„Le compte est bon“ lösen mit Computeralgebra

Michael Cuntz (Hannover)

cuntz@math.uni-hannover.de

Einleitung

Welche Möglichkeiten gibt es, die Zahl 438 aus den Zahlen $2, 3, 7, 7, 10, 100$ zu berechnen, indem man nur die Operationen $+, -, \cdot$ und $/$ verwendet? Jede der Ausgangszahlen darf hierbei höchstens einmal verwendet werden. Also z. B.

$$438 = (7 - 3) \cdot (10 + 100) - 2$$

oder

$$438 = (3 + 7/7) \cdot (10 + 100) - 2.$$

Oder wie sieht es aus, wenn man die Zahl 652 aus den Zahlen $3, 3, 5, 6, 9, 10$ erhalten will? In diesem Fall ist es unmöglich, nur wie kann man das entscheiden? Immerhin kann man mit diesen Zahlen die Zahl 651 erreichen,

$$651 = ((10 + 3 \cdot 6 \cdot 9) - 5) \cdot 3,$$

aber „näher“ kommt man nicht.

Das Fernsehspiel „Des chiffres et des lettres“, das ich aus meiner Kindheit noch kenne und in Frankreich auch in der Schule gespielt wird, wird schon seit etwa 50 Jahren im französischen Fernsehen ausgestrahlt. In dem Spiel müssen die Kandidaten in zwei Disziplinen antreten: Einerseits sollen sie aus gegebenen Buchstaben ein möglichst langes Wort kombinieren („Le mot le plus long“), andererseits einer gegebenen Zahl durch elementare Operationen wie in den obigen Beispielen möglichst nahe kommen („Le compte est bon“). Natürlich werden mittlerweile in der Sendung Computer benutzt, um optimale Lösungen zu berechnen. Wir wollen hier kurz besprechen, wie „Le compte est bon“ mit Computeralgebra gelöst werden kann.

Kombinatorik

Spielregeln

Zunächst müssen wir präzisieren, wie das Spiel definiert ist. Im Spiel werden eine Zahl $N \in \{101, \ldots, 999\}$ und sechs Zahlen $a_1, \ldots, a_6 \in \{1, \ldots, 10, 25, 50, 75, 100\}$ vorgegeben. Hierbei dürfen die Zahlen $1, \ldots, 10$ eventuell doppelt vorkommen. Das Ziel des Spiels ist es, die Zahl N ausgehend von $a_1, \ldots, a_6$ durch die elementaren Operationen $+, -, \cdot$ und $/$ zu berechnen.
Nicht ganz klar aus den Regeln ersichtlich ist, dass die Zwischenergebnisse immer ganze Zahlen bleiben müssen. Will man etwa die Zahl 15 aus $1, 5, 6, 7$ berechnen, muss man zwischendurch rationale Zahlen verwenden. Eine Lösung wäre $15 = 6/(7/5 - 1)$; diese ist aber nicht regelkonform.
Außerdem sind meines Wissens negative Zwischenergebnisse nicht erlaubt. Da ich diese Regel aber nicht explizit gefunden habe, gehen wir hier davon aus, dass negative Zahlen erlaubt sind, dass aber der Operator ‘$-$’ nicht unär verwendet werden darf.

Catalan-Zahlen

Es gibt eine Menge Symmetrien, die in der Lösung des Spiels verwendet werden können. Zuerst müssen wir uns aber klar machen, wie die Ausdrücke am Ende geklammert werden können. Will man z. B. drei Elemente a, b, c mit einer Operation $\circ$ (z. B. ‘$-$’) verknüpfen, gibt es bei fester Reihenfolge der Elemente die beiden Möglichkeiten $(a \circ b) \circ c$ und $a \circ (b \circ c)$. Bei vier Elementen a, b, c, d sind es die fünf Ausdrücke

$$((a \circ b) \circ c) \circ d, \quad (a \circ (b \circ c)) \circ d, \quad (a \circ b) \circ (c \circ d),$$

$$a \circ ((b \circ c) \circ d), \quad a \circ (b \circ (c \circ d)).$$

Es sei C_n die Anzahl der Ausdrücke, in denen die Operation n-mal vorkommt. Da wegen der Klammerung die Operation an einer eindeutigen Stelle zuletzt durchgeführt wird, sieht man die Rekursion

$$C_n = \sum_{k=1}^{n} C_{k-1} C_{n-k}.$$

Zum Beispiel mithilfe von erzeugenden Funktionen kann man $C_n = \frac{1}{n+1}\binom{2n}{n}$ folgern (siehe etwa [Aig04]). Die Anzahl der Ausdrücke, die man bei einer (nicht assoziativen) Operation also betrachten muss, ist die berühmte *Catalan-Zahl*.

