Dimitri Meier, Saeed Shanidar, Andreas Berks

dimitri.meier@haw-hamburg.de,  
saeed.shanidar@haw-hamburg.de,  
andreas.berks@haw-hamburg.de

ISP Aufgabe 2

Gruppe3

Inhaltsverzeichnis

[1) Constraint-Netz 2](#_Toc481071137)

[1.1) Schritt 1: Definieren der Variablen 2](#_Toc481071138)

[1.2) Schritt 2: Definieren der Wertebereiche für Variablen 3](#_Toc481071139)

[1.3) Schritt 3: Constraints für Spalten 4](#_Toc481071140)

[1.4) Schritt 4: Constraints für Zeilen 5](#_Toc481071141)

[1.5) Schritt 5: Constraints für Blöcke 6](#_Toc481071142)

[1.6) Gesamtes Contraint-Netz 7](#_Toc481071143)

[2) Lösung in Prolog 8](#_Toc481071144)

[3) Optimierung der Performance 11](#_Toc481071145)

[3.1) Schritt 1: maplist durch Rekursion ersetzen 11](#_Toc481071146)

[3.2) Schritt 2: Rekursionen durch explizierte Unifikation ersetzen 12](#_Toc481071147)

[3.3) Schritt 3: Entfernen von Unterfunktionen 15](#_Toc481071148)

[3.4) Verworfene Schritte 17](#_Toc481071149)

[3.4.1) Änderung 1: Ohne Nutzung von Flatten 17](#_Toc481071150)

[3.4.2) Änderung 2: Ohne Unifikation der Zeilen 18](#_Toc481071151)

# Constraint-Netz

## Schritt 1: Definieren der Variablen

Wir definieren für jedes Feld des Sudoku-Rätsels eine Variable. Wir bezeichnen diese Variablen als **si,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).



## Schritt 2: Definieren der Wertebereiche für Variablen

Der Wertebereich jeder Variablen **si,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.

