universität
wien

BACHELORARBEIT

Titel der Bachelorarbeit

Logik erster Stufe ist nicht entscheidbar

Verfasser

Max Haunschmidt

angestrebter akademischer Grad

Bachelor of Science (BSc.)

Wien, im Juli 2019

Studienkennzahl lt. Studienblatt:	A 033621
Studienrichtung lt. Studienblatt:	Mathematik
Betreuerin:	Dr. Sandra Müller, MSc MSc

Abstract

Ziel dieser Arbeit ist es, den *Satz über die Unentscheidbarkeit der Logik erster Stufe* zu beweisen. Dazu werden zuerst eher naiv einige Begriffe aus der Berechenbarkeitstheorie eingeführt, die später anhand von Registermaschinen konkretisiert werden. Es folgt eine kurze Einführung zu Turingmaschinen. Danach wird die Unentscheidbarkeit des Halteproblems bewiesen, welche wiederum im Beweis des Hauptresultats eine wichtige Rolle spielt.

Inhaltsverzeichnis

1 Einleitung

> „Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt. [...] Das Entscheidungsproblem muss als das Hauptproblem der mathematischen Logik bezeichnet werden“
>
> –David Hilbert, Wilhelm Ackermann (1928) [3]

Bereits 8 Jahre nach der Veröffentlichung dieser motivierenden Aussage, wurde das „Entscheidungsproblem“ unabhängig und beinahe zeitgleich von Alonzo Church und Alan Turing gelöst – und zwar mit einer negativen Antwort. Sie beendeten dadurch eine Suche nach maschinell verifizierbarer Wahrheit, von der bereits Leibniz träumte (Stichwort *calculus ratiocinator*). Beide entwickelten dafür ihre eigene Konkretisierung eines *Verfahrens*: ersterer den λ-Kalkül, letzterer die später nach ihm benannte Turingmaschine. In der vorliegenden Arbeit habe ich einen dritten Zugang über Registermaschinen gewählt.

Der Text richtet sich an Studierende der Mathematik und Informatik, die bereits einige Grundlagen der mathematischen Logik erlernt haben. Insbesondere werden die Begriffe *Sprache, Struktur, Term, Formel, Allgemeingültigkeit* und *Beweisbarkeit* verwendet, ohne sie zu erklären. Man findet aber alle auftretenden Begriffe in den meisten einführenden Büchern zur mathematischen Logik, wie z.B. in [2]. Dieses Buch war auch die Hauptquelle für die vorliegende Arbeit, wobei kleine Änderungen in der verwendeten Notation vorgenommen wurden, um das Verständnis zu erleichtern.

2 Grundlagen

Dieser Abschnitt orientiert sich am Kapitel 10.1 aus [2] und liefert alle nötigen Grundlagen, um über Entscheidbarkeit, Aufzählbarkeit und deren Zusammenhang zu sprechen. Wichtig ist hierbei der Begriff des Verfahrens, der nur schwer mathematisch exakt zu definieren ist. Am besten denkt man bei Verfahren an Algorithmen, also an schrittweise verlaufende Anweisungen, die auf Zeichenreihen operieren und auf einer geeigneten Maschine durchführbar sind. Die exakte Definition eines Verfahrens folgt im Abschnitt über Registermaschinen.

Definition 1. Ein *Alphabet* $\mathbb{A}$ ist eine nicht leere Menge von Zeichen (Symbolen). Die Menge $\mathbb{A}^* = \bigcup_{n\in\mathbb{N}} \mathbb{A}^n$ bezeichne die Menge der *Wörter* über A. Ein Wort $\zeta \in \mathbb{A}^n$ hat Länge n. Das leere Wort, also die einzige Zeichenkette mit Länge null, bezeichnen wir mit $\diamond$.

Definition 2. Sei $\mathbb{A}$ ein Alphabet, W eine Menge von Wörtern über $\mathbb{A}$ und $\mathfrak{V}$ ein Verfahren.

(i). $\mathfrak{V}$ heißt *Entscheidungsverfahren* für W, falls $\mathfrak{V}$ bei jeder Eingabe $\zeta \in \mathbb{A}^*$ schließlich stoppt und zuvor genau einmal eine Ausgabe $\eta \in \mathbb{A}^*$ liefert, wobei

$$\eta = \diamond, \text{ falls } \zeta \in W \text{ und } \eta \neq \diamond, \text{ falls } \zeta \notin W.$$

(ii). W heißt *entscheidbar*, falls es ein Entscheidungsverfahren für W gibt.

Die folgenden Beispiele illustrieren die Definition:

Beispiel 1. $W :=$ „Die Menge der Primzahlen“ ist entscheidbar. Als Alphabet wähle $\mathbb{A} := \{0, \ldots, 9\}$. Ein $\zeta \in \mathbb{A}^*$ entspricht also einer natürlichen Zahl im Dezimalsystem. Das folgende Verfahren ist ein Entscheidungsverfahren für W:

(i). Falls $\zeta = 0$ oder $\zeta = 1$, gib $\eta = 1$ aus.

(ii). Falls $\zeta = 2$, gib $\eta = \diamond$ aus.

(iii). Falls $\zeta \geq 3$, prüfe der Reihe nach, ob $i \mid \zeta$, für $i \in \{2, \ldots, \zeta - 1\}$. Falls dies nie der Fall ist, gib $\eta = \diamond$ aus, ansonsten $\eta = 1$.

Bemerkung 1. Dieses Verfahren ist natürlich nicht effizient, aber hier begnügen wir uns mit der Tatsache, *dass* etwas möglich ist, und nicht mit Fragen der Speichernutzung oder Rechengeschwindigkeit.

Beispiel 2. Falls W entscheidbar ist, so ist auch $\mathbb{A}^* \setminus W$ entscheidbar. Denn für W existiert per Definition ein Entscheidungsverfahren $\mathfrak{V}$, das für alle eingegebenen $\zeta \in W$ $\eta = \diamond$ ausgibt und $\eta \neq \diamond$, sonst. Durch Vertauschen der Ausgaben in $\mathfrak{V}$ erhält man ein Entscheidungsverfahren $\bar{\mathfrak{V}}$, das genau $\mathbb{A}^* \setminus W$ entscheidet.

Definition 3. (Aufzählbarkeit) Sei $\mathbb{A}$ ein Alphabet, $W \subset \mathbb{A}^*$ und $\mathfrak{V}$ ein Verfahren.

(i). $\mathfrak{V}$ heißt ein *Aufzählungsverfahren* für W, falls $\mathfrak{V}$ genau die Wörter aus W ausgibt. Dabei spielt weder die Reihenfolge noch mehrfache Ausgaben des gleichen Wortes eine Rolle.

(ii). W heißt *aufzählbar*, falls es ein Aufzählungsverfahren für W gibt.

Bemerkung 2. Eine aufzählbare Menge ist also automatisch abzählbar. Die Umkehrung gilt jedoch im Allgemeinen nicht, wie am Ende der Arbeit gezeigt wird.

Um die Nicht-Entscheidbarkeit der Logik erster Stufe beweisen zu können, muss diese erst definiert werden. Wieder beziehe ich mich dabei auf [2], genauer auf Kapitel 2.2.

