Sudoku as a SAT Problem

Inês Lynce
IST/INESC-ID, Technical University of Lisbon, Portugal
ines@sat.inesc-id.pt

Joël Ouaknine
Oxford University Computing Laboratory, UK
joel@comlab.ox.ac.uk

Abstract

Sudoku is a very simple and well-known puzzle that has achieved international popularity in the recent past. This paper addresses the problem of encoding Sudoku puzzles into conjunctive normal form (CNF), and subsequently solving them using polynomial-time propositional satisfiability (SAT) inference techniques. We introduce two straightforward SAT encodings for Sudoku: the minimal encoding and the extended encoding. The minimal encoding suffices to characterize Sudoku puzzles, whereas the extended encoding adds redundant clauses to the minimal encoding. Experimental results demonstrate that, for thousands of very hard puzzles, inference techniques struggle to solve these puzzles when using the minimal encoding. However, using the extended encoding, unit propagation is able to solve about half of our set of puzzles. Nonetheless, for some puzzles more sophisticated inference techniques are required.

1 Introduction

Sudoku is a puzzle that became popular in Japan in 1986 and achieved international popularity in 2005. It is often described as "the Rubik's cube of the 21st century".

The attraction of the puzzle is that the completion rules are simple, yet the line of reasoning required to reach completion may be difficult. Each puzzle has a unique solution and does not require the use of trial and error or guessing, i.e., each puzzle is solved merely with 'reasoning'.

A recent paper by Helmut Simonis [10] presents the Sudoku problem as a Constraint Satisfaction Problem (CSP). The author compares different propagation schemes for solving Sudoku, and suggests additional techniques to solve the problem without search.

Sudoku can also be encoded as a SAT problem. In this paper, we present two very intuitive SAT encodings for Sudoku puzzles: the minimal encoding and the extended encoding. The minimal encoding suffices to characterize Sudoku puzzles, whereas the extended encoding adds redundant clauses to the minimal encoding. These additional clauses can be justified in terms of resolution, and therefore do not alter the solution of a problem instance.

Furthermore, we give experimental results on solving Sudoku puzzles with polynomial-time reasoning, using not only unit propagation but also more sophisticated techniques. The minimal encoding is far from being a good option. However, for the extended encoding we observe that unit propagation is sufficiently powerful to solve many problems, and that more sophisticated techniques are able to solve all our Sudoku puzzles.

This paper is organized as follows. In the next section we describe Sudoku problem. Afterwards, we present our SAT encodings for Sudoku, and describe a number of SAT inference techniques. Finally, we give experimental results, and conclude suggesting future research work.

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

Figure 1: A Sudoku puzzle.

2 The Sudoku Problem

Definition 1 *A Sudoku puzzle is* represented *by a* 9×9 *grid, which comprises nine* 3×3 *sub-grids (also called boxes). Some of the entries in the grid are filled with numbers from 1 to 9, whereas other entries are left blank.*

Figure 1 is an example of a Sudoku puzzle. Puzzles are often also assigned a difficulty level, which usually depends on the number of initial non-blank entries provided. This number may be as few as 17 to test expert players. As we shall see, the numbers 1 through 9 are used solely for convenience; arithmetic relationships between them are completely irrelevant. Hence, any set of distinct symbols could have been used.

Definition 2 *A Sudoku puzzle is* solved *by assigning numbers from 1 to 9 to the blank entries such that every row, every column, and every* 3×3 *sub-grid contains each of the nine possible numbers.*

Interestingly, this rule explains why Sudoku means "single number" in Japanese. Although the definition above characterizes Sudoku, the puzzles available in the entertainment literature have two additional properties. In the remainder of this paper, we will only consider Sudoku puzzles that have these properties, i.e.:

Property 1 Sudoku puzzles that have only one solution.

Property 2 Sudoku puzzles that can be solved with only reasoning, i.e., with no search.

