ISP AUFGABE 2

Gruppe 3, Team 5
15.05.2017

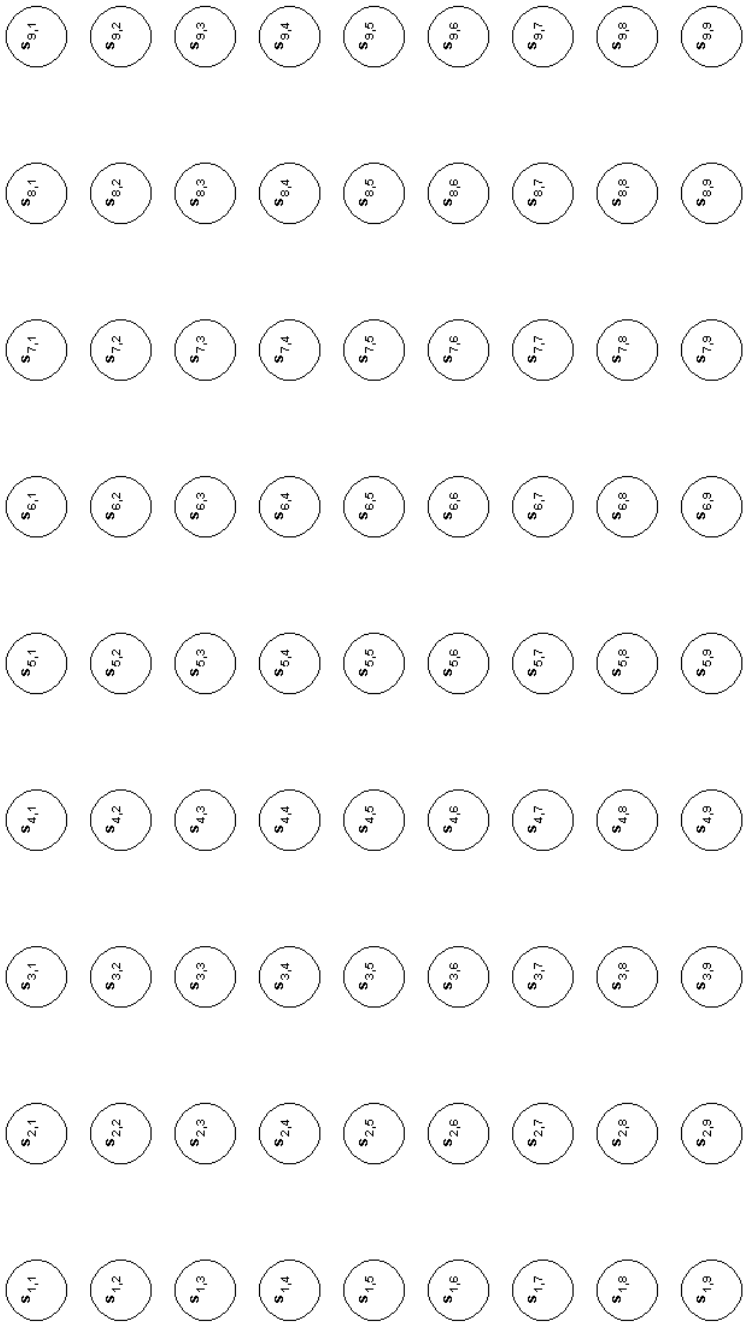
Dimitri Meier, Saeed Shanidar, Andreas Berks
dimitri.meier@haw-hamburg.de,
saeed.shanidar@haw-hamburg.de,
andreas.berks@haw-hamburg.de

Inhaltsverzeichnis

1) Constraint-Netz

1.1) Schritt 1: Definieren der Variablen

Wir definieren für jedes Feld des Sudoku-Rätsels eine Variable. Wir bezeichnen diese Variablen als $s_{i,j}$. Dabei stellt **i** die Nummer der Zeile dar (von einschließlich 1 bis einschließlich 9) und **j** die Nummer der Spalte (ebenfalls von einschließlich 1 bis einschließlich 9).



