7down – Zusatzaufgaben

Mathias Ziebarth und Joachim Breitner

13. März 2008

Problem 1

Unser Programm hat folgende Lösungen berechnet:

Testfall 1	**Testfall 2**	**Testfall 3**
153131	279841	576676
141441	321619	272422
973493	121114	985465
330529	877444	711348
869017	368888	232574
876927	646421	961361

Problem 2

Aufgabe (a)

Unsere Lösung für das Problem „Die Zahl a ist nicht negativ“ ist:

$$\texttt{=(qs(plus(mal(}a\texttt{,10),1)),plus(qs(}a\texttt{),1))}$$

Dieses Prädikat nennen wir für die weiteren Teilaufgaben `nn(`a`)`.

BEWEIS. Wir machen eine Fallunterscheidung:

Fall 1: $a \geq 0$:

Sei

$$a = \sum_{i=0}^{n} a_i \cdot 10^i$$

mit $n \in \mathbb{N}$ und $a_i \in \{0, \ldots, 9\}$ für $i = 0, \ldots, n$ die Dezimaldarstellung von a. Dann ist

$$a \cdot 10 + 1 = \sum_{i=0}^{n} a_i \cdot 10^{i+1} + 1$$

die Dezimaldarstellung von $a \cdot 10 + 1$. Für die Quersummen gilt

$$qs(a \cdot 10 + 1) = \sum_{i=0}^{n} a_i + 1 = qs(a) + 1$$

und unsere Bedingung ist erfüllt.

Fall 2: $a < 0$: Sei

$$a = -\sum_{i=k}^{n} a_i \cdot 10^i$$

mit $n \in \mathbb{N}$, $k \leq n$, $a_i \in \{0, \ldots, 9\}$ für $i = 0, \ldots, n$ und $a_k \neq 0$ die Dezimaldarstellung von a ohne Schlussnullen. Sei weiter

$$a \cdot 10 + 1 = -\sum_{i=0}^{n+1} b_i \cdot 10^i$$

die Dezimaldarstellung von $a \cdot 10 + 1$.

Das Addieren der Eins zu einer negativen Zahl bedeutet das Abziehen der Eins vom Betrag der Zahl, so dass es zu Überträgen kommt. Also ist $b_i = 9$ für $i = 0, \ldots k$, $b_{k+1} = a_k - 1$ und $b_{i+1} = a_i$ für $i = k + 1, \ldots, n$. Für die Quersummen folgt

$$qs(a) = \sum_{i=k}^{n} a_i$$

sowie

$$qs(a \cdot 10 + 1) = (k + 1) \cdot 9 + a_k - 1 + \sum_{k+1}^{n} a_i = (k + 1) \cdot 9 - 1 + qs(a).$$

Es ist $(k + 1) \cdot 9 - 1 \neq 0$ für alle $k \in \mathbb{N}$, also kann die Bedingung nicht erfüllt sein und unsere Bedingung ist korrekt. ■

Aufgabe (b)

Unsere Lösung für das Problem „Die Zahl a enthält keine Null“ ist:

```
nn(minus(qp(a),1))
```

oder ausgeschrieben:

`=(qs(plus(mal(minus(qp(`a`),1),10),1)),plus(qs(minus(qp(`a`),1)),1))`

Dieses Prädikat nennen wir `k0(`a`)` für die Teilaufgabe (c).

BEWEIS. Das Querprodukt einer Zahl ist offensichtlich genau dann Null, wenn die Zahl eine Null als Ziffer enthält. Damit gilt:

$$\begin{aligned} a \text{ enthält keine Null} &\iff qp(a) \neq 0 \\ \text{(Querprodukte sind nicht-negativ)} &\iff qp(a) > 0 \\ &\iff qp(a) - 1 \geq 0 \\ &\iff \texttt{nn(minus(qp(}a\texttt{),1))} \end{aligned}$$

womit die Richtigkeit unserer Bedingung gezeigt ist. ■

Aufgabe (c)

Ein 9×9-Sudoku muss insgesamt 27 Bedingungen erfüllen, nämlich die Einmaligkeit der Ziffern in den 9 Zeilen, den 9 Spalten und den 9 Kästchen. Hat man nur 18 Variablen zur Verfügung, über die man Aussagen machen kann, so muss man die restlichen Bedingungen geschickter erfüllen.

Unser Ansatz ist hierbei, mit A waagerecht bis I waagerecht die Zeilen und mit J senkrecht bis R senkrecht die Spalten zu bezeichnen. Mittels `k0(A_w)`,...,`k0(I_w)` schließt man nun die ungültige Ziffer 0 aus und erfüllt die ersten 18 Sudoku-Bedingungen mittels `U(A_w)`,...,`U(I_w)`, `U(J_s)`,..., `U(R_s)`.

