Prof. Dr. H. Brenner **Osnabrück SS 2014**

Einführung in die mathematische Logik

Vorlesung 19

Entscheidbarkeit und Berechenbarkeit

In der letzten Vorlesung haben wir verschiedene mathematische Operationen (wie Addition und Multiplikation) durch Registerprogramme berechnet und ebenso mathematische Prädikate (etwa das Prädikat, eine Primzahl zu sein) durch Registerprogramme charakterisiert. Die Fähigkeit eines Registerprogramms, bestimmte Funktionen bzw. Prädikate zu berechnen bzw. zu charakterisieren, führt zu den folgenden Begriffen.

Definition 19.1. Eine k-stellige Funktion

$$F\colon \mathbb{N}^k \longrightarrow \mathbb{N}$$

heißt *R-berechenbar* (oder *Register-berechenbar*), wenn es ein Programm P für eine Registermaschine gibt, die bei jeder Eingabe $(r_1, \ldots, r_k)$ (in den ersten k Registern) anhält und $F(r_1, \ldots, r_k)$ als (einzige) Ausgabe besitzt.

Definition 19.2. Es sei $T \subseteq \mathbb{N}$ eine Teilmenge der natürlichen Zahlen. Man sagt, dass diese Menge *R-entscheidbar* (oder *Register-entscheidbar*) ist, wenn es ein Programm P für eine Registermaschine gibt, die bei jeder Eingabe anhält und für die die Äquivalenz

$$n \in T \text{ genau dann, wenn } P(n) \text{ die Ausgabe } 0 \text{ besitzt}$$

gilt.

Eine Teilmenge $T \subseteq \mathbb{N}$ ist genau dann R-entscheidbar, wenn die zugehörige Indikatorfunktion R-berechenbar ist.

Die Churchsche These

Wir haben in der letzten Vorlesung für einige recht einfache Aufgaben Registerprogramme angegeben, die diese Aufgabe lösen. Diese Beispiele vermitteln eine Vorstellung davon, was alles mit Registermaschinen berechnet werden kann. Zur Tragweite von algorithmischer Berechenbarkeit überhaupt ist die sogenannte *Churchsche These* von Bedeutung.

Bemerkung 19.3. Die *Churchsche These* (nach Alonzo Church, manchmal auch *Church-Turing-These*) behauptet, dass die intuitiv berechenbaren Funktionen (bzw. die intuitiv entscheidbaren Prädikate) mit den Registerberechenbaren Funktionen übereinstimmt. Da es sich bei „intuitiv berechenbar“ um einen nicht präzisen Begriff handelt, lässt sich diese These nicht

beweisen. Sie ist dennoch weitgehend akzeptiert, wobei die folgenden Gründe angeführt werden.

Alle Präzisierungen des Berechenbarkeitsbegriffs, nämlich durch Registermaschine, Turingmaschine, primitiv-rekursive Funktionen, λ-Kalkül, führen zu einer übereinstimmenden Klasse von berechenbaren Funktionen. Dies beruht darauf, dass man die algorithmischen Verfahren wechselseitig simulieren kann.

Konkrete, intuitiv berechenbare Funktionen lassen sich stets durch ein Registerprogramm realisieren.

In der Praxis ist die Churchsche These vor allem eine Erleichterung, da man aufgrund eines häufig naheliegenden intuitiven Algorithmus auf die Existenz eines Registerprogramms schließen kann, und so die oft mühevolle „Programmier-Arbeit“ umgeht.

Das Halteproblem

Nicht jedes Programm hält an. Ein einfaches Beispiel mit zwei Registern R_1, R_2 und leerer Belegung für R_2 ist

(1) $1+$
(2) $C(2,1)$
(3) Halte an

Im Allgemeinen wird es sehr schnell schwierig, zu einem gegebenen konkreten Programm zur Eingabe $r_1 = 0$ zu entscheiden, ob es den Haltebefehl schließlich erreicht oder nicht. Ebenso ist es schwierig zu entscheiden, für welche Eingabedaten in R_1 (den *Input*) das konkrete Programm stoppt.

Ein qualitativ anderes Problem ist allerdings die Frage, ob es ein Verfahren gibt, mit dem man für jedes Programm (bzw. jedes Programm und jede Eingabe) entscheiden kann, ob es anhält oder nicht.

Hier deutet sich eine selbstbezügliche Fragestellung an: Gibt es ein Programm, das Aussagen über alle Programme machen kann? Welche Aussage macht dann dieses Programm über sich selbst?

Um einen solchen Ansatz präzise machen zu können, müssen wir Programme als Eingabe für ein Programm interpretieren können. Das Programm einer Registermaschine erlaubt nur die Eingabe einer Zahl. Daher müssen wir ein Programm durch eine Zahl kodieren. Dies geschieht in zwei Schritten.