Definition 4. Das zu Grunde liegende Alphabet $\mathbb{A}_\infty$ soll mächtig genug sein, um „alle“ mathematischen Sachverhalte ausdrücken zu können. $\mathbb{A}_\infty$ enthält also

(i). $v_0, v_1, v_2, \ldots$ (abzählbar viele Variablen)

(ii). $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

(iii). $\exists, \forall$

(iv). $\doteq$

(v).),(
Außerdem braucht es noch Symbole für abzählbar viele Konstanten, Funktionen und Relationen, letztere auch in beliebiger endlicher Stelligkeit. Wir fügen also noch

(vi). $c_0, c_1, c_2, \ldots, f_0^n, f_1^n, f_2^n, \ldots$ *und* $R_0^n, R_1^n, R_2^n, \forall n \in \mathbb{N}$
hinzu, wobei das hochgestellte n für die Stelligkeit von Funktions- bzw. Relationszeichen steht.

Punkt (vi) definiert also insbesondere eine Sprache S_∞. Aus diesem Alphabet lassen sich nun beliebige *Terme* formen, welche wiederum zu *Formeln*, (manchmal auch *Ausdrücken*) verbunden werden können. Eine Formel, in der alle vorkommenden Variablen durch einen Quantor gebunden sind, heißt *Satz* (manchmal auch *Aussage*).

$L_n^{S_\infty}$ steht für die Menge der S_∞-Formeln mit n freien Variablen. Für die weiteren Ausführungen besonders wichtig ist $L_0^{S_\infty}$, die Menge der S_∞ Sätze.

Bemerkung 3. Man kann jedes Wort aus $\mathbb{A}_\infty^*$ auch mit Symbolen des endlichen Alphabets $\mathbb{A}_0$ schreiben. Wähle

$$\mathbb{A}_0 := \{v, \underline{0}, \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}, \underline{8}, \underline{9}, \bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}, \bar{7}, \bar{8}, \bar{9}, \neg, \vee, \exists, \doteq,), (, c, f, R\}.$$

Das Wort $R^2_{317}f^1_{42}c_{444}v_1$ wird so zu $R\underline{3}\underline{1}\underline{7}\bar{2}f\underline{4}\underline{2}\bar{1}c\underline{4}\underline{4}\underline{4}v\underline{1}$. Dies impliziert auch eine *lexikographische Ordnung* für Wörter aus $\mathbb{A}_0^*$: Ein Wort ist lexikographisch vor einem anderen, falls es kürzer ist (also aus weniger Symbolen besteht). Sollten beide gleich lang sein, ordnet man sie ihren Symbolen nach. Beispielsweise ist $\vee\underline{7}$ lexikographisch vor $\vee\bar{5}$, weil $\underline{7}$ in der Definition von $\mathbb{A}_0$ vor $\bar{5}$ geschrieben wurde.

Notation. Da sich auf ähnliche Art und Weise beinahe jedes Alphabet in ein endliches Alphabet verwandeln lässt, reicht es aus, endliche Alphabete zu betrachten. Sei deshalb ab jetzt $\mathbb{A} := \{a_0, \ldots, a_r\}$ immer ein endliches Alphabet.

Lemma 2.1. *Die Menge* $\Phi := \{\varphi \in L_0^{S_\infty} \mid \models \varphi\}$ *der allgemeingültigen* S_∞*-Sätze ist aufzählbar.*

Beweis. Der Vollständigkeitssatz liefert uns $\models \varphi \iff \vdash \varphi$. Gesucht ist also ein Verfahren $\mathfrak{V}$, dass systematisch die beweisbaren φ auflistet. $\mathfrak{V}$ arbeitet induktiv: für $n = 1, 2, 3, \ldots$ stellt es zuerst die lexikographisch ersten n S_∞ Terme und Formeln her. Dabei verwenden wir das in Bemerkung 3 definierte, endliche Alphabet $\mathbb{A}_0$. Mit diesen Termen und Formeln bildet das Verfahren alle möglichen Ableitungen der Länge $\leq n$, die jeweils aus höchstens n Sequenzen bestehen. Eine Sequenz ist dabei eine endliche Aneinanderreihung von Formeln. Falls nun die letzte Sequenz einer Ableitung aus einem einzigen Satz φ besteht, so ist φ beweisbar und wird von $\mathfrak{V}$ ausgegeben □

Entscheidbarkeit und Aufzählbarkeit sind miteinander verwandte, aber nicht gleiche Eigenschaften. Es gilt etwa:

Lemma 2.2. *Jede entscheidbare Teilmenge* W *von Wörtern aus einem endlichen Alphabet* $\mathbb{A}$ *ist aufzählbar*

Beweis. Mit Hilfe des Entscheidungsverfahrens $\mathfrak{V}$ von W ist es leicht, ein Aufzählungsverfahren $\mathfrak{W}$ zu konstruieren. $\mathfrak{W}$ arbeitet induktiv über die lexikographische Ordnung aller Wörter aus $\mathbb{A}^*$, prüft sie mit $\mathfrak{V}$ auf ihre Zugehörigkeit zu W und gibt sie bei positiver Antwort aus. □

Bemerkung 4. Dass die Rückrichtung im Allgemeinen nicht gilt, kann man sich wie folgt veranschaulichen: Angenommen, man hat ein Aufzählungsverfahren $\mathfrak{V}$ für W und möchte wissen, ob für ein $\zeta \in \mathbb{A}^*$ auch $\zeta \in W$ gilt. Wird ζ schließlich von $\mathfrak{V}$ ausgegeben, so gehört es sicherlich zu W. Solange ζ aber nicht ausgegeben wird wissen wir nicht, ob das daran liegt, dass $\zeta \notin W$, oder dass $\mathfrak{V}$ einfach

noch nicht lange genug gelaufen ist. Das Hauptresultat dieser Arbeit (Satz 6.1) zusammen mit Lemma 2.1 liefert ein formales Beispiel.

Hat man jedoch zusätzlich die Aufzählbarkeit des Komplements von W, so folgt die Äquivalenz:

Lemma 2.3. *Eine Teilmenge W von $\mathbb{A}^*$ ist entscheidbar genau dann, wenn W und $\mathbb{A}^* \setminus W$ aufzählbar sind.*

Beweis. „$\Longrightarrow$“: Aus Beispiel 2 folgt die Entscheidbarkeit von $\mathbb{A}^* \setminus W$. Doppelte Anwendung des vorherigen Lemmas liefert die Aufzählbarkeit von W und $\mathbb{A}^* \setminus W$.
„$\Longleftarrow$“: Das Entscheidungsverfahren für W startet die beiden Aufzählungsverfahren für W und dessen Komplement parallel. Jedes $\zeta \in \mathbb{A}^*$ muss irgendwann von genau einem der beiden Aufzählungsverfahren aufgezählt werden. So kann für jedes ζ entschieden werden, ob es zu W gehört oder nicht. □

3 Registermaschinen

In diesem Abschnitt wird die abstrakte *Registermaschine* eingeführt, was es auch ermöglicht den Begriff des Verfahrens zu konkretisieren. Dabei bildete das Kapitel 10.2 in [2] meine Hauptquelle. Im Folgenden sei $\mathbb{A} = \{a_0, a_1, \ldots, a_r\}$ wieder ein endliches Alphabet.

Eine Registermaschine besteht aus beliebig vielen Speicherplätzen (oder Registern) $R_0, \ldots, R_m$. In jedem dieser Speicherplätze befindet sich zu jedem Zeitpunkt genau ein Wort beliebiger Länge aus $\mathbb{A}^*$. Registermaschinen können Anweisungen ausführen, die in einer bestimmten, einfachen „Programmiersprache“ geschrieben sind.

