Programmatische Berechenbarkeit von Modulo und ganzzahliger Division

Alexander Köster

10. August 2018

1 Grundlagen

Die ganzen Zahlen sind ein *Euklidischer Ring*[1], d.h. es gibt eine Division mit Rest.
Das heißt: Für ganze Zahlen $n \in \mathbb{Z}$, $m \in \mathbb{Z} \setminus \{0\}$ gibt es $q, r \in \mathbb{Z}$, sodass $n = q \cdot m + r$ und $|r| < |m|$.
r heißt **Rest** und q heißt **(ganzzahliger) Quotient**.
Man definiert $n \bmod m := r$ (**Modulo**) und $n \operatorname{div} m := q$ (**ganzzahlige Division**).

Für unsere Anwendungen in der theoretischen Informatik schränken wir diese ein auf die Teilmenge der natürlichen Zahlen $\mathbb{N}$, da mit dieser Menge üblicherweise Berechenbarkeiten definiert werden. In den natürlichen Zahlen existiert diese Division mit Rest ebenso[2].

Wie man mit den klassischen Rechenoperationen Modulo und ganzzahlige Division berechnen kann, stellt man z.B. gut mit der Methode der schriftlichen Division mit Rest[3] dar.
Eine übliche schriftliche Division könnte folgendermaßen aussehen:

```
 257 : 12 = 21 R 5
-24
---
 17
-12
---
  5
```

Ganz primitiv betrachtet, zieht man hier von der 257 so oft 12 ab, bis eine Zahl kleiner als 12 übrig bleibt. Diese ist dann der Rest, und die Anzahl der Schritte, die man gebraucht hat, der ganzzahlige Quotient.

2 Turing-Berechenbarkeit

Die Turing-Berechenbarkeit ergibt sich somit fast von selbst. Mit dem oben beschriebenen Algorithmus kann man sehr leicht ein WHILE-Programm angeben, das schriftliche Division durchführt; und WHILE-Berechenbarkeit ist äquivalent zu Turing-Berechenbarkeit.

[1] ein kommutativer Ring R mit Eins mit einer Abbildung $|\cdot| : R \to \mathbb{N}_0$, sodass für alle $a, b \in R$ gilt:

- $|a| = 0 \Leftrightarrow a = 0$
- $|ab| = |a||b|$
- $\exists q \in R,\ r \in R \setminus \{0\} : a = qb + r \wedge |r| < |b|$

[2] beweisbar durch vollständige Induktion (Existenz) und Superposition (Eindeutigkeit).

[3] `https://de.wikiversity.org/wiki/Schriftliche_Division/Eigenschaften/Fakt`

Ein simples WHILE-Programm wäre zum Beispiel:

```
x_1 := n ∈ ℕ; x_2 := m ∈ ℕ \ {0}; (Eingabe)
x_3 := 0;
WHILE x_1 ≠ 0 DO
    x_4 := x_1;
    x_1 := x_1 − x_2;
    x_3 := x_3 + 1;
END;
x_3 := x_3 − 1;
x_1 := x_2 − x_4;
IF x_1 = 0 THEN
    x_4 := 0;
    x_3 := x_3 + 1;
END
x_1 := x_4;
x_1 =: r; x_3 =: q (Ausgabe, d.h. Ergebnis für n mod m in x_1 und für n div m in x_3)
```

Wenn man beide Ergebnisse benötigt, ist das gleichzeitige Berechnen am effizientesten. Benötigt man **nur Modulo**, kann man das Zählen aus dem Algorithmus streichen:

```
x_1 := n ∈ ℕ; x_2 := m ∈ ℕ \ {0}; (Eingabe)
WHILE x_1 ≠ 0 DO
    x_3 := x_1;
    x_1 := x_1 − x_2;
END;
x_1 := x_3;
x_3 := x_2 − x_1;
IF x_3 = 0 THEN
    x_1 := 0;
END
x_1 =: r (Ausgabe, Ergebnis für n mod m in x_1)
```

Wenn man den ganzzahligen Quotienten braucht, vereinfacht sich der Algorithmus nicht. Man schreibt dann im allgemeinen WHILE-Programm oben am Ende bloß $x_1 := x_3$ statt $x_1 := x_4$ und erhält q in x_1.