Zuerst führen wir für die erlaubten Befehle abkürzende Schreibweisen ein. Wir arbeiten mit dem Alphabet

$$\prime - \ I\ D\ C\ P\ H\ ,$$

Die einzelnen Befehle werden folgendermaßen notiert

(1) Inkrementierung von R_i
    $I\prime\prime\cdots\prime$ (mit i Strichen).
(2) Dekrementierung von R_i
    $D\prime\prime\cdots\prime$ (mit i Strichen).
(3) Sprunganweisung $C(i,j)$
    $C\prime\prime\cdots\prime,\prime\prime\cdots\prime$ (mit i Strichen vor dem Komma und j Strichen nach dem Komma).
(4) Druckanweisung: P.
(5) Halteanweisung: H.

Das Symbol $\prime$ wird also benutzt, um sowohl die Registernummern als auch die Zeilennummern (in der Sprunganweisung) auszudrücken. Da in jeder Befehlszeile eines konkreten Programmes konkrete Register bzw. Zeilen adressiert werden, stehen da jeweils natürliche Zahlen (keine Variablen), die problemlos durch eine Strichfolge ausgedrückt werden können.

Ein Programm, das aus den durchnummerierten Befehlszeilen $B_1, B_2, \ldots, B_h$ besteht, wird dann insgesamt durch die Zeichenfolge

$$b_1 - b_2 - \ldots - b_h$$

wiedergegeben, wobei die b_j die soeben angeführte Kodierung der j-ten Befehlszeile ist. Das Zeichen $-$ wird also verwendet, um die Zeilen voneinander zu trennen. Das Mitschleppen der Zeilennummern ist nicht nötig, da sich die Nummer aus der Reihenfolge rekonstruieren lässt.

Das oben angegebene Programm hätte demnach die symbolische Kodierung

$$I\prime - C\prime\prime,\prime - H$$

In einem zweiten Schritt ersetzen wir diese symbolische Kodierung durch eine numerische Kodierung. Dafür gibt es verschiedene Möglichkeiten. Da unser Alphabet, mit dem wir jedes Programm schreiben können, 8 Symbole verwendet, liegt eine Repräsentierung im Achtersystem nahe. Da die 0 als Anfangsnummer etwas problematisch ist, arbeiten wir lieber im Neunersystem (man kann die folgenden Zahlen genau so gut im Zehnersystem auffassen) und ordnen den Symbolen von oben in der obigen Reihenfolge die Ziffern

$$1, 2, 3, 4, 5, 6, 7, 8$$

zu. Das Programm von oben wird dann zur Ziffernfolge

$$3125118127\,.$$

Die einem jeden Programm P auf diese Weise zugeordnete Zahl (also der Zahlwert, nicht die Ziffernfolge) nennen wir $c(P)$. Man spricht von der *Gödelnummer* des Programms.

Bemerkung 19.4. Es ist algorithmisch überprüfbar, ob eine als Strichfolge gegebene natürliche Zahl ein Code für ein Registerprogramm ist. Dazu muss zuerst die Zahl in ihre Ziffernentwicklung (im Neunersystem) übersetzt

werden. Da der Trennstrich, der die einzelnen Befehle trennt, durch eine bestimmte Ziffer codiert wird, muss die Ziffernfolge zwischen zwei Trennstrichziffern einen Befehl codieren. Die syntaktische Korrektheit dieser einzelnen Befehlsziffernfolgen muss dann der Reihe nach überprüft werden. Dazu muss man für jeden der Einzelbefehle einen Algorithmus entwerfen. Wenn beispielsweise die Anfangsziffer einer Befehlsziffernfolge eine 3 (also ein I) ist, so muss es sich um einen Inkrementierungsbefehl handeln und alle nachfolgenden Ziffern (bis zum nächsten Trennstrich) müssen eine 1 sein.

Für ein Registerprogramm P und eine natürliche Zahl n verstehen wir unter $P(n)$ das Programm angesetzt auf n im ersten Register (und leeren anderen Registern).

Lemma 19.5. *Die Menge*

$$\{n \in \mathbb{N} |\, n \textit{ ist die Nummer eines Registerprogramms } P \textit{ und } P(n) \textit{ hält an}\}$$

ist nicht R-entscheidbar.

Beweis. Wir nehmen an, dass es ein Programm U gibt, das diese Menge entscheidet (der erste Teilaspekt, ob es sich überhaupt um ein valides Programm handelt, ist entscheidbar). Wir ändern dieses Programm ab zu einem Programm U', indem wir den letzten Befehl von U (also den Haltebefehl) durch den Programmabschnitt (mit der relativen Nummerierung und einem neuen Register R_i)

(1) $C(1,3)$
(2) Gehe zu 5
(3) $i+$
(4) $C(1,3)$
(5) Halte an

ersetzen. Dies bedeutet, dass das Programm U' genau dann in eine Endlosschleife hineinkommt und nicht anhält, wenn das Programm U die Ausgabe 0 hat. Daher gilt die Äquivalenz, dass ein Programm P bei Eingabe der eigenen Programmnummer $c(P)$ genau dann anhält, wenn U' bei Eingabe der Programmnummer $c(P)$ von P nicht anhält. Diese Äquivalenz ergibt bei Anwendung auf das Programm $P = U'$ einen Widerspruch. □

Satz 19.6. *Die Menge*

$$\{n \in \mathbb{N} |\, n \textit{ ist die Nummer eines Registerprogramms } P \textit{ und } P(0) \textit{ hält an}\}$$

ist nicht R-entscheidbar.

