Turingmaschinen und Registermaschinen

Wie man eine Registermaschine in eine Turingmaschine übersetzt und welche Erkenntnis man daraus gewinnt

Theorieseminar bei Prof. Dr. Heinrich Braun, DHBW-Karlsruhe

Florian Loch

17.03.2015

Das abstrakte Modell der Turingmaschine ist ein essenzieller Teil des theorischen Fundaments der Informatik. Weit besser für die Abbildung realer Prozessoren eignet sich jedoch das Modell der Registermaschine. Durch den Beweis der Überführbarkeit von Registermaschinen hin zu Turingmaschinen und die Skizzierung des Beweises für den umgekehrten Weg soll der „Kreis der Mächtigkeit“ aufgezeigt und geschlossen sowie die Gleichmächtigkeit der beiden wichtigen Modellen festgestellt werden.

Inhaltsverzeichnis

1 Einleitung

Diese schriftliche Ausarbeitung ist, zusammen mit der am 17.03.2015 gehaltenen Präsentation, Teil der Prüfungsleistung des Theorieseminars an der DHBW Karlsruhe unter der Leitung von Prof. Dr. Heinrich Braun und Dr. Martin Holzer.

Ziel dieser Arbeit soll sein zu zeigen, dass das Konzept der Registermaschine in das Konzept der Turingmaschine, dem abstrakten, schon lange vorhandenen und grundlegenden Maschinenmodell der Informatik, überführt werden kann. Der Weg der Transformation soll dabei gründlich aufgezeigt werden und Analysen bzgl. Laufzeit- und Speicherkomplexität durchgeführt werden. Durch eine Beweisskizze für die Übersetzung einer Turingmaschine hinzu einer Registermaschine soll der Kreis geschlossen und die „Gleichmächtigkeit“ vieler Konzepte und Programmiersprachen gezeigt werden.

Diese Arbeit orientiert sich an Ingo Wegeners Buch „Theoretische Informatik : eine algorithmenorientierte Einführung“ [1].

2 Motivation

Die Idee hinter der Überführung von Registermaschine zu Turingmaschine ist der Beweis, dass eine Turingmaschine (TM) *mindestens* so mächtig ist wie eine Registermaschine (RAM). Der Bedarf dieses Beweises begründet sich u. A. dadurch, dass die TM zwar als Teil des theoretischen Fundaments der Informatik angesehen wird, in der Praxis Algorithmen jedoch für RAMs entwickelt werden. Durch den Beweis dieser „Hierarchie“ würden viele Feststellungen bzgl. der TM auch für die RAM gelten, da eine jede RAM als eine TM betrachtet werden könnte.
Durch einen solchen Beweis ergeben sich jedoch noch einige weitere Schlüsse: Ein Algorithmus und dessen Laufzeitkomplexität sind prinzipiell unabhängig von der verwendeten Rechnerarchitektur (ARM, i386, x86, amd64, MIPS, etc.) bzw. dem Maschinenkonzept (RAM, TM)[1] sowie dass ein Algorithmus in jeder Turing-vollständigen Sprache implementiert werden kann. Durch den umgekehrten Beweis, die Überfühhrung der TM auf die RAM, lässt sich der „Kreis der Mächtigkeit“ (siehe Abschnitt **??**) schließen und die „Gleichmächtigkeit“ der beiden theoretischen Konzepte (sowie der darauf aufsetzenden Konzepte und Sprachen) feststellen.

[1] i. Allg. hier mit einer polynomialen Zeitkomplexitätsverschlechterung, bspw. wird aus $T(n) \Rightarrow T^3(n)$)[1, S. 7]

3 Wiederholung Turingmaschine

Die Turingmaschine wurde grundlegend 1936 durch Alan Turing eingeführt und ist heute wichtiger Teil der theoretischen Informatik. Viele Problemstellungen, wie bspw. das Halteproblem oder die Entscheidbarkeit (in polynomialer Zeit), werden auf ihrer Basis untersucht und definiert.
Von der deterministischen „Basis-Turingmaschine“ (TM bzw. DTM) ausgehend gibt es viele Varianten wie die Mehrband-, Mehrspur- und Keller-Turingmaschinen. Sie basieren i. R. allerdings auf derselben Definition (siehe Abbildung 3), erweitern sie lediglich. Neben den klassischen deterministischen Turingmaschinen gibt es noch nichtdeterministische und probabilistische Turingmaschinen; diese sind für diese Arbeit jedoch ohne Bedeutung.