Definition 5. Ein *(Register-)Programm* über $\mathbb{A}$ für eine Registermaschine besteht aus endlich vielen, geordneten Zeilen $z_0, \ldots, z_k$, die jeweils eine Anweisung enthalten. Jede Zeile z_i beginnt mit einer Zeilennummer i, gefolgt von genau einer der nachstehenden Anweisungen:

(i). $R_l = R_l + a_j$ („Verlängerungsanweisung“)

(ii). $R_l = R_l - a_j$ („Verkürzungsanweisung“)

(iii). IF $R_l = \diamond$ THEN z_j ELSE z_{j_0} OR ... OR z_{j_r} („Sprunganweisung“)

(iv). PRINT („Druckanweisung“)

(v). STOP („Stoppanweisung“)

Die Verlängerungsanweisung hängt das Symbol a_j an das Ende des Wortes im Register R_l.
Falls das Wort im Register R_l mit dem Symbol a_j endet, löscht die Verkürzungsanweisung dieses letzte Symbol.
Die Sprunganweisung liest den Eintrag im Register R_l. Falls es das leere Wort $\diamond$ ist, führt die Maschine als nächstes Zeile z_j aus, sie *springt* also. Ansonsten ist das letzte Symbol des Wortes in R_i entscheidend. Falls das Wort mit a_0 endet, wird in die Zeile z_{j_0} gesprungen. Falls es mit a_1 endet, wird in die Zeile z_{j_1} gesprungen usw. die einzelnen z_{j_i} müssen dabei nicht notwendigerweise verschieden sein, aber immer kleiner oder gleich der Nummer der letzten Zeile z_k. Falls $z_j = z_{j_i}$ für alle i – wird also in jedem Fall in Zeile z_j gesprungen – so schreibe GOTO z_j.
Die Druckanweisung gibt das im Register R_0 stehende Wort aus.
Die Stoppanweisung beendet das Programm. Sie ist immer genau die letzte Zeile z_k eines Programms.

Bemerkung 5. Die Definition von Registermaschinen ist in der Literatur nicht eindeutig. Diese Definition orientiert sich stark an [2], während in [4] eine ähnliche und in [1] eine sehr abweichende Definition gegeben werden.

Bemerkung 6. Bis jetzt war ein Verfahren eine Art Algorithmus: Es besteht aus schrittweise verlaufenden Anweisungen, die auf Zeichenreihen operieren und auf einer geeigneten Rechenmaschine durchführbar sind. Eine Möglichkeit für jene „geeignete Rechenmaschine“ ist eine Registermaschine, an welche wir für den Rest dieser Arbeit bei einem Verfahren denken. Eine andere Möglichkeit für eine sog. *abstrakte Maschine* wäre z.B. die Turingmaschine.

Definition 6. Ab jetzt sei ein Verfahren $\mathfrak{V}$ ein Registerprogramm P, das auf einer Registermaschine ausgeführt wird. Eine eventuelle Eingabe (manchmal auch *Input*) des Verfahrens steht immer im Register R_0, die übrigen Register enthalten am Beginn der Berechnungen das leere Wort.

Beispiel 3. Sei $\mathbb{A} = \{|\}$, wobei ein $\zeta \in \mathbb{A}^*$ über seine Länge mit einer natürlichen Zahl identifiziert wird. Diese Tatsache schreiben wir als $\mathbb{A}^* \simeq \mathbb{N}$. Das folgende Programm P_{parity} entscheidet, ob eine Eingabe, also das Wort, das zu Beginn der Berechnung im Register R_0 steht, eine gerade oder eine ungerade Zahl ist. Bei einer geraden Anzahl an Strichen wird $\diamond$ ausgegeben, ansonsten $|$

0 IF $R_0 = \diamond$ THEN 6 ELSE 1
1 $R_0 = R_0 - |$
2 IF $R_0 = \diamond$ THEN 5 ELSE 3
3 $R_0 = R_0 - |$
4 IF $R_0 = \diamond$ THEN 6 ELSE 1
5 $R_0 = R_0 + |$
6 PRINT
7 STOP

Notation. Obiges Programm P_{parity} wird bei jedem Input $\zeta \in \mathbb{A}^*$ schließlich stoppen (manchmal verwendet man auch den Begriff *terminieren*). Falls ein Programm P für einen bestimmten Input ζ terminiert, schreibe

$$P : \zeta \longrightarrow \text{stop},$$

andernfalls

$$P : \zeta \longrightarrow \infty.$$

Falls P bei Input ζ genau das Wort η ausgibt und schließlich stoppt schreibe

$$P : \zeta \longrightarrow \eta,$$

und lese „P angesetzt auf ζ liefert η“. Die gleiche Notation wird ab jetzt auch für allgemeine Verfahren $\mathfrak{V}$ verwendet, die im vorigen Kapitel besprochen wurden.

Beispiel 4. Ein Programm P über $\mathbb{A} = \{a_0, a_1, a_2\}$, sodass

$$P : \zeta \longrightarrow \text{stop}, \quad \text{falls } \zeta = a_0a_0a_2 \text{ und}$$

$$P : \zeta \longrightarrow \infty, \quad \text{falls } \zeta \neq a_0a_0a_2$$

könnte so aussehen:

0 IF $R_0 = \diamond$ THEN 1 ELSE 1 ELSE 1 ELSE 2
1 GOTO 1
2 $R_0 = R_0 - a_2$
3 IF $R_0 = \diamond$ THEN 1 ELSE 4 ELSE 1 ELSE 1
4 $R_0 = R_0 - a_0$
5 IF $R_0 = \diamond$ THEN 1 ELSE 6 ELSE 1 ELSE 1
6 $R_0 = R_0 - a_0$
7 IF $R_0 = \diamond$ THEN 8 ELSE 1 ELSE 1 ELSE 1
8 STOP

Bemerkung 7. Da ein Verfahren nun exakt definiert ist, müsste man alle der im ersten Kapitel eingeführten Begriffe erneut definieren. Hier kommt uns aber die *Church-Turing-These* zu Hilfe. Sie besagt, dass die intuitiven Definitionen von Aufzählbarkeit und Entscheidbarkeit genau die gleichen Ergebnisse liefern, wie Definitionen, die sich auf den präzisierten Verfahrensbegriff stützen. Dabei spielt es keine Rolle, ob hinter dem Verfahren eine Registermaschine, eine Turingmschine oder ein sonstiges, gleichwertiges Rechenmodell steht. Manchmal möchte man aber trotzdem betonen, dass eine Menge W durch ein Registerprogramm aufgezählt oder entschieden wird. In diesem Fall sagt man, W ist *R-aufzählbar* bzw. *R-entscheidbar*.

4 Alternative Zugänge

Am Ende des vorherigen Kapitels wurde die Turingmaschine erwähnt, welche im folgenden Kapitel gemeinsam mit dem Begriff *berechenbar* eingeführt und diskutiert wird. Das gesamte Kapitel ist jedoch für die Formulierung und den Beweis des Hauptresultats nicht notwendig und dient lediglich einem besseren Verständnis der theoretischen Informatik im Allgemeinen. Als Quelle dienten hauptsächlich [1], sowie [2] und [4].

4.1 Berechenbarkeit

Definition 7. Seien $\mathbb{A}$ und $\mathbb{B}$ Alphabete, $f : \mathbb{A}^* \to \mathbb{B}^*$ eine (partielle)[1] Funktion. Die (partielle) Funktion f heißt *berechenbar*, falls es ein Berechnungsverfahren $\mathfrak{V}$ über $\mathbb{A} \cup \mathbb{B}$ gibt, mit $\mathfrak{V} : \zeta \to f(\zeta),\ \forall \zeta \in dom(f) \subseteq \mathbb{A}^*$, das also für jedes Wort ζ aus dem Definitionsbereich von f als Eingabe, die Ausgabe $f(\zeta) \in \mathbb{B}^*$ liefert.