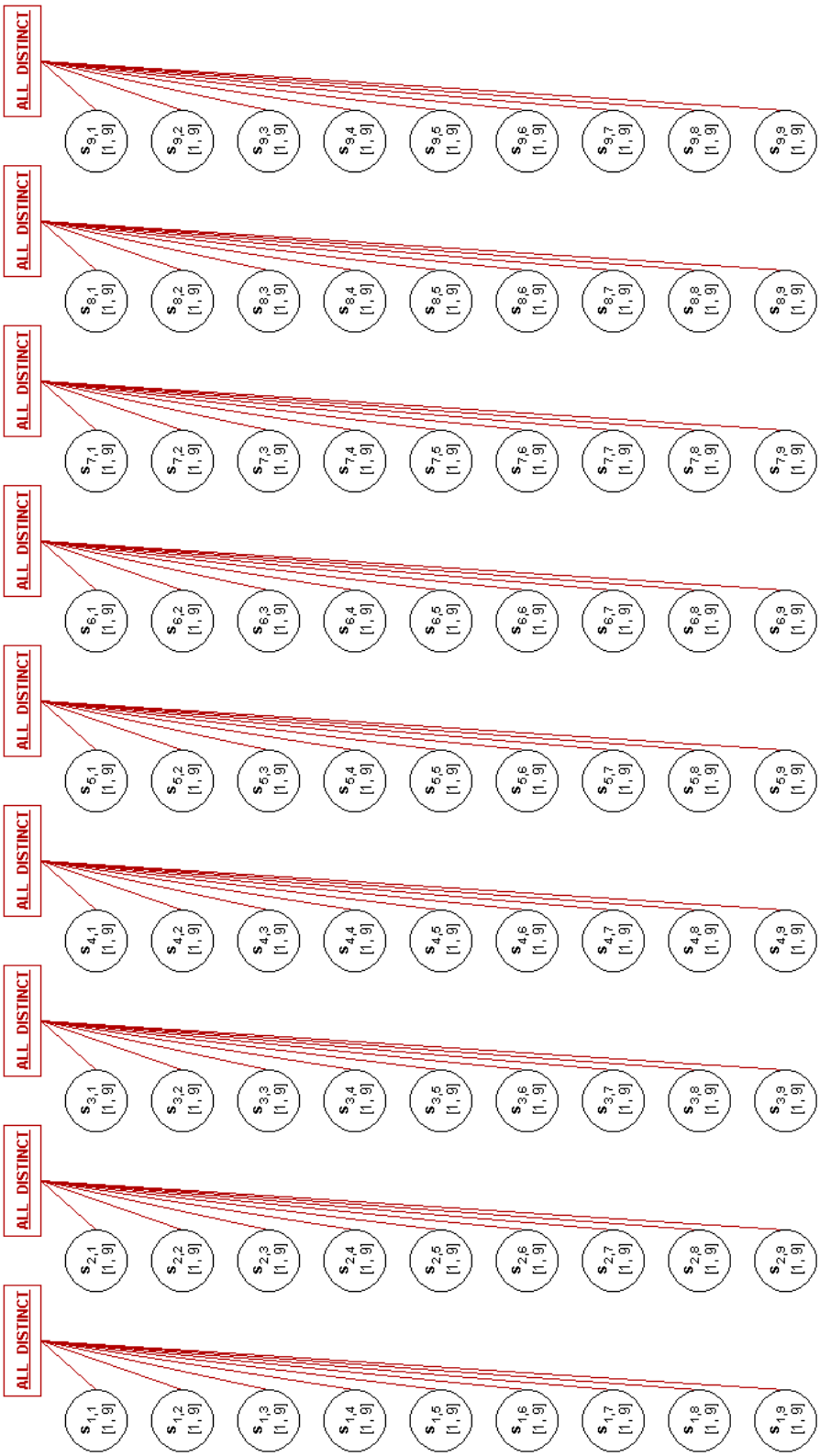
1.2) Schritt 2: Definieren der Wertebereiche für Variablen

Der Wertebereich jeder Variablen $s_{i,j}$ geht von einschließlich 1 bis einschließlich 9.

Im Folgenden definieren wir nun Schritt für Schritt die einzelnen Contraints, welche zusammen unser Contraint-Netz (siehe Abschritt 1.6) bilden.