Having only one solution means that all the numbers to be assigned to the blank entries are *necessary* assignments. The second property requires in addition that, at any stage in the course of solving the puzzle, there should always be at least one blank entry that can be assigned to merely by considering what is immediately implied by the set of non-blank entries. Hence, reasoning consists in using inference rules in such a way that all of the assignments are found.

To illustrate how to solve a Sudoku puzzle, let us consider entry *(a)* in the left-hand grid of Figure 2. Considering the relevant 3×3 sub-grid, this is the only position where number 3 can be placed. Also, consider entry *(b)* in the right-hand grid of the same figure. It is clear that *(b)* is the only position where number 7 can be placed in the second row.

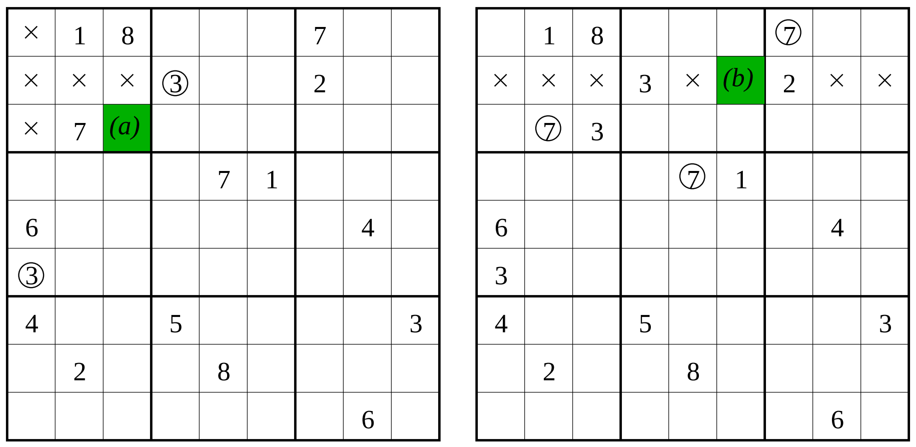

Figure 2: Solving a Sudoku puzzle.

Finally, observe that we can generalize Sudoku to $n \times n$ grids (for n a square), where n is referred to as the order of the puzzle. The generalized Sudoku problem is NP-Complete, as has been shown in [12]. The proof uses a simple reduction from the Latin Squares problem, which has been proved to be NP-complete by Colbourn et *al.* in [4]. Note, however, that generalized Sudoku puzzles satisfying Properties 1 and 2 are polynomial-time solvable.

3 SAT Encodings for the Sudoku Problem

A SAT problem is represented using n propositional variables $x_1, x_2, \ldots, x_n$, which can be assigned truth values 0 (false) or 1 (true). A literal l is either a variable x_i (i.e., a positive literal) or its complement $\neg x_i$ (i.e., a negative literal). A clause ω is a disjunction of literals and a CNF formula φ is a conjunction of clauses. A literal l_j of a clause ω_a that is assigned truth value 1 satisfies the clause, and the clause is said to be *satisfied*. If the literal is assigned truth value 0 then it is removed from the clause. A clause with a single literal is said to be *unit* and its literal has to be assigned value 1 for the clause to be satisfied. The derivation of an *empty* clause indicates that the formula is *unsatisfied* for the given assignment. The formula is *satisfied* if all its clauses are satisfied. The SAT problem consists of deciding whether there exists a truth assignment to the variables such that the formula becomes satisfied.

A Sudoku puzzle can easily be represented as a SAT problem, albeit one requiring a significant number of propositional variables. If we were using variables with arbitrary finite domains, then $9 \cdot 9$ variables with domain [1..9] would be the most adequate option. Nevertheless, encoding Sudoku puzzles into CNF requires $9 \cdot 9 \cdot 9 = 729$ propositional variables. For each entry in the 9×9 grid $\mathcal{S}$, we associate 9 variables. Let us use the notation s_{xyz} to refer to variables. Variable s_{xyz} is assigned true *if and only if* the entry in row x and column y is assigned number z. Hence, $s_{483} = 1$ means that $\mathcal{S}[4, 8] = 3$. Naturally, the pre-assigned entries of the Sudoku grid will be represented as unit clauses.