Beispiel 5. Seien $\mathbb{A} = \{a_0, a_1, \ldots, a_r\}$ und $\mathbb{B} = \{0, 1, \ldots, 9\}$. Die Funktion $f : \mathbb{A}^* \to \mathbb{B}^*$, die jedem Wort ζ aus $\mathbb{A}^*$ die eigene Länge – also die Anzahl der Symbole in Dezimaldarstellung – zuordnet, ist berechenbar.

Definition 8. Jedes Registerprogramm P über $\mathbb{A}$ definiert eine berechenbare partielle Funktion $f : \mathbb{A}^* \to \mathbb{A}^*$. Diese partielle Funktion heißt die von P berechnete partielle Funktion. Eine (partielle) Funktion, die von einem Registerprogramm berechnet werden kann, heißt *R-berechenbar*.

Zwischen den bereits definierten Begriffen Aufzählbarkeit, Entscheidbarkeit und Berechenbarkeit besteht wieder eine Verbindung:

Beispiel 6. Sei $\mathfrak{V}$ ein Entscheidungsverfahren für $W \subseteq \mathbb{A}^*$. Dann ist die Funktion $f : \mathbb{A}^* \to \mathbb{A}^*$, mit $f(\zeta) = \diamond$, falls $\zeta \in W$ und $f(\zeta) \neq \diamond$, falls $\zeta \notin W$ berechenbar, denn das Berechnungsverfahren ist genau das Entscheidungsverfahren.

Die Verbindung ist sogar noch stärker:

Lemma 4.1. *Seien $\mathbb{A}$ und $\mathbb{B}$ Alphabete, $\# \notin \mathbb{A} \cup \mathbb{B}$ und $f : \mathbb{A}^* \to \mathbb{B}^*$. Dann sind die folgenden Aussagen äquivalent:*

(i). f ist berechenbar.

(ii). $W := \{\zeta \# f(\zeta) | \zeta \in \mathbb{A}^*\} \subseteq (\mathbb{A} \cup \mathbb{B} \cup \{\#\})^*$ ist entscheidbar.

(iii). W ist aufzählbar.

[1] Zur Erinnerung: Eine partielle Funktion ist zwar rechts-eindeutig (oder funktional) aber nicht notwendigerweise links-total.

Beweis. (i) $\Longrightarrow$ (ii): Da f berechenbar ist, folgt per Definition die Existenz eines Verfahrens $\mathfrak{V}$ mit $\mathfrak{V} : \zeta \to f(\zeta)$, $\forall \zeta \in \mathbb{A}^*$. Ein Entscheidungsverfahren für W könnte so aussehen:

Schritt 1: Zähle, wie oft das Symbol # in der Eingabe $\zeta' \in (\mathbb{A} \cup \mathbb{B} \cup \{\#\})^*$ vorkommt. Falls # genau ein Mal vorkommt, mache weiter mit Schritt 2, ansonsten gilt sicher $\zeta' \notin W$.

Schritt 2: Prüfe, ob die Zeichenkette ζ vor dem Symbol # ein Element von $\mathbb{A}^*$ ist. Falls ja, mache weiter mit Schritt 3, ansonsten gilt sicher $\zeta' \notin W$.

Schritt 3: Berechne mit $\mathfrak{V}$ den Funktionswert $f(\zeta)$ und vergleiche das Ergebnis mit der Zeichenkette in ζ' nach dem Symbol #. Falls beide übereinstimmen, gilt $\zeta' \in W$, ansonsten gilt sicher $\zeta' \notin W$.

(ii) $\Longrightarrow$ (iii): Aus Entscheidbarkeit folgt immer Aufzählbarkeit, siehe Lemma 2.2.

(iii) $\Longrightarrow$ (i): Sei $\mathfrak{V}$ ein Aufzählungsverfahren für W. Ein Berechnungsverfahren für f könnte so aussehen:

Schritt 1: Nach dem Einlesen der Eingabe $\zeta \in \mathbb{A}^*$ starte das Aufzählungsverfahren $\mathfrak{V}$

Schritt 2: Da $\mathfrak{V}$ ganz W aufzählt, muss auch $\zeta \# f(\zeta)$ irgendwann ausgegeben werden und man kann $f(\zeta)$ ablesen und ausgeben.

□

4.2 Turingmaschinen

Die im vorigen Kapitel definierte Registermaschine ist ein sehr stark reduziertes Modell eines Computers, wie wir ihn heute, zu Beginn des 21. Jahrhunderts, kennen und täglich nutzen. Zur Zeit der Publikation von Alan Turings Text „On Computable Numbers with an Application to the Entscheidungsproblem" (1936), verstand der Autor unter einem Computer noch etwas ganz anderes: „We may now construct a machine to do the work of this computer."[5] Ein Computer war nichts anderes, als ein Mensch, der mit Zettel und Stift Berechnungen durchführte. Die Turingmaschine ist eine Abstraktion eben dieser Art von Computer. Aus dem zweidimensionalen Blatt Papier wird ein unendlich langes, eindimensionales, in Kästchen unterteiltes Band. Auf diesem Band rechnet nun aber kein Mensch, sondern ein sogenannter *Lese-/Schreibkopf*, der durch ein Programm gesteuert wird und zu jedem Zeitpunkt über genau einem der Kästchen steht. Ob das Band auf beiden Seiten des Lese-/Schreibkopfes unendlich lang ist oder nur auf einer, ist einerseits Geschmackssache, andererseits für die erzielten Ergebnisse irrelevant (siehe z.B. [1] Seite 59).

Die Turingmaschine arbeitet zyklisch: Zuerst wird vom Lese-/Schreibkopf der Inhalt jenes Kästchens, über dem er sich gerade befindet, eingelesen. Der Inhalt eines Kästchens ist im Gegensatz zur Registermaschine nicht beliebig lang, sondern besteht aus höchstens einem Symbol des *Bandalphabets* $\mathbb{A} = \{a_0, \ldots, a_r\}$. Nach dem Einlesen eines Symbols a_{i_1} kommt das angesprochene Programm ins Spiel. Es besteht nicht wie bei Registermaschinen aus hintereinander auszuführenden Anweisungszeilen, sondern aus *Zuständen* $q_1, \ldots, q_n$ und einer *Übergangsfunktion* δ. Die Maschine befindet sich zu jedem Zeitpunkt in genau einem Zustand q_{j_1}, welcher gemeinsam mit dem eingelesenen Symbol a_{i_1} den Funktionswert der Übergangsfunktion $\delta(q_{j_1}, a_{i_1})$ bestimmt. Dieser Funktionswert ist ein Tripel (q_{j_2}, a_{i_2}, d), welches den neuen Zustand q_{j_2}, das in das aktuelle Kästchen zu schreibende Symbol a_{i_2} und die Richtung, in die sich der Lese-/Schreibkopf als nächstes bewegen soll, vorgibt. Der Lese-/Schreibkopf überschreibt das Symbol a_{i_1} mit dem Symbol a_{i_2} und bewegt sich danach höchstens ein Kästchen in die vorgegebene Richtung. Dabei bedeutet $d = L$ ein Kästchen nach links, $d = R$ ein Kästchen nach rechts und $d = N$ keine Bewegung. Schreiben wir für die Menge der Zustände $Q := \{q_1, \ldots, q_n\}$, so können wir die Übergangsfunktion als $\delta : Q \times \mathbb{A} \to Q \times \mathbb{A} \times \{L, R, N\}$ definieren. Nach dem Bewegen des Lese-/Schreibkopfes beginnt der Zyklus mit dem Einlesen des neuen Symbols wieder von vorn.