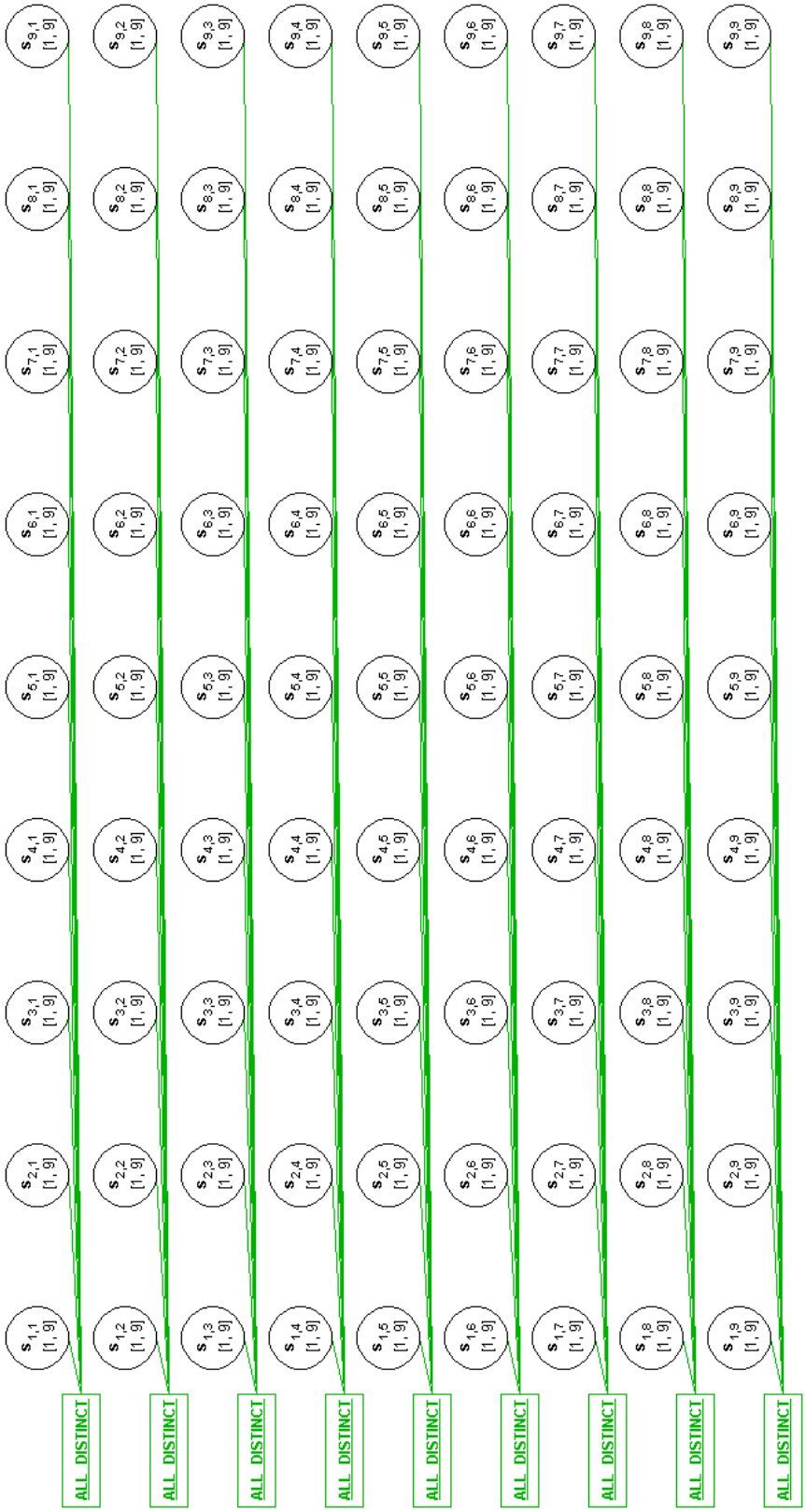
1.3) Schritt 3: Constraints für Spalten

Wir legen für jede Spalte ein **`ALL_DISTINCT`**-Constraint fest, welches bedingt, dass sich alle Werte der jeweiligen Spalte unterscheiden.



