Solving Kenken using an SMT (integer) solver

Preprint · June 2020

DOI: 10.13140/RG.2.2.30170.36808

1 author:

Clive England
British Computer Society
14 PUBLICATIONS **0** CITATIONS

SEE PROFILE

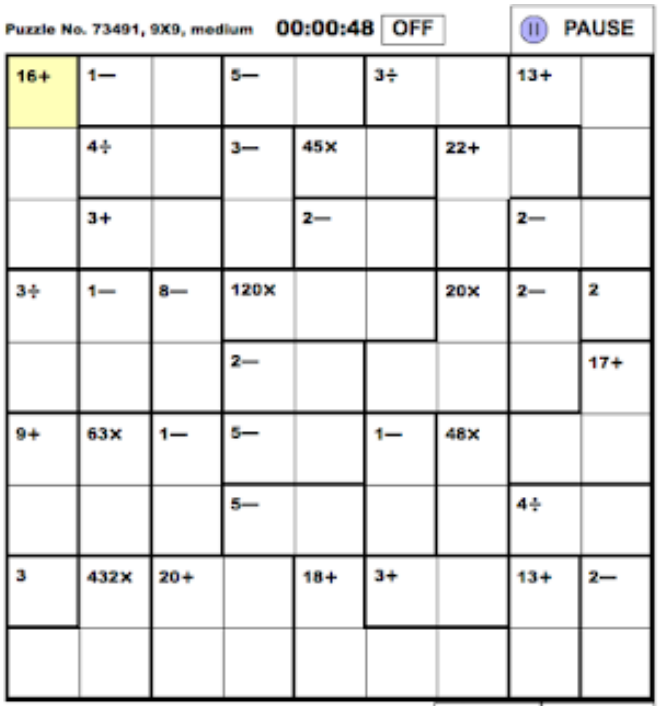
Friday, 20 March 2020

Solving Kenken using an SMT (integer) solver

An Article by Clive England, CEng, MBCS
See Original and updates on https://gannett-hscp.blogspot.com June 2020

Continuing in the short series of reframing puzzles and using an SMT integer solver to find a solution we look now at solving the Kenken (sometimes known as killer Sudoku) class of problem. This popular number placement puzzle seen in the UK papers Daily Telegraph and Daily Mail. This puzzle is similar to killer-Sudoku using a rectangular grid of numbers where target sums are set for small regions of cells. The numbers 1..9 are placed in the grid such that the sum of the numbers equals the targets provided for each region. Any number can only used once in each row and column.

This 9 by 9 hard example is from www.Kenken.com. Looking at this puzzle we see regions of cells each of which has a total and an operation +,-,*,/ in the top left most region cell.



The encoding of this puzzle relies on a region identifier being placed in each group of cells that belong to the same sum. The target sum and operation are encoded with the region letter. The top left cell is labeled a16+ and the two below as just a. The encoding is {group identification letters}{target number}{operation} for the first cell in a group then just the {group letter identification letter} for the rest.

When the operand of a sum is - or / the cells can be used in either order. This is the encoding for the puzzle above.

```
#kenken www.kenken.com kk_91.png 73491
a16+,b1-,b,e5-,e,f3/,f,g13+,g
a,c4/,c,h3-,j45*,j,l22+,l,g
a,d3+,d,h,k2-,k,l,m2-,m
n3/,p1-,q8-,r120*,r,r,t20*,u2-,v2
n,p,q,s2-,s,t,t,u,w17+
aa9+,bb63*,cc1-,dd5-,dd,ff1-,gg48*,w,w
aa,bb,cc,ee5-,ee,ff,gg,hh4/,hh
kk3,mm432*,xx20+,xx,yy18+,zz3+,zz,ab13+,cd2-
```

```
mm,mm,xx,xx,yy,yy,yy,ab,cd
```

A script is used to generate the SMT2 lines for the solver. For this puzzle because of the inclusion of * and / operands the UFNIA: Non-linear integer arithmetic with uninterpreted sort and function symbols logic is used. The words after a ; on a line are comments.

Set the logic to UFNIA and declare the variables V0 .. V80 then set the range for each variable.

```
(set-logic UFNIA)
(set-option :produce-models true)
(set-option :produce-assignments true)
(declare-const V0 Int)
(declare-const V1 Int)
(declare-const V2 Int)
```

.....

```
(declare-const V79 Int)
(declare-const V80 Int)
(assert (and (> V0 0) (< V0 10)))
(assert (and (> V1 0) (< V1 10)))
```

.....

Next the constraints for each number has to be unique on the row and unique in a column. There are 18 of these lines.

```
(assert (distinct V0 V1 V2 V3 V4 V5 V6 V7 V8 )) ; line 0
1

(assert (distinct V0 V9 V18 V27 V36 V45 V54 V63 V72 )) ;
line 0 9
```

.....

