Jede TM wird von unendlichen vielen Strings dargestellt wir erlauben das eine beliebige Anzahl von 1 die ignoriert werden am Ende einer Kodierten TM sind.

2.2 Die Unverselle Turingmaschine

Theorem 1.9

Es existiert eine Turingmaschine U so das für jedes $x, a \epsilon \{0,1\}^*$ gilt. $U(x,a) = M_a(x)$ wobei M_a die Turingmaschine repräsentiert die durch a codiert ist. Wenn $M_a(x)$ in T schritten hält benötigt $U(x,a)$ $CT \log T$ Schritte. Wobei C eine konstante ist in Abhängigkeit von $M_a s$ Alphabet , Anzahl der Zustände und Anzahl der Bänder.

Beweisskizze

Um ebenfalls auf den $TlogT$ Faktor zu kommen für die Simulation müssen wir für die k-Bänder ein bessere Modellierung finden. Im ersten Schritt legen wir fest das wir nur mit einem Lesekopf für alle k-Bänder arbeiten so das der Lesekopf mit einmal alle k-Bänder einliest. Dies lässt sich wie in Claim 1.5 mit geringen Mehraufwand bewerkstelligen. Die unabhängige Bewegung der einzelnen Leseköpfe wird nun dadurch simuliert das das betreffende Band in die entgegen gesetzte Richtung geschoben wird. Dies bringt offensichtlich wieder quadratischen Mehraufwand in die Laufzeitabschätzung diesesmal nicht beim Einlesen der Symbole sondern nur bei der Kopfbewegung. Um nun für eine Kopfbewegung nicht das gesamte Band zu bewegen wird folgende Kodierung/Partitionierung des Bandes vorgenohmen.

- Die Position des Lese/Schreibkopfes wird mit Position 0 bezeichnet unter dem Kopf dürfen sich ausschließlich Symbole und Leerzeichen des Ursprünglichen Alphabets befinden.
- Das Band wird nun ausgehend von Position 0 in beide Richtung in Partitionen L_0 $..L_{log(n)}$ und R_0 .. $R_{log(n)}$ unterteilt. Wobei Partition L_i genau $2 * 2^i$ Symbole groß ist.

Zusätzlich legen wir folgende Eigenschaften fest die ggf. nach jeder Kopfbewegung wieder hergestellt werden müssen.

- Die Summe aller Symbole und Leerzeichen des Ursprünglichen Alphabet von Partition R_i und L_i darf maximal $2 * 2^i$ betragen
- Eine Partition L_i oder R_i ist entweder genau halb gefüllt ,leer oder voll. Sollte eine Partition nicht voll sein wird sie entsprechend mit einem neuen zusätzlichen Leerzeichen aufgefüllt was nicht bestandteil des Ursprünglichen Alphabets ist.

Betrachten wir nun die einzelnen Schritte die nötig sind um eine Kopfbewegung nach rechts eines der k-Bänder zu realisieren.

- Suche einen Index i so das R_i nicht leer ist. Dies ist am Rande bemerkt auch der erste Index an dem L_i nicht voll ist.

- Wenn R_i nur halb voll ist werden alle Symbole aus R_i in die Partitionen R_0 bis R_{i-1} verschoben. Sollte R_i voll sein wird nur die hälfte der Zeichen in die Partitionen R_0 bis R_{i-1} verschoben. Das erste Symbol aus dem Ursprünglichen Alphabet wird in Position 0 kopiert alle Zusätzlich Leerzeichen die nicht zum Ursprünglichen Alphabet gehöhren werden entfernt. Beim kopieren der Zeichen aus Partition R_i wird darauf geachtet das die Partitionen R_0 bis R_{i-1} nur halb gefüllt werden.