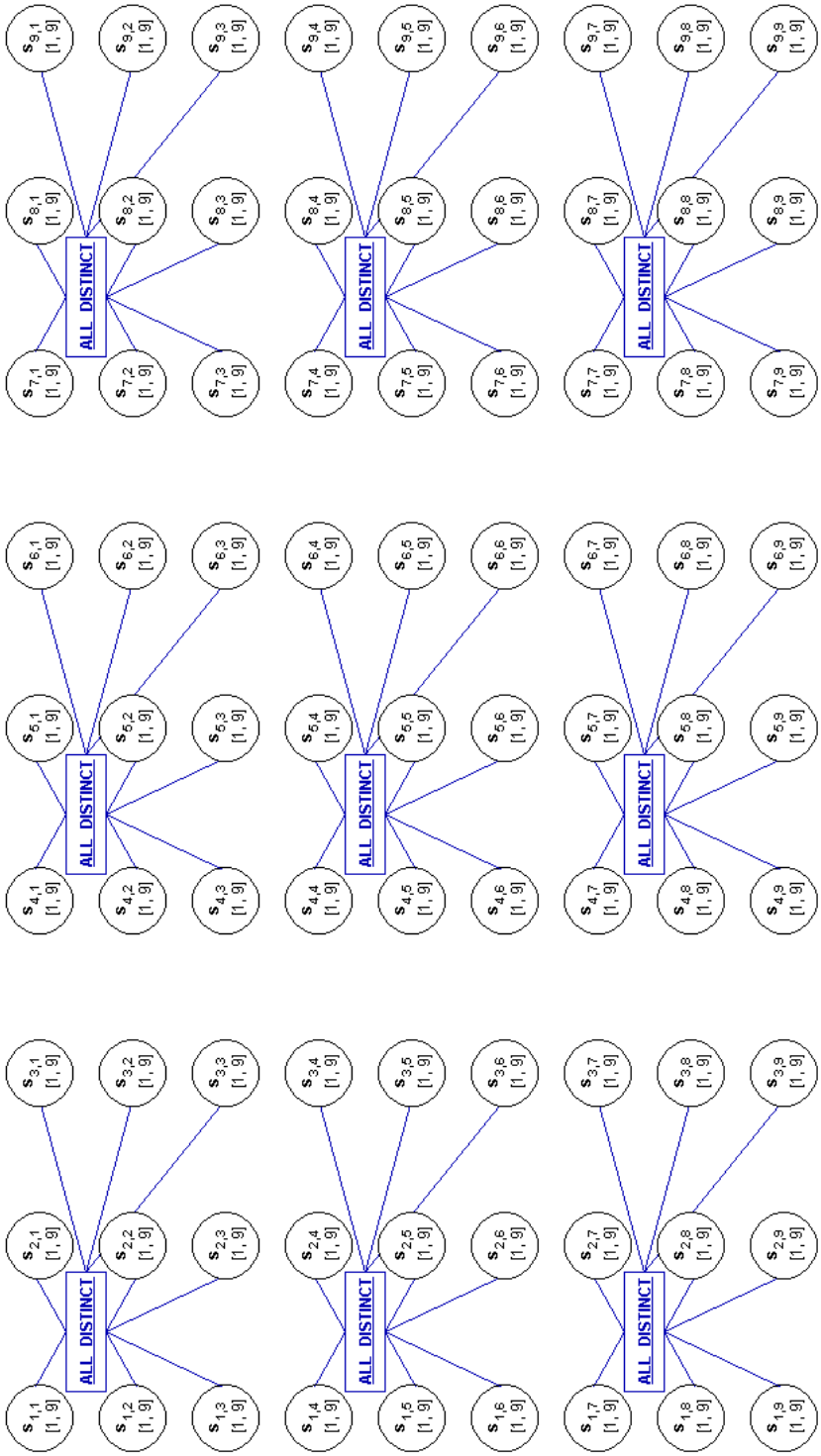
1.4) Schritt 4: Constraints für Zeilen

Wir legen für jede Zeile ein **`ALL_DISTINCT`**-Constraint fest, welches bedingt, dass sich alle Werte der jeweiligen Zeile unterscheiden.



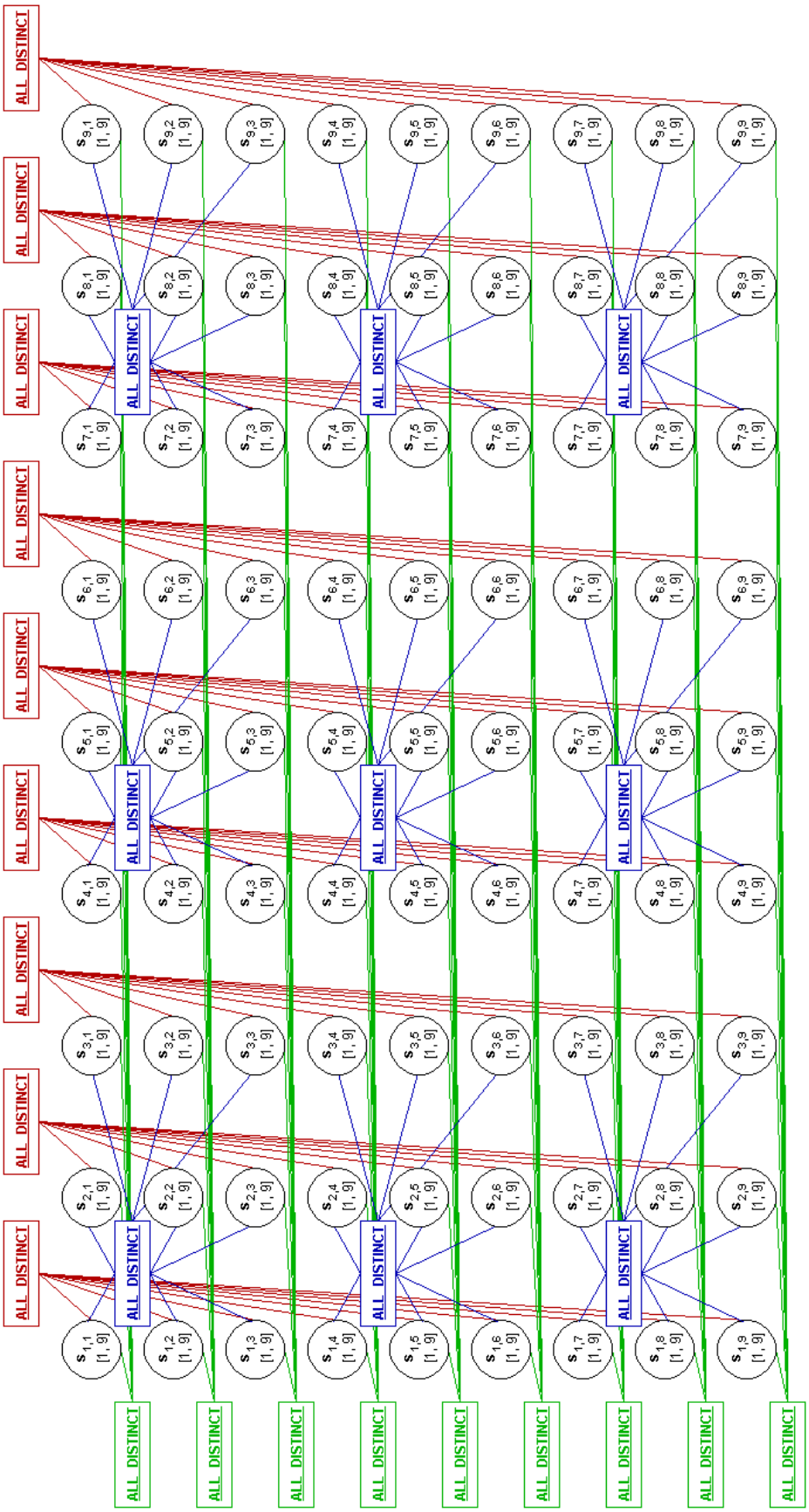
1.5) Schritt 5: Constraints für Blöcke

Wir legen für jeden 3x3-Block ein **`ALL_DISTINCT`**-Constraint fest, welches bedingt, dass sich alle Werte des jeweiligen Blocks unterscheiden.