Ist $m = 0$, so würden diese WHILE-Programme nie terminieren und so korrekterweise „undefiniert“ sein.

3 LOOP-Berechenbarkeit

Verzichtet man darauf, bei der Eingabe von $m = 0$ ein sinnvolles Verhalten zu bekommen, kann man Modulo und ganzzahlige Division auch leicht in ein LOOP-Programm überführen. Damit ist diese Form von Modulo und ganzzahliger Division sogar im schwächeren Fall als Turing-Berechenbarkeit, nämlich der LOOP-Berechenbarkeit oder primitiv rekursiven Berechenbarkeit[4].

Hierzu verwendet man z.B. die Annahme, dass bei der iterativen Subtraktion einer positiven Zahl m von n maximal m Schritte benötigt werden, um n auf 0 zu reduzieren, nämlich dann, wenn $m = 1$. Ist $m > 1$, so werden weniger Schritte benötigt, da n sich schneller der 0 nähert.

Auch wenn dadurch bei großen m deutlich zu viele Rechenschritte verwendet werden und der Algorithmus hier nicht mehr sehr effizient ist, kann man den gleichen Algorithmus benutzen (LOOP-Programme dienen sowieso ja nur der theoretischen Betrachtung der Berechenbarkeit, nicht der Performanz).
Uns ist zwar klar, dass wir nicht m Schritte, sondern eigentlich $(n \operatorname{div} m) + 1$ Schritte benötigen, aber der Algorithmus berechnet ja erst $n \operatorname{div} m$, daher kann man dies im LOOP -Schritt noch nicht benutzen.

[4] d.h. berechenbar durch eine primitiv-rekursive Funktion; äquivalent zu LOOP-Berechenbarkeit.

Folgendes LOOP-Programm ergibt sich für die schriftliche Division:

```
x_1 := n ∈ ℕ; x_2 := m ∈ ℕ \ {0}; (Eingabe)
x_3 := 0;
LOOP x_2 DO
    x_5 := 1; x_5 := x_5 − x_1;
    IF x_5 = 0 DO
        x_4 := x_1;
        x_1 := x_1 − x_2;
        x_3 := x_3 + 1;
    END;
END;
x_3 := x_3 − 1;
x_1 := x_2 − x_4;
IF x_1 = 0 THEN
    x_4 := 0;
    x_3 := x_3 + 1;
END
x_1 := x_4;
x_1 =: r; x_3 =: q (Ausgabe, d.h. Ergebnis für n mod m in x_1 und für n div m in x_3)
```

Und folgendes LOOP-Programm ergibt sich für **nur Modulo**:

```
x_1 := n ∈ ℕ; x_2 := m ∈ ℕ \ {0}; (Eingabe)
LOOP x_2 DO
    x_4 := 1; x_4 := x_4 − x_1;
    IF x_5 = 0 DO
        x_3 := x_1;
        x_1 := x_1 − x_2;
    END;
END;
x_1 := x_3;
x_3 := x_2 − x_1;
IF x_3 = 0 THEN
    x_1 := 0;
END
x_1 =: r (Ausgabe, Ergebnis für n mod m in x_1)
```

Der Fall $m = 0$ muss in den obigen beiden Programmen dann auf äußere Weise verhindert werden; in dem Falle $m = 0$ würde sonst dieses Programm $q = m,\ r = 0$ ausgeben.

Ebenso wie bei den WHILE-Programmen ist für die Berechnung des ganzzahligen Quotienten der ganze Algorithmus des ersten Programms vonnöten, mit $x_1 := x_3$ anstelle $x_1 := x_4$ als letzten Ausführungsschritt, um $x_1 =: q = n \operatorname{div} m$ zu erhalten.