Definition einer DTM M

Q	Menge aller Zustände, die M annehmen kann
Γ	Bandalphabet, Menge aller Zeichen, die von M verarbeitet werden kann bzw. aller Zeichen die jemals in einer Zelle abgelegt werden können
Σ	Eingabealphabet, Menge aller Zeichen, die eingegeben werden können, $\Sigma \subseteq \Gamma$
B	Blank-Zeichen, $B \in \Gamma \setminus \Sigma$
q_0	Startzustand von M, $q_0 \in Q$
δ	Zustandsüberführungsfunktion, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$
T	Implizite Menge aller Endzustände. Umfasst q, die bei beliebigem a keine Zustandsänderung oder Bewegung des Kopfes bewirken: $\delta(q, a) = (q, a, N)$
F	Menge der Finalzustände, $F \subseteq T$, nur bei binär antwortendem (Eingabe akzeptiert, Eingabe abgelehnt) M

Abbildung 1: Definition einer DTM

Die TM verfügt in ihrer ursprünglichen Form über ein unendliches, sogenanntes Turingband. Die einzelnen Zellen können mit $a \in \Gamma, |a| = 1$ beschrieben werden. Üblicherweise ist jede Zelle des Bandes zu Anfang mit dem sogenannten „Blank“-Zeichen (B) gefüllt, mit Ausnahme der durch die Eingabe ($w \in \Sigma*$) belegten Zellen. Mithilfe eines Lese-/Schreibkopfes, welcher schrittweise auf dem Band entlang bewegt werden kann, liest die TM zu Beginn eines Zyklus' a ein. Zum Zeitpunkt t_0 steht dieser am Anfang der Eingabe.
Entsprechend dem Zustand q in dem sich die TM gerade befindet ergibt sich durch die Überführungsfunktion δ (das „Programm“) der Folgezustand q', die Ausgabe $a' \in \Gamma$, welche in die aktuelle Zelle geschrieben wird, sowie die Angabe, ob und in welche Richtung $d \in \{R, L, N\}$ der Kopf um 1 Zelle bewegt werden soll.

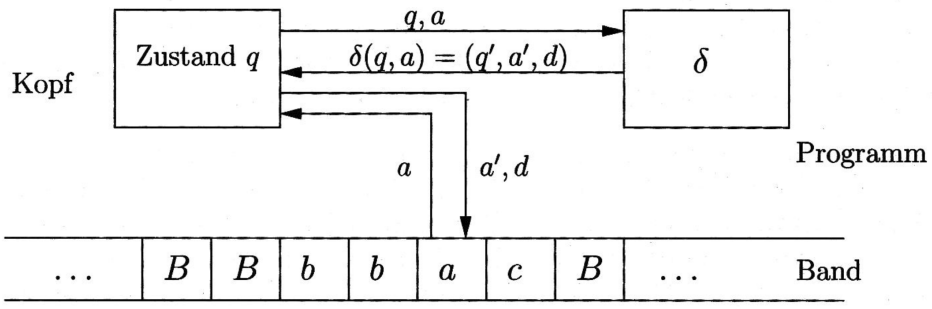

Abbildung 2: Üblicher Aufbau einer DTM [1, S. 10]

4 Vorstellung Registermaschine

Registermaschinen (im englischen *Random Access Maschine*, daher RAM) sind ein in der theoretischen Informatik verwendetes Modell, welches sich – im Gegensatz zur Turingmaschine – an realen, mit Assembler programmierten CPUs orientiert. In der Praxis vorhandene Probleme wie Over- und Underflows, bspw. durch arithmetische Operationen, finden keine Berücksichtigung. Ebenso verfügt die Registermaschine über einen, in der Praxis natürlich nicht vorhandenen, unendlichen Speicher (in Form der Registermenge R) mit wahlfreiem Zugriff in konstanter Zeit ($O(1)$). Dieser Zugriff kann entweder *direkt* oder *indirekt* erfolgen.

Definition einer Registermaschine

R	Unbegrenzte Menge an Registern mit wahlfreiem Zugriff, können beliebige $x \in \mathbb{N}$ enthalten
R_0	Ist per Definition der Akkumulator
P	Programm mit endlicher Länge, $p = \|P\|$
b	Befehlszähler, enthält die aktuelle Programmzeile

Abbildung 3: Definition einer Registermaschine

Im Gegensatz zur TM ist die RAM, bzw. der Inhalt der Register, nicht durch ein (endliches) Alphabet oder eine Längenbegrenzung beschränkt. Ein Register kann ein beliebiges $x \in \mathbb{N}$ als Wert annehmen. Zur Speicherung von Elementen aus $\mathbb{Z}$ bedarf man daher zweier Register (codiertes Vorzeichen, Betrag); für Elemente aus $\mathbb{Q}$ drei (codiertes Vorzeichen, Betrag, Nachkommaanteil).
Das Verhalten einer RAM bedingt sich aus ihrem Programm P, wobei die einzelnen Anweisungen von ihrer Funktion her in etwa den Zuständen einer Turingmaschine entsprechen. Entsprechend der Harvard-Architektur und im Gegensatz zur Von-Neumann-Architektur kann eine RAM nicht ihr eigenes Programm P verändern oder erweitern – ebenso wie eine TM ihre Zustände nicht verändern kann.

