Sudoku as a Satisfiability-Problem

Andreas Klappenecker

Sudoku Constraints

Sudoku is an $n^2 \times n^2$ array with some fields containing entries with numbers in the range $[1..n^2]$. The goal is to find numbers from $[1..n^2]$ for each of the empty fields such that

- each column contains all numbers from 1 to n^2

- each row contains all numbers from 1 to n^2

- each of the n^2 blocks contains all numbers from 1 to n^2

The goal is to decide whether this can be done or not.

Reduce Sudoku to SAT

Goal: Translate a Sudoku problem into a propositional formula that is satisfiable if and only if the Sudoku has a solution.

Formalizing the Constraints

We define a predicate valid that is true if and only if the n^2 array entries specified in its arguments contain all numbers in $R=[1..n^2]$.

Suppose that $x_1, x_2, \ldots, x_{n^2}$ are entries of the array. Then

$$\text{valid}(x_1, x_2, \ldots, x_{n^2}) = \forall\, d \in R\ \exists\, i \in R\ (x_i = d)$$

Sudoku Constraints Predicate

$N = n^2$, $B = \{1, n+1, 2n+1, \ldots, n^2 - (n-1)\}$ beginning of blocks

$\text{sudoku}(\ (x_{ij})_{i,j \in R}\ ) =$

$\forall\ (i \text{ in } R)\ \text{valid}(x_{i1}, \ldots, x_{iN}) \ \wedge\ \forall\ (j \text{ in } R)\ \text{valid}(x_{1j}, \ldots, x_{Nj}) \ \wedge$

$\forall\ (i,j \text{ in } B)\ \text{valid}(\ x_{ij}, x_{i(j+1)}, \ldots, x_{i(j+n-1)},$

$x_{(i+1)j}, x_{(i+1)(j+1)}, \ldots, x_{(i+1)(j+n-1)}, \ldots,$

$x_{(i+n-1)j}, x_{(i+n-1)(j+1)}, \ldots, x_{(i+n-1)(j+n-1)})$

Rows
Columns
Blocks

Variations

The previous approach can be used as an input to a theorem prover such as Isabelle/HOL.

There are variations that express everything over a boolean domain. This leads to many more clauses, but apparently this can be efficient.

Encoding Sudoku

Choose n^2 boolean variables for each entry of the array.

We denote by p_{ijd} the truth value of $x_{ij} = d$.

Cell (i,j) takes a value in R: $\exists$ (d in R) p_{ijd}

Cell (i,j) takes at most one value:

$\forall d \; \forall d' \; (1 <= d < d' <= n^2) \rightarrow \neg(p(i,j,d) \wedge p(i,j,d'))$

[This translates into $C(n^2,2)$ clauses for the SAT solver]

Further Reading

The approach outlined here followed

T. Weber, A SAT-based Sudoku Solver.

There are ways to optimize the CNF boolean solvers.