

1.1 Parser- Reihenfolge

1. $A \rightarrow (B \rightarrow A)$
valid
2. $(A \rightarrow B) \rightarrow ((\neg B) \rightarrow (\neg A))$
valid
3. $(A \vee B) \rightarrow (A \wedge B)$
valid
4. $A \vee (\neg B)$
satisfiable
5. $((\neg A) \wedge B)$
satisfiable

1.2 Modelle von Formeln

1. $A \wedge B \wedge \neg D$
Es gibt nur dieses eine Modell.
2. $(A \vee B \vee D) \wedge (A \wedge B)$
Modell: $A \wedge B$
3. $(A \wedge B \wedge D \wedge \neg A \wedge \neg B) \vee (D \Leftrightarrow \neg D)$
Es gibt keine Modelle hierzu.
4. $((A \rightarrow B) \wedge (B \rightarrow D)) \rightarrow (A \rightarrow D)$
Jede mögliche Interpretation ist ein richtiges Modell

1.3 Xor- Verkettung

$A \text{ xor } B \text{ xor } C$ ist wahr, wenn genau ein Eingangssignal wahr ist.

Da es sich um den gleichen Operanten handelt, gibt es keine Gewichtung der Bindung und es wird von links nach rechts geparkt. Daher wird zuerst $A \text{ xor } B$ betrachtet. Der Ausgang dieser Operation wird wahr, wenn entweder A oder B wahr ist. Das zweite xor gibt Wahr aus, wenn entweder C oder das erste xor Wahr ist. Da das erste aber nur dann Wahr ist, wenn entweder A oder B Wahr ist, darf immer nur ein Wert Wahr sein, damit eine wahre Aussage entsteht.


```

29     }
30     }
31 }
32
33     UpdateSudokuNotes();
34
35 } // */
36
37 //printSudoku();
38 //printf("\nsolved: %hu\n", solved);
39
40 //Ausgabe:
41 if (changed == 0) return 2; //nicht eindeutig
42 if (changed < 0) return 3; //widerspruch
43 return 0;
44 }

```

code/sudokuSolver.cpp

2.2 Feld-Tester

```

1
2 uint8_t testNumbers(int x, int y) {
3     z = x / 3 + 3 * (y / 3);
4
5     //checke, ob es nur noch eine Zahl fr das Feld gibt:
6     if (checkPow2(sudoku[x][y] & ~0b1111)) {
7         activeNumber = highestBit(sudoku[x][y]) - 3;
8         num = 1 << activeNumber;
9         setField(x, y, z);
10        return activeNumber;
11    }
12
13    uint8_t x_start = 3 * (x / 3);
14    uint8_t y_start = 3 * (y / 3);
15    activeNumber = 0;
16    uint8_t multipleChoices = 0;
17
18    //checke, ob sich aus den Reihen, Spalten und Feldern die
19    //Zahl ergibt:
20    for (int i = 1; i <= 9; i++) { //aktiver Zahlenwert
21        num = 1 << i;
22        if ((row[y] & num) && (column[x] & num) && (field[z] &

```

```

num)) {//Fehlt die Zahl noch in der Reihe & Spalte &
Feld?
22
23 //checke, ob eine Zahl die einzige in der
Reihe/Spalte/Feld ist:
24 num <= 3;
25 if (!(sudoku[x][y] & num)) continue;
26 uint8_t r_count = 0, c_count = 0, f_count = 0;
27 for (int j = 0; j < 9; j++) {
28     if (sudoku[j][y] & num) r_count++;
29     if (sudoku[x][j] & num) c_count++;
30     if (sudoku[x_start + j % 3][y_start + j / 3] & num)
        f_count++;
31     //if (x == 0 && y == 0) printf("\n%i: %i:: %4hx, %4hx,
        %4hx", i, j, sudoku[j][y], sudoku[x][j],
        sudoku[x_start + j % 3][y_start + j / 3]);
32 }
33 //if (x == 0 && y == 0) printf("\n%i:fnd: %4hhu, %4hhu,
        %4hhu", i, r_count, c_count, f_count);
34
35 if (r_count == 1 || c_count == 1 || f_count == 1) {
        activeNumber = i; multipleChoices = 0; break;
        }//genau eine mglichkeit?
36 if ((r_count > 1 && c_count > 1 && f_count > 1) ||
        activeNumber != 0) multipleChoices = 1;//mehr als
        eine mglichkeit?
37 activeNumber = i;
38 }
39 }
40
41 if (multipleChoices) return 1 << 7;//mehr als eine
        mglichkeit!
42 //if (activeNumber == 0) return 0;keine Mglichkeiten,
        sollte nicht passieren!
43
44 //Streiche die Zahl aus der Reihe/Spalte/Feld:
45 num = 1 << activeNumber;
46 setField(x, y, z);
47 return activeNumber;
48 }