Beginnend bei der ersten Programmzeile ($b = 0$) arbeitet sich die RAM durch das Programm, wobei abhängig von der jeweiligen Programmzeile bzw. Anweisung (siehe Abbildung 5) unterschiedliche Operationen durchzuführen sind. Logische und arithmetische Operationen erfordern zwingend, dass sich der (bei unären Operatoren) bzw. einer der beiden Eingangswerte (bei binären Operatoren) im Akkumulator befindet. Das Programm endet sobald $b > |P|$ (implizit oder explizit durch END-Befehl) gilt.

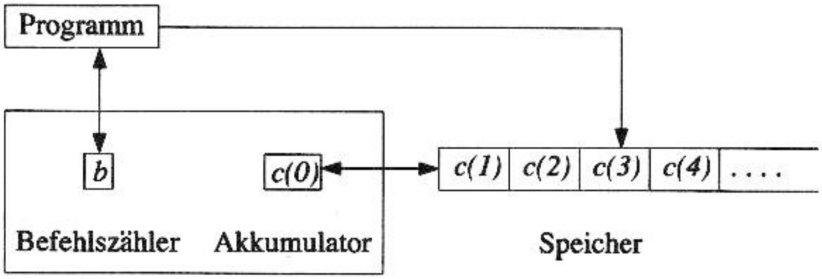

Abbildung 4: Schematischer Aufbau einer RAM [1, S. 7]

$LOAD\ i:$	$c(0) := c(i),\ b := b+1.$
$STORE\ i:$	$c(i) := c(0),\ b := b+1.$
$ADD\ i:$	$c(0) := c(0)+c(i),\ b := b+1.$
$SUB\ i:$	$c(0) := \max\{c(0)-c(i),0\},\ b := b+1.$
$MULT\ i:$	$c(0) := c(0)*c(i),\ b := b+1.$
$DIV\ i:$	$c(0) := \lfloor c(0)/c(i) \rfloor,\ b := b+1.$
$GO\ TO\ j$	$b := j.$
$IF\ c(0)?\ l\ GO\ TO\ j$	$b := j$ falls $c(0)?\ l$ wahr ist, und $b := b+1$ sonst. (Dabei ist $? \in \{=, <, \leq, >, \geq\}$).
END	$b := b.$

Abbildung 5: Schematischer Aufbau einer RAM [1, S. 8]

5 Simulation von Registermaschine auf Turingmaschine

Satz 1 *Jede logarithmisch $t(n)$-zeitbeschränkte Registermaschine kann für ein Polynom q durch eine $O(q(n+t(n)))$-zeitbeschränkte Turingmaschine simuliert werden. [1, S. 17s]*

Hinter der Idee der Transformation einer Registermaschine M_{RAM} in eine Turingmaschine M_{ES} steht die Idee der schrittweisen Simulationen: Eine, vom Befehl abhängige, Anzahl an Schritten von M_{ES} bewirkt einen Schritt von M_{RAM}. Um diese Simulation anschaulicher und nachvollziehbarer zu gestalten, soll sie in mehreren Stufen erfolgen. Dabei wird im Nachfolgenden immer gezeigt werden, wie sich eine Konfiguration der Eingangsmaschine auf der Ausgangsmaschine darstellt gefolgt von der Beschreibung eines Schrittes in der Eingangs- auf der Ausgangsmaschine.

$$\textbf{RAM } (M_{RAM}) \rightarrow \textbf{Mehrband-TM } (M_{MB}) \rightarrow \textbf{Mehrspur-TM } (M_{MS}) \Rightarrow \\ \textbf{(Einspur-)TM } (M_{ES})$$

Diese Schritte werden in den nachfolgenden Unterkapiteln ausführlich abgehandelt.

5.1 Registermaschine auf Mehrband-Turingmaschine

Die Mehrband-Turingmaschine verfügt über eine endliche Anzahl (k]) unendlicher Turing-Bänder, wobei für jedes ein eigener Lese-/Schreibkopf existiert, der unabhängig bewegt werden kann. Daraus ergibt sich, dass anstelle eines Eingabesymbols a mit der Länge 1 nun ein Eingabewort $w \in \Gamma^k$ mit der Länge k eingelesen wird. Ebenfalls verändert sich die Zustandsüberführungsfunktion: $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R, L, N\}^k$. Sie bildet nun anstatt auf einen 3 dimensionalen auf einen $2k + 1$ dimensionalen Raum ab.
Die hier Verwendung findende Mehrband-TM besteht aus vier Bändern mit verschiedenen Aufgaben:

- Band 1, auf dem die Ein- und Ausgabe der Turingmaschine bzw. der simulierten Registermaschine abgelegt wird
- Band 2, welches den Speicher/die Register R der Registermaschine M_{RAM} repräsentiert. Da die Mehrband-TM zwar endlich viele Bänder besitzen darf, diese Zahl jedoch nicht während der Ausführung wachsen darf und eine Registermaschine per Definition über unendlich viele Register verfügen kann (und sich die Anzahl der tatsächlich benutzen Register zur Laufzeit variieren kann), ist es nicht möglich pro Register ein einzelnes Band zu verwenden.
- Band 3, Hilfsband, welches den Akkumulator darstellt. Registerinhalte werden während der Ausführung hierhin kopiert und mit logischen/arithmetischen Operationen (siehe Abbildung 5) verarbeitet.
- Band 4, Band um die Adresse eines einzulesenden Registers zu vermerken. Bei direktem Zugriff kann dies sinnvollerweise auch über die Zustände direkt erfolgen

(ansonsten müsste der Inhalt des Bands ohnehin entsprechend der in den Zuständen enthaltenen Informationen geschrieben werden – Zustandsanzahl wächst durch diese Vorgehensweise also nicht), unter Verwendung der indirekten Adressierung können jedoch zur Laufzeit weitere Register alloziert werden und dies kann in der endlichen Zustandsmenge nicht berücksichtigt werden – daher bedarf es dieses Hilfsbands.

Die Registermaschine kann, wie bereits erwähnt, in ihren Registern beliege $p \in \mathbb{N}$ enthalten. D. h. ein Register kann unendlich viele verschiedene Werte annehmen. Der Inhalt einer Zelle auf dem Turing-Band ist jedoch durch das endliche Bandalphabet Γ (und damit die Länge 1) beschränkt. Daher muss der Registerinhalt Ziffernweise auf dem Band abgelegt werden[2]. Um es bei logischen und arithmetischen Operatoren einfacher zu haben sowie um das Bandalphabet klein zu halten, codieren wir die in den Registern enthaltenen ganzen Zahlen mit dem Binärsystem bevor sie auf dem Turing-Band abgelegt werden.

Da die Länge der Registerinhalte nicht konstant ist, ist es nicht möglich, den Anfang eines bestimmten Registers auf Band 2 a priori zu kennen. Die einzelnen Inhalte können daher nicht kontinuierlich auf dem Band abgelegt werden, sondern müssen durch Trennzeichen von einander abgegrenzt werden. Um keine festgelegte Reihenfolge der Registerrepräsentation auf dem Band zu erzwingen wird außerdem vor jeden Registerinhalt noch dessen Index geschrieben. Daraus ergibt sich eine Bandbelegung wie die folgende, wobei \$ Register untereinander und # den Index vom zugehörigen Inhalt abtrennt (zur Veranschaulichung definieren wir die Funktion $c(i)$, welche den Inhalt von Register R_i liefert. Des Weiteren definieren wir $bin(i)$, welche die Binärdarstellung von i zurückliefert. $bin(c(i))$ liefert folglich die Binärdarstellung des in R_i gespeicherten Wertes zurück):

$$\&bin(i_1)\#bin(c(i_1))\& \ldots \&bin(i_m)\#bin(c(i_m))\#\#$$

Die Realisierung des endlichen Programmzählers b erfolgt über die Zustände der simulierenden Turingmaschine, ebenso wie die Abbildung des Programms P selbst. In unserem abstrahierten Model stellen wir uns jede Zeile von P als ein Unterprogramm[3] der Turingmaschine vor. Eine Registermaschine mit einem p Zeilen/Anweisungen umfassenden Programm resultiert dabei in $p + 2$ Unterprogrammen – nach der Konstruktion einer Turingmaschine, die die „Hardware“ der Registermaschine emuliert, ist die Übersetzung der Befehle die eigentliche Simulation:

- P_0: Einlesen der Eingabe von Band 1 nach dem oben genannten Schema in einen festgelegten Bereich, beispielsweise beginnend am Anfang des Bandes, auf Band 2

[2] Ein Register kann theoretisch also unendlich viele Zellen auf dem Turing-Band belegen

[3] Eine Funktion oder ein Unterprogramm innerhalb einer Turingmaschine ist eine Menge von Zuständen mit dem Ziel der Abarbeitung einer Teilaufgabe. Dienen als abstraktes Modell um (theoretische) Programmierung zu vereinfachen. [1] geht soweit, über definierte Bandbereiche Parameter und Funktionsrückgaben zu realisieren.

- P_1 bis P_p: Unterprogramme, die aus der Übersetzung der entsprechenden Programmzeilen der Registermaschine resultieren.
- P_{p+1}: Schreibt die Ausgabe (d. h. den Inhalt eines festzulegenden Registers bzw. mehrerer Register) von Band 2 auf Band 1

Die Unterprogramme werden anschließend sequentiell, entsprechend ihrer Indexnummer, ausgeführt.

Auf ein ausführliches Beispiel, wie es bereits Teil des zugehörigen Vortrags war, muss an dieser Stelle leider, dem vorgegeben Umfang der Arbeit bzw. dessen Einhaltung geschuldet, verzichtet werden.