Next the constrains for the number regions. Notice the "or" logic for the - and / operands. There is one line for each cell region, 36 in all for this puzzle example.

```
(assert (= 48 (* V51 V60))) ;   Region gg

(assert (or (= 2 (- V34 V43))(= 2 (- V43 V34)))) ;
Region u

(assert (= 3 (+ V19 V20))) ;    Region d

(assert (or (= 2 (- V39 V40))(= 2 (- V40 V39)))) ;
Region s

(assert (= 13 (+ V70 V79))) ;   Region ab


(assert (or (= 3 (/ V5 V6))(= 3 (/ V6 V5)))) ;  Region f
```

.....

and finally the wrap up and answers extract.

```
(check-sat)
(get-value (V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13
V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27
```

```
V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40 V41
V42 V43 V44 V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55
V56 V57 V58 V59 V60 V61 V62 V63 V64 V65 V66 V67 V68 V69
V70 V71 V72 V73 V74 V75 V76 V77 V78 V79 V80 ))
(exit)
```

As with the previous puzzles the script was coded to read in the solvers output and then generate output using the original puzzle and check the answers provided. Using display colours help identify the cells of a region. Amusingly we could have done the cell colouring with just four colours using the solver technique described here.

The performance of the two solvers used vary dramatically MathSat5 took about 7 minutes

```
424.79 real
421.93 user
1.75 sys
```

but Z3 solver took substantially longer at 45 minutes but gave the same answers.

```
real 45m38.547s
user 45m21.082s
sys 0m10.940s
```

In short mode the script just provides region name and cell values.

a 7	b 5	b 4	e 6	e 1	f 9	f 3	g 2	g 8
a 1	c 2	c 8	h 4	j 9	j 5	l 7	l 6	g 3
a 8	d 1	d 2	h 7	k 4	k 6	l 9	m 3	m 5
n 6	p 4	q 9	r 5	r 3	r 8	t 1	u 7	v 2
n 2	p 3	q 1	s 8	s 6	t 4	t 5	u 9	w 7
aa 4	bb 7	cc 5	dd 3	dd 8	ff 2	gg 6	w 1	w 9
aa 5	bb 9	cc 6	ee 2	ee 7	ff 3	gg 8	hh 4	hh 1
kk 3	mm 6	xx 7	xx 9	yy 5	zz 1	zz 2	ab 8	cd 4
mm 9	mm 8	xx 3	xx 1	yy 2	yy 7	yy 4	ab 5	cd 6

In verbose mode each cell has R= Region identifier, Ro= Operand, Rn= region total and Val=cell value.

R=a Ro=+ Rn=16 Val=7	R=b Ro=- Rn=1 Val=5	R=b Ro=- Rn=1 Val=4	R=e Ro=- Rn=5 Val=6	R=e Ro=- Rn=5 Val=1	R=f Ro=/ Rn=3 Val=9	R=f Ro=/ Rn=3 Val=3	R=g Ro=+ Rn=13 Val=2	R=g Ro=+ Rn=13 Val=8