1.6) Gesamtes Contraint-Netz

Fügen wir nun alle Contraints zusammen, ergibt sich folgendes Contraint-Netz:



2) Lösung in Prolog

Zuerst haben wir eine passende Repräsentation für ein Sudoku-Rätsel gesucht.
Wir haben ein Prädikat **`sudoku/2`** definiert, welches als ersten Parameter eine eindeutige Bezeichnung für das Rätsel erwartet und als zweiten Parameter das Rätsel selbst. Dieses wird in einer geschachtelten 9x9-elementrigen Liste erwartet, in der zu lösende Felder als Variablen definiert sind und festgelegte Felder durch den entsprechenden Wert von einschließlich 1 bis einschließlich 9 angegeben werden.

Beispiel:

```
% https://users.informatik.haw-hamburg.de/~schumann/pub/IS/Praktikum/Praktikum2.pdf
sudoku(sudoku_1, [
    [6,_,_,7,_,_,5,_,_],
    [_,2,8,_,_,_,_,_,_],
    [_,_,_,6,4,_,3,_,_],
    [7,4,_,_,_,_,_,2,_],
    [_,_,1,_,_,_,8,_,_],
    [_,5,_,_,_,_,_,3,7],
    [_,_,3,_,7,6,_,_,_],
    [_,_,_,_,_,_,1,9,_],
    [_,_,4,_,_,5,_,_,8]
]).
```

Zur Lösung eines Rätsels definieren wir nun ein Prädikat **`sudoku/1`**, welche als Parameter die eindeutige Bezeichnung für das Rätsel erwartet (siehe **`sudoku/2`**). Das Prädikat nutzt **`sudoku/2`**, um das Spielfeld zu „laden“ und ruft dann die Unterprädikate **`solveSudoku/1`** (zum Lösen des Rätsels) und **`printSudoku/1`** (zur Ausgabe der Lösung) auf.

```
sudoku(Number) :-
    sudoku(Number, Sudoku),
    solveSudoku(Sudoku),
    printSudoku(Sudoku).
```

Das Unterprädikat **`solveSudoku/1`** prüft nun in jeweils einem Unterprädikat die entsprechenden Contraints ab. Die Unterprädikate werden im Anschluss erläutert.

```
solveSudoku(Sudoku) :-
    checkLength(Sudoku),
    checkDomain(Sudoku),
    checkRows(Sudoku),
    checkCols(Sudoku),
    checkBlocks(Sudoku).
```

Das Unterprädikat **checkLength/1** prüft ob das Rätsel eine gültige Struktur besitzt, das heißt, ob es eine 9x9-Matrix darstellt. Dazu wird zuerst die Anzahl der Zeilen geprüft und anschließend iterativ durch **maplist/2** die Anzahl der Felder einer Zeile.

```
checkLength(Sudoku) :- length(Sudoku, 9), maplist(checkLengthInner, Sudoku).
checkLengthInner(Row) :- length(Row, 9).
```

Das Unterprädikat **checkDomain/1** prüft die Wertebereiche jeder Variablen. Dazu wird zuerst das Spielfeld durch Nutzung von **append/2** geflattet und anschließend geprüft, ob jedes Element der entstandenen eindimensionalen Liste einen Wert von einschließlich 1 bis einschließlich 9 besitzt.

```
checkDomain(Sudoku) :-
    append(Sudoku, SudokuFlatted),
    SudokuFlatted ins 1..9.
```

Das Unterprädikat **checkRows/1** prüft iterativ durch **maplist/2** die **ALL_DISTINCT**-Constraints für jede Zeile. Das **ALL_DISTINCT**-Constraint wird durch das clpfd-Prädikat **all_distinct/1** umgesetzt.

```
checkRows(Sudoku) :- maplist(all_distinct, Sudoku).
```

Das Unterprädikat **checkCols/1** prüft die **ALL_DISTINCT**-Constraints für jede Spalte. Dazu wird zuerst die 9x9-Matrix transponiert und anschließend das Unterprädikat **checkRows/1** genutzt.

```
checkCols(Sudoku) :-
    transpose(Sudoku, SudokuTransposed),
    checkRows(SudokuTransposed).
```