Will man nun Betrachtungen bezüglich der Laufzeitkomplexität anstellen, so sollten vorher noch die Funktionen $s(n)$ und $t(n)$ definiert werden. $t(n)$ beschreibt die Zeitkomplexität der ursprünglichen Registermaschine hinsichtlich der Eingabelänge. Die Simulation eines Schrittes von M_{RAM} wird mit $O(s(n))$ Schritten der simulierenden Turingmaschine abgeschätzt. Die obere Grenze von $s(n)$ kann wiederum, bei logarithmischem Zeitkomplexitätsmaß[4], durch $t(n)$ (da die Komplexität eines Schrittes zwingend geringer/gleich der der gesamten Registermaschine ist) abgeschätzt werden.
Bei genauem Betrachten können einige Eigenschaften der Umwandlung festgestellt werden. Ein ganz offenkundiger Nachteil ist die Emulation des Registerspeichers, denn der Zugriff erfolgt nun nicht mehr wahlfrei in konstanter Zeit $O(1)$ sondern sequentiell mit $O(n)$. Ein Schritt der M_{RAM} benötigt daher, wie oben erwähnt, $O(s(n))$ Schritte der Turingmaschine. Bei einer Registermaschine mit Zeitkomplexität von $O(t(n))$ und der vorher beschriebenen Abschätzung $s(n) \leq t(n)$ folgt für die Turingmaschine eine Zeitkomplexität von $O(t(n)^2)$. Anders gesagt werden aus $t(n)$ Schritten der Registermaschine bis zu $t(n)^2$ Schritte der Turingmaschine. Das Quadrat ist die Folge des sequentiellen Zugriff auf die Registerinhalte bzw. der Begrenzung von $s(n)$ durch $t(n)$.
Die Speicherverbrauchsklasse ändert sich hingegen nicht – es kommt lediglich ein Faktor durch Ablegen der Registerindizes und der Trennzeichen hinzu.

Es wurde also gezeigt, dass es zu einer jeden Registermaschine M_{RAM} eine äquivalente Mehrband-Turingmaschine M_{MB} gibt.

5.2 Mehrband-Turingmaschine auf Mehrspur-Turingmaschine

Der nächste Schritt von einer RAM hinzu einer „klassischen“, Einband-Turingmaschine ist der konstruktive Beweis, dass eine Mehrband-Turingmaschine, wie die weiter oben

[4]Das logarithmische Maß beschreibt hier die Länge der Eingabe als Anzahl der Stellen der entsprechenden Binärrepräsentation ($ceil(\log_2(p))+1 = |bin(p)|$). Das logarithmische Kostenmaß berücksichtigt also die Darstellungsgröße; das häufiger verwendete uniforme Maß hingegen nimmt für jede Operation eine gleiche, konstante Zeiteinheit an.

beschriebene, in eine Mehrspur-Turingmaschine übersetzt bzw. von dieser simuliert werden kann (M_{MB} kann auf M_{MS} reduziert werden).

Satz 2 *Eine k-Band-Turingmaschine M, die mit Rechenzeit $t(n)$ und Speicherplatz $s(n)$ auskommt, kann von einer Turingmaschine M' mit Zeitbedarf $O(t^2(n))$ und Speicherplatz $O(s(n))$ simuliert werden. [1, S. 15]*

Eine Mehrspur-Turingmaschine verfügt über ein unendliches Turing-Band und, wie der Name impliziert, über eine beliebige, jedoch endliche Anzahl k' an Spuren[5] k). Für jede Spur existiert ein eigener Lese-/Schreibkopf. Diese können, anders als bei der Mehrband-TM, jedoch nicht unabhängig voneinander bewegt werden („Kopfstrang“). Anstelle eines einzelnen Eingabesymbols liest auch sie ein k' Zeichen langes Eingabewort ein – dies gilt analog für die Ausgabe. Die Überführungsfunktion lautet: $\delta : Q \times \Gamma^{k'} \rightarrow Q \times \Gamma^{k'} \times \{R, L, N\}$. Den einzigen Unterschied zur Mehrband-TM stellt daher die Einschränkung hinsichtlich der „Bewegungsfreiheit“ der Lese-/Schreibköpfe dar.

Es liegt nahe, für jedes Band der Mehrband-Maschine M_{MB} eine Spur der Mehrspur-Maschine M_{MS} zu verwenden. Neben den Zuständen und ihren Bändern kann M_{MB} jedoch auch die Positionen der Köpfe verwenden um Informationen zu speichern – die Idee ist nun, diesen Unterschied durch Verwendung zusätzlicher Spuren in M_{MS} auszugleichen. Wie in [1] beschrieben verwenden wir daher $2k + 1$ (k bezieht sich auf M_{MB}) Spuren zur Simulation von M_{MB} auf M_{MS}: k Spuren für k Bänder, k Spuren für die Markierung der Kopfpositionen der k Bänder und eine weitere Spur um das Fenster festzulegen, innerhalb dessen sich alle Köpfe von M_{MB} nach Ausführung eines Schrittes befinden. Dies ist notwendig, um ohne „Abzählen“ der bereits besuchten Markierungen auszukommen und gleichzeitig nicht unendlich lange in eine Richtung zu laufen, in der Hoffnung eine weitere Markierung zu finden. Siehe Abbildung 6.