The constraints to be added to our SAT encoding refer to each entry, each row, each column and each 3x3 sub-grid, as illustrated in Figure 3. Similarly to the SAT encoding for the partial latin squares completion problem [7], we may either use the minimal or the extended encoding. The minimal encoding is enough for specifying the problem, whereas the extended encoding adds redundant constraints to the minimal encoding. For both encodings, the size of the encoding is $\mathcal{O}(n^4)$, where n is the order of the Sudoku puzzle. Observe that in this

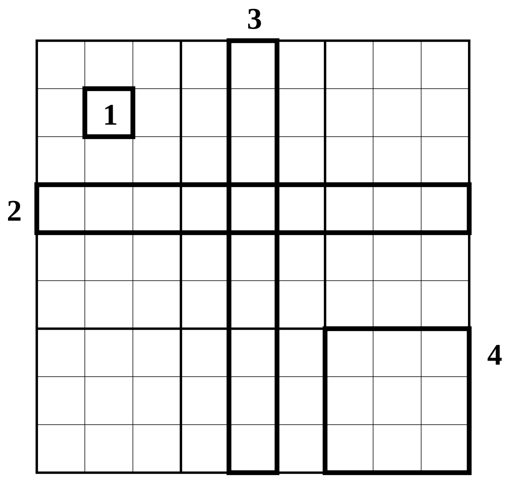

Figure 3: Constraints on the Sudoku problem.

case the redundant constraints are quite intuitive, and therefore the extended encoding is the one that most naturally comes to one's mind (see, e.g., the work by Ivor Spence available from `http://www.cs.qub.ac.uk/∼I.Spence/SuDoku/SuDoku.html`).

The minimal encoding asserts that there is at least one number in each entry, and that each number appears at most once in each row, in each column and in each 3×3 sub-grid. More formally, we have:

- There is at least one number in each entry:
  $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$
- Each number appears at most once in each row:
  $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$
- Each number appears at most once in each column:
  $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$
- Each number appears at most once in each 3x3 sub-grid:
  $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$
  $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$.

In the minimal encoding, the resulting CNF formula will have 8,829 clauses, apart from the unit clauses representing the pre-assigned entries. Of these clauses, 81 clauses are nine-ary and the remaining 8,748 clauses are binary. The nine-ary clauses result from the set of at-least-one clauses ($9 \cdot 9 = 81$), whereas the 8,748 binary clauses result from the three sets of at-most-one clauses ($3 \cdot 9 \cdot 9 \cdot \mathcal{C}_{9,2}$).

The extended encoding explicitly asserts that each entry in the grid has exactly one number, and likewise for each row, each column, and each 3x3 sub-grid. The extended encoding includes all the clauses of the minimal encoding, as well as the following constraints:

- There is at most one number in each entry:
  $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigwedge_{z=1}^{8} \bigwedge_{i=z+1}^{9} (\neg s_{xyz} \vee \neg s_{xyi})$

- Each number appears at least once in each row:

  $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$
- Each number appears at least once in each column:

  $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{y=1}^{9} s_{xyz}$
- Each number appears at least once in each 3×3 sub-grid:

  $\bigvee_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}.$

In the extended encoding, the resulting CNF formula will have 11,988 clauses, apart from the unit clauses representing the pre-assigned entries. From these clauses, 324 clauses are nine-ary and the remaining 11,664 clauses are binary. The nine-ary clauses result from the four sets of at-least-one clauses ($4 \cdot 9 \cdot 9 = 324$). The 11,664 binary clauses result from the four sets of at-most-one clauses ($4 \cdot 9 \cdot 9 \cdot \mathcal{C}_{9,2}$).

The extended encoding described above is similar to the CSP encoding introduced in [10]. Sudoku is modeled as a CSP problem by a combination of *alldifferent* constraints [11]. Observe that for the particular case of a Sudoku puzzle, where all variables have the same domain and all values in the domain must occur, the *alldifferent* constraint infers as many assignments as unit propagation.