- Nun wird auf der linken Seite um die oben genannten Eigenschaften wieder herzustellen entsprechende Kopieroperationen durchgeführt bzw mit neuen Leerzeichen aufstockt. Betroffen von diesen Operationen sind auch hier nur Partitionen L_0 bis L_i

Bei genauer Betrachtung fällt auf das bei diesen Operationen maximal $2 * 2^i$ Zeichen des ursprünglichen Alphabets und nochmal die selbe Anzahl der neuen Leerzeichen betroffen sind. So das sich asymptotisch der Aufwand mit $O(2^i)$ abschätzen lässt. Wenn wir das Band nach dieser Kopfbewegung betrachten fällt auf das bis zur nächsten Kopfbewegung die eine Partition mit Index i betrifft mindestens $(2^{i-1}) - 1$ Kopfbewegung in eine Richtung nötig sind. So das die Häufigkeit einer Kopfbewegung die eine Partition mit Index i betrifft sich durch $\frac{1}{2^{i-1}}$ berechnen lässt. Das ganze läst sich nun in folgende Formel zusammenfassen : $\Sigma_{i=0}^{logT}(k \times \frac{T*2^i}{2^{i-1}})$ Nach streichen der konstanten Faktoren bleibt so amortisiert ein asymptotischer Mehraufwand von $TlogT$ übrig wobei T hier die länge der Eingabe $|n|$ steht.

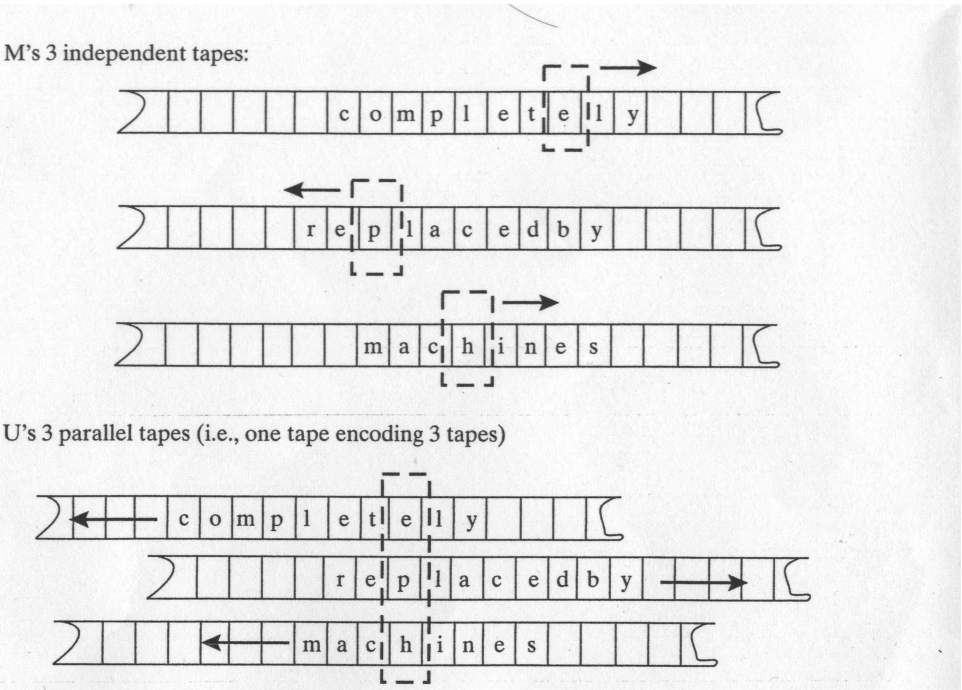


Figure 1.8. Packing k tapes of M into one tape of $\mathcal{U}$. We consider $\mathcal{U}$'s single work tape to be composed of k parallel tapes, whose heads move in unison, and hence we shift the contents of these tapes to simulate independent head movement.