Ein Schritt von M_{MB} läuft in M_{MS} nun wie folgt ab: Beginnend in der Spalte von ˆ fährt M_{MS} alle mit # markierten Spalten an und liest das auf der jeweils zugehörigen „Datenspur“ gespeicherte Zeichen ein. Dies hat so zu geschehen, dass die Reihenfolge der Zeichen innerhalb des Wortes w gegenüber dem Einlesen der Mehrband-Maschine unverändert bleibt – M_{MS} und M_{MB} also in allen Situationen identische Wörter einlesen. Die Speicherung von w kann entweder in Form von Zuständen erfolgen ($|\Gamma|^k \Rightarrow$ kombinatorische Explosion[6]), oder über eine weitere Hilfsspur – in letzterem Fall müsste die Information, was nach dem eingelesen Wort zu tun ist bzw. dessen Erkennung, dennoch in den Zuständen gespeichert sein; die Zahl an Zuständen würde daher eher noch wachsen. Nachdem w nun bekannt ist, ist auch bekannt in welchen Zustand M_{MB} übergehen, welche Ausgabe M_{MB} schreiben und in welche Richtung M_{MB} jeden einzelnen Kopf bewegen würde. Zustandsübergang und Ausgabe können von M_{MB} übernommen werden – allerdings nur in der Form, dass der Nachfolgezustand von q_{MB} bzw. dessen Pendant der

[5] Vgl. Anzahl der Bänder bei Mehrband-Turingmaschinen

[6] Es sei erneut angemerkt, dass sich k auf die Anzahl der Bänder von M_{MB} bezieht und nicht etwas auf die Anzahl der Spuren k' von M_{MS}

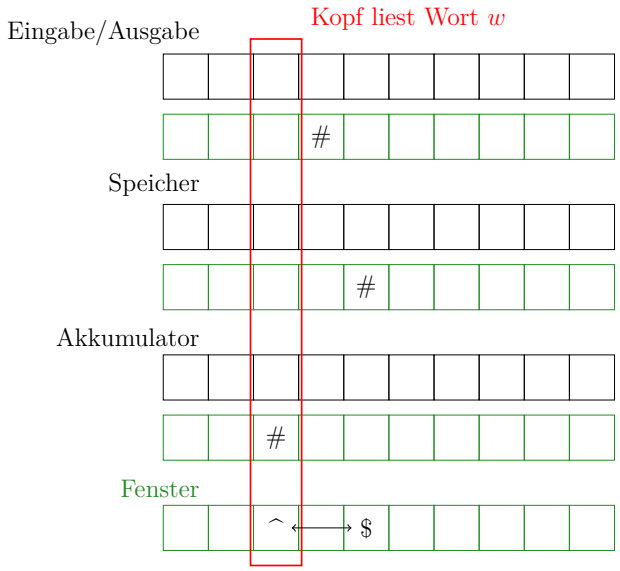

Abbildung 6: Schematische Darstellung des Aufbaus der Mehrband-TM

Folgezustand von q_{MS} sein muss, wobei jedoch vorher in Form von „Hilfszuständen“ die Ausgabe bzw. die Simulation der Kopfpositionen durchgeführt werden muss. Zur Bewegung der Köpfe bzw. zum Verschieben der Markierungen auf den Hilfsspuren bedarf es (womöglich) einer Kette von Hilfszuständen, die der Reihe nach die Markierungen um maximal eine Zelle nach rechts oder links verschieben.

Im schlimmsten Fall (erster Kopf nach rechts, zweiter Kopf nach links, dritter nach rechts ... und letztlich auf dem Fenster-Band nach links zu ^) kann eine solche, zur Ausgabe bzw. Markierung der entsprechenden Stellen notwendige, Kette $1+2(k-1)+2$ Zustände benötigen[7].

Abschließend muss der Kopfstrang von M_{MS} wieder an die Position von ^ bewegt werden. Der den Simulationsschritt abschließende Zustand von M_{MS} muss, wie bereits erwähnt, dem von M_{MB} entsprechen. Dies wird solange wiederholt, bis M_{MB} bzw. M_{MS} terminiert (gesetzt, M_{MB} terminiert überhaupt).

5.3 Mehrspur-Turingmaschine auf Einspur-Turingmaschine

Letzter Schritt ist der Beweis der Übertragbarkeit einer Mehrspur-Turingmaschine M_{MS}, wie sie im letzten Abschnitt vorgestellt wurde, hinzu einer Einspur-Turingmaschine M_{ES}. Dieser letzte Schritt basiert auf der simplen Idee einer „Codierung des Bandalphabets“.