## 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.



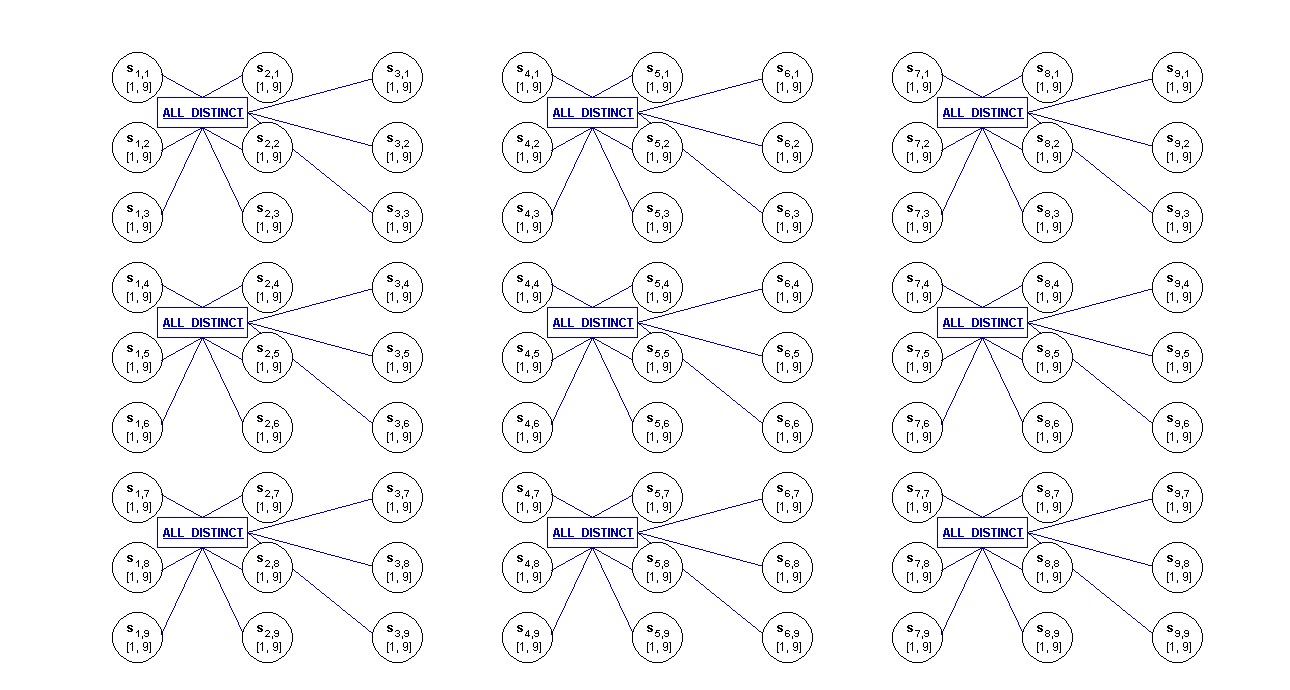
## Schritt 4: Constraints für Zeilen

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



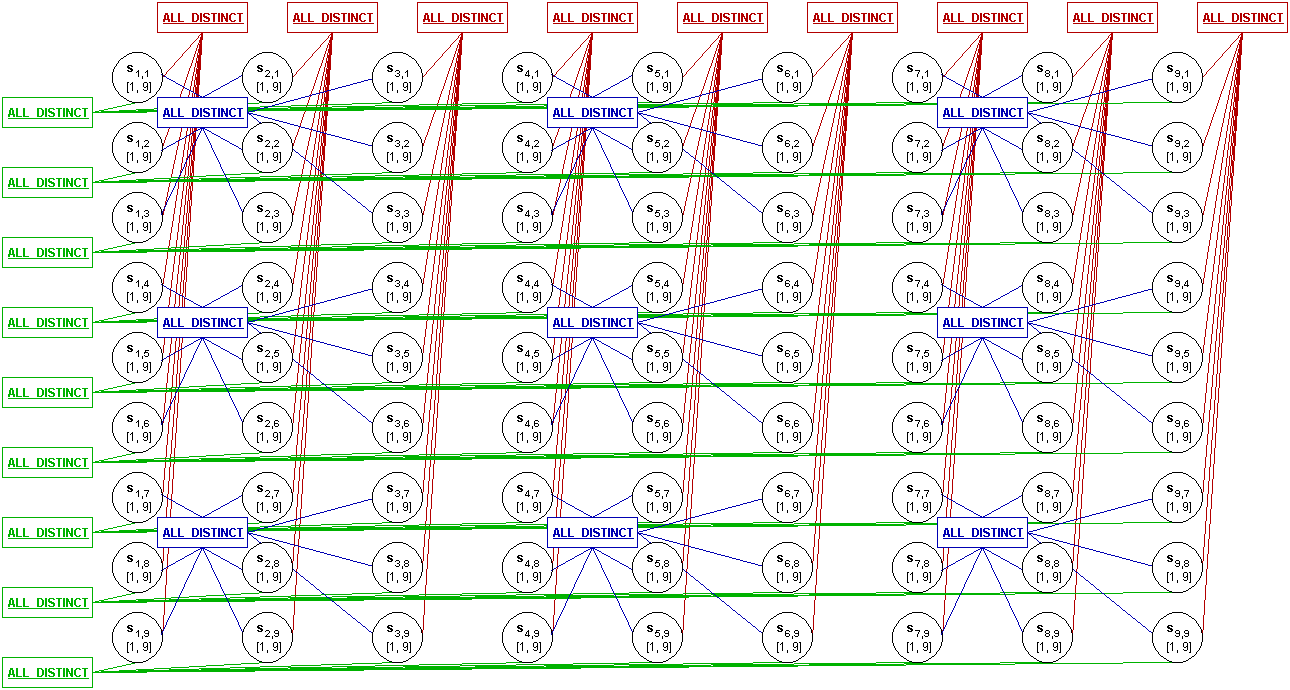
## Schritt 5: Constraints für Blöcke

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



## Gesamtes Contraint-Netz

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



# 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:**



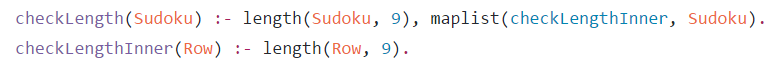
Zur Lösung eines Rätsels definieren wir nun ein Prädikat **sudoku/1**, welche als Parameter die eindeutige Bezeichnung für das Rätsel erwartet (siehe **sudoku/2**). Das Prädikat nutzt **sudoku/2**, um das Spielfeld zu „laden“ und ruft dann die Unterprädikate **solveSudoku/1** (zum Lösen des Rätsels)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



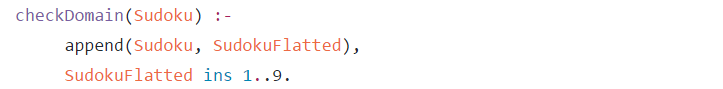
Das Unterprädikat **solveSudoku/1** prüft nun in jeweils einem Unterprädikat die entsprechenden Contraints ab. Die Unterprädikate werden im Anschluss erläutert.