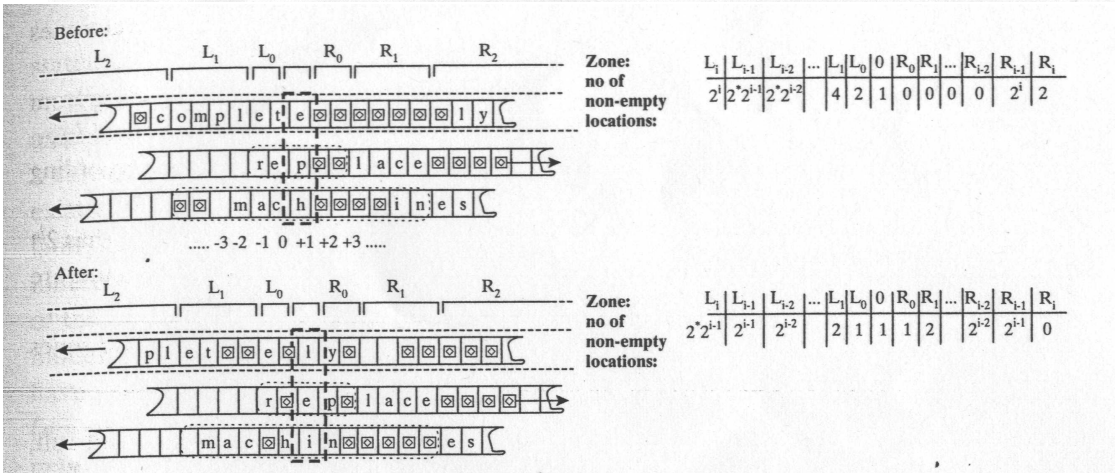


Figure 1.9. Performing a shift of the parallel tapes. The left shift of the first tape involves zones $R_0, L_0, R_1, L_1, R_2, L_2$, the right shift of the second tape involves only R_0, L_0, while the left shift of the third tape involves zones R_0, L_0, R_1, L_1. We maintain the invariant that each zone is either empty, half-full, or full and that the total number of nonempty cells in $R_i \cup L_i$ is $2 \cdot 2^i$. If before the left shift zones $L_0, \ldots, L_{i-1}$ were full and L_i was half-full (and so $R_0, \ldots, R_{i-1}$ were full and R_i half-full), then after the shift zones $R_0, L_0, \ldots, R_{i-1}, L_{i-1}$ will be half-full, L_i will be full and R_i will be empty.

3 Die unberechenbaren Funktionen

3.1 Die Funktion UC

Theorem 1.10

Es existiert eine Funktion $UC : \{0,1\}* \rightarrow \{0,1\}$ die durch keine TM berechnet werden kann.

UC:

Für jedes a aus $\{0,1\}^*$ ist $UC(a) = 0$ wenn $M_a(a) = 1$. Sollte $M_a(a)$ etwas anderes ausgeben oder einen Endlosschleife eintreten so ist $UC(a) = 1$.

Beweisskizze

Die Beweisidee ist davon auszugehen das die Funktion UC berechenbar ist und dann mit Hilfe der Diagonalisierung zum wiederspruch zu führen. Wenn UC berechenbar ist existiert eine TM_{UC} die UC berechnet. Wenn man diese TM_{UC} nun als Eingabe für sich selbst verwendet, wird klar das dies gegen die Defenition der Funktion UC verstösst.

3.2 Das Halte Problem

Für jedes Paar x,a aus $\{0,1\}^*$ berechnet $HALT(x,a) = 1$ wenn $M_a(x)$ nach einer Endlichen Anzahl von Schritten zum Stillstand kommt. Andernfalls berechnet $HALT(x,a) = 0$.

Beweisskizze

Dies lässt sich durch Reduktion auf die Funktion UC beweisen. Angenommen eine TM Halt(x,a) existiert so lässt eine sich TM konstruieren die mit Hilfe von Halt und der UTM erlaubt UC berechnet.

3.3 Gödels Theorem(Unvollständigkeitssatz)

Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig.

1. Gödelscher Unvollständigkeitssatz

In jedem formalen System der Zahlen, das zumindest eine Theorie der natürlichen Zahlen enthält, gibt es einen unentscheidbaren Satz, also einen Satz, der nicht beweisbar und dessen Wiederlegung ebenso wenig beweisbar ist.

2. Gödelscher Unvollständigkeitssatz

Kein formales System der Zahlen, das zumindest eine Theorie der natürlichen Zahlen enthält, lässt sich innerhalb seiner selbst als widerspruchsfrei beweisen.