Nun muss noch die Bedingung, dass in den 3×3-Kästchen keine Ziffer doppelt vorkommt, modelliert werden. Dazu stellen wir zunächst eine Bedingung auf, die besagt, dass die k_1-te Ziffer einer Zahl $a = \sum_{i=0}^{9} a_i \cdot 10^i$ und die k_2-te Ziffer einer Zahl $b = \sum_{i=1}^{9} b_i \cdot 10^i$ verschieden sind. Wir zählen dabei die Ziffern der einfacherern Notation wegen von hinten mit Null beginnend.

Wir werden mit den Überträgen an der k_1-ten (bzw. k_2-ten) Stelle arbeiten. Damit uns Überträge an der $(k_1 + 1)$-ten Stelle nicht reinpfuschen (wenn $a_{k_1+1} = 9$ ist), verringern wir diese Ziffer um eins, falls $k_1 < 9$ ist.

$$a' \coloneqq a - 1 \cdot 10^{k_1+1} = \sum_{i=0}^{9} a'_i \quad (k_i < 9)$$

Ist $k_1 = 9$ so kann an der $k_1 + 1 = 10$-ten Stelle natürlich kein Übertrag passieren, und wir verwenden $a' \coloneqq a$ unverändert.

Wir beobachten, dass der Ausdruck

$$s(a,l) \coloneqq qs(a' + l \cdot 10^{k_1}) - qs(a') \quad l \in \{1,\dots,9\},$$

also die Veränderung der Quersumme beim Erhöhen der k_1-ten Stelle um l, gerade l ist, wenn $a_{k_1} + l < 10$ ist und kein Übertrag passiert, und gerade $l - 10 + 1 = l - 9$ ist, wenn wegen $a_{k_1} + l \geq 10$ ein Übertrag passiert. Der Wert von $s(a,l)$ hängt also (dank unserer Vorverarbeitung) nur von der Stelle a_{k_1} ab.

Vergleichen wir nun dies zwischen den gegebenen Zahlen mit

$$\Delta_l \coloneqq s(a,l) - s(b,l), \quad l \in \{0,\dots,9\}$$

so gilt

$$\Delta_l = \begin{cases} -9, & \text{falls nur } a \text{ bei } k_1 \text{ einen Übertrag hat} \\ 0, & \text{falls } a \text{ und } b \text{ einen Übertrag hat} \\ 0, & \text{falls weder } a \text{ noch } b \text{ einen Übertrag hat} \\ +9, & \text{falls nur } b \text{ bei } k_2 \text{ einen Übertrag hat} \end{cases}$$

Es gilt also $\Delta_l \neq 0$ genau für die l, bei denen die größere der Ziffern a_{k_1} und b_{k_2} schon einen Übertrag hat, die kleinere aber noch nicht. Genauer:

$$\Delta \coloneqq \Delta_1 + \Delta_2 + \cdots + \Delta_9 = (-9) \cdot (a_{k_1} - b_{k_2})$$

Offensichtlich gilt nun

$$\begin{aligned} a_{k_1} \neq b_{k_2} &\iff \Delta \neq 0 \\ &\iff qs(\Delta) > 0 \\ &\iff qs(\Delta) - 1 \geq 0 \end{aligned}$$

Der letzte Term lässt sich nun mittels `nn(·)` darstellen. Auch kamen bei der Konstruktion von Δ nur Operationen vor, die wir verwenden dürfen. Damit ist gezeigt, dass man die Ungleichheit von beliebigen Zellen konstruieren kann – wenn auch sehr umständlich. Eine Aufschlüsselung der Bedingung „Die 3. Stelle von `A_w` unterscheidet sich von der 4. Stelle von `B_w`“ finden sie in Abbildung 1.

Mit dieser Konstruktion lässt sich nun die Ungleichheit der Ziffern in einem 3×3-Kästchen sicherstellen, indem sie einfach auf alle Paare von Ziffern im Kästchen angewandt wird. Wer will, kann sich die Paare sparen, die schon in einer Zeile oder einer Spalte liegen.