Am Beginn der Berechnungen ist die Turingmaschine im *Startzustand* $q_0 \in Q$ und der Lese-/Schreibkopf befindet sich über jenem Kästchen, in dem das erste Symbol der Eingabe steht. Die restlichen Symbole stehen in den Kästchen rechts davon, alle übrigen Kästchen enthalten das leere Wort $\diamond$. Eine Turingmaschine beendet ihre Berechnung, wenn sie einen der festgelegten *Endzustände* $q_{e_j} \in Q_e \subseteq Q$ erreicht. Die Ausgabe ist die Aneinanderreihung aller Symbole rechts des Lese-/Schreibkopfes, beginnend mit dem Symbol in dem Kästchen unter dem Lese-/Schreibkopf. Es folgt die formale Definition:

Definition 9. Eine Turingmaschine ist ein Tupel

$$T = (Q, \mathbb{A}, \delta, \zeta, q_0, Q_e),$$

wobei

- $Q := \{q_1, \ldots, q_n\}$ eine endliche Menge von Zuständen,
- $\mathbb{A} := \{a_0, \ldots, a_r\}$ ein endliches Bandalphabet (meist verwendet man $\mathbb{A} = \{0, 1\}$),
- $\delta : Q \times \mathbb{A} \to Q \times \mathbb{A} \times \{L, R, N\}$ die Übergangsfunktion,
- $\zeta \in \mathbb{A}^*$ die Eingabe zu Beginn der Berechnungen,

- q_0 der Startzustand und
- $Q_e \subseteq Q$ eine endliche Menge von Endzuständen ist.

Bemerkung 8. Wem diese Definition zu trocken erscheint und gerne eine Turingmaschine in Aktion sehen möchte, sei auf die Website https://turingmachinesimulator.com/ hingewiesen.

Bemerkung 9. Wie schon bei der Definition von Registermaschinen ist sich die Fachliteratur auch bei Turingmaschinen nicht einig, wie sie zu definieren ist. Die gegebene Definition orientiert sich stark an [1], wobei in [4] wieder ein ähnlicher Weg gegangen wird.

Uns steht nun eine weitere Möglichkeit zur Verfügung, Verfahren (und somit Entscheidbarkeit, Aufzählbarkeit und Berechenbarkeit) exakt zu definieren. Analog zu R-Entscheidbarkeit, R-Aufzählbarkeit und R-Berechenbarkeit, sei eine Menge T-aufzählbar, T-entscheidbar, bzw. eine (partielle) Funktion T-berechenbar, falls das zugehörige Verfahren durch eine Turingmaschine realisiert wird.

Man kann auch eine *mehrbändige* Turingmaschine definieren. Diese besteht aus $k \in \mathbb{N} \setminus \{0\}$ Bändern und Lese-/Schreibköpfen, die sich unabhängig von einander auf ihrem jeweiligen Band bewegen können. Die Übergangsfunktion δ muss dann angepasst werden zu $\delta : Q \times \mathbb{A}^k \to Q \times (\mathbb{A} \times \{L, R, N\})^k$.

Es folgen noch zwei Resultate ohne Beweise, welche aber in [1] nachgeschlagen werden können. Sie zeigen, dass die von (mehrbändigen) Turingmaschinen und Registermaschinen berechenbaren Funktionen (und damit die durch sie entscheidbaren und aufzählbaren Mengen) exakt die selben sind. Beide Abstraktionen des *Computers* haben also die gleichen Möglichkeiten und Grenzen.

Satz 4.2. *Jede mehrbändige Turingmaschine kann durch eine einbändige Turingmaschine simuliert werden.*

Beweis. Siehe [1] Seite 53 ff. □

Satz 4.3. *Sei $f : \mathbb{A}^* \to \mathbb{A}^*$ eine (partielle) Funktion. Dann gilt:*

$$f \text{ ist R-berechenbar} \iff f \text{ ist T-berechenbar.}$$

Beweis. Siehe [1] Seite 64 ff. □

5 Halteproblem

In diesem Kapitel wird das sogenannte *Halteproblem* speziell für Registermaschinen erörtert, wobei ich mich stark an den Ausführungen in Kapitel 10.3 von [2] orientierte. Grob gesprochen lautet seine Aussage:
Es gibt keine Maschine, die für alle Maschinen immer richtig entscheidet, ob sie für einen bestimmten Input terminieren oder nicht.
Um den Satz exakt formulieren zu können, wird die *Gödelisierung* von Programmen eingeführt. Es sei wie zuvor $\mathbb{A} = \{a_0, \ldots, a_r\}$ ein endliches Alphabet.

Definition 10. Sei

$$\mathbb{B} := \mathbb{A} \cup \{A, B, C, \ldots, X, Y, Z, 0, 1, \ldots, 8, 9, =, +, -, \diamond, |\}$$

eine Erweiterung des Alphabets um alle Zeichen, die in einem Registerprogramm vorkommen. Nun kann jedes Programm P über $\mathbb{A}$ als ein Wort in $\mathbb{B}^*$ dargestellt werden - man schreibt einfach die Zeilen hintereinander anstatt untereinander und trennt sie mit dem Symbol $|$. In der lexikographischen Reihenfolge von $\mathbb{B}^*$ (siehe Bemerkung 3) hat dieses Wort einen eindeutigen Platz n. Als *Gödelnummer* ξ_P von P definiert man die n-malige Aneinanderreihung des Symbols a_0. Den Übergang von P zu ξ_P bezeichnet man als *Gödelisierung*.

Notation. Setze:

$$\Pi := \{\xi_P \mid P \text{ Programm über } \mathbb{A}\},$$

$$\Pi'_{stop} := \{\xi_P \mid P \text{ Programm über } \mathbb{A} \text{ und } P : \xi_P \longrightarrow \text{stop}\} \text{ und}$$

$$\Pi_{stop} := \{\xi_P \mid P \text{ Programm über } \mathbb{A} \text{ und } P : \diamond \longrightarrow \text{stop}\}$$

Lemma 5.1. Π *ist R-entscheidbar*

Beweis. Wir stützen uns wieder auf die Church-Turing-These und geben ein allgemeines Verfahren anstatt eines Registerprogramms an. Falls ein $\xi \in \mathbb{A}$ nicht ausschließlich aus mehreren a_0 besteht, so ist es sicherlich keine Gödelnummer. Falls doch, so lässt sich ξ eindeutig in ein Wort aus $\mathbb{B}^*$ zurückübersetzen, welches wiederum in einzelne Zeilen umgeschrieben werden kann, von denen man die Eigenschaften aus Definition 5 überprüfen kann. □

Im Beweis der Unentscheidbarkeit des Halteproblems spielt ein *Diagonalargument* eine entscheidende Rolle. Zwei einfache Beispiele dienen als Aufwärmübung.

Beispiel 7. Sei $M \neq \emptyset$ und $R \subseteq M \times M$ eine Relation. Definiere $D := \{b \in M \mid \neg\, Rbb\}$ und für $a \in M$ sei $M_a := \{b \in M \mid Rab\}$. Dann gilt

$$\forall a \in M \; M_a \neq D$$

Denn ein fixes a ist in M_a genau dann, wenn es in Relation mit sich selbst ist. Dann ist es aber sicher nicht in D. Ist a nicht in Relation mit sich selbst, so ist es nicht in M_a, aber dafür in D, also können die beiden Mengen nicht gleich sein.