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.



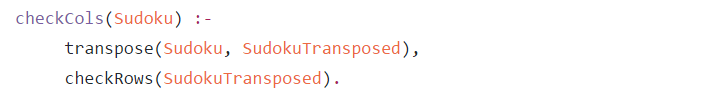
Das Unterprädikat **checkDomain/1** prüft die Wertebereiche jeder Variablen. Dazu wird zuerst das Spielfeld durch Nutzung von **append/2** geflattet und anschließend geprüft, ob jedes Element der entstandenen eindimensionalen Liste einen Wert von einschließlich 1 bis einschließlich 9 besitzt.



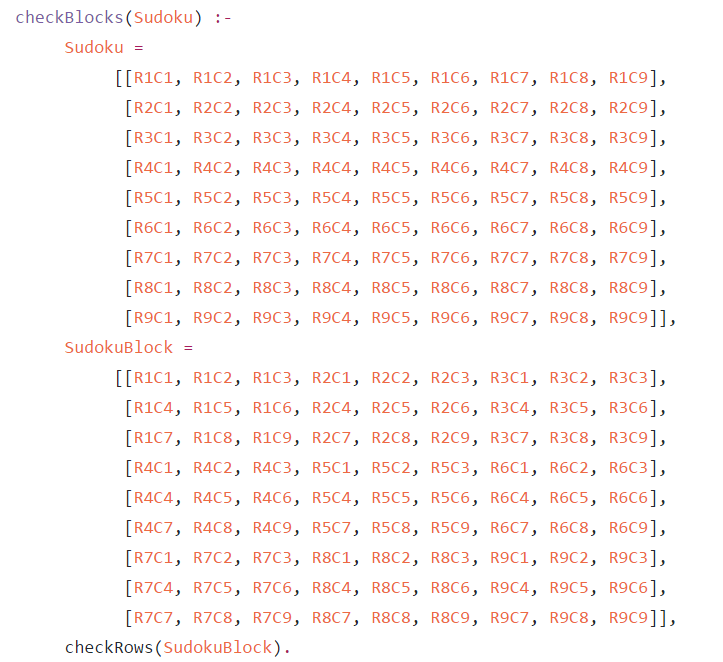
Das Unterprädikat **checkRows/1** prüft iterativ durch **maplist/2** die  
**ALL\_DISTINCT**-Constraints für jede Zeile. Das **ALL\_DISTINCT**-Constraint wird durch das  
clpfd-Prädikat **all\_distinct/1** umgesetzt.



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.



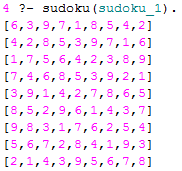
Das Unterprädikat **checkBlocks/1** prüft die **ALL\_DISTINCT**-Constraints für jeden Block. Dazu werden zuerst alle 9 Variablen eines Blocks explizit in eine Liste geschrieben. Diese Listen werden nun in einer 9x9-Matrix zusammengefasst und anschließend wird erneut das Unterprädikat **checkRows/1**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



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.

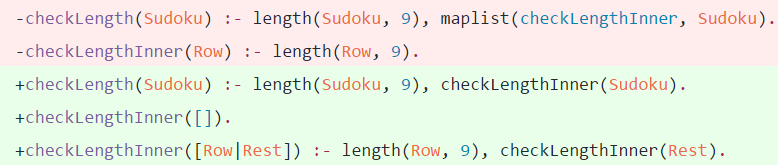


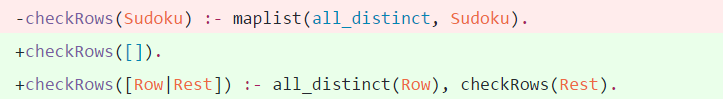
Die Ausgabe der Lösung sieht damit in unserem Beispiel wie folgt aus:

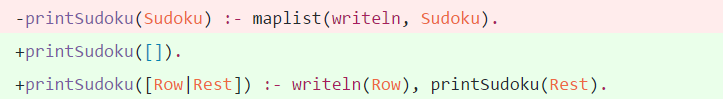


# Optimierung der Performance

## 3.1) Schritt 1: maplist durch Rekursion ersetzen



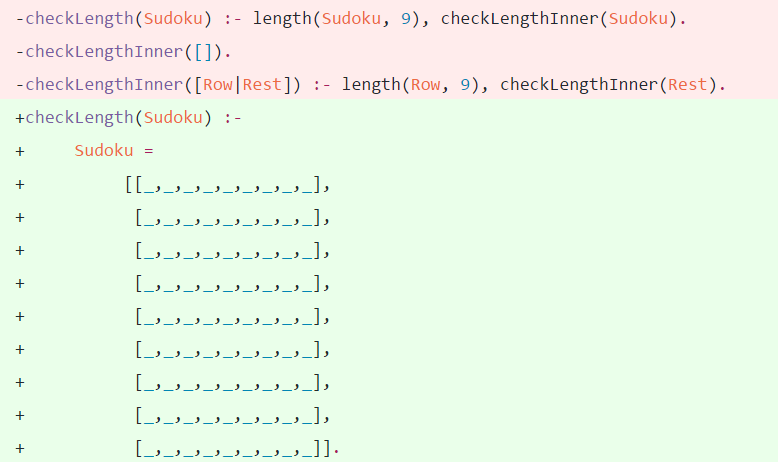


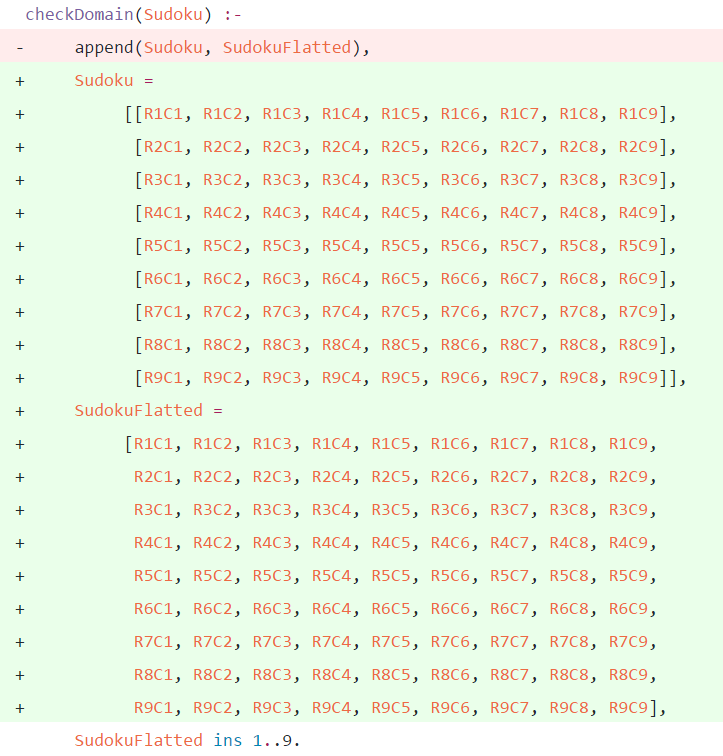


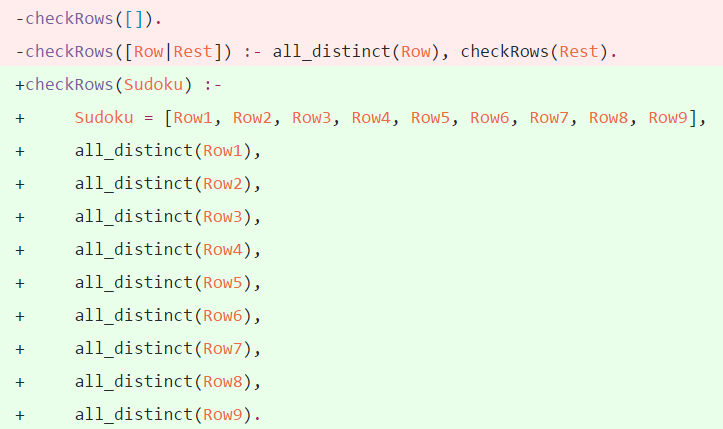
Durch diese Änderung konnten **13 Inferenzen** eingespart werden:

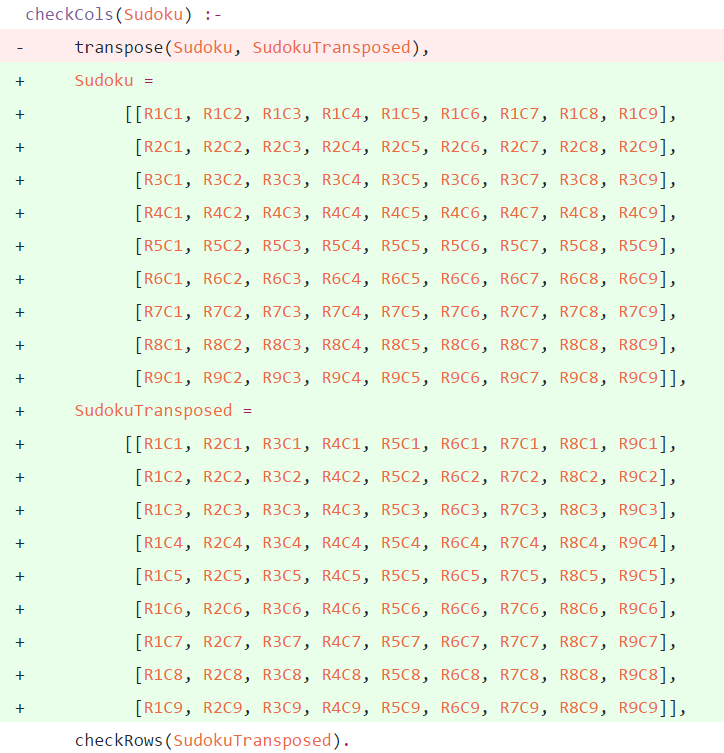
|  |  |  |  |
| --- | --- | --- | --- |
| **Funktion** | **Inferenzen vorher** | **Inferenzen nachher** | **Differenz** |
| checkLength | 50 | 41 | -9 |
| checkDomain | 4010 | 4010 | 0 |
| checkRows | 42635 | 42634 | -1 |
| checkCols | 214651 | 214650 | -1 |
| checkBlocks | 276664 | 276663 | -1 |
| printSudoku | 20 | 19 | -1 |
| **Gesamt** | **538030** | **538017** | **-13** |

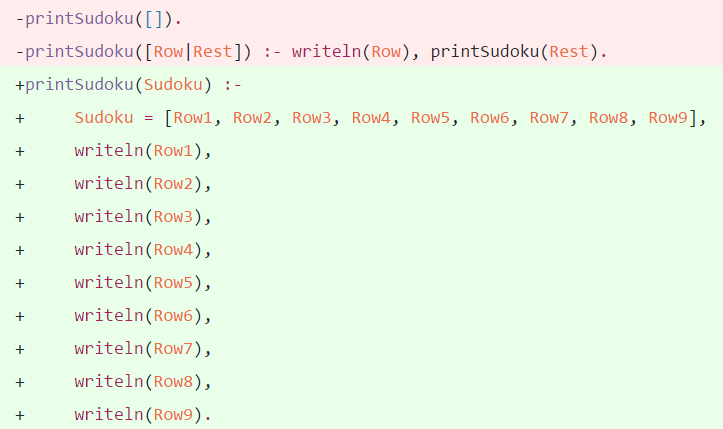
## 3.2) Schritt 2: Rekursionen durch explizierte Unifikation ersetzen