```

code/sudokuSolver.cpp

2.3 Update-Funktion von möglichen Eingaben

```

1
2 void UpdateSudokuNotes() {
3     //Trage nderungen nach:
4     for (int y = 0; y < 9; y++) {
5         for (int x = 0; x < 9; x++) {
6             sudoku[x][y] &= ((row[y] & column[x] & field[x / 3 + 3 *
              (y / 3)]) << 3) | 0b1111;
7         }
8     }
9
10    //Trage aus nderungen resultierende nderungen nach:
11    //eliminateBlockedField(); //ist logisch impliziert und ist
        daher oboslet!
12    eliminateDoubleField();
13
14
15 }

```

code/sudokuSolver.cpp

2.4 Elimination von möglichen Eingaben

```

1
2 void eliminateDoubleField() {
3     uint16_t positions;
4     uint16_t numMask;
5     uint16_t xAdd, yAdd;
6     uint16_t rowInc, colInc, fldInc;
7     uint16_t* lineMask;
8     for (int x = 0; x < 9; x++) { //reihe/spalte/feld
9         uint16_t counter[3][9] = { 0 }; //fr jeden Zahlenwert ein
            counter + feld-positionsmaske
10         uint16_t count = 0;
11         uint16_t size = 0;
12
13         xAdd = 3 * (x % 3);
14         yAdd = 3 * (x / 3);
15         for (int n = 0; n < 9; n++) { //Zahlenfeld
16             //Summiere Feld auf:
17             for (int i = 0; i < 9; i++) { //Zahlenwert-1

```

```

18     rowInc = 0b1 & (sudoku[n][x] >> (i + 4));
19     colInc = 0b1 & (sudoku[x][n] >> (i + 4));
20     fldInc = 0b1 & (sudoku[xAdd + n % 3][yAdd + n / 3] >>
        (i + 4));
21     counter[0][i] += rowInc; //inkrementiere counter bei
        gesetzten Zahlenwert-bit
22     counter[1][i] += colInc;
23     counter[2][i] += fldInc;
24     counter[0][i] |= rowInc << (n + 4); //setze
        feld-positionsbit
25     counter[1][i] |= colInc << (n + 4);
26     counter[2][i] |= fldInc << (n + 4);
27 }
28 }
29
30 //Teste auf 2- bis 5fache Wertepaare (mehr Anwendungsfälle
    gibt es nicht):
31 for (int k = 0; k < 3; k++) {
32     lineMask = k > 0 ? (k == 2 ? field : column) : row;
33     for (int i = 0; i < 9; i++) {//aktiver Zahlenwert-1
34         size = counter[k][i] & 0b1111;
35
36         if (size < 2 || size > 5) continue;
37
38         count = 0;
39         numMask = 0;
40         positions = counter[k][i] & ~0b1111;
41
42         for (int j = 0; j < 9; j++) {//Zahlenwerte der Reihe
43             if ((lineMask[x] & (1 << j+1)) && ((counter[k][j] &
                positions) == (counter[k][j] & ~(0b1111)))) {
                numMask |= 1 << j; count++; }//summiere, wenn
                Zahlenwert der Reihe Teil der Menge des aktuellen
                Zahlenwertes ist
44         }
45
46         //if (k == 1 && i == 0 && x == 1) printf("count: %hu,
            counter: %hx, mask: %hx, line: %hx, field: %hx\n",
            count, counter[k][i], numMask, lineMask[x],
            field[x]);
47
48         //*/
49         //count < size: nicht genug Dopplungen um Rückschlüsse