Beispiel 8. Sei $M := \mathbb{A}^*$ und R folgende zweistellige Relation auf M: $R\xi\eta \iff \xi$ ist Gödelnummer eines Programms, das eine Menge aufzählt, die η enthält.
Dann ist $D := \{\eta \mid \neg\, R\eta\eta\}$ nicht aufzählbar.
Denn angenommen D wäre aufzählbar. Dann gäbe es ein Programm P, das nach und nach alle η aufzählt, die -falls sie die Gödelnummer eines Programms sind, das eine Menge aufzählt- nicht in dieser aufgezählten Menge sind. Was ist aber mit der Gödelnummer ξ_P von eben diesem Programm? Angenommen P zählt ξ_P auf, dann darf P ξ_P nicht aufzählen. Zählt umgekehrt P ξ_P nicht auf, dann muss ξ_P in der von P aufgezählten Menge sein, ein Widerspruch. Also ist die Menge aller Gödelnummern von Programmen, die nicht ihre eigene Gödelnummer ausgeben, nicht aufzählbar.

Satz 5.2 (Unentscheidbarkeit des Halteproblems)**.**

a) Π'_{stop} *ist nicht R-entscheidbar*

b) Π_{stop} *ist nicht R-entscheidbar*

Beweis. a): Angenommen, Π'_{stop} wäre R-entscheidbar. Dann gäbe es auch ein Programm P_0, das Π'_{stop} entscheidet, für das also

$$\begin{aligned} P_0 : \xi_P \longrightarrow \diamond, \quad falls\ P : \xi_P \longrightarrow \text{stop}, \\ P_0 : \xi_P \longrightarrow \eta \neq \diamond, \quad falls\ P : \xi_P \longrightarrow \infty \end{aligned} \tag{5.1}$$

gilt. Durch leichte Modifikation von P_0 erhalten wir ein neues Programm P_1 das statt im ersten Fall von (5.1) das leere Wort zu drucken in einer Endlosschleife weiter rechnet. Anstelle der finalen STOP Anweisung in Zeile k setze dazu die Zeilen

k IF $R_0 = \diamond$ THEN k ELSE $k+1$ ... ELSE $k+1$
$k+1$ STOP

an das Ende des Programms. Eine zweite kleine Änderung -jede Druckanweisung

wird zu GOTO k umgeschrieben- bewirkt folgendes:

$$P_1 : \xi_P \longrightarrow \infty, \quad falls\ P : \xi_P \longrightarrow \text{stop},$$
$$P_1 : \xi_P \longrightarrow \text{stop}, \quad falls\ P : \xi_P \longrightarrow \infty.$$

Da dies für jedes Programm P gilt, muss es insbesondere für P_1 gelten, was aber zum Widerspruch

$$P_1 : \xi_{P_1} \longrightarrow \infty \iff P_1 : \xi_{P_1} \longrightarrow \text{stop}$$

führt, also kann P_0 nicht existieren und damit ist Π'_{stop} nicht R-entscheidbar.
b): Wir führen wieder einen Widerspruchsbeweis. Angenommen es gäbe ein Programm P_0 das Π_{stop} entscheidet. Das lässt sich folgendermaßen für die Entscheidbarkeit von Π'_{stop} verwenden:

(i). Für ein $\xi \in \mathbb{A}^*$ entscheide zunächst mit Lemma 5.1, ob ξ die Gödelnummer eines Programms ist. Falls nicht, kann es auch nicht in Π'_{stop} sein.

(ii). Stelle das zu ξ gehörende, eindeutige Programm P her. Es gilt also $\xi = \xi_P$.

(iii). Forme P zu $P^\diamond$ um, einem Programm mit der Eigenschaft

$$P : \xi_P \longrightarrow \text{stop} \iff P^\diamond : \diamond \longrightarrow \text{stop}.$$

(iv). Mit P_0 kann nun entschieden werden, ob $P^\diamond$ mit Input $\diamond$ hält, und damit ob P mit Input ξ_P hält, was Teil a) widerspricht.

Um in Punkt (iii) von P zu $P^\diamond$ zu kommen, fügen wir am Anfang von P n Mal die Zeile $R_0 = R_0 + a_0$ ein, wobei n die Länge von ξ_P ist. □

Bemerkung 10. In Umgangssprache bedeutet Teil b): Es gibt kein Programm, das für alle Programme entscheidet, ob sie bei leerem Input halten. Ähnlich wie im Übergang von P zu $P^\diamond$ kann man dieses Ergebnis auf jeden beliebigen Input erweitern. Gemeinsam mit der Church-Turing-These führt das zur Formulierung am Beginn des Kapitels: Es gibt keine (Turing-vollständige) Maschine, die für alle Maschinen immer richtig entscheidet, ob sie für einen bestimmten Input terminieren oder nicht.

6 Unentscheidbarkeit der Logik erster Stufe

In diesem Kapitel, das wieder an Kapitel 10.4 von [2] angelehnt ist, wird das Hauptresultat dieser Arbeit behandelt: Es gibt kein Verfahren, das für jeden S_∞-Satz entscheidet, ob er allgemeingültig ist.

Satz 6.1. *Die Menge* $\Phi := \{\varphi \in L_0^{S_\infty} \mid \models \varphi\}$ *ist nicht R-entscheidbar*

Beweis. Wir führen einen Widerspruchsbeweis. Ziel ist es, aus der Annahme der Entscheidbarkeit der Prädikatenlogik erster Stufe die Entscheidbarkeit des Halteproblems zu folgern, also einen Widerspruch zu Satz 5.2 b). Sei dazu wie in Beispiel 3 $\mathbb{A} = \{|\}$ und somit $\mathbb{A}^* \simeq \mathbb{N}$. Aus einem Entscheidungsverfahren P_0 für Φ erhalten wir ein Entscheidungsverfahren P_1 für Π_{stop}, wenn wir zu jedem Programm P einen S_∞-Satz φ_P konstruieren können, der

$$\models \varphi_P \iff P : \diamond \longrightarrow \text{stop} \tag{6.1}$$

erfüllt. P_1 könnte grob so aussehen:

(i). Für ein $\xi \in \mathbb{A}^*$ entscheide zunächst mit Lemma 5.1, ob ξ die Gödelnummer eines Programms ist. Falls nicht, kann es auch nicht in Π_{stop} sein.

(ii). Stelle das zu ξ gehörende, eindeutige Programm P her. Es gilt also $\xi = \xi_P$.

(iii). Forme aus P den Satz φ_P mit der Eigenschaft 6.1.

(iv). Mit P_0 kann nun entschieden werden, ob $\models \varphi_P$ und damit ob P mit leerem Input hält, Widerspruch zu 5.2 b).

Es bleibt wie im Beweis von 5.2 b) den Punkt (iii) zu konkretisieren, der Übergang von P zu φ_P. Hier ist es aber etwas anspruchsvoller.

Für den Rest des Beweises fixieren wir ein Programm P mit k Zeilen. Wir definieren zuerst die *Konfiguration von P nach s Schritten*, die wir als $K_{P,s}$ schreiben. Ein Schritt ist dabei die Ausführung einer Zeile des Programms und wir gehen immer davon aus, dass P auf das leere Wort angesetzt wurde. $K_{P,s}$ soll Aufschluss darüber geben, was nach s Schritten in jedem der Register steht und welche Zeile als nächstes ausgeführt wird. Sei dazu n die kleinste Zahl, sodass P nur Register im Bereich $R_0, \ldots, R_n$ verwendet. Dann definiere $K_{P,s}$ als das $(n+2)$-Tupel $(i, m_0, \ldots, m_n)$, wobei i die Nummer der nächsten auszuführenden Zeile z_i ist und $m_0, \ldots, m_n$ die Wörter in den jeweiligen Registern. Als Beispiel betrachte das Programm P_{parity} aus Beispiel 3 nach einem Schritt. Es gilt: $K_{P_{parity},1} = (6, \diamond)$, da als nächstes Zeile 6 ausgeführt wird und im ganzen Programm nur das Register R_0 verwendet wird, in welchem zu diesem Zeitpunkt das leere Wort steht.