4 SAT Inference Techniques

In this section we review some inference techniques for CNF formulas. Such techniques allow one to infer additional information from the original formula. For example, one may infer new assignments, new clauses, or replace variables due to the identification of equivalent variables. These techniques are also referred to as simplification techniques, to the extent that the additional information should ultimately reduce the number of variables and clauses in a formula.

Resolution is one of the most well-known inference techniques for satisfiability [6]. In general, given $\omega_r = (x_i \vee \alpha)$ and $\omega_s = (\neg x_i \vee \beta)$, where α and β are disjunctions of literals, the resolution operation ρ allows deriving the clause $\omega_t = \rho(\omega_r, \omega_s, x_i) = \rho((x_i \vee \alpha), (\neg x_i \vee \beta), x_i) = (\alpha \vee \beta)$.

Resolution is known to be a sound and complete proof procedure for SAT [8]. Each variable x is eliminated by applying resolution between each pair of clauses containing x and $\neg x$. Either the empty clause is derived and the formula is shown to be unsatisfiable or a set of clauses containing only pure literals[1] is obtained and the formula is shown to be satisfiable.

However, it is in general not computationally feasible to use resolution as a complete procedure. Hence, different algorithms for simplifying propositional formulas restrict the application of resolution in various ways. For example, we may consider a *restricted* form of resolution by which *only* simplified clauses can be derived. Formally, given $\omega_r = (x_i \vee \alpha)$ and $\omega_s = (\neg x_i \vee \alpha)$, where α is a disjunction of literals, derive the clause $\omega_t = \rho(\omega_r, \omega_s, x_i) = \rho((x_i \vee \alpha), (\neg x_i \vee \alpha), x_i) = (\alpha)$.

In what follows we describe some of the most well-know inference techniques, which have proved useful for solving Sudoku. All these techniques are restricted forms of resolution.

4.1 Unit Propagation

The most widely used simplification rule in backtrack-search SAT algorithms is certainly the *unit clause rule* [6]. This rule applies whenever a unit clause $\omega_i = (l_j)$ is identified. In this case, all clauses

[1] A literal is said to be pure whenever its variable only occurs in either positive or negative form throughout the formula. In this situation, the literal can be assigned truth value 1, satisfying all of its clauses.

containing literal l_j are declared satisfied, and literal $\neg l_j$ is removed from all clauses containing it. The unit clause rule can be iterated, and this process is referred to as *unit propagation*.

4.2 Failed Literal Rule

The *failed literal rule* [5] is applied as follows: if forcing an assignment to a variable $x = \nu(x)$ (where $\nu(x) \in \{0, 1\}$) and then performing unit propagation yields a conflict[2], then $x = 1 - \nu(x)$ is a necessary assignment. Observe that applying the failed literal rule with *arbitrary* depth provides a complete proof procedure for SAT.

4.3 Binary Failed Literal Rule

The *binary failed literal rule* [5] corresponds to the failed literal rule when considering two variables instead of one. Formally, if the assignments $x_i = \nu(x_i)$ and $x_j = \nu(x_j)$ yield a conflict, then add the clause $(x_i = 1 - \nu(x_i) \vee x_j = 1 - \nu(x_j))$ to the formula.

4.4 Hyper-Binary Resolution

Hyper-binary resolution is a restricted form of resolution introduced in [2]. Given the set of clauses $(\neg l_1 \vee x_i) \wedge (\neg l_2 \vee x_i) \wedge \ldots \wedge (\neg l_k \vee x_i) \wedge (l_1 \vee l_2 \vee \ldots \vee l_k \vee x_j)$, hyper-binary resolution allows to infer $(x_i \vee x_j)$. It is clear that the same clause could be obtained following a sequence of resolution steps to be applied to the same clauses. We should also note that hyper-binary resolution covers the 2-closure, i.e., the resolution between every two-clauses containing a literal and its negation.