Ohne Symmetrien zu verwenden, können wir nun also zunächst abschätzen, dass wir C_n mögliche Klammerungen betrachten müssen, wenn wir $n+1$ Elemente einsetzen. Für jede Operation haben wir vier Möglichkeiten, und die $n+1$ Elemente werden aus den $a_1, \ldots, a_6$ auf $\binom{6}{n+1}$ Weisen ausgewählt und können in $(n+1)!$ verschiedenen Möglichkeiten eingesetzt werden. Wir erhalten also (höchstens)

$$\sum_{n=0}^{5} 4^n \binom{6}{n+1} (n+1)! C_n = 33665406$$

Ausdrücke, die wir auswerten müssen, um das Spiel komplett zu lösen. Das ist für einen modernen Computer nicht viel; einen einfachen Algorithmus geben wir weiter unten an.

Symmetrien
Aber eigentlich würden wir mit diesen Ausdrücken viel zu viele Rechnungen probieren:
Assoziativität: Es gilt etwa $(a+b)+c = a+(b+c)$ oder $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
Kommutativität: Außerdem haben wir $a+b = b+a$ und $a \cdot b = b \cdot a$.
Distributivität: Wenn die Zahlen a, a, b, c gegeben sind, ist zudem $a \cdot (b+c) = a \cdot b + a \cdot c$. Wenn es nur darum geht, irgendeine Lösung zu finden, könnte man sich also hier einen Fall sparen.
Multiplizitäten: Gilt $a_1 = a_2$, so wird der Vorfaktor $(k+1)!$ in der Formel kleiner. Dasselbe gilt in komplizierterer Form allgemeiner, wenn die Zahlen $a_1, \ldots, a_6$ nicht verschieden sind.
Statt einer Catalan-Kombinatorik dürfen wir eigentlich ein allgemeineres kombinatorisches Modell verwenden. Solche Modelle sind in der Kombinatorik bekannt und basieren auf gewissen Pfaden mit Labels in einem Gitter (oder Bäumen mit Labels).

Algorithmus
Folgenden Algorithmus kann man verwenden, um alle Lösungen zu finden, die exakt N ergeben:

`LeCompteEstBon`(A, N, F, (i_0, j_0), s)
Alle Lösungen des Spiels „Le compte est bon“

Eingabe: Zahlen $A = [a_1, \ldots, a_n]$ und N, Flags $F = [f_1, \ldots, f_n]$, ein Paar von Indizes (i_0, j_0), ein Ausgabestring s.
Ausgabe: alle Ausdrücke in A, die N ergeben.

Für alle Paare $(i,j) \in \{1, \ldots, n\}$ mit $i < j$, $(i,j) >_{lex} (i_0, j_0)$ und $f_i = f_j = 0$:
• Setze $F' := [f'_1, \ldots, f'_n, 0]$, mit $f'_k = f_k$ für $k \neq i, j$ und $f'_i = f'_j = 1$.
• Bilde einen neuen Ausgabestring s' aus s und der Rechnung $a_i + a_j = (a_i + a_j)$.
• Falls $a_i + a_j = N$, gib s' aus.
• `LeCompteEstBon`($A + [a_i + a_j]$, N, F', (i,j), s').
• [... die drei letzten Zeilen für alle anderen Operationen nochmal ...]

Die Funktion `LeCompteEstBon` ruft man nun mit der Liste $A = [a_1, \ldots, a_n]$ der gegebenen Zahlen und der Zielzahl N auf, sowie mit $f_i = 0$ für $1 \leq i \leq n$, $i_0 = j_0 = 0$ und s dem leeren String.

Dieser Vorschlag berücksichtigt bereits einen Teil der oben aufgelisteten Symmetrien. Die Tatsache, dass das neue Paar (i,j) lexikographisch größer als das letzte sein soll, erspart eine Menge überflüssiger Rechnungen, da gemeinsame Teile von Ausdrücken nicht mehrfach ausgewertet werden.