Da in jedem Registerprogramm mit k Zeilen genau die letzte Zeile eine Stoppanweisung ist, erhalten wir ein Kriterium für das Terminieren eines Programms P:

$$P : \diamond \longrightarrow \text{stop} \iff \exists s \exists m_0 \ldots \exists m_n \; K_{P,s} = (k, m_0, \ldots, m_n). \tag{6.2}$$

Für ein schließlich terminierendes Programm P bezeichne dieses eindeutige s mit s_P.

Unser Ziel ist immer noch, jedes Programm in eine Formel mit der Eigenschaft 6.1 umzuwandeln. Um Formeln schreiben und später auch interpretieren zu können ist erstens eine Sprache $\mathscr{L} \subseteq S_\infty$ und zweitens eine Struktur $\mathfrak{A}$ inklusive Universum A notwendig, um die Zeichen aus $\mathscr{L}$ zu interpretieren. Als Schreibweise für die Interpretation eines Relations-/Funktions-/Konstantenzeichens aus $\mathscr{L}$ in der Struktur $\mathfrak{A}$ benutzen wir $R^{\mathfrak{A}}, f^{\mathfrak{A}}$ bzw. $c^{\mathfrak{A}}$. Wir wählen $\mathscr{L} := \{R_0^{n+3}, R_0^2, f_0^1, c_0\}$ und schreiben aus Gründen der besseren Lesbarkeit $\mathscr{L} := \{\underline{R}, \underline{\leq}, \underline{f}, \underline{0}\}$.[2] Dies spiegelt außerdem die Interpretation durch die Struktur $\mathfrak{A}$ wieder, die wir jedoch abhängig von P definieren müssen. Um das zu kennzeichnen schreiben wir $\mathfrak{A}_P$ bzw. A_P.
Fall 1, $P : \diamond \longrightarrow \infty$: Wähle $A_P = \mathbb{N}$, interpretiere $\underline{\leq}$ als die gewöhnliche kleiner Relation $<$ auf $\mathbb{N}$, $\underline{f}$ als die Nachfolgerfunktion $f : \mathbb{N} \to \mathbb{N},\ f(n) = n+1$ und $\underline{0}$ als $0 \in \mathbb{N}$. Das $n+3$-Tupel $(s, i, m_0, \ldots, m_n) \in \mathbb{N}^{n+3}$ erfüllt die Relation $R := \underline{R}^{\mathfrak{A}_P}$, falls $(i, m_0, \ldots, m_n) = K_{P,s}$.
Fall 2, $P : \diamond \longrightarrow$ *stop*: In diesem Fall brauchen wir nur ein endliches Universum, und zwar bis zur Zahl $e := \max\{k, s_P\}$. Wähle also $A_P = \{0, 1, \ldots, e\}$. Die Relation $<:= \underline{\leq}^{\mathfrak{A}_P}$ und die Funktion $f := \underline{f}^{\mathfrak{A}_\mathfrak{P}}$ bleiben im Vergleich zu Fall 1 beinahe unverändert. Sie sind lediglich auf Elemente von A_P beschränkt, wobei $f(e) = e$ gelten soll. Die Interpretationen von $\underline{0}$ und $\underline{R}$ bleiben gänzlich unverändert. Da in jedem Schritt ein Wort aus einem Register höchstens um ein | erhöht werden kann (was wir mit $+1$ interpretieren), und alle Register zu Beginn des Programms leer sind, können die auftretenden Zahlen bis zum Ende des Programms nicht größer als e werden. R ist also tatsächlich eine Relation auf A_P. Gleiches gilt für jede Zeilennummer i einer Zeile z_i aus dem Programm, $i \leq e$ und daher $i \in A_P$.

Da $\{|\}^* \simeq \mathbb{N}$ schreiben wir der Einfachheit halber auch $\underline{1}$ für $\underline{f}\underline{0}$, $\underline{2}$ für $\underline{f}\underline{f}\underline{0}$ usw. Mit der Struktur $\mathfrak{A}_P$ können wir jede Zeile z_i des Programms P in eine $\mathscr{L}$-Formel ψ_{z_i} umwandeln, die genau den Effekt der Zeile widerspiegelt. Je nach Typ der Zeile, wird ψ_{z_i} anders definiert:

[2] $\underline{\leq}$ ist ein unterstrichenes „kleiner“ Symbol, und *kein* „kleiner oder gleich“ Symbol wie $\leq$.

Fall 1, z_i ist die Verlängerungsanweisung $i\ R_l = R_l + |$:

$$\psi_{z_i} := \forall x \forall y_0 \ldots \forall y_n (\underline{R} x \underline{i} y_0 \ldots y_n \to (x \underline{\leq} \underline{f} x \wedge \underline{R} \underline{f} x \underline{i+1} y_0 \ldots y_{l-1} \underline{f} y_l y_{l+1} \ldots y_n)).$$

Fall 2, z_i ist die Verkürzungsanweisung $i\ R_l = R_l - |$:

$$\begin{aligned}\psi_{z_i} := \forall x \forall y_0 \ldots \forall y_n (\underline{R} x \underline{i} y_0 \ldots y_n \to (x \underline{\leq} \underline{f} x \ \wedge\ ((y_l \doteq \underline{0} \ \wedge\ \underline{R} \underline{f} x \underline{i+1} y_0 \ldots y_n) \vee \\ (\neg y_l \doteq \underline{0} \ \wedge\ \exists u (f u \doteq y_i \ \wedge\ \underline{R} \underline{f} x \underline{i+1} y_0 \ldots y_{l-1} u y_{l+1} \ldots y_n))))).\end{aligned}$$

Fall 3, z_i ist die Sprunganweisung i IF $R_l = \diamond$ THEN z_j ELSE z_{j_0}:

$$\begin{aligned}\psi_{z_i} := \forall x \forall y_0 \ldots \forall y_n (\underline{R} x \underline{i} y_0 \ldots y_n \to (x \underline{\leq} \underline{f} x \ \wedge\ ((y_l \doteq \underline{0} \ \wedge\ \underline{R} \underline{f} x \underline{j} y_0 \ldots y_n) \vee \\ (\neg y_l \doteq \underline{0} \ \wedge\ \underline{R} \underline{f} x \underline{j_0} y_0 \ldots y_n)))).\end{aligned}$$

Fall 4, z_i ist die Druckanweisung i PRINT:

$$\psi_{z_i} := \forall x \forall y_0 \ldots \forall y_n (\underline{R} x \underline{i} y_0 \ldots y_n \to (x \underline{\leq} \underline{f} x \ \wedge\ \underline{R} \underline{f} x \underline{i+1} y_0 \ldots y_n)).$$

Die Stoppanweisung muss nicht in eine Formel übersetzt werden, da sie genau die letzte Zeile des Programms ist. Für sie werden wir das Kriterium 6.2 verwenden.