```

```

        zu schlieen
50    //count > size: Fehler, da es nicht mehr Zahlen als
        size sein knnen
51    if (count == size) { //alle anderen Werte knnen gelscht
        werden
52        uint16_t fieldpos = positions >> 4;
53        uint16_t n = 0; //Zahlenfeld
54        numMask <= 4;
55
56        /*/
57        printf("\n\nmatch:\n");
58        printf("x: %i, k: %i, i: %i, count: %i\n", x, k,
            i+1, count);
59        printf("fldpos: %hx, mask: %hx\n", fieldpos,
            numMask);
60        /*/
61        do {
62            if (fieldpos & 1) {
63                switch (k)
64                {
65                    case 0: sudoku[n][x] &= numMask; break;
66                    case 1: sudoku[x][n] &= numMask; break;
67                    case 2: sudoku[xAdd + n % 3][yAdd + n / 3] &=
                        numMask; break;
68                }
69            }
70            n++;
71        } while (fieldpos >= 1);
72    } // */
73    }
74    }
75    }
76
77    /*/
78    printf("\nnach double:");
79    printSudoku();
80    printf("\n");
81    /*/
82 }

```

code/sudokuSolver.cpp

2.5 Ergebnis

Es lassen sich alle Sudokus (selbst die ganz schweren) mit dem Algorithmus lösen.

The screenshot shows a web browser window with a Sudoku game interface on the left and a terminal window on the right. The browser window displays a Sudoku grid with some numbers filled in. The terminal window shows the output of the Sudoku solver, including the input grid, a confirmation prompt, and the solved grid.

Sudoku Game Interface (Left):

The interface shows a 9x9 grid with the following numbers (row by row):

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

The interface also includes buttons for "New Start", "New Game", and "Break", along with a timer showing "07:11".

Sudoku Solver Terminal (Right):

The terminal window shows the following output:

```

Auswählen C:\Users\miese_Kakalake\source\repos
Sudoku- Solver v0.1
Bitte gebe die Werte des Sodokus ein:
3 4 6
7 9 3
8 3
3 5 2 1 9 4
2 3
4 8 2 1
6 7 4
Eingabe korrekt? (y/n)y
Starte Auswertung...
Lösung ermittelt:
3 1 2 4 7 8 6 5 9
7 6 4 2 9 5 8 1 3
8 9 5 3 1 6 2 7 4
4 3 9 5 2 1 7 6 8
5 8 1 7 6 4 3 9 2
6 2 7 8 3 9 5 4 1
1 4 8 6 5 2 9 3 7
2 7 6 9 4 3 1 8 5
9 5 3 1 8 7 4 2 6
  
```

Der Algorithmus erkennt auch, wenn falsche Eingaben getätigt werden:

The image displays two screenshots of a web-based Sudoku solver interface and its terminal output.

Top Screenshot (Successful Solve):

The web interface shows a Sudoku puzzle on the left and a terminal window on the right. The puzzle is a 9x9 grid with some numbers pre-filled. The terminal window shows the solver's output:

```

Sudoku- Solver v0.1
Bitte gebe die Werte des Sodokus ein:
6 1 8 7
9 8 6 6
5 9 6
7 9
Eingabe korrekt? (y/n)y
Starte Auswertung...
Es konnte keine eindeutige Lösung gefunden werden.
Wurde vielleicht eine Zahl vergessen?

```

Bottom Screenshot (Error Message):

The web interface shows the same Sudoku puzzle. The terminal window shows the solver's output after an invalid input:

```

Sudoku- Solver v0.1
Bitte gebe die Werte des Sodokus ein:
6 1 8 7
9 8 6 6
5 9 6 2
7 9 5
1 4 8
2 8
8 5 1
Eingabe korrekt? (y/n)y
Starte Auswertung...
Widerspruch gefunden: Das Sodoku ist nicht lösbar!
Wurde vielleicht eine Zahl falsch eingegeben?

```

Die Lösung für dieses Sudoku:

M 03: A x
M 03: A x
Downl: x
Sudoku x

ku-very-difficult/
HTW webmail :: Wil...
Microsoft Word Onl...
HTW-Berlin: Service...
Moodle @ HTW Be...