Das Unterprädikat **checkBlocks/1** prüft die **ALL_DISTINCT**-Constraints für jeden Block. Dazu werden zuerst alle 9 Variablen eines Blocks explizit in eine Liste geschrieben. Diese Listen werden nun in einer 9x9-Matrix zusammengefasst und anschließend wird erneut das Unterprädikat **checkRows/1** genutzt.

```
checkBlocks(Sudoku) :-
    Sudoku =
        [[R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9],
         [R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9],
         [R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9],
         [R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9],
         [R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9],
         [R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9],
         [R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9],
         [R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9],
         [R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]],
    SudokuBlock =
        [[R1C1, R1C2, R1C3, R2C1, R2C2, R2C3, R3C1, R3C2, R3C3],
         [R1C4, R1C5, R1C6, R2C4, R2C5, R2C6, R3C4, R3C5, R3C6],
         [R1C7, R1C8, R1C9, R2C7, R2C8, R2C9, R3C7, R3C8, R3C9],
         [R4C1, R4C2, R4C3, R5C1, R5C2, R5C3, R6C1, R6C2, R6C3],
         [R4C4, R4C5, R4C6, R5C4, R5C5, R5C6, R6C4, R6C5, R6C6],
         [R4C7, R4C8, R4C9, R5C7, R5C8, R5C9, R6C7, R6C8, R6C9],
         [R7C1, R7C2, R7C3, R8C1, R8C2, R8C3, R9C1, R9C2, R9C3],
         [R7C4, R7C5, R7C6, R8C4, R8C5, R8C6, R9C4, R9C5, R9C6],
         [R7C7, R7C8, R7C9, R8C7, R8C8, R8C9, R9C7, R9C8, R9C9]],
    checkRows(SudokuBlock).
```

Während der Testphase haben wir festgestellt, dass einige Variablen in der Lösung nicht belegt wurden. Daher belegen wir diese explizit mit dem Prädikat **labeling/2**. Dieses erzwingt eine Belegung der noch nicht belegten Variablen durch Backtracking.
Der erste Parameter ist eine Liste von Optionen, der zweite Parameter eine Liste aller Variablen.

Damit sind alle Unterprädikate des Prädikates **solveSudoku/1** erläutet.
Zur Ausgabe der Lösung nutzen wir das Prädikat **printSudoku/1**, welches iterativ durch **maplist/2** die einzelnen Zeilen der 9x9-Matrix ausgibt.

```
printSudoku(Sudoku) :- maplist(writeln, Sudoku).
```

Die Ausgabe der Lösung sieht damit in unserem Beispiel wie folgt aus:

```
4 ?- sudoku(sudoku_1).
[6,3,9,7,1,8,5,4,2]
[4,2,8,5,3,9,7,1,6]
[1,7,5,6,4,2,3,8,9]
[7,4,6,8,5,3,9,2,1]
[3,9,1,4,2,7,8,6,5]
[8,5,2,9,6,1,4,3,7]
[9,8,3,1,7,6,2,5,4]
[5,6,7,2,8,4,1,9,3]
[2,1,4,3,9,5,6,7,8]
```

3) Optimierung der Performance

3.1) Schritt 1: maplist durch Rekursion ersetzen

```
-checkLength(Sudoku) :- length(Sudoku, 9), maplist(checkLengthInner, Sudoku).
-checkLengthInner(Row) :- length(Row, 9).
+checkLength(Sudoku) :- length(Sudoku, 9), checkLengthInner(Sudoku).
+checkLengthInner([]).
+checkLengthInner([Row|Rest]) :- length(Row, 9), checkLengthInner(Rest).
```

```
-checkRows(Sudoku) :- maplist(all_distinct, Sudoku).
+checkRows([]).
+checkRows([Row|Rest]) :- all_distinct(Row), checkRows(Rest).
```

```
-printSudoku(Sudoku) :- maplist(writeln, Sudoku).
+printSudoku([]).
+printSudoku([Row|Rest]) :- writeln(Row), printSudoku(Rest).
```

Durch diese Änderung konnten **13 Inferenzen** eingespart werden:

Funktion	**Inferenzen vorher**	**Inferenzen nachher**	**Differenz**
checkLength	50	41	-9
checkDomain	4010	4010	0
checkRows	42635	42634	-1
checkCols	214651	214650	-1
checkBlocks	276664	276663	-1
labeling	669	669	0
printSudoku	20	19	-1
Gesamt	**538699**	**538686**	**-13**