Da sowohl die Anzahl an Spuren k' als auch der Umfang des Bandalphabets Γ von M_{MS} endlich und konstant sind, ist es möglich jede der endlich-vielen Kombination

[7]Diese Abschätzung bezieht sich ausschließlich auf den Aufwand, der nötig ist um die Fähigkeit der Mehrband-Turingmaschine, Köpfe unabhängig voneinander zu bewegen, zu simulieren – nicht auf die Simulation der Mehrband-Turingmaschine selbst!

$w \in {\Gamma_{M_{MS}}}^{k'}$ auf ein neues Zeichen $a \in \Gamma_{M_{ES}}$ abzubilden: $alph(w) \rightarrow a$; $\forall w \in {\Gamma_{M_{MS}}}^{k'} \rightarrow \exists a \in \Gamma_{M_{ES}}$ Die Größe des Bandalphabets $\Gamma_{M_{ES}}$ wächst exponentiell mit der Anzahl an Spuren k'.

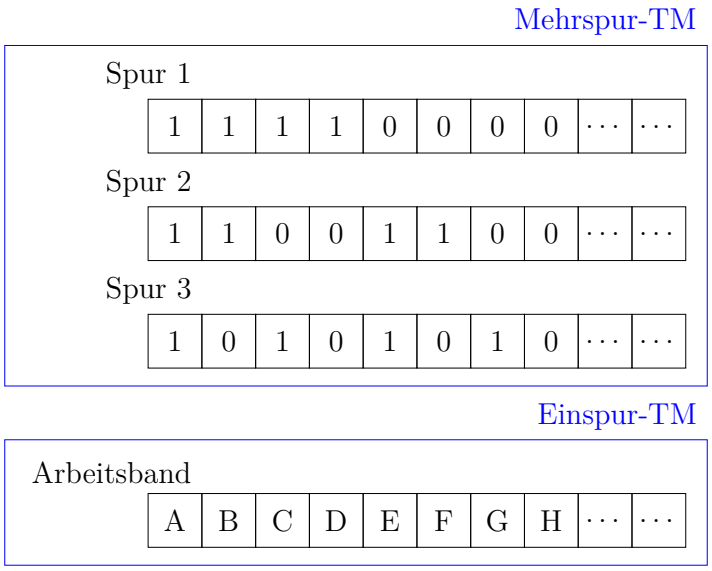

Abbildung 7: Schematische Abbildung des Eingabewortes w in M' auf ein Eingabesymbol in M''

Das Tripel $\{1, 1, 1\}$ würde dementsprechend auf A abgebildet, $\{1, 1, 0\}$ auf B usw..

Alternativ zu diesem Ansatz könnte die Maschine auch so umgebaut werden, dass sie direkt mit der, in k' langen Blöcken hintereinander geschriebenen, Binärdarstellung arbeitet.

6 Simulation von TM auf RAM

Im Gegensatz zur Überführung der Registermaschine hinzu einer Turingmaschine soll die Simulation einer Turingmaschine auf einer Registermaschine nur sehr kurz behandelt werden. Die nachfolgende Beweisskizze, dass eine TM innerhalb einer RAM ausgeführt werden kann, ist konstruktiver Art – daher gibt es auch einige Implementierungen.

Idee ist es, einen Turingmaschinen-Simulator in einer Sprache zu schreiben, der in Assembler (oder direkt Maschinencode) kompiliert und damit auf realen, registerbasierten CPUs ausgeführt werden kann. Da das theoretische Modell der RAM lediglich eine abstrahierte Vereinfachung der Realität ist, können beide hinsichtlich ihrer Mächtigkeit als gleich angesehen werden. Eine solche Sprache könnte bspw. eine Hochsprache wie Java (u. A. imperativ), Haskell (funktional) oder Prolog (logisch) sein. Kann in diesen ein solcher Simulator geschrieben werden, und solche Implementierungen gibt es wie „Sand am Meer“, ist gezeigt, dass eine abstrakte/reale Registermaschine *mindestens* so mächtig ist wie eine Turingmaschine und daher als Turing-vollständig bezeichnet werden kann.

Die weiteren Schlüsse, die aus diesem Ansatz gezogen werden können, werden im nachfolgenden Kapitel besprochen.

7 Zusammenfassung

Nachdem ihm Rahmen dieser Arbeit gezeigt wurde, dass sowohl eine Registermaschine in eine klassische Turingmaschine überführt, als auch, dass eine Turingmaschine auf einer Registermaschine emuliert werden kann – ohne dass sich die Laufzeit dabei superpolynomiell verschlechtern würde[1, S. 7] – sollen nun einige Schlüsse daraus gezogen werden.
8 Durch den Beweis der beidseitigen Übersetzbarkeit wird gezeigt, dass das theoretische Modell der Turingmaschine und die Grundidee moderner Rechnerarchitekturen gleichmächtig, und daher in der Lage sind, alle entscheidbaren Probleme zu lösen.
Nun kann gesagt werden, dass die Registermaschine (und durch die gegebene Transitivität auch die Turingmaschine) *mindestens* so mächtig ist, wie alle Programmiersprachen PS die auf ihr ausgeführt werden können, sprich in ausführbaren Maschinencode kompiliert werden können. Dadurch gilt, dass jede turingmächtig Sprache[8] *mindestens* so mächtig ist, wie alle anderen Sprachen $\in L^*$.

