(Noch offen: 7.4)

Einheit 8: Rekursive Funktionen

Church-Turing These. Dieser Abschnitt dient zu ihrer Information und Unterhaltung.

Für die späteren Übungen brauchen sie aus diesem Abschnitt nur:

- die Notation "berechenbar" := "goto-berechenbare partielle Funktion",
- die Versicherug dass "berechenbar" genau "berechenbar in Ihrer Lieblings-Programmiersprache" bedeutet (Church-Turing-These).

In Abschnitt 3.1 und 3.2 des Skriptums werden rekursive Funktionen und goto-berechenbare (partielle) Funktionen definiert (und gezeigt dass sie gleich sind).

Es stellt sich heraus dass dieser Begriff enorm robust ist: Alle üblichen Begriffe von "Computer", "Programmiersprache" etc führen immer zum selben Begriff der berechenbaren Funktion. Die goto-berechenbaren partiellen Funktionen sind genau diejenigen die durch eine Turingmaschine berechenbar sind, durch eine Registermaschine, mit einem C-Programm (oder Assembler, Basic, Pascal, Lisp, Prolog, Ruby, Perl, Java etc etc). Und dieser Begriff entspricht auch genau dem des "(mathematischen) Algorithmus" (der ja viel älter ist als Computermodelle): Ein mathematisches Problem lässt sich "algorithmisch" (durch eine endliche, "mechanische" Rechenvorschrift) lösen genau dann wenn das Problem durch einen Computer gelöst werden kann.

Manchmal wird diese Behauptung auch Church-Turing-These genannt: "Die berechenbaren Funktionen sind genau die goto-berechenbaren Funktionen." Das ist natürlich kein Theorem (weil "berechenbar" nicht definiert ist). Es ist auch nicht (nur) eine Definition von "berechenbar", vielmehr ist es eben die (informelle) Behauptung dass sämtliche sinnvollen Begriffe von berechenbar äquivalent sind.

Wir werden daher in diesem Übungsblatt auch einfach "berechenbare Funktion" statt "goto-berechenbare partielle Funktion" sagen.

Vorerst ein paar Bemerkungen, die die behauptete Universalität des Berechenbarkeitsbegriffs etwas relativieren:

- Es gibt natürlich Programmiersprachen bzw. Computermodelle die schwächer sind als der universelle Berechenbarkeitsbegriff. Wir haben ja schon den Begriff "primitiv rekursiv" kennengelernt, daneben gibt es zB regular expressions, kontextfreie Grammatiken, endliche Automaten, etc.
- Es gibt natürlich auch (theoretische) Computermodelle, die stärker sind: Z.B. kann man einfach eine unberechenbare funktion g als zusätzliche Grundfunktion in der goto-Sprache verwenden.

  Das führt zu wichtigen Begriffen der Berechenbarkeitstheorie, insbesondere zur Ordnung der Turing-Grade: $g \geq f$ (sprich: die Funktion g ist zumindest so kompliziert wie f), wenn f berechnet werden kann wenn g als Grundfunktion zugelassen wird.

  Aber solche Computermodelle haben keine physikaliche Entsprechung, es gibt keine Vorschläge wie eine physikalische Maschine aussehen könnte die nicht-rekursive Funktionen berechnet.
- Apropos real existierende physikalische Systeme: Sämtliche real existierenden Computer sind natürlich im Unterschied zu unserem idealisierten Modell endlich: Der Speicher ist endlich, die maximale sinnvolle Berechnungszeit ist auch begrenzt (z.B. durch das (vermutliche) Ende des Universums) etc.

  Das sind in der Praxis enorm wichtige Einschränkungen, die aber mathematisch sehr schwer zu fassen sind.

  Zum Beispiel: Als MathematikerIn wird man ja gerne glauben dass die Addition (zweier beliebiger natürlicher Zahlen) algorithmisch (bzw. mit einem Computer) durchführbar ist. Aber jeder endliche Computer kann natürlich nur mit Zahlen bis zu einer gewissen Grösse umgehen; wenn man also die physikalische Endlichkeit mit einbezieht, müsste man eigentlich sagen: Die Addition ist bis zu einer gewissen (nicht spezifizierten) endlichen Schranke berechenbar. Das ist es aber natürlich auch nicht was man meint (endlich viele Werte kann ich ja von jeder beliebigen Funktion berechnen, durch Fallunterscheidung).

Es wird also alles kompliziert und unbefriedigend, daher beschäftigen wir uns lieber mit dem idealisierten und unrealistischen Computermodell wo wir beliebig viel Speicher und Zeit zur Verfügung haben (aber in jeder konkreten erfolgreichen Berechnung natürlich nur endlich viel davon verwenden).

- Weil es wichtig ist will ich den letzten Punkt nochmals wiederholen: Die Aussage "f ist berechenbar" heisst nicht unbedingt, dass f in irgendeinem sinnvollen praktischen Sinn berechenbar ist (es könnte ja sein dass es für f nur ein Programm gibt dass auf jedem Input länger rechnet als das Universum exisiteren wird).