EN
f 344

Sudoku
Easy Medium Hard **Very difficult**

	6			1		8	7	
	9			8		6		
			6	2				
			9	6				
		5						2
	7	9						5
					4			
2							8	
8							5	1

Sudoku- Solver v0.1

Bitte gebe die Werte des Sodokus ein:

6	1	8 7
9	8	6
	9 6	
5		2
7 9		5
2	4	8
8		5 1

Eingabe korrekt? (y/n)y

Starte Auswertung...

Lösung ermittelt:

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

Kapitel 3. Kompletter Code:

```
1 #include "sudokuSolver.h"
2 #include <stdint.h>
3 #include <stdio.h>
4 #include <windows.h>
5
6 //uint16_t sudoku[3][3][3][3]; // Die ersten beiden beschreiben
   die Position und die letzten beiden den Inhalt des
   3x3-Feldes
7 uint16_t sudoku[9][9]; // Array mit allen Zahlenwerten und
   bitmasken der jeweiligen Zahlenfelder
8
9
10 const uint16_t emptyRow = 0b1111111110;
11 const uint16_t numberBits = 0b1111;
12 uint16_t activeNumber;
13
14 //Reihen, Spalten und 3x3-Felder:
15 uint16_t row[9];
16 uint16_t column[9];
17 uint16_t field[9];
18
19 int z; // das entsprechende Feld
20 uint16_t num; // bitmaske der aktuellen Zahl
21 uint16_t solved; // Anzahl der Zahlen, die ermittelt wurden. Bei
   solved = 81 wird der Algorithmus erfolgreich beendet
22
23 char pressedButton;
24
25 //initiiert alle Felder des Sodokus
26 //void initSudoku(){ for (int i = 0; i < 81; i++) { sudoku[i %
   3][(i / 3) % 3][(i / 9) % 3][(i / 27) % 3] = emptyField; } }
27 void initSudoku() { for (int i = 0; i < 81; i++) sudoku[i %
   9][i / 9] = emptyField; }
28 void setSudoku(COORD pos, uint16_t value) {
```

```

        sodoku[pos.X][pos.Y] = value; }
29 uint16_t getSudoku(COORD pos) { return sodoku[pos.X][pos.Y]; }
30 uint16_t getSudoku(int x, int y) { return sodoku[x][y]; }
31
32 /*=====
33  * Helper      *
34  *=====*/
35 //Gibt 1 aus, wenn sie eine exakte potenz von 2 ist, sonst 0
36 uint16_t checkPow2(uint16_t number) { return number &&
        !(number & (number - 1)); }
37 //Gibt die Stelle des hchsten Bits an (startend von 0)
38 int highestBit(uint16_t number) { int res = 0; while (number
        >= 1) { res++; } return res; }
39
40 //setzt das Feld mit der bitmaske num
41 void setField(int x, int y, int z) {
42     sodoku[x][y] = activeNumber;
43     num = 1 << activeNumber;
44
45     row[y]     ^= num;
46     column[x] ^= num;
47     field[z]   ^= num;
48 }
49 void setField(int x, int y) { setField(x, y, x / 3 + 3 * (y /
        3)); }
50
51 //testet ein Feld des Sodokus auf einen bestimmten Wert und
        gibt entweder die gefundene Zahl, bei mehreren Zahlen 1<<7
        und bei keiner Zahl 0 aus
52 uint8_t testNumbers(int x, int y) {
53     z = x / 3 + 3 * (y / 3);
54
55     //checke, ob es nur noch eine Zahl fr das Feld gibt:
56     if (checkPow2(sodoku[x][y] & ~0b1111)) {
57         activeNumber = highestBit(sodoku[x][y]) - 3;
58         num = 1 << activeNumber;
59         setField(x, y, z);
60         return activeNumber;
61     }
62
63     uint8_t x_start = 3 * (x / 3);
64     uint8_t y_start = 3 * (y / 3);
65     activeNumber = 0;