Für alle turingmächtigen Konzepte und Sprachen gilt, dass sie gleichmächtig sind – die Grundkonzepte der Programmierung lassen sich daher auf die „simplen“ Operationen der Turingmaschine herunterbrechen. Das heißt exemplarisch, dass Shell-Skript genauso

[8] Eine Sprache gilt als Turing-vollständig bzw. turingmächtig, wenn sie in der Lage ist, alle Berechnungen auszuführen, die von einer universellen Turingmaschine ausgeführt werden könnten. Üblicherweise wird Turing-Vollständigkeit durch Implementierung einer Turingmaschine in der jeweiligen Sprache bewiesen.

mächtig ist wie Java oder Conways „Game of Life“[2].

Der große Vorteil, der sich durch diese Beweismöglichkeit der Gleichmächtigkeit ergibt ist, dass die Implementierung einer universellen Turingmaschine in einer Hochsprache eine relative einfache Aufgabe ist – im Gegensatz zur alternativen Simulation der jeweiligen, zu vergleichenden Sprache. Auch ist die Implementierung der Turingmaschine keine „Einzelfallbetrachung“ sondern eine allgemeine Aussage gegenüber allen anderen Sprachen in der Menge der auf Registermaschinen ausführbaren Sprachen PS. Wie in Abbildung 8 gezeigt lassen sich die allermeisten Konzepte und Sprachen alle ineinander überführen und damit „schließt sich der Kreis“.

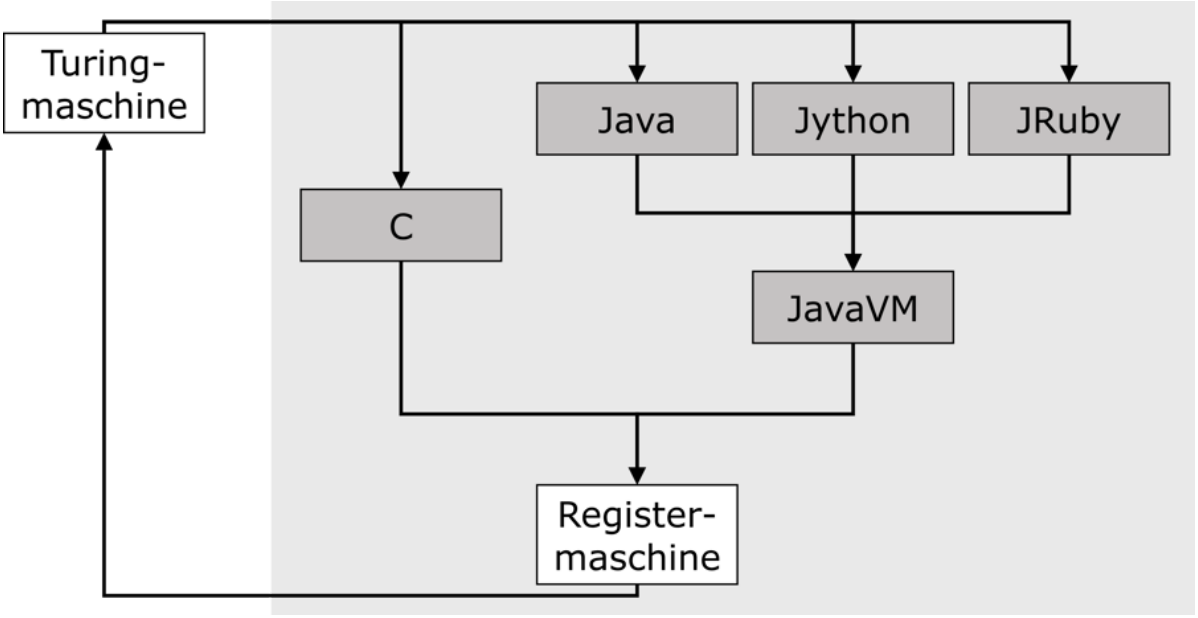

Abbildung 8: „Mächtigkeitskreis“ verschiedener, beispielhaft gewählter, Konzepte und Sprachen

8 Quellen- & Abbildungsverzeichnis

[1] Ingo Wegener. Theoretische Informatik : eine algorithmenorientierte Einführung. 3., überarb. Aufl. Leitfäden der Informatik. Wiesbaden: Teubner, 2005. ISBN: 3-8351-0033-5.

[2] Paul Rendell. A Turing Machine in Conway's Game of Life. URL: `http://rendell-attic.org/gol/tm.htm`.