A SAT-based Sudoku Solver*

Tjark Weber

Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
webertj@in.tum.de

Abstract. This paper presents a SAT-based *Sudoku* solver. A *Sudoku* is translated into a propositional formula that is satisfiable if and only if the *Sudoku* has a solution. A standard SAT solver can then be applied, and a solution for the *Sudoku* can be read off from the satisfying assignment returned by the SAT solver. No coding was necessary to implement this solver: The translation into propositional logic is provided by a framework for finite model generation available in the Isabelle/HOL theorem prover. Only the constraints on a *Sudoku* solution had to be specified in the prover's logic.

1 Introduction

Sudoku, also known as *Number Place* in the United States, is a placement puzzle. Given a grid – most frequently a 9×9 grid made up of 3×3 subgrids called "regions" – with various digits given in some cells (the "givens"), the aim is to enter a digit from 1 through 9 in each cell of the grid so that each row, column and region contains only one instance of each digit. Fig. 1 shows a *Sudoku* on the left, along with its unique solution on the right [12]. Note that other symbols (e.g. letters, icons) could be used instead of digits, as their arithmetic properties are irrelevant in the context of *Sudoku*. This is currently a rather popular puzzle that is featured in a number of newspapers and puzzle magazines [1, 3, 9].

Several *Sudoku* solvers are available already [6, 10]. Since there are more than $6 \cdot 10^{21}$ possible *Sudoku* grids [5], a naïve backtracking algorithm would be infeasible. *Sudoku* solvers therefore combine backtracking with – sometimes complicated – methods for constraint propagation. In this paper we propose a SAT-based approach: A *Sudoku* is translated into a propositional formula that is satisfiable if and only if the *Sudoku* has a solution. The propositional formula is then presented to a standard SAT solver, and if the SAT solver finds a satisfying assignment, this assignment can readily be transformed into a solution for the original *Sudoku*. The presented translation into SAT is simple, and requires minimal implementation effort since we can make use of an existing framework for finite model generation [11] available in the Isabelle/HOL [8] theorem prover.

* This work was supported by the PhD program Logic in Computer Science of the German Research Foundation.

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

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

Fig. 1. *Sudoku* example and solution

2 Implementation in Isabelle/HOL

An implementation of the *Sudoku* rules in the interactive theorem prover Isabelle/HOL is straightforward. Digits are modelled by a datatype with nine elements $1, \ldots, 9$. We say that nine grid cells $x_1, \ldots, x_9$ are *valid* iff they contain every digit.

Definition 1 (valid).

$$\mathrm{valid}(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) \equiv \bigwedge_{d=1}^{9} \bigvee_{i=1}^{9} x_i = d.$$

Labeling the 81 cells of a 9×9 grid as shown in Fig. 2, we can now define what it means for them to be a *Sudoku* solution: each row, column and region must be valid.

Definition 2 (sudoku).

$$\begin{aligned} \mathrm{sudoku}(\{x_{ij}\}_{i,j\in\{1,\ldots,9\}}) \equiv & \bigwedge_{i=1}^{9} \mathrm{valid}(x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}, x_{i6}, x_{i7}, x_{i8}, x_{i9}) \\ & \wedge \bigwedge_{j=1}^{9} \mathrm{valid}(x_{1j}, x_{2j}, x_{3j}, x_{4j}, x_{5j}, x_{6j}, x_{7j}, x_{8j}, x_{9j}) \\ & \wedge \bigwedge_{i,j\in\{1,4,7\}} \mathrm{valid}(x_{ij}, x_{i(j+1)}, x_{i(j+2)}, x_{(i+1)j}, x_{(i+1)(j+1)}, x_{(i+1)(j+2)}, \\ & \qquad x_{(i+2)j}, x_{(i+2)(j+1)}, x_{(i+2)(j+2)}). \end{aligned}$$

The next section describes the translation of these definitions into propositional logic.

3 Translation to SAT

We encode a *Sudoku* by introducing 9 Boolean variables for each cell of the 9×9 grid, i.e. $9^3 = 729$ variables in total. Each Boolean variable p_{ij}^d (with

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	x_{57}	x_{58}	x_{59}
x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	x_{67}	x_{68}	x_{69}
x_{71}	x_{72}	x_{73}	x_{74}	x_{75}	x_{76}	x_{77}	x_{78}	x_{79}
x_{81}	x_{82}	x_{83}	x_{84}	x_{85}	x_{86}	x_{87}	x_{88}	x_{89}
x_{91}	x_{92}	x_{93}	x_{94}	x_{95}	x_{96}	x_{97}	x_{98}	x_{99}

Fig. 2. *Sudoku* grid

$1 \leq i, j, d \leq 9$) represents the truth value of the equation $x_{ij} = d$. A clause

$$\bigvee_{d=1}^{9} p_{ij}^{d}$$

ensures that the cell x_{ij} denotes one of the nine digits, and 36 clauses

$$\bigwedge_{1 \leq d < d' \leq 9} \neg p_{ij}^{d} \vee \neg p_{ij}^{d'}$$

make sure that the cell does not denote two different digits at the same time.

Since there are just as many digits as cells in each row, column, and region, Def. 1 is equivalent to the following characterization of validity, stating that the nine grid cells $x_1, \ldots, x_9$ contain distinct values.