3.2) Schritt 2: Rekursionen durch explizierte Unifikation ersetzen

```
-checkLength(Sudoku) :- length(Sudoku, 9), checkLengthInner(Sudoku).
-checkLengthInner([]).
-checkLengthInner([Row|Rest]) :- length(Row, 9), checkLengthInner(Rest).
+checkLength(Sudoku) :-
+     Sudoku =
+          [[_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_],
+           [_,_,_,_,_,_,_,_,_]].
```

```
 checkDomain(Sudoku) :-
-     append(Sudoku, SudokuFlatted),
+     Sudoku =
+          [[R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9],
+           [R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9],
+           [R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9],
+           [R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9],
+           [R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9],
+           [R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9],
+           [R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9],
+           [R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9],
+           [R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]],
+     SudokuFlatted =
+          [R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9,
+           R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9,
+           R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9,
+           R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9,
+           R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9,
+           R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9,
+           R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9,
+           R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9,
+           R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9],
      SudokuFlatted ins 1..9.
```

```
-checkRows([]).
-checkRows([Row|Rest]) :- all_distinct(Row), checkRows(Rest).
+checkRows(Sudoku) :-
+     Sudoku = [Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9],
+     all_distinct(Row1),
+     all_distinct(Row2),
+     all_distinct(Row3),
+     all_distinct(Row4),
+     all_distinct(Row5),
+     all_distinct(Row6),
+     all_distinct(Row7),
+     all_distinct(Row8),
+     all_distinct(Row9).

 checkCols(Sudoku) :-
-     transpose(Sudoku, SudokuTransposed),
+     Sudoku =
+          [[R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9],
+           [R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9],
+           [R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9],
+           [R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9],
+           [R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9],
+           [R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9],
+           [R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9],
+           [R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9],
+           [R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]],
+     SudokuTransposed =
+          [[R1C1, R2C1, R3C1, R4C1, R5C1, R6C1, R7C1, R8C1, R9C1],
+           [R1C2, R2C2, R3C2, R4C2, R5C2, R6C2, R7C2, R8C2, R9C2],
+           [R1C3, R2C3, R3C3, R4C3, R5C3, R6C3, R7C3, R8C3, R9C3],
+           [R1C4, R2C4, R3C4, R4C4, R5C4, R6C4, R7C4, R8C4, R9C4],
+           [R1C5, R2C5, R3C5, R4C5, R5C5, R6C5, R7C5, R8C5, R9C5],
+           [R1C6, R2C6, R3C6, R4C6, R5C6, R6C6, R7C6, R8C6, R9C6],
+           [R1C7, R2C7, R3C7, R4C7, R5C7, R6C7, R7C7, R8C7, R9C7],
+           [R1C8, R2C8, R3C8, R4C8, R5C8, R6C8, R7C8, R8C8, R9C8],
+           [R1C9, R2C9, R3C9, R4C9, R5C9, R6C9, R7C9, R8C9, R9C9]],
      checkRows(SudokuTransposed).
```

```
-printSudoku([]).
-printSudoku([Row|Rest]) :- writeln(Row), printSudoku(Rest).
+printSudoku(Sudoku) :-
+     Sudoku = [Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9],
+     writeln(Row1),
+     writeln(Row2),
+     writeln(Row3),
+     writeln(Row4),
+     writeln(Row5),
+     writeln(Row6),
+     writeln(Row7),
+     writeln(Row8),
+     writeln(Row9).
```

Durch diese Änderung konnten weitere **405 Inferenzen** eingespart werden:

Funktion	**Inferenzen vorher**	**Inferenzen nachher**	**Differenz**
checkLength	41	1	-40
checkDomain	4010	3906	-104
checkRows	42634	42625	-9
checkCols	214650	214417	-233
checkBlocks	276663	276653	-10
labeling	669	669	0
printSudoku	19	10	-9
Gesamt	**538686**	**538281**	**-405**

3.3) Schritt 3: Entfernen von Unterfunktionen

```
sudoku(Number) :-
    sudoku(Number, Sudoku),
    time(solveSudoku(Sudoku)).

solveSudoku(Sudoku) :-
    % check length
        Sudoku =
         [[R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9],
          [R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9],
          [R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9],
          [R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9],
          [R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9],
          [R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9],
          [R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9],
          [R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9],
          [R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]],

    % check domain
    SudokuFlatted =
         [R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9,
          R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9,
          R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9,
          R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9,
          R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9,
          R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9,
          R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9,
          R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9,
          R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9],
    SudokuFlatted ins 1..9,

    % check rows
    Sudoku = [Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9],
    all_distinct(Row1),
    all_distinct(Row2),
    all_distinct(Row3),
    all_distinct(Row4),
    all_distinct(Row5),
    all_distinct(Row6),
    all_distinct(Row7),
    all_distinct(Row8),
    all_distinct(Row9),
```

```
% check cols
all_distinct([R1C1, R2C1, R3C1, R4C1, R5C1, R6C1, R7C1, R8C1, R9C1]),
all_distinct([R1C2, R2C2, R3C2, R4C2, R5C2, R6C2, R7C2, R8C2, R9C2]),
all_distinct([R1C3, R2C3, R3C3, R4C3, R5C3, R6C3, R7C3, R8C3, R9C3]),
all_distinct([R1C4, R2C4, R3C4, R4C4, R5C4, R6C4, R7C4, R8C4, R9C4]),
all_distinct([R1C5, R2C5, R3C5, R4C5, R5C5, R6C5, R7C5, R8C5, R9C5]),
all_distinct([R1C6, R2C6, R3C6, R4C6, R5C6, R6C6, R7C6, R8C6, R9C6]),
all_distinct([R1C7, R2C7, R3C7, R4C7, R5C7, R6C7, R7C7, R8C7, R9C7]),
all_distinct([R1C8, R2C8, R3C8, R4C8, R5C8, R6C8, R7C8, R8C8, R9C8]),
all_distinct([R1C9, R2C9, R3C9, R4C9, R5C9, R6C9, R7C9, R8C9, R9C9]),

% check blocks
all_distinct([R1C1, R1C2, R1C3, R2C1, R2C2, R2C3, R3C1, R3C2, R3C3]),
all_distinct([R1C4, R1C5, R1C6, R2C4, R2C5, R2C6, R3C4, R3C5, R3C6]),
all_distinct([R1C7, R1C8, R1C9, R2C7, R2C8, R2C9, R3C7, R3C8, R3C9]),
all_distinct([R4C1, R4C2, R4C3, R5C1, R5C2, R5C3, R6C1, R6C2, R6C3]),
all_distinct([R4C4, R4C5, R4C6, R5C4, R5C5, R5C6, R6C4, R6C5, R6C6]),
all_distinct([R4C7, R4C8, R4C9, R5C7, R5C8, R5C9, R6C7, R6C8, R6C9]),
all_distinct([R7C1, R7C2, R7C3, R8C1, R8C2, R8C3, R9C1, R9C2, R9C3]),
all_distinct([R7C4, R7C5, R7C6, R8C4, R8C5, R8C6, R9C4, R9C5, R9C6]),
all_distinct([R7C7, R7C8, R7C9, R8C7, R8C8, R8C9, R9C7, R9C8, R9C9]),

% print rows
writeln(Row1),
writeln(Row2),
writeln(Row3),
writeln(Row4),
writeln(Row5),
writeln(Row6),
writeln(Row7),
writeln(Row8),
writeln(Row9).
```