```

```

66  uint8_t multipleChoices = 0;
67
68  //checke, ob sich aus den Reihen, Spalten und Feldern die
    Zahl ergibt:
69  for (int i = 1; i <= 9; i++) { //aktiver Zahlenwert
70      num = 1 << i;
71      if ((row[y] & num) && (column[x] & num) && (field[z] &
        num)) { //Fehlt die Zahl noch in der Reihe & Spalte &
        Feld?
72
73          //checke, ob eine Zahl die einzige in der
        Reihe/Spalte/Feld ist:
74          num <= 3;
75          if (!(sudoku[x][y] & num)) continue;
76          uint8_t r_count = 0, c_count = 0, f_count = 0;
77          for (int j = 0; j < 9; j++) {
78              if (sudoku[j][y] & num) r_count++;
79              if (sudoku[x][j] & num) c_count++;
80              if (sudoku[x_start + j % 3][y_start + j / 3] & num)
                f_count++;
81              //if (x == 0 && y == 0) printf("\n%i: %i:: %4hx, %4hx,
                %4hx", i, j, sudoku[j][y], sudoku[x][j],
                sudoku[x_start + j % 3][y_start + j / 3]);
82          }
83          //if (x == 0 && y == 0) printf("\n%i:fnd: %4hhu, %4hhu,
                %4hhu", i, r_count, c_count, f_count);
84
85          if (r_count == 1 || c_count == 1 || f_count == 1) {
                activeNumber = i; multipleChoices = 0; break;
            } //genau eine mglichkeit?
86          if ((r_count > 1 && c_count > 1 && f_count > 1) ||
                activeNumber != 0) multipleChoices = 1; //mehr als
                eine mglichkeit?
87          activeNumber = i;
88      }
89  }
90
91  if (multipleChoices) return 1 << 7; //mehr als eine
    mglichkeit!
92  //if (activeNumber == 0) return 0; //keine Mglichkeiten,
    sollte nicht passieren!
93
94  //Streiche die Zahl aus der Reihe/Spalte/Feld:

```

```

95  num = 1 << activeNumber;
96  setField(x, y, z);
97  return activeNumber;
98 }
99
100 //initialisiert reihen, spalten und felder mit den
   eingetragenen werten:
101 int initSolveSudoku() {
102     solved = 0;
103
104     //initialisiere reihen, spalten und felder:
105     for (int y = 0; y < 9; y++) {
106         for (int x = 0; x < 9; x++) {
107             row[y] = emptyRow;
108             column[x] = emptyRow;
109             field[x % 3 + 3 * (y / 3)] = emptyRow;
110         }
111     }
112
113     //Trage alle vorgegebenen Zahlen ein:
114     for (int y = 0; y < 9; y++) {
115         for (int x = 0; x < 9; x++) {
116             activeNumber = sudoku[x][y] & 0b1111;
117
118             //Falls die Zahl schon eingetragen wurde:
119             if (activeNumber != 0) {
120                 z = x / 3 + 3 * (y / 3);
121                 num = 1 << activeNumber;
122                 if ((row[y] & num) == 0 //Teste, ob die Zahl schon
                     vorkommt
123                     || (column[x] & num) == 0
124                     || (field[z] & num) == 0) { printf("\ninit
                     failure!\n"); return 3; } //Widerspruch: Eine Zahl
                     kann nur einmal vorkommen!
125                 //else { printf("\n(%2i;%2i;%2i) correct", x, y, z); }
126
127                 //setze den Wert:
128                 setField(x,y,z);
129                 solved++;
130             }
131         }
132     }
133

```

```

134 UpdateSudokuNotes();
135
136 return 0;
137 }
138
139 void UpdateSudokuNotes() {
140     //Trage nderungen nach:
141     for (int y = 0; y < 9; y++) {
142         for (int x = 0; x < 9; x++) {
143             sudoku[x][y] &= ((row[y] & column[x] & field[x / 3 + 3 *
144                 (y / 3)]) << 3) | 0b1111;
145         }
146     }
147     //Trage aus nderungen resultierende nderungen nach:
148     //eliminateBlockedField(); //ist logisch impliziert und ist
149     //daher oboslet!
150     eliminateDoubleField();
151
152 }
153 //Eliminierung durch doppelnde bitmasken der felder:
154 void eliminateDoubleField() {
155     uint16_t positions;
156     uint16_t numMask;
157     uint16_t xAdd, yAdd;
158     uint16_t rowInc, colInc, fldInc;
159     uint16_t* lineMask;
160     for (int x = 0; x < 9; x++) { //reihe/spalte/feld
161         uint16_t counter[3][9] = { 0 }; //fr jeden Zahlenwert ein
162         //counter + feld-positionsmaske
163         uint16_t count = 0;
164         uint16_t size = 0;
165
166         xAdd = 3 * (x % 3);
167         yAdd = 3 * (x / 3);
168         for (int n = 0; n < 9; n++) { //Zahlenfeld
169             //Summiere Feld auf:
170             for (int i = 0; i < 9; i++) { //Zahlenwert-1
171                 rowInc = 0b1 & (sudoku[n][x] >> (i + 4));
172                 colInc = 0b1 & (sudoku[x][n] >> (i + 4));
173                 fldInc = 0b1 & (sudoku[xAdd + n % 3][yAdd + n / 3] >>
174                     (i + 4));

