Bemerkungen/Lösungen zu Aufgabe 2.2

Einige Bemerkungen zu Blatt 2 (vieles wird hier nur vage angedeutet und wird im Kapitel 3 der Vorlesung klarer werden).

Aufgabe:

- Kann man die LOOP Programme als $P_0, P_1, \ldots$ aufzählen? Folgere: Es gibt nicht LOOP-berechenbare Funktionen.

Antwort: Es gibt nur abzählbar viele LOOP Programme, daher kann man sie offenbar als $P_0, P_1, \ldots$ aufzählen. Da es überabzählbar viele Funktionen gibt, muss es auch nicht-LOOP-berechenbare Funktionen geben.

Bem: Genau dasselbe Argument funktioniert auch für alle anderen "sinnvollen" Berechenbarkeitsbegriffe. Der wesentliche Punkt ist dass Programme endliche Zeichenketten (aus einem endlichen oder abzählbaren Alphabet) sind.

Bem: Läßt man unendliche Programme zu, dann bekommt man keinen sinnvollen Berechenbarkeitsbegriff: Jede Funktion f ist dann berechenbar, durch das folgende Programm P_f: Wenn input $= 0$, dann output $= f(0)$, ...

Aufgabe:

- Kann man die LOOP Programme "effektiv", d.h., "mit einem Computer" aufzählen? Genauer: Gibt es so eine Aufzählung P_n und ein Computerprogramm M, das auf Input n, m den Output $P_n(m)$ liefert (d.h., den output, den das n-ten Loop Programms auf Input m liefert)? Kann es ein entsprechendes Loop Programm M geben mit $M(n, m) = P_n(m)$?

Antwort:

- Ja, man kann die LOOP Programme "effektiv", d.h. mit einem Algorithmus, aufzählen: Gegeben ein Programm P und einen inptut x kann man einfach durch Ausrechnen simulieren, was das Programm P auf input x für eien Output liefert. (Wichtig ist dabei dass ein LOOP Programm immer nach endlich vielen Rechenschritten hält, oder: "terminiert".) Natürlich haben wir noch nicht definiert, was ein allgemeines Computerprogramm bzw ein Algorithmus ist, daher ist diese Behauptung etwas vage.
- Nein, dieses "Simulationsprogramm" (nichts anderes als ein sogenannter "Interpreter", in der Berechenbarkeitstheorie auch "universelles Programm" genannt) kann selbst kein LOOP Programm sein, weil man ansonsten das Cantorsche Diagonalisierungsargument verwenden könnte um einen Widerspruch zu erzeugen.
- In der Sprache der "angewandten Informatik": Natürlich kann man einen LOOP Interpreter schreiben, z.B. in C oder Java oder Basic etc (auf der homepage von Hans Adler finden Sie zB einen Javascript interpreter), aber man kann keinen LOOP interpreter in der LOOP Sprache schreiben.

Aufgabe:

- Der vorherige Punkt impliziert, dass Loop Programme nicht den allgemeinen Begriff der Berechenbarkeit umfassen: $P_n(m)$ ist berechenbar, aber nicht Loop berechenbar. Folgt allgemein, dass es gar keinen formalen "universellen Begriff der Berechenbarkeit" geben kann (so wie das Richard Paradoxon zeigt, dass es keinen formalen "universellen Begriff der Definierbarkeit" geben kann)?

Antwort:

- Das ist etwas diffizil. Es gibt sehr wohl einen natürlichen, universellen Begriff der Berechenbarkeit, "rekursiv" genannt; bzw ein universelles mathematisches Modell eines (idealisierten) Computers. Dieser Begriff spricht allerdings über "partielle Funktionen", d.h. Funktionen die nicht auf jedem Input definiert sind. Anders formuliert: Ein Computerprogramm terminiert i.A. nicht auf jedem Input (anders als ein LOOP Programm). Das ist kein Defekt der Programmiersprache, sondern unvermeidbar: Für den universellen Begriff der Berechenbarkeit gibt es ein universelles Programm, d.h. eine berechenbare Funktion $f(x, y)$ so dass es zu jeder anderen berechenbaren Funktion $g(y)$ eine Zahl e (den "source code") gibt so dass $g(y) = f(e, y)$ für alle y. Das Cantorsche Diagonalisierungsargument

liefert keinen Widerspruch: Die Funktion $g(x) = f(x,x)+1$ ist rekursiv, also von der Form $f(e,x)$ für ein x; damit ist also $f(e,e) = g(e) = f(e,e)+1$. Das sieht zwar wie ein Widerspruch aus, ist aber keiner: $f(e,e)$ ist in diesem Fall schlichtwegs (zwingendermassen) "undefiniert", d.h. das Computerprogramm M terminiert nicht, und das Computerprogramm "führe erst M aus und zähle dann 1 zum output dazu" terminiert dementsprechend ebenfalls nicht, und die Gleichung $f(e,e) = f(e,e)+1$ besagt nur undefiniert = undefiniert.

- Diese Überlegung zeigt auch, dass das Halteproblem nicht durch einen Computer entschieden werden kann: Könnte man rekursiv feststellen ob $f(x,x)$ terminiert oder nicht, dann könnte man $g(x)$ definieren als $f(x,x)+1$ falls $f(x,x)$ terminiert, und als 0 sonst. Dieses g würde tatsächlich einen Widerspruch bewirken.
- In der Sprache der "praktischen Informatik": Eine Funktion ist rekursiv genau dann wenn sie durch ein C-Programm berechnet werden kann, und dasselbe gilt für jede andere ("normale") Programmiersprache (Perl, Java, Lisp, was immer Sie wollen). Man kann nun in C einen C-Interpreter schreiben (d.h. ein Programm das als Input einen source code A und eine Zahl x verwendet und als output den output von A auf input x liefert.) (Zur Erinnerung: In der LOOP Sprache kann man keinen LOOP Interpreter schreiben, wie wir oben gesehen haben.)