Beweis. Wir nehmen an, dass es ein Registerprogramm V gibt, dass die in Frage stehende Menge entscheidet, also stets anhält und angesetzt auf eine Zahl n genau dann die Ausgabe 0 liefert, wenn $n = c(P)$ für ein Programm P ist (also wenn n die Programmnummer eines Registerprogramms ist) und wenn dieses Programm P, angesetzt auf 0, anhält. Wir entwickeln aus V

ein Programm U, das genau dann die Ausgabe 0 hat, wenn $n = c(P)$ für ein Programm P ist und wenn P, angesetzt auf n, anhält. Dies ergibt einen Widerspruch zu Lemma 19.5.

Dazu wird U folgendermaßen konstruiert: Wenn n keine Programmnummer ist, so hält das Programm U mit der Ausgabe 1 an (hier gibt es also keinen Unterschied zu V). Wenn $n = c(P)$ eine Programmnummer ist, so wird das Programm P' aufgestellt, das dem Programm P die n-fache Inkrementierung des ersten Registers voranstellt und dessen (in einem bedingten Sprungbefehl in einer Befehlszeile) adressierten Befehlszeilennummern sich um n erhöhen. Für die Programmnummer $n' = C(P')$ wird nun mittels V überprüft, welche Ausgabe P', angesetzt auf 0, besitzt. Aufgrund der Konstruktion von P' besitzt P' bei Eingabe 0 die Ausgabe 0 genau dann, wenn P bei Eingabe von n die Ausgabe 0 besitzt. □

Aufzählbarkeit von Programmen

Wir führen einen weiteren Berechenbarkeitsbegriff ein.

Definition 19.7. Es sei $T \subseteq \mathbb{N}$ eine Teilmenge der natürlichen Zahlen. Man sagt, dass diese Menge *R-aufzählbar* (oder *Register-aufzählbar*) ist, wenn es ein Programm P für eine Registermaschine gibt, die bei Eingabe von 0 nach und nach genau die Zahlen aus T ausdruckt (dabei dürfen Zahlen aus T auch mehrfach ausgedruckt werden).

Zwischen Entscheidbarkeit und Aufzählbarkeit besteht der folgende Zusammenhang.

Lemma 19.8. *Es sei $T \subseteq \mathbb{N}$ eine Teilmenge von natürlichen Zahlen. Dann ist T genau dann R-entscheidbar, wenn sowohl T als auch das Komplement $\mathbb{N} \setminus T$ R-aufzählbar ist.*

Beweis. Wenn P ein Programm ist, das T entscheidet, so kann man einfach ein T aufzählendes Programm konstruieren. Man lässt der Reihe nach jede natürliche Zahl mittels P auf ihre Zugehörigkeit zu T überprüfen und druckt sie aus, falls sie dazu gehört (dazu muss man den Haltebefehl von P zu einer Druckausgabe modifizieren). Entsprechend konstruiert man ein Aufzählungsprogramm für das Komplement.

Es seien nun T als auch $\mathbb{N} \setminus T$ aufzählbar, und es seien P und Q Programme, die dies leisten. Dann liefert die folgende Kombination der beiden Programme ein Entscheidungsverfahren: Man schreibt die Programme P und Q hintereinander (wobei man natürlich die adressierten Register und Programmzeilen umnummerieren muss) und lässt sie abwechselnd bis zu einer Druckausgabe laufen. Sobald eine Druckausgabe eines Programmteils mit der zu überprüfenden Zahl n übereinstimmt, weiß man, ob n zu T gehört oder

nicht. Da n entweder zu T oder zum Komplement gehört, muss einer dieser Fälle eintreten. □

LEMMA 19.9. *Die Menge der Programmnummern von Register-Programmen, die angesetzt auf* 0 *anhalten, ist R-aufzählbar.*

Beweis. Die Idee für ein algorithmisches Aufzählverfahren geht so: Zu jeder natürlichen Zahl n berechnet man sämtliche Programme P mit $c(P) \leq n$. Jedes dieser Programme lässt man, angesetzt auf 0, n Schritte (also n Befehlszeilenwechsel) lang laufen. Wenn P anhält, so druckt man $c(P)$ aus. Wenn all diese Programme n Schritte gelaufen sind, so erhöht man auf $n+1$. □

Daraus folgt insbesondere, dass die nicht haltenden Programme nicht aufzählbar sind.

Abbildungsverzeichnis