

# Sudoku mit SMT

## Formale Methoden und Werkzeuge WS24/25 - Oscar Friske

Dieses Projekt implementiert einen Sudoku Solver, sowie einen Generator für neue Sudoku Instanzen. Zur Modellierung des SMT Problems wurde die Bibliothek "hasmtlib" sowie Haskell als Hostsprache genutzt.

## SMT Spezifikationen

- Solver: Z3
- Logik: QF\_LIA - linear integer arithmetic  
QF\_LIA bietet alles, was für die Modellierung des Sudokus benötigt wird. Zur Darstellung des Problems reichen ganze Zahlen, sowohl für die Werte der Felder (1..9) als auch die Koordinaten (0..8), aus. Desweiteren werden hauptsächlich Listen auf die Einzigartigkeit ihrer Element untersucht. Auch hierfür reichen die linearen Constraints, welche QF\_LIA zur Verfügung stellt. Somit können wir mit einer verhältnismäßig kleinen Logik arbeiten, was sich auch auf die Performance positiv auswirkt.

## Implementierung

Im wesentlichen wird das Sudoku Problem als 9x9 Matrix und die einzelnen Werte als Tupel von Wert und Koordinate implementiert.

```
board <- replicateM 9 $ replicateM 9 $ var @IntSort

let setCells = [(n, (row,column)), ...]
```

Das Projekt ist grundsätzlich in 2 Teile unterteilt:

### (1) Solve.hs

Dieser Teil implementiert den eigentlichen Lösungsalgorithmus. Dabei wird das Problem in SMT modelliert und anschließend von z3 gelöst. Für die Modellierung bzw. für das Spiel an sich gibt es 3 wesentliche Constraints:

#### (a) Row Constraint

Jede Zeile muss genau die Zahlen 1-9 enthalten (keine Doppelungen)

Dies ist bei der gegebenen Datenstruktur leicht umsetzbar, da das Feld bereits als Liste von Zeilen vorliegt. Durch das SMT Keyword "distinct" kann die Eigenschaft leicht überprüft werden.

#### (b) Column Constraint

Jede Spalte muss genau die Zahlen 1-9 enthalten (keine Doppelungen)

Auch dieses Constraint folgt grundsätzlich dem gleichen Aufbau des Row Constraints. Allerdings muss hierfür die Matrix zunächst transponiert werden. Somit erhalten wir eine Liste von Spalten.

#### (c) Subgrid Constraint

Jedes 3x3 Feld muss genau die Zahlen 1-9 enthalten (keine Doppelungen)

Hierfür müssen die 3x3 Subgrids vorher in eigene Listen gefiltert werden. Dafür werden

beginnt bei einer Koordiante "(row, column)" und die nächsten 3 Zeilen jeweils die nächsten 3 Spalten gefiltert

Des weiteren gibt es zwei Constraint, welchen den Zahlenraum einschränken (1-9), ein Constraint, welches die vordefinierten Felder voraussetzt und ein optionales Constraint, welches eine Lösung ausschließt. Dies wird benötigt um später eine Lösung auf ihre Einzigartigkeit zu überprüfen und kommt beim Generieren eines neuen Rätsels zum Einsatz

## (2) Generate.hs

Auf Basis des Solvers können neue Instanzen eines Sudokus generiert werden. Die grundsätzliche Idee ist dabei folgende:

- (1) Generiere  $n$  zufällige Felder und Löse basierend auf diesen ein Sudokubrett.  
Sollten die zufälligen Felder eine Lösung des Sudokus verhindern, werden neue Zufallsfelder generiert. Hier muss abgewogen werden zwischen der Lösungsgeschwindigkeit des Solvers (mehr vorgegebene Werte beschleunigen das Lösen) und der möglichen Anzahl der Neugenerierungen, da die Felder keine gültige Lösung zulassen. 10 Zufällige Anfangswerte haben sich als gutes Mittelmaß herausgestellt.
- (2) Entferne ein zufälliges Feld  
Die Lösbarkeit bleibt dadurch erhalten.
- (3) Überprüfe, ob das Sudoku weiterhin nur eine mögliche Lösung hat <sup>1</sup>.
  - i. Falls ja, gehe zu Schritt (2)
  - ii. Falls nein, mache den letzten Schritt rückgängig und gehe zu (2).  
Das ist allerdings nur  $n$  mal möglich. Sollte nach  $n$  Wiederholungen keine bessere Instanz (mit weniger Feldern) gefunden wurden sein, beende die Generierung. Mehr Wiederholungen können das Ergebnis verbessern, wobei sich 15 erlaubte Wiederholungen als gute Anzahl herausgestellt haben. Je mehr Wiederholungen erlaubt sind, desto länger dauert auch die Generierung. Dies macht sich vor allem bemerkbar, wenn nur noch 30 Felder vorgegeben sind.

<sup>1</sup> Um zu überprüfen, ob das aktuelle Brett nur eine mögliche Lösung hat, existiert die `checkIfUniqueSolution` Funktion. Dabei durchläuft das aktuelle (unvollständige Brett) den Solver zwei mal. Im ersten Durchlauf wird das Brett auf seine generelle Lösbarkeit untersucht. Das Ergebnis wird anschließend im zweiten Durchlauf dem Solver übergeben mit der Bedingung, dass dies keine Lösung sein darf. Wird dennoch eine Lösung gefunden, war die Lösung des ersten Durchlaufs nicht die einzige. Wird hingegen keine weitere Lösung gefunden weiß man, dass das Sudoku nur eine Lösung hat.

## Anwendung

Zum generieren eines neuen Sudokus kann das Programm über "cabal build" gebaut und anschließend über "cabal run" gestartet werden. Z3 läuft dabei nicht im debug modus.

## Auswertung

Standartmäßig läuft das Programm mit 10 zufälligen Anfangswerten und 20 erlaubten Wiederholungen bei der Reduzierung des Sudokus. Dadurch ergeben sich neue Sudoku Instanzen mit 23-27 vorgegebenen Felder (im Schnitt ca. 24-25) Dies kann auf Grund des Zufallsaspektes variieren, erlaubt aber dadurch die Generierung sehr vieler neuer Rätsel.