R=a Ro=+ Rn=16 Val=1	R=c Ro=/ Rn=4 Val=2	R=c Ro=/ Rn=4 Val=8	R=h Ro=- Rn=3 Val=4	R=j Ro=* Rn=45 Val=9	R=j Ro=* Rn=45 Val=5	R=l Ro=+ Rn=22 Val=7	R=l Ro=+ Rn=22 Val=6	R=g Ro=+ Rn=13 Val=3
R=a Ro=+ Rn=16 Val=8	R=d Ro=+ Rn=3 Val=1	R=d Ro=+ Rn=3 Val=2	R=h Ro=- Rn=3 Val=7	R=k Ro=- Rn=2 Val=4	R=k Ro=- Rn=2 Val=6	R=l Ro=+ Rn=22 Val=9	R=m Ro=- Rn=2 Val=3	R=m Ro=- Rn=2 Val=5
R=n Ro=/ Rn=3 Val=6	R=p Ro=- Rn=1 Val=4	R=q Ro=- Rn=8 Val=9	R=r Ro=* Rn=120 Val=5	R=r Ro=* Rn=120 Val=3	R=r Ro=* Rn=120 Val=8	R=t Ro=* Rn=20 Val=1	R=u Ro=- Rn=2 Val=7	R=v Ro= Rn=2 Val=2
R=n Ro=/ Rn=3 Val=2	R=p Ro=- Rn=1 Val=3	R=q Ro=- Rn=8 Val=1	R=s Ro=- Rn=2 Val=8	R=s Ro=- Rn=2 Val=6	R=t Ro=* Rn=20 Val=4	R=t Ro=* Rn=20 Val=5	R=u Ro=- Rn=2 Val=9	R=w Ro=+ Rn=17 Val=7
R=aa Ro=+ Rn=9 Val=4	R=bb Ro=* Rn=63 Val=7	R=cc Ro=- Rn=1 Val=5	R=dd Ro=- Rn=5 Val=3	R=dd Ro=- Rn=5 Val=8	R=ff Ro=- Rn=1 Val=2	R=gg Ro=* Rn=48 Val=6	R=w Ro=+ Rn=17 Val=1	R=w Ro=+ Rn=17 Val=9
R=aa Ro=+ Rn=9 Val=5	R=bb Ro=* Rn=63 Val=9	R=cc Ro=- Rn=1 Val=6	R=ee Ro=- Rn=5 Val=2	R=ee Ro=- Rn=5 Val=7	R=ff Ro=- Rn=1 Val=3	R=gg Ro=* Rn=48 Val=8	R=hh Ro=/ Rn=4 Val=4	R=hh Ro=/ Rn=4 Val=1
R=kk Ro= Rn=3 Val=3	R=mm Ro=* Rn=432 Val=6	R=xx Ro=+ Rn=20 Val=7	R=xx Ro=+ Rn=20 Val=9	R=yy Ro=+ Rn=18 Val=5	R=zz Ro=+ Rn=3 Val=1	R=zz Ro=+ Rn=3 Val=2	R=ab Ro=+ Rn=13 Val=8	R=cd Ro=- Rn=2 Val=4
R=mm Ro=* Rn=432 Val=9	R=mm Ro=* Rn=432 Val=8	R=xx Ro=+ Rn=20 Val=3	R=xx Ro=+ Rn=20 Val=1	R=yy Ro=+ Rn=18 Val=2	R=yy Ro=+ Rn=18 Val=7	R=yy Ro=+ Rn=18 Val=4	R=ab Ro=+ Rn=13 Val=5	R=cd Ro=- Rn=2 Val=6

The check output looks like this where the occurrence of each digit is counted. There should be 9 each of the 1..9 digits.

1=9,2=9,3=9,4=9,5=9,6=9,7=9,8=9,9=9,

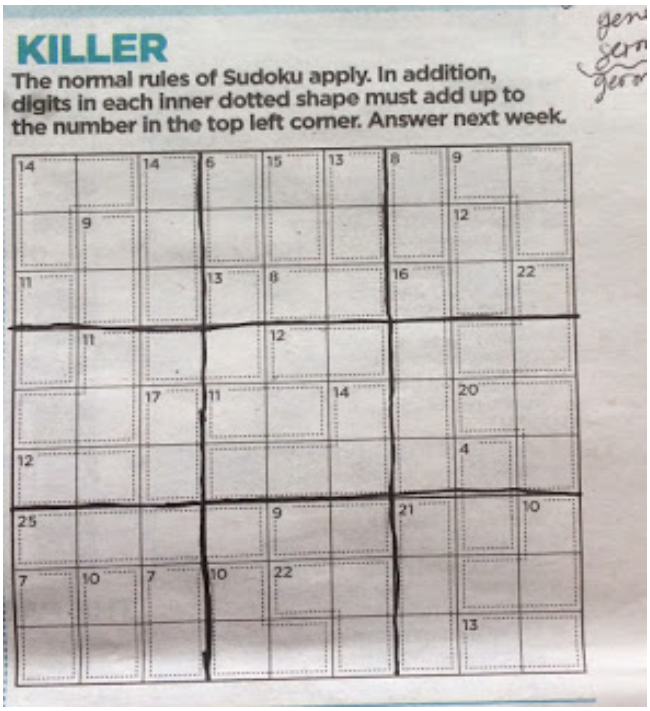
then the region formulas are checked.
Region a = OK as 16 + over 0 9 18
Region aa = OK as 9 + over 45 54
Region ab = OK as 13 + over 70 79
Region b = OK as 1 - over 1 2
Region bb = OK as 63 * over 46 55
.....

Region v = OK as 2=2
Region w = OK as 17 + over 44 52 53
Region xx = OK as 20 + over 65 66 74 75
Region yy = OK as 18 + over 67 76 77 78
Region zz = OK as 3 + over 68 69

This encoding and solver technique can be used for many similar integer constraint puzzles such as the DM Killer (which just has addition operations) and Telegraph Killer Sudoku Pro.

Killer

Killer is a combination of sudoku and kenken but the sub regions just use +.



The same methodology as kenken is used. The encoding becomes:

```
#Killer Sudoku _01 DM
a14+,a,c14+,d6+,e15+,f13+,g8+,h9+,h
a,b9+,c,d,e,f,g,i12+,h
j11+,b,c,k13+,l8+,l,m16+,i,p22+
j,s11+,k,k,n12+,n,m,p,p
s,s,t17+,v11+,v,w14+,m,q20+,q
u12+,u,t,w,w,w,m,r4+,q
x25+,x,x,x,y9+,y,z21+,r,aa10+
cc7+,dd10+,ee7+,ff10+,gg22+,gg,z,aa,aa
cc,dd,ee,ff,ff,gg,z,bb13+,bb
```

For the output region letters are displayed and coloured.