  Umgekehrt ist "f ist nicht berechenbar" eine recht starke Aussage: nicht einmal mit dem idealisierten potentiell unendlichen Computer kann f berechnet werden.

  (Die Praxis ist aber hier, wie immer, kompliziert: Es könnte aber selbst dann noch passieren dass es ein Programm gibt das f für "fast alle relevanten Inputs" berechnet; oder f "mit in der Praxis ausreichender Genauigkeit berechnet" etc. Die Praxis ist weniger klar als das theoretische Modell.)

  Für die praktische Berechenbarkeit macht es jedenfalls keinen Unterschied ob man beweist: "f ist unberechenbar" oder "f ist zwar berechenbar, aber nur in exponentieller Laufzeit".

Aufgabe 8.1 Partielle Funktionen. (Skriptum-Übungsaufgabe 8.1). Wir betrachten partielle Funktionen, d.h. Funktionen deren Definitionsbereich nur Teilmenge von $\mathbb{N}^n$ ist (oder äquivalent: für die wir einen zusätzlichen Funktionswert, "undefiniert", zulassen). Wie muss die Definition von "Verknüpfung von Funktionen", "primitiver Rekursion" und "μ-Rekursion" auf partielle Funktionen erweitert werden, damit die so definierten rekursiven Funktionen (d.h. Grundfunktionen, abgeschlossen unter Einsetzung, primitiver und μ-Rekursion) genau die goto-berechenbaren partiellen Funktionen sind?

Hinweis: Das führt auch zu seltsamen Effekten; wenn zB f die (berechenbare) konstante Null-Funktion ist, dann ist $f \circ g$ im allgemeinen nicht konstant Null. Wenn zB g nirgends definiert ist, dann auch $f \circ g$.

Aufgabe 8.2 Das universelle Programm. Im Abschnitt 3.2 des Skriptums wird jedem goto-Programm P ein code (=source code) $\mathcal{P} \in \mathbb{N}$ zugeordnet. Zeige dass es ein universelles goto-Programm gibt: Ein Programm U dass auf Input $(\mathcal{P}, n)$ den output liefert, den das Programm P auf input n liefern würde. (Wenn P auf input n nicht terminiert, dann terminiert auch U auf $(\mathcal{P}, n)$ nicht; wenn $\mathcal{P}$ kein gültiger source code ist, dann ist es uns egal was U auf $(\mathcal{P}, n)$ liefert.)

Bemerkung: dieses Programm U ist nichts anderes als ein goto-Interpreter (der in der goto-Programmiersprache implementiert ist). Genauso gibt es natürlich Java-Interpreter die in Java geschrieben sind etc.

Aufgabe 8.3 Nochmals Diagonalisierung. Sei $f(n)$ der output vom universellen Programm U auf input (n, n).

- Ist f rekursiv?
- Sei $g(n) = f(n) + 1$. Ist g rekursiv? Gibt es ein goto-Programm für g? Wenn ja, sei $\mathcal{P}$ der source code dieses Programms. Was liefert U auf input $(\mathcal{P}, \mathcal{P})$?
- Sei nun $h(n) = f(n) + 1$ falls $f(n)$ definiert ist, und ansonsten 17. Ist h rekursiv?

Bemerkung: Das zeigt, dass das sogenannte Halteproblem (hält U auf input (n, n)?) nicht rekursiv ist. Das wird man, in etwas anderer Formulierung, auch in Abschnitt 3.3 sehen.

Codierung. Rekursive Funktionen sind (partielle) Funktionen von $\mathbb{N}^n$ nach $\mathbb{N}$. Durch die üblichen Codierungen kann man auch von der Berechenbarkeit von Funktionen mit anderen Definitions- oder Wertebereichen sprechen. Im Kapitel 2.3 des Vorlesungsskripts wird das zB für Terme und Formeln der Prädikatenlogik gemacht; so kann man dann darüber sprechen ob zB Funktionen von Terme nach Termen berechenbar sind; oder ob eine Eigenschaft (=Teilmenge) von Sätzen rekursiv ist etc.

Auf ähnliche Weise kann man zB rationale Zahlen kodieren; oder Polynome mit ganzzahligen (oder rationalen) Koeffizienten, etc.

Im Folgenden wird stillschweigend vorausgesetzt dass irgendeine “sinnvolle” Codierung gewählt wurde. Verwenden Sie zur Lösung der folgenden Beispiele die Church-Turing-These (mit anderen Worten: argumentieren Sie informell dass die Probleme algorithmisch lösbar sind, und verweisen Sie darauf dass sie deswegen rekursiv bzw goto-berechenbar sind).

Aufgabe 8.4.

- Argumentiere: Die Funktion f die das paar (n, m) auf den größten gemeinsamen Teiler von n und m abbildet ist rekursiv.
- Wir betrachten die Menge der Polynome in einer Variablen mit ganzzahligen Koeffizienten. Argumentiere: Die Teilmenge derjenigen Polynome, die eine ganzzahlige Nullstelle haben, ist rekursiv.
- Argumentiere: Die Addition und Multiplikation rationaler Zahlen sind rekursive Funktionen. Was ist mit der Exponentiation zweier rationaler Zahlen?