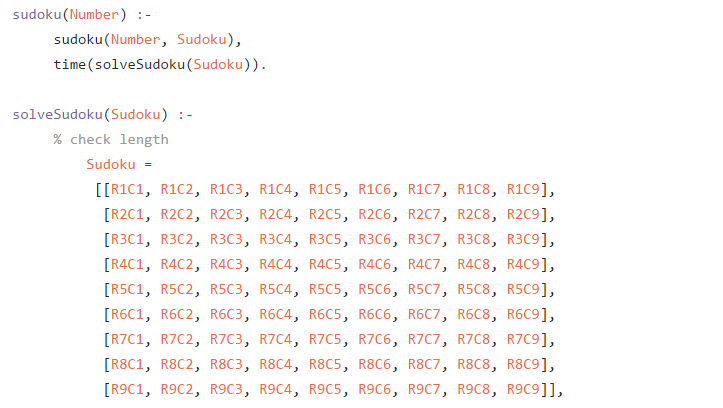


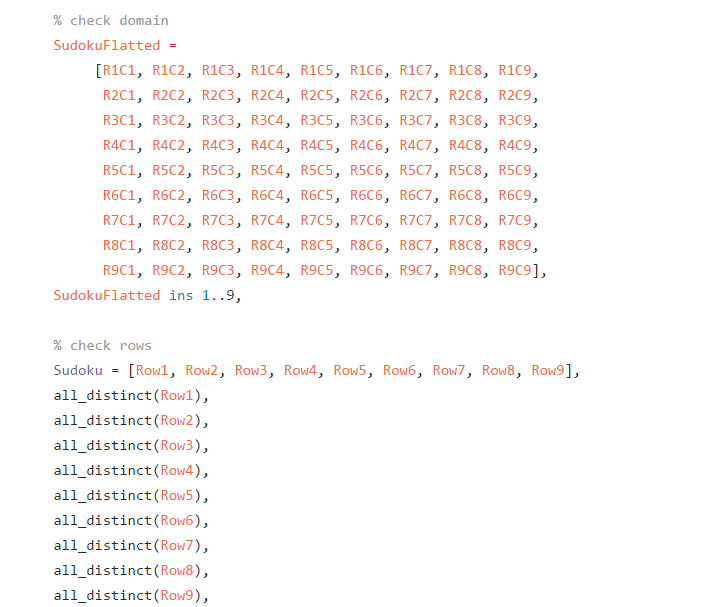


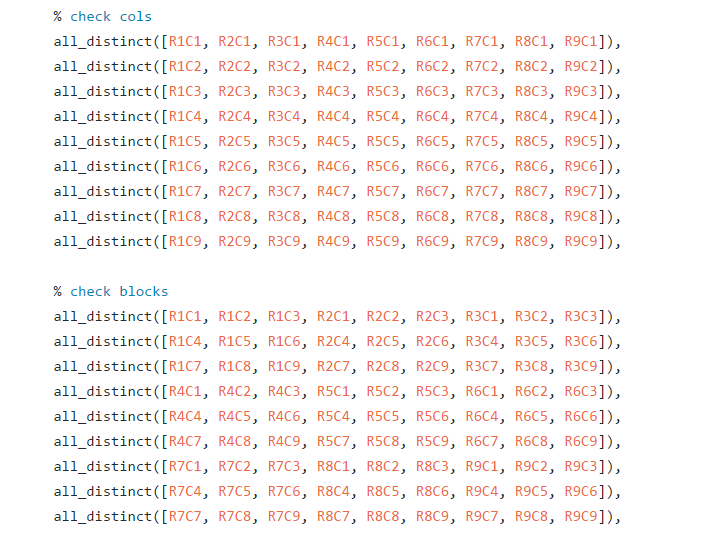
Durch diese Änderung konnten weitere **405 Inferenzen** eingespart werden:

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |
| printSudoku | 19 | 10 | -9 |
| **Gesamt** | **538017** | **537612** | **-405** |

## 3.3) Schritt 3: Entfernen von Unterfunktionen









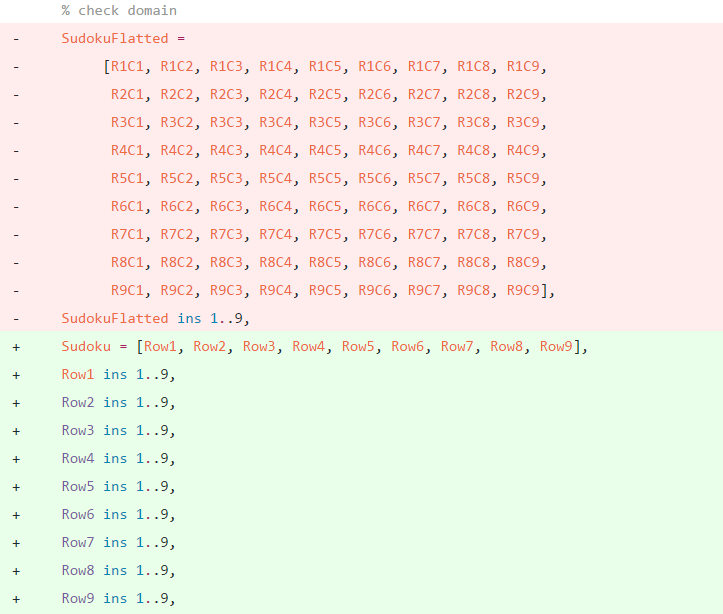
Durch diese Änderung konnten weitere **7 Inferenzen** eingespart werden:

|  |  |  |  |
| --- | --- | --- | --- |
| **Funktion** | **Inferenzen vorher** | **Inferenzen nachher** | **Differenz** |
| checkLength | 1 |  |  |
| checkDomain | 3906 |  |  |
| checkRows | 42625 |  |  |
| checkCols | 214417 |  |  |
| checkBlocks | 276653 |  |  |
| printSudoku | 10 |  |  |
| **Gesamt** | **537612** | **537605** | **-7** |

**Die Lösung stellt unsere endgültige Lösung dar.**

## 3.4) Verworfene Schritte

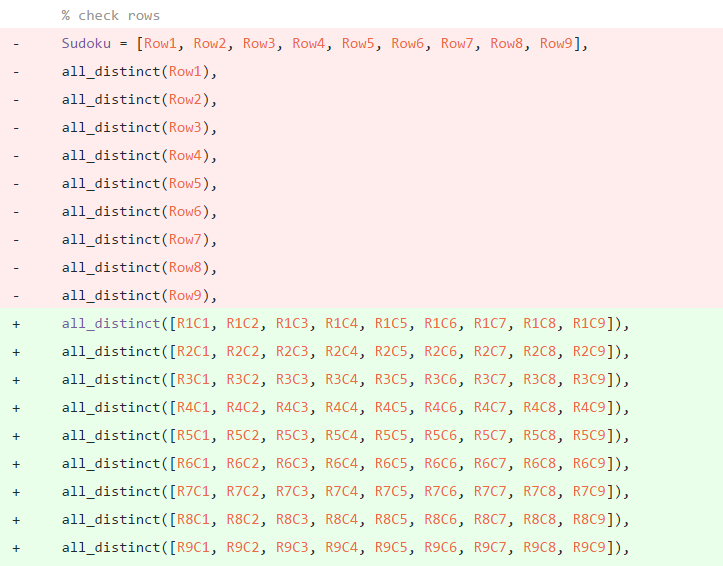
### 3.4.1) Änderung 1: Ohne Nutzung von Flatten

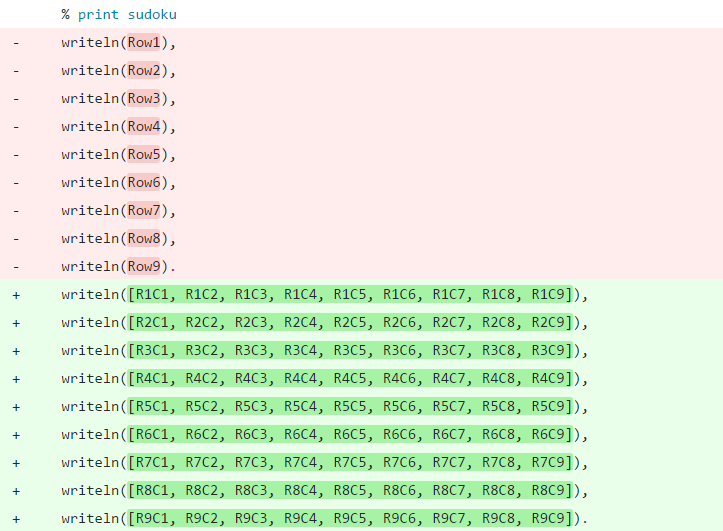


Diese Änderung wurde verworfen, da das zeilenweise Prüfen der Wertebereiche bedeutend aufwendiger ist, als die Wertebereiche der Variablen als eine geflattete Liste zu prüfen  
(+**280 Inferenzen**!).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Inferenzen vorher** | **Inferenzen nachher** | **Differenz** |
| **Gesamt** | **537605** | **537885** | **+280** |

### 3.4.2) Änderung 2: Ohne Unifikation der Zeilen





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** | **537605** | **537605** | **0** |