1=9,2=9,3=9,4=9,5=9,6=9,7=9,8=9,9=9,

a 9	a 3	c 7	d 1	e 8	f 4	g 5	h 2	h 6

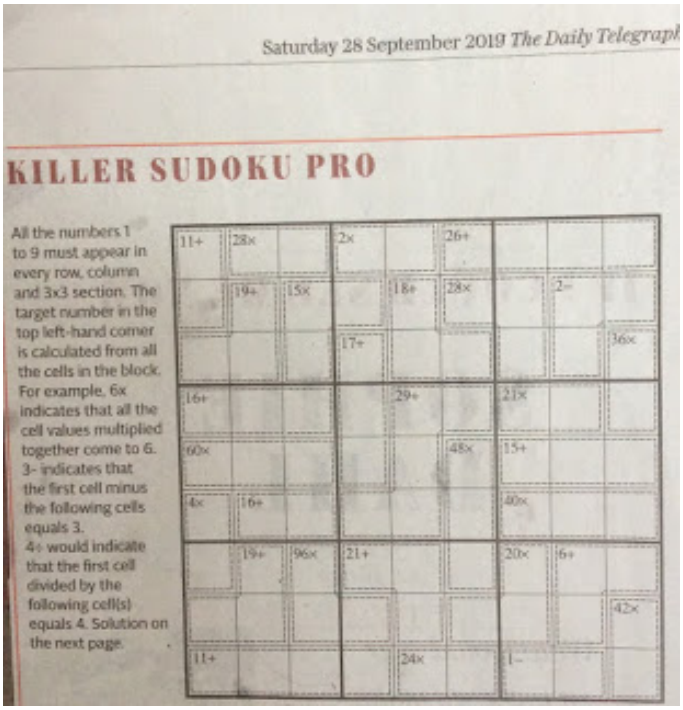
a 2	b 4	c 6	d 5	e 7	f 9	g 3	i 8	h 1
j 8	b 5	c 1	k 3	l 6	l 2	m 9	i 4	p 7
j 3	s 1	k 2	k 8	n 5	n 7	m 4	p 6	p 9
s 4	s 6	t 8	v 9	v 2	w 3	m 1	q 7	q 5
u 5	u 7	t 9	w 6	w 4	w 1	m 2	r 3	q 8
x 7	x 9	x 5	x 4	y 3	y 6	z 8	r 1	aa 2
cc 1	dd 2	ee 4	ff 7	gg 9	gg 8	z 6	aa 5	aa 3
cc 6	dd 8	ee 3	ff 2	ff 1	gg 5	z 7	bb 9	bb 4

Region a = OK as 14 + over 0 1 9

Region a = OK as 14 + over 0 1 9
Region aa = OK as 10 + over 62 70 71
Region b = OK as 9 + over 10 19
....

Example of Killer Sudoku

Encoded and works the same as kenken.



Appendix Usage of logic generation script

```
Usage: KenKen {options} < puzzel.txt  or
```

```
    KenKen {options} -f puzzel.txt < SolverOutput
    -v N :Be verbose to level N ( use v=1 for detailed output )
    -f puzzel.txt :Puzzel file needed when reading Solver output use
STDIN otherwise

    Programs reads STDIN looking for either puzzelFile format lines
or Solver output lines
        If the input is in puzzelFile format the program will
generate solver input lines.
        If the input is in Solver output lines format the program
will expect a -f puzzelFile paremeter.
            Using both these input streams program will then
generate display .html output.
Game Rules
Each cell can be 1..N only. Repeats NOT allowed in any row or
column.
Encoding Rules
Each cell belongs to a single region/cage that has a letter and
formula in the first occurance of the region.
Region names must be unique. Puzzels must be square. N*N puzzel has
answers 1..N.
    For Example
    b2-,b, ...  the values in the b labeled cells must satisfy b-b=2
    Numbers may repeat with a region but not in a row or column.
```

```
Each KenKen puzzel Input as follows each caged squares must belong
to a region
        region_lable{Value}{operation+-*/}

        #KenKen DM x
        a2/,b2-,b,c3/,c,d4
        a,e72*,f4,g90*,g,h3+
        e,e,i5+,j5,g,h
        j17+,j,i,i,k4,l2/
        j,m7+,m,n7+,n,l
        j,o5-,o,n,p2-,p
....
to generate an SMT2 input file. Use a .. z and A .. Z or aa .. zz
for region/cage names
```

```
OR Process results from solver in the following format
    sat
    ( (V0 9)
    (V1 7)
    (V2 5)
    (V3 3)
    ....
to generate an html table as output
```