Eine Implementierung löst je nach Programmiersprache das Problem in mehr oder weniger einer Sekunde. Das obige Beispiel $A = [2, 3, 7, 7, 10, 100]$ und $N = 438$ hat laut `LeCompteEstBon` 15 Lösungen.

Computeralgebra

Rationale Funktionen
Es gibt jedoch eine einfache Methode, um alle Symmetrien auf einmal zu berücksichtigen ohne die Kombinatorik genauer zu verstehen. Jeder Ausdruck, den wir oben betrachten, kann „symbolisch“ als eine rationale Funktion in den Einträgen $x_1, \ldots, x_6$ angesehen werden. Wir können also mit einem Computeralgebrasystem ein für alle Mal alle diese rationalen Funktionen ausrechnen und abspeichern.
Für Ausdrücke mit $1, 2, 3, 4, 5$ bzw. 6 Zahlen erhalten wir so $1, 4, 24, 176, 1440$ bzw. 12608 verschiedene rationale Funktionen; das sind übrigens auch genau die Zahlen, die bei der genaueren Analyse der Kombinatorik herauskommen (vergleiche [CHN16]).
Bei jeder dieser Funktionen können noch die Variablen permutiert werden. Wir erhalten stattdessen $1, 6, 68, 1170, 27142$ bzw. 793002 verschiedene Funktionen (man beachte, dass hier wieder automatisch Symmetrien eingehen).

Nun müssen jedoch noch für jede Funktion auf n Variablen die $\binom{6}{n}$ Teilmengen von $\{a_1, \ldots, a_6\}$ eingesetzt werden. Damit erhalten wir

$$6 + 90 + 1360 + 17550 + 162852 + 793002 = 974860$$

rationale Funktionen in sechs Variablen. Insgesamt genügt es demnach 974860 Ausdrücke bei $a_1, \ldots, a_6$ auszuwerten. Der naive Ansatz ohne Symmetrien musste mehr als 34-mal so viele Ausdrücke ausprobieren.

Feinheiten
Den Ansatz mit Computeralgebra kann man noch auf viele Arten verbessern. Zum Beispiel haben die 974860 rationalen Funktionen deutlich weniger verschiedene Zähler; man könnte also die Zähler sammeln und mehrfaches Einsetzen vermeiden. Es gibt allerdings auch mehrere kleine Probleme:
Ein Problem ist, dass wir durch die rationalen Funktionen nicht mehr die Vorgabe einhalten, dass alle Zwischenergebnisse ganzzahlig bleiben. So müssen wir also nach Auswertung nochmal prüfen, ob der Ausdruck regelkonform ist.
Ein zweites Problem ist, dass es nicht offensichtlich ist, wie man aus der rationalen Funktion einen passenden Ausdruck rekonstruiert, z. B. ist

$$(x \cdot u + y \cdot u)/(z \cdot w + u \cdot v \cdot w) = (x+y)/(w \cdot (z/u + v)).$$

Das Problem kann man umgehen, indem man sich zu jeder Funktion a priori einen Ausdruck merkt.
Schließlich spart Algorithmus `LeCompteEstBon` von oben durch die Rekursion mehrfaches Auswerten von Teilen von Ausdrücken, was bei den rationalen Funktionen nicht so einfach zu realisieren ist.

Tatsächlich verhält es sich hier so wie bei vielen Problemen: Mit Computeralgebra läßt sich ein Problem oft in wenigen Zeilen Code lösen, die allerdings nicht unbedingt eine optimale Laufzeit haben. Trotz der obigen Anmerkungen benötigen wir am Ende etwas weniger Auswertungen als im ersten Ansatz, dieser Vorsprung geht aber leider komplett dadurch verloren, dass die meisten Computeralgebra-Systeme den Code interpretieren.

Literatur

[Aig04] M. Aigner, *Diskrete Mathematik*, fifth ed., Vieweg Studium: Aufbaukurs Mathematik. Friedr. Vieweg & Sohn, Wiesbaden, 2004.

[CHN16] Frédéric Chapoton, Florent Hivert, and Jean-Christophe Novelli, *A set-operad of formal fractions and dendriform-like sub-operads*, Journal of Algebra **465** (2016), 322 – 355.