Note that the binary failed literal rule supersedes the failed literal rule. Moreover, hyper-binary resolution supersedes the failed literal rule [3]. Indeed, let us consider $\mathcal{C}$ to be the set of clauses used for applying the failed literal rule, triggered by satisfying literal l and leading to the inference of the empty clause. There must then be a sequence of hyper-binary resolution steps using $\mathcal{C}$ (and the clauses inferred meanwhile) which, applied together with unit propagation, allow deriving $\neg l$. In addition, the binary failed literal rule supersedes hyper-binary resolution: applying the binary failed literal rule to the set of clauses $(\neg l_1 \vee x_i) \wedge (\neg l_2 \vee x_i) \wedge \ldots \wedge (\neg l_k \vee x_i) \wedge (l_1 \vee l_2 \vee \ldots \vee l_k \vee x_j)$ given the assignments $x_i = 0$ and $x_j = 0$ implies deriving clause $(x_i \vee x_j)$; the same clause would have been derived by applying hyper-binary resolution.

A key observation is that the failed literal rule, the binary failed literal rule, and hyper-binary resolution, combined with unit propagation, all saturate in polynomial time. More precisely, repeatedly applying these rules on an arbitrary CNF formula will lead in polynomially many steps (in the size of the original formula) to a formula from which no further progress can be made.

5 Experimental Results

The main goal of this section is to evaluate how different encodings combined with different inference techniques perform on a large set of Sudoku puzzles. We were interested in solving these puzzles without search, i.e., only by repeatedly using inference techniques. Of course, we did not consider full resolution among these techniques since it is complete (and, in general, exponential-time). For all the experiments, we used an Intel Pentium M processor 2.00GHz with 1GB of memory running Linux Fedora.

The time taken to solve each puzzle instance was typically less than 10 milliseconds. These values are comparable to the ones reported in [10]. Indeed, this result is not surprising. Latin

[2]A conflict is reached when a unit clause is derived.

squares instances of order up to 40 are easily solved by SAT solvers, even faster than by CSP solvers [1]. Since 9×9 Sudoku puzzles are solved so easily by SAT solvers, a meaningful comparison with other approaches would make more sense over larger grids, something which we plan to carry out in the near future.

We performed experiments on using our two encodings for solving a set of 24,260 very hard Sudoku puzzles. This collection of puzzles with 17 pre-assigned entries is due to Gordon Royle and is available from `http://www.csse.uwa.edu.au/∼gordon/sudokumin.php`. According to the author [9], these problems have no special property other than having 17 pre-assigned entries and a unique solution. Since there are no known puzzles with 16 pre-assigned entries, none of these puzzles contain redundancies. In other words, removing any of the entries yields more than one solution. Moreover, these puzzles are guaranteed to be mathematically inequivalent in that no two of them can be translated to each other by any combination of the following operations:

- Permutations of the 9 symbols;
- Transposing the matrix (i.e., exchanging rows and columns);
- Permuting (i.e., rearranging) rows within a single block;
- Permuting columns within a single block;
- Permuting the blocks row-wise;
- Permuting the blocks column-wise.

For solving these Sudoku puzzles, we ran each problem under four different configurations. These configurations all include the most widely used inference technique, unit propagation, as well as at most one of the other techniques described in the previous section. The four configurations were the following:

1. Unit propagation (`up`);
2. Unit propagation + failed literal rule (`up+flr`);
3. Unit propagation + hyper-binary resolution (`up+hypre`);
4. Unit propagation + binary failed literal rule (`up+binflr`).

For running the first and second configurations we adapted `compactor`, a preprocessor by Chu Min Li available from `http://www.laria.u-picardie.fr/∼cli`. For running the `up+hypre` configuration, we adapted `hypre`, a preprocessor by Fahiem Bacchus described in [3] and available from `http://www.cs.toronto.edu/∼fbacchus/sat.html`[3]. The results for the `up+binflr` configuration were obtained using `compact` [5], a polynomial-time preprocessor by James Crawford available from `http://www.cirl.uoregon.edu/crawford/crawford.html`. `Compact` implements a limited version of the binary failed literal rule in order not to compromise the efficiency of the tool.