Andererseits kann man diese Konstruktion auch für etwa die Spalten nutzen – dann kommt man mit nur den 9 Variablen für die Zeilen aus. Diese braucht man in jedem Fall für das Keine-Null-Kriterium und um die einzelnen Zellen zu adressieren. Wir konnten also die Vorgabe um 50% unterbieten, wenn auch mit extrem vielen und aufwendigen Regel-Konstruktionen.

```
=(qs(plus(mal(minus(qs(plus(plus(plus(plus(plus( plus(plus(plus(plus(0,minus(minus(
qs(plus(minus(A_w,10000),1000)), qs(minus(A_w,10000))), minus(
qs(plus(minus(B_w,100000),10000)), qs(minus(B_w,100000))))),
minus(minus( qs(plus(minus(A_w,10000),2000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),20000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),3000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),30000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),4000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),40000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),5000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),50000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),6000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),60000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),7000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),70000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),8000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),80000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),9000)), qs(minus(A_w,10000))), minus(
qs(plus(minus(B_w,100000),90000)), qs(minus(B_w,100000)))))), 1),10),1)),plus(
qs(minus( qs(plus(plus(plus(plus(plus(plus(plus(plus(plus(0,minus(minus(
qs(plus(minus(A_w,10000),1000)), qs(minus(A_w,10000))), minus(
qs(plus(minus(B_w,100000),10000)), qs(minus(B_w,100000))))),
minus(minus( qs(plus(minus(A_w,10000),2000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),20000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),3000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),30000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),4000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),40000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),5000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),50000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),6000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),60000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),7000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),70000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),8000)), qs(minus(A_w,10000))),
minus( qs(plus(minus(B_w,100000),80000)), qs(minus(B_w,100000)))),
minus(minus( qs(plus(minus(A_w,10000),9000)), qs(minus(A_w,10000))), minus(
qs(plus(minus(B_w,100000),90000)), qs(minus(B_w,100000)))))), 1)),1))
```

Abbildung 1: Die 3. Stelle von `A_w` unterscheidet sich von der 4. Stelle von `B_w`

Sudoku-Konstrukteur

Um die Konstruktion von Aufgabe 2c) durchzuführen haben wir ein Haskell-Programm geschrieben. Dieses ist hier abgedruckt. Die Ausgabe selbst ist zu groß – 527 Zeilen, 1 178 843 Bytes – und wird vermutlich von keinem Solver ohne weiteres geschluckt.

Man beachte wie direkt sich die oben konstruierten Formeln in Code gießen lassen, wenn man die eingebauten Operationen für Addition, Subtraktion und Multiplikation geeignet überläd. Sogar die `sum`-Funktion funktioniert so. Weiter kann man mit dem Listen-Monad in Haskell einfach Permutationen von Koordinaten erzeugen.

```
{-# LANGUAGE FlexibleInstances #-}

{- (c) 2008 Joachim Breitner
 -
 - This haskell Program generates input in the informaticup-syntax
 - for Kreuzzahlenraetsel that simulate a sudoku
 -}

import Control.Monad

unary name v = name++"("++v++")"
binary name a b = name++"("++a++","++b++")"

-- Use nice syntax instead of ugly syntax
instance Num [Char] where
        (+) = binary "plus"
        (-) = binary "minus"
        (*) = binary "mal"
        abs = undefined
        signum = undefined
        fromInteger = show

-- powers of 10
l `e` p = fromInteger (l * 10^p)

-- Carry-Over safe Version of v
safe v 8 = v
safe v k = v -  1 `e` (k+1)

-- More operators
qs = unary "qs"
qp = unary "qp"
```

```
u  = unary "U"

-- equality
eq  = binary "eq"

-- not negative
nn v = eq (qs (v*10+1)) (qs v + 1)

-- no zero
nz v = nn(qp v - 1)

s v k l = qs(safe v k + l `e` k) - qs(safe v k)

delta' a b k1 k2 l = s a k1 l - s b k2 l

delta a b k1 k2  = sum $ map (delta' a b k1 k2) [1..9]

digits_differ (a, b, k1, k2) = nn (qs (delta a b k1 k2) - 1)

rows = map (:"_w") ['A'..'I']

column_pairs = do
        row1 <- rows
        row2 <- rows
        guard  $ row1 < row2
        k <- [0..8]
        return (row1, row2, k, k)

block_pairs = do
        x1 <- [0..8]
        x2 <- [0..8]
        y1 <- [0..8]
        y2 <- [0..8]
        guard $ y1 < y2 -- Rows handled by U(..)
        guard $ x1 /= x2 -- Columns already handled as well
        guard $ x1 `div` 3 == x2 `div` 3 && -- In same block?
                y1 `div` 3 == y2 `div` 3

        let row1 = rows !! y1
            row2 = rows !! y2
        return (row1, row2, x1, x2)

all_pairs = column_pairs ++ block_pairs
```

```
pair_rules = map digits_differ all_pairs

uniqe_rules = map u rows

nz_rules = map nz rows

all_rules = uniqe_rules ++ nz_rules ++ pair_rules

cells = zipWith (\c n -> c:"␣"++show n ++"␣1") ['A'..'I'] [1..9]

descr = ["9␣9", "9"] ++ cells ++
        ["9"] ++ map (:[]) ['A'..'I'] ++
        ["0", show (length all_rules) ] ++
        all_rules

main = mapM_ putStrLn descr
```