```



```

173         counter[0][i] += rowInc; // inkrementiere counter bei
           gesetzten Zahlenwert-bit
174         counter[1][i] += colInc;
175         counter[2][i] += fldInc;
176         counter[0][i] |= rowInc << (n + 4); // setze
           feld-positionsbit
177         counter[1][i] |= colInc << (n + 4);
178         counter[2][i] |= fldInc << (n + 4);
179     }
180 }
181
182 // Teste auf 2- bis 5fache Wertepaare (mehr Anwendungsfälle
   gibt es nicht):
183 for (int k = 0; k < 3; k++) {
184     lineMask = k > 0 ? (k == 2 ? field : column) : row;
185     for (int i = 0; i < 9; i++) { // aktiver Zahlenwert-1
186         size = counter[k][i] & 0b1111;
187
188         if (size < 2 || size > 5) continue;
189
190         count = 0;
191         numMask = 0;
192         positions = counter[k][i] & ~0b1111;
193
194         for (int j = 0; j < 9; j++) { // Zahlenwerte der Reihe
195             if ((lineMask[x] & (1 << j+1)) && ((counter[k][j] &
               positions) == (counter[k][j] & ~(0b1111)))) {
               numMask |= 1 << j; count++; } // summiere, wenn
               Zahlenwert der Reihe Teil der Menge des aktuellen
               Zahlenwertes ist
196         }
197
198         // if (k == 1 && i == 0 && x == 1) printf("count: %hu,
           counter: %hx, mask: %hx, line: %hx, field: %hx\n",
           count, counter[k][i], numMask, lineMask[x],
           field[x]);
199
200         // */
201         // count < size: nicht genug Dopplungen um Rückschlüsse
           zu schließen
202         // count > size: Fehler, da es nicht mehr Zahlen als
           size sein können
203         if (count == size) { // alle anderen Werte können gelöscht

```

```

        werden
204     uint16_t fieldpos = positions >> 4;
205     uint16_t n = 0; //Zahlenfeld
206     numMask <= 4;
207
208     /*/
209     printf("\n\nmatch:\n");
210     printf("x: %i, k: %i, i: %i, count: %i\n", x, k,
        i+1, count);
211     printf("fldpos: %hx, mask: %hx\n", fieldpos,
        numMask);
212     /*/
213     do {
214         if (fieldpos & 1) {
215             switch (k)
216             {
217                 case 0: sodoku[n][x] &= numMask; break;
218                 case 1: sodoku[x][n] &= numMask; break;
219                 case 2: sodoku[xAdd + n % 3][yAdd + n / 3] &=
                    numMask; break;
220             }
221         }
222         n++;
223     } while (fieldpos >= 1);
224 } // */
225 }
226 }
227 }
228
229 /*/
230 printf("\nnach double:");
231 printSudoku();
232 printf("\n");
233 /*/
234 }
235 //Eliminierung durch Blockieren einer Reihe/Spalte:
236 void eliminateBlockedField() {
237     uint16_t mask, rmask, cmask;
238     uint16_t xAdd, yAdd;
239     for (int x = 0; x < 9; x++) { //feld
240         xAdd = 3 * (x % 3);
241         yAdd = 3 * (x / 3);
242         //printf("%i:\n", x + 1);