Table 1 gives the percentage of puzzles solved by each configuration, using either the minimal or the extended encoding. First of all, it is clear that unit propagation is not powerful enough to solve the entire set of puzzles: using the extended encoding, almost half of the puzzles are solved, whereas for the minimal encoding there are no puzzles solved.

When using the minimal encoding, the maximum percentage of puzzles solved (69%) is achieved with the `up+binflr` configuration, followed by the `up+hypre` configuration with 25% and then by the `up+flr` configuration with only 1%.

Under the extended encoding, however, the failed literal rule (as mentioned in [10]), hyper-binary resolution, and the binary failed literal rule could be used to effectively solve all the puzzles that were not solved by unit propagation.

[3]Hypre also implements variable equivalence, which was disabled for the results given in Table 1. However, even if variable equivalence had been enabled, the results would have been exactly the same.

	`up`	`up+flr`	`up+hypre`	`up+binflr`
Minimal	0%	1%	25%	69%
Extended	47%	100%	100%	100%

Table 1: Percentage of puzzles solved for the minimal and the extended encodings.

	`up`	`up+flr`	`up+hypre`	`up+binflr`
Minimal	54%	50%	47%	46%
Extended	20%	–	–	–

Table 2: Percentage of variables in the resulting formula (mean).

For the configurations that were not able to solve all puzzles, a simplified formula was obtained (in the unsolved instances), whose free variables correspond to variables of the original formulas that were not given assignments. The ratio between the number of such variables and the initial number of variables is a measure of how far the formula has been simplified, i.e., how much information has been inferred—or rather, how much information is not yet known. Table 2 gives the average values of these ratios as percentages. Obviously, when all the puzzles were solved no ratio was computed.

From the results in Table 2, some conclusions can be drawn. First, for the `up` configuration, in the case of unsolved formulas much less progress towards the solution is recorded under the minimal encoding than under the extended encoding. This result is not surprising: the extended encoding allows additional inferences. Second, results for the minimal encoding using the `up` and the `up+flr` configurations are quite similar, which means that there seems to be no significant advantage in applying the failed literal rule under the minimal encoding. Finally, we observe that the `up+hypre` and `up+binflr` configurations do not yield significantly smaller unsolved formulas, even though when combining the minimal encoding with these configurations the likelihood of solving a puzzle instance is far more promising.

6 Conclusions and Future Work

In this paper we have introduced two straightforward SAT encodings for representing and solving Sudoku puzzles. Such puzzles are meant to be solved without search, i.e., merely with reasoning. Empirical results show that while unit propagation is not powerful enough to solve all the puzzle instances, more sophisticated—yet polynomial-time—inference techniques are very successful in solving them.

For future work we would like to carry out a systematic comparison of the effectiveness of the techniques described in this paper with other known approaches for solving Sudoku, especially over larger (generalized) puzzles.

References

[1] C. Ansótegui, A. del Val, I. Dotú, C. Fernández, and F. Manyá. Modeling choices in quasigroup completion: SAT vs. CSP. In *Proceedings of the National Conference on Artificial Intelligence*, 2004.

[2] F. Bacchus. Exploiting the computational tradeoff of more reasoning and less searching. In *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, pages 7–16, May 2002.

[3] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, pages 183–192, May 2003.

[4] C. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8:25–30, 1984.

[5] J. M. Crawford and L. Auton. Experimental results on the cross-over point in satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 22–28, 1993.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, July 1960.

[7] C. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.

[8] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.

[9] G. Royle. Personal Communication, 2005.

[10] H. Simonis. Sudoku as a constraint problem. In *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems*, October 2005. `http://www.icparc.ic.ac.uk/∼hs/sudoku.pdf`.

[11] W. J. van Hoeve. The alldifferent constraint: a survey. In *6th Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[12] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. In *Proceedings of the National Meeting of the Information Processing Society of Japan (IPSJ)*, 2002.