Jede beliebige Zeile z_i (exklusive der letzten Zeile) eines Programms kann also in eine $\mathscr{L}$-Formel übersetzt werden. Verbinden wir diese Formeln mit $\wedge$, also $\psi_{z_0} \wedge \cdots \wedge \psi_{z_{k-1}}$, dann erhalten wir eine sehr lange Formel, die schon fast das ganze Programm übersetzt. Da wir Programme nur auf $\diamond$ ansetzten, ist $K_{P,0} = (0, \ldots, 0)$, für alle Programme. Also muss in unserer Übersetzungsformel zusätzlich auch $\underline{R0} \ldots \underline{0}$ gelten. Außerdem muss $<$ eine transitive, totale Ordnung mit 0 als kleinstem Element sein, x muss immer kleiner oder gleich fx sein und falls $x \neq e$, dann muss fx die kleinste Zahl sein, die größer als x ist. Diese letzten Forderungen fassen wir in ψ_0 zusammen:

$$\begin{aligned}\psi_0 :=& \forall x \forall y \forall z\ ((x \underline{\leq} y \wedge y \underline{\leq} z) \to x \underline{\leq} z) \ \wedge\ ((x \neq x) \to x \underline{\leq} y \vee y \underline{\leq} x) \wedge \\ & (\neg\ x \underline{\leq} x) \ \wedge\ (\underline{0} \underline{\leq} x \vee \underline{0} \doteq x) \ \wedge\ (x \underline{\leq} \underline{f} x \vee x \doteq \underline{f} x) \wedge \\ & (\exists u\ x \underline{\leq} u \to (x \underline{\leq} \underline{f} x \wedge \forall w (x \underline{\leq} w \to (\underline{f} x \underline{\leq} w \vee \underline{f} x \doteq w)))).\end{aligned}$$

Jetzt können wir eine Formel ψ_P angeben, die das Programm P codiert:

$$\psi_P := \psi_0 \ \wedge\ \underline{R0} \ldots \underline{0} \ \wedge\ \psi_{z_0} \ \wedge\ \ldots \ \wedge\ \psi_{z_{k-1}}.$$

Es gilt:

$$\mathfrak{A}_P \models \psi_P, \quad \textit{sowie} \tag{6.3}$$

$$(\mathfrak{A} \models \psi_P \wedge K_{P,s} = (i, m_0, \ldots, m_n)) \implies \mathfrak{A} \models \underline{R} \underline{s} \underline{i} \underline{m_0} \ldots \underline{m_n}. \tag{6.4}$$

Ersteres folgt direkt aus der Definition von ψ_P. Wir haben $\mathfrak{A}$ und die einzelnen ψ_{z_i} genau so gewählt, dass 6.3 gilt. Für 6.4 wird ein Induktionsbeweis über s geführt:

Beweis. *Induktionsanfang* ($s = 0$): Da P auf $\diamond$ angesetzt wird und ein Programm immer mit Zeile z_0 beginnt, muss $K_{P,0} = (0, \ldots, 0)$ gelten. Laut Voraussetzung gilt $\mathfrak{A} \models \psi_P$, also insbesondere $\mathfrak{A} \models R\underline{0} \ldots \underline{0}$.
Induktionsschluss ($s \to s + 1$): Das Programm laufe ohne Beschränkung der Allgemeinheit mindestens $s + 1$ Schritte. Falls nicht, wähle ein kleineres s; selbst das kürzest mögliche Programm besteht aus der Zeile 0 $STOP$, läuft also einen Schritt. Aus der Induktionsvoraussetzung wissen wir, dass für s die Implikation 6.4 gilt, wir können also annehmen, dass $Rsim_0 \ldots m_n$ gilt. Je nach Befehl in der Zeile z_i werden sich $s, i, m_0, \ldots, m_n$ durch die Ausführung der Zeile im s-ten Schritt verändern. Falls z_i beispielsweise eine Verlängerungsanweisung für das Register R_0 ist, gilt $K_{P,s+1} = (i + 1, m_0 + 1, \ldots, m_n)$. Die durch das Ausführen der Zeile z_i veränderten Werte von $(i, m_0, \ldots, m_n)$ bezeichne mit $(i', m'_0, \ldots, m'_n)$. Was genau das Programm in der Zeile z_i macht, ist aber auch in ψ_{z_i} „abgespeichert“. Da $\mathfrak{A} \models \psi_P$, also insbesondere $\mathfrak{A} \models \psi_{z_i}$, folgt aus $K_{P,s+1} = (i', m_0, \ldots, m_n)$ auch $\mathfrak{A} \models \underline{Rs+1i'}\underline{m'_0} \ldots \underline{m'_n}$, und damit die Induktionsbehauptung. □

Im finalen Schritt setzen wir

$$\varphi_P := \psi_P \to \exists x \exists y_0 \ldots \exists y_n \underline{R}x\underline{k}y_0 \ldots y_n \tag{6.5}$$

und erhalten dadurch den gewünschten Satz φ_P, der 6.1 erfüllt.

Um dies zu überprüfen sei zunächst φ_P allgemeingültig. Also gilt auch $\mathfrak{A}_P \models \varphi_P$. Wegen 6.3 muss daher auch $\mathfrak{A}_P \models \exists x \exists y_0 \ldots \exists y_n \underline{R}x\underline{k}, y_0, \ldots, y_n$ gelten. Es gibt also Zeugen $s, m_0, \ldots, m_n \in A_P$, für die $K_{P,s} = (k, m_0, \ldots, m_n)$ gilt und mit dem Kriterium 6.2 folgt somit $P : \diamond \to \text{stop}$.

Für die Rückrichtung sei P ein schließlich terminierendes Programm. Mit 6.2 folgt die Existenz von $s, m_0, \ldots, m_n$, sodass $K_{P_s} = (k, m_0, \ldots, m_n)$. Sei $\mathfrak{A}$ eine beliebige Struktur mit $\mathfrak{A} \models \psi_P$, dann gilt wegen 6.4 auch $\mathfrak{A} \models \underline{Rsk}\underline{m_0} \ldots \underline{m_n}$. Zusammengefasst ergibt das $\mathfrak{A} \models \varphi_P$. □

Bemerkung 11. Nun können wir ein Beispiel für eine abzählbare aber nicht aufzählbare Menge geben, wie in Bemerkung 2 angekündigt. In Lemma 2.1 wurde gezeigt, dass die Menge aller allgemeingültigen S_∞-Sätze aufzählbar ist. Da sie aber -wie soeben gezeigt- nicht entscheidbar ist, kann wegen Lemma 2.3 das Komplement $\Phi^c := \{\varphi \in L_0^{S_\infty} \mid \not\models \varphi\}$ nicht aufzählbar sein. Die Menge aller Formeln ist jedoch abzählbar, also muss deren Teilmenge Φ^c ebenfalls abzählbar sein.

7 Literatur

Literatur

[1] Alexander Asteroth und Christel Baier. *Theoretische Informatik*. 1. Aufl. München: Pearson Studium, 2002. URL: https://www.pearson-studium.de/theoretische-informatik.html.

[2] Heinz-Dieter Ebbinghaus, Wolfgang Thomas und Dieter Flum. *Einführung in die mathematische Logik*. 6. Aufl. Berlin: Springer Spektrum, 2018. URL: https://link.springer.com/book/10.1007%2F978-3-662-58029-5.

[3] David Hilbert und Wilhelm Ackermann. *Grundzüge der theoretischen Logik*. 1. Aufl. Berlin: Springer, 1928.

[4] Dirk W Hoffmann. *Grenzen der Mathematik*. 3. Aufl. Berlin: Springer Spektrum, 2018.

[5] Alan Turing. *On computable numbers with an application to the Entscheidungsproblem*. 1936.