Beweisskizze

Die Beweisidee ist ähnlich der Beweisidee für die TM die UC berechnet. Gödel hatte gezeigt das sich jede mathematische Aussage innerhalb eines Axiomssystem eindeutig durch eine "Gödelnummer" kodieren lässt. Gödel konstruiert nun eine Aussage der Form: "Der Ausdruck X ist nicht beweisbar". Wobei X eine Variable ist. Die Beweißidee ist nun die Gödelnummer des Ausdrucks dem Ausdruck selbst als Variable zu übergeben. So das die Aussage nun lautet: "Ich bin nicht beweisbar". Nun gibt es nur 2 Möglichkeiten. Entweder das Axiomssystem kann diese Aussage beweißen dann ist es wiedersprüchlich oder das Axiomssystem kann diese Aussage nicht beweisen und muss *glauben* das er war ist.

4 Laufzeitkomplexitätsklassen

4.1 Definitionen

Entscheidungsprobleme

Wir sagen eine Maschine entscheidet eine Sprache L aus $\{0,1\}^*$ wenn es die Funktion $f_L : \{0,1\}^* \to \{0,1\}$ berechnet sodass , $f_L(x) = 1$ genau dann wenn x Element von L ist.

Definition 1.12 (Die Klasse DTime)

Sei $T : N \to N$ eine Funktion. Eine Sprache L ist in $Dtime(T(n))$ wenn eine Turingmaschine exisitiert die in $c \times T(n)$ die Sprache L entscheidet, für ein $c > 0$.

Definition 1.13(Die Klasse P)

Alle Sprachen L aus $\{0,1\}^*$ die sich für eine Konstante $c > 0$ in $Dtime(n^c)$ von einer deterministischen Turingmaschine entscheiden lassen.

Bedeutung von P

P wird auch die Klasse der Effizient berechenbaren Probleme bezeichnet. Auch wenn man argumentieren könnte das eine Laufzeit von $O(n^{100})$ in der Praxis keinen praktikablen Algorithmus beschreibt, so zeigt doch die selbige das wenn einmal ein Algorithmus mit polynomieller Laufzeit gefunden und einer relativ großen konstanten im Exponenten, so folgen darauf schnell weitere Verbesserungen. Die Laufzeitklasse P ermöglicht es Algorithmen in unterschiedlichen Rechnungsmodellen zu vergleichen.

4.2 Kritik an P

- Worse-case Betrachtung ist zu strikt.
- Genauigkeit, Verwendung von Realen Zahlen

- Zufall, Berechnungsmodelle die zufällige Werte berücksichtigen
- Exotische Modelle, Quantum mechanics, String Theorie

4.3 Berechnungsmodelle

Church-Turing These:

Die Klasse der Turing-berechenbaren Funktionen ist genau die Klasse der intuitiv berechenbaren Funktionen.

In anderen Worten

Dies ist keine These die bewiesen werden kann aber in der Informatik wird allg. angenohmen das diese war ist. Der Hintergrund für diese These geht zurück auf Alonzo Church und Alan Turing. Turing hatte als erstes gezeigt das das Lambda Kalkül und die Turingmaschine äquivalent sind. Nach der Turingmaschine wurde noch viele weitere Rechnungsmodelle aufgestellt. Jedoch konnte gezeigt werden das all diese in ihrer Berechnungsfähigkeit nicht mehr leisten konnten als die Turingmaschine.

Fazit

Obwohl die Turingmaschine dem Gedankenprozess eines Menschen beim Rechnen nach empfunden ist zeigte Turing das es Funktionen gibt die ein Mensch sich ausdenken kann aber keine Turingmaschine berechnen kann. Turing wurde bei seiner Arbeit unter anderem durch Gödel inspiriert/motiviert, der durch die Gödelnummer und dem Unvollständigkeitssatz eine ähnliche Aussage trifft. Nähmlich das Mathematik genauer formale Zahlensystem nicht vollständig von Maschinen mittels der Axiome des Zahlensystems beweisbar sind und das Mathematik von daher wohl auch in Zukunft von Menschen und nicht von Maschinen betrieben wird.