```

```

243
244     for (int n = 4; n < 13; n++) {//Zahlenwert
245         mask = 0;
246         for (int i = 0; i < 3; i++) {
247             uint16_t r1 = sudoku[xAdd][yAdd + i];
248             uint16_t r2 = sudoku[xAdd+1][yAdd + i];
249             uint16_t r3 = sudoku[xAdd+2][yAdd + i];
250             uint16_t c1 = sudoku[xAdd + i][yAdd];
251             uint16_t c2 = sudoku[xAdd + i][yAdd + 1];
252             uint16_t c3 = sudoku[xAdd + i][yAdd + 2];
253
254             //mask |= (((r1 & (r2 | r3)) | (r2 & r3)) >> n) &
0b1) << i;
255             //mask |= (((c1 & (c2 | c3)) | (c2 & c3)) >> n) &
0b1) << j;
256             rmask |= (((r1 | r2 | r3) >> n) & 0b1) << i;
257             cmask |= (((c1 | c2 | c3) >> n) & 0b1) << i;
258         }
259         mask = rmask ^ cmask;
260         if (checkPow2(mask) == 0 || mask == 0) continue;
261         mask = checkPow2(rmask) ? rmask : cmask << 3;
262
263         uint16_t excludeMask = ~(1<n);
264         int line = highestBit(mask);
265         //printf("blocked in field %i at %i by: %i\n", x+1,
line, n - 3);
266
267         if (line > 2) {//spalten
268             line += xAdd - 3;
269             uint16_t addblock = 3 * ((1 + x / 3) % 3);
270             for (int i = 0; i < 6; i++) {
271                 sudoku[line][(addblock + i) % 9] &= excludeMask;
272             }
273         }
274         else {//reihen
275             line += yAdd;
276             uint16_t addblock = 3 * ((x + 1) % 3);
277             for (int i = 0; i < 6; i++) {
278                 sudoku[(addblock + i) % 9][line] &= excludeMask;
279             }
280         }
281     }
282 }

```

```

283  /*/
284  printf("\nnach block:");
285  printSudoku();
286  printf("\n");
287  /**/
288 }
289
290 //lst das sodoku mit brute force:
291 int solveSudoku() {
292
293     int changed = 1;//1, wenn in dem aktuellen Zyklus nderungen
        vorgenommen wurden
294
295     //initiiere
296     int err = initSolveSudoku();
297     if (err != 0) return err;
298
299     /**/
300     //Ermittle Lsung:
301     while (solved != 81 && changed > 0) {
302         changed = 0;
303
304         //Laufe jede einzelne Zahl ab
305         for (int y = 0; y < 9; y++) {
306             for (int x = 0; x < 9; x++) {
307                 if (sudoku[x][y] == 0) { /*printSudoku();*/ return 3;
                    //widerspruch!!!
308                 activeNumber = sudoku[x][y] & 0b1111;
309                 if (activeNumber == 0) {
310                     activeNumber = testNumbers(x, y);
311                     if (activeNumber == 1 << 7) continue;
312                     else if (activeNumber == 0) changed = -1;
313                     else {
314                         //printf("\n%i: found (%2i;%2i): %hhu\n", changed,
                            x, y, activeNumber);
315                         changed = 1;
316                         solved++;
317                     }
318                 }
319             }
320         }
321
322         UpdateSudokuNotes();

```

```
323
324 } // */
325
326 // printSudoku();
327 // printf("\nsolved: %hu\n", solved);
328
329 // Ausgabe:
330 if (changed == 0) return 2; // nicht eindeutig
331 if (changed < 0) return 3; // widerspruch
332 return 0;
333 }
334
335 void printSudoku() {
336     for (int y = 0; y < 9; y++) {
337         printf("\n");
338         if (y % 3 == 0) printf("\n");
339         for (int x = 0; x < 9; x++) {
340             printf("%4hx ", sudoku[x][y]);
341             if (x % 3 == 2) printf(" ");
342         }
343     }
344     printf("\nrow:\n");
345     for (int y = 0; y < 9; y++) {
346         printf("%4hx ", row[y]);
347     }
348     printf("\ncolumn:\n");
349     for (int y = 0; y < 9; y++) {
350         printf("%4hx ", column[y]);
351     }
352     printf("\nfield:\n");
353     for (int y = 0; y < 9; y++) {
354         printf("%4hx ", field[y]);
355     }
356
357     printf("\n\n");
358 }
```

code/sudokuSolver.cpp