Durch diese Änderung konnten weitere **7 Inferenzen** eingespart werden:

Funktion	**Inferenzen vorher**	**Inferenzen nachher**	**Differenz**
checkLength	1		
checkDomain	3906		
checkRows	42625		
checkCols	214417		
checkBlocks	276653		
labeling	699		
printSudoku	10		
Gesamt	**538311**	**538304**	**-7**

Die Lösung stellt unsere endgültige Lösung dar.

3.4) Verworfene Schritte

3.4.1) Änderung 1: Ohne Nutzung von Flatten

```
    % check domain
-   SudokuFlatted =
-       [R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9,
-        R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9,
-        R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9,
-        R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9,
-        R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9,
-        R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9,
-        R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9,
-        R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9,
-        R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9],
-   SudokuFlatted ins 1..9,
+   Sudoku = [Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9],
+   Row1 ins 1..9,
+   Row2 ins 1..9,
+   Row3 ins 1..9,
+   Row4 ins 1..9,
+   Row5 ins 1..9,
+   Row6 ins 1..9,
+   Row7 ins 1..9,
+   Row8 ins 1..9,
+   Row9 ins 1..9,
```

Diese Änderung wurde verworfen, da das zeilenweise Prüfen der Wertebereiche bedeutend aufwendiger ist, als die Wertebereiche der Variablen als eine geflattete Liste zu prüfen (**+280 Inferenzen**!).

	Inferenzen vorher	**Inferenzen nachher**	**Differenz**
Gesamt	**538304**	**538584**	**+280**

3.4.2) Änderung 2: Ohne Unifikation der Zeilen

```
    % check rows
-   Sudoku = [Row1, Row2, Row3, Row4, Row5, Row6, Row7, Row8, Row9],
-   all_distinct(Row1),
-   all_distinct(Row2),
-   all_distinct(Row3),
-   all_distinct(Row4),
-   all_distinct(Row5),
-   all_distinct(Row6),
-   all_distinct(Row7),
-   all_distinct(Row8),
-   all_distinct(Row9),
+   all_distinct([R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9]),
+   all_distinct([R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9]),
+   all_distinct([R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9]),
+   all_distinct([R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9]),
+   all_distinct([R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9]),
+   all_distinct([R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9]),
+   all_distinct([R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9]),
+   all_distinct([R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9]),
+   all_distinct([R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]),
```

```
    % print sudoku
-   writeln(Row1),
-   writeln(Row2),
-   writeln(Row3),
-   writeln(Row4),
-   writeln(Row5),
-   writeln(Row6),
-   writeln(Row7),
-   writeln(Row8),
-   writeln(Row9).
+   writeln([R1C1, R1C2, R1C3, R1C4, R1C5, R1C6, R1C7, R1C8, R1C9]),
+   writeln([R2C1, R2C2, R2C3, R2C4, R2C5, R2C6, R2C7, R2C8, R2C9]),
+   writeln([R3C1, R3C2, R3C3, R3C4, R3C5, R3C6, R3C7, R3C8, R3C9]),
+   writeln([R4C1, R4C2, R4C3, R4C4, R4C5, R4C6, R4C7, R4C8, R4C9]),
+   writeln([R5C1, R5C2, R5C3, R5C4, R5C5, R5C6, R5C7, R5C8, R5C9]),
+   writeln([R6C1, R6C2, R6C3, R6C4, R6C5, R6C6, R6C7, R6C8, R6C9]),
+   writeln([R7C1, R7C2, R7C3, R7C4, R7C5, R7C6, R7C7, R7C8, R7C9]),
+   writeln([R8C1, R8C2, R8C3, R8C4, R8C5, R8C6, R8C7, R8C8, R8C9]),
+   writeln([R9C1, R9C2, R9C3, R9C4, R9C5, R9C6, R9C7, R9C8, R9C9]).
```

Diese Änderung wurde verworfen, weil sie keine Optimierung der Inferenzen mit sich bringt, dafür aber duplizierten Code enthält, welcher immer als fehleranfällig einzustufen ist.

	Inferenzen vorher	**Inferenzen nachher**	**Differenz**
Gesamt	**538304**	**538304**	**0**