Lemma 1 (Equivalent characterization of validity).

$$\text{valid}(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) \iff \bigwedge_{1 \leq i < j \leq 9} x_i \neq x_j$$

$$\iff \bigwedge_{1 \leq i < j \leq 9} \bigwedge_{d=1}^{9} x_i \neq d \vee x_j \neq d.$$

The latter characterization turns out to be much more efficient when translated to SAT. While Def. 1, when translated directly, produces 9 clauses with 9 literals each (one literal for each equation), the formula given in Lemma 1 is translated into 324 clauses (9 clauses for each of the 36 inequations $x_i \neq x_j$), but each clause of length 2 only. This allows for more unit propagation [14] at the Boolean level, which – in terms of the original *Sudoku* – corresponds to cross-hatching [12] of digits, a technique that is essential to reduce the search space.

The 9 clauses obtained from a direct translation of Def. 1 could still be used as well; unit propagation on these clauses would correspond to counting the digits 1 – 9 in regions, rows, and columns to identify missing numbers. However, in our experiments we did not experience any speedup by including these clauses.

This gives us a total of 11745 clauses: 81 definedness clauses of length 9, $81 \cdot 36$ uniqueness clauses of length 2, and $27 \cdot 324$ validity clauses,[1] again of length 2. However, we do not need to introduce Boolean variables for cells whose value is given in the original *Sudoku*, and we can omit definedness and uniqueness clauses for these cells as well as some of the validity clauses – therefore the total number of variables and clauses used in the encoding of a *Sudoku* with givens will be less than 729 and 11745, respectively.

Note that our encoding already yields a propositional formula in conjunctive normal form (CNF). Therefore conversion into DIMACS CNF format [4] – the standard input format used by most SAT solvers – is trivial. Isabelle can search for a satisfying assignment using either an internal DPLL-based [2] SAT solver, or write the formula to a file in DIMACS format and execute an external solver. We have employed zChaff [7] to find the solution to various *Sudoku* classified as "hard" by their respective authors (see Fig. 3 for an example), and in every case the runtime was only a few milliseconds.

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

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

Fig. 3. hard *Sudoku* example and solution

4 Concluding Remarks

We have presented a straightforward translation of a *Sudoku* into a propositional formula. The translation can easily be generalized from 9×9 grids to grids of arbitrary dimension. It is polynomial in the size of the grid, and since *Sudoku* is NP-complete [13], no algorithm with better complexity is known. The translation, combined with a state-of-the-art SAT solver, is also practically successful: 9×9 *Sudoku* puzzles are solved within milliseconds.

Traditionally the givens in a *Sudoku* are chosen so that the puzzle's solution is unique; nevertheless our algorithm can be extended to enumerate all possi-

[1] This number includes some duplicates, caused by the overlap between rows/columns and regions: certain cells that must be distinct because they belong to the same row (or column) must also be distinct because they belong to the same region.

ble solutions (by explicitly disallowing all solutions found so far, and perhaps using an incremental SAT solver that allows adding clauses on-the-fly to avoid searching through the same search space multiple times).

Particularly remarkable is the fact that our solver, while it can certainly compete with hand-crafted *Sudoku* solvers, some of which use rather complex patterns and search heuristics, required very little implementation effort. Aside from Lemma 1, no domain-specific knowledge was used. The impressive performance is largely due to the SAT solver. Even the translation into propositional logic was not written by hand, but is an instance of a framework for finite model generation that is readily available in the Isabelle/HOL theorem prover. Only the *Sudoku* rules had to be defined in the prover, and this was a trouble-free task.

References

[1] Col Allan, editor. *New York Post*. News Corporation, New York City, NY, USA, 2005.

[2] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[3] Giovanni di Lorenzo, editor. *Die Zeit*. Zeitverlag Gerd Bucerius GmbH & Co. KG, Hamburg, Germany, 2005.

[4] DIMACS satisfiability suggested format, 1993. Available online at `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc`.

[5] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible Sudoku grids, June 2005. Available online at `http://www.shef.ac.uk/~pm1afj/sudoku/`.

[6] DeadMan's Handle Ltd. Sudoku solver, September 2005. Available online at `http://www.sudoku-solver.com/`.

[7] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, Las Vegas, June 2001.

[8] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[9] Robert James Thomson, editor. *The Times*. Times Newspapers Ltd., London, UK, 2005.

[10] Pete Wake. Sudoku solver by logic, September 2005. Available online at `http://www.sudokusolver.co.uk/`.

[11] Tjark Weber. Bounded model generation for Isabelle/HOL. In Wolfgang Ahrendt, Peter Baumgartner, Hans de Nivelle, Silvio Ranise, and Cesare Tinelli, editors, *Selected Papers from the Workshops on Disproving and the Second International Workshop on Pragmatics of Decision Procedures (PDPAR 2004)*, volume 125(3) of *Electronic Notes in Theoretical Computer Science*, pages 103–116. Elsevier, July 2005.

[12] Wikipedia. Sudoku – Wikipedia, the free encyclopedia, September 2005. Available online at `http://en.wikipedia.org/wiki/Sudoku`.

[13] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. In *IPSJ SIG Notes 2002-AL-87-2*. IPSJ, 2002.

[14] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In Andrei Voronkov, editor, *Proceedings of the 8th International Conference on Computer Aided Deduction (CADE 2002)*, volume 2392 of *Lecture Notes in Computer Science*. Springer, 2002.