proceedings

MDPI

Proceeding Paper

Isomorphism between Sudoku and Proof Systems and Its Application in Sudoku Solving †

Jakub Dakowski

Department of Logic and Cognitive Science, Adam Mickiewicz University, 60-568 Poznań, Poland; jakubdakowski@gmail.com

† Presented at Philosophy and Computing Conference, IS4SI Summit 2021, Online, 12–19 September 2021.

Abstract: (1) Introduction: While automatic Sudoku solvers are a well-known area of study in formal sciences, there has been little to no progress when it comes to describing the proving process as analogous to Sudoku solving. (2) Materials and Methods: This paper proposes two methods of solving Sudokus automatically: one using Hilbert systems, the other with an additional contradiction rule. (3) Results: While the first algorithm was not complete, it seems that the second one is. It was able to solve most of the provided test cases in under a second. (4) Discussion: Different work already suggests this concept for a Sudoku solver. However, it comes from a different theoretic standpoint. Future work in this field might include incorporating the results of proof theory or searching for a Sudoku solvable for every possible substitution.

Keywords: logic; proof theory; sudoku; puzzle; consequence operation

Citation: Dakowski, J. Isomorphism between Sudoku and Proof Systems and Its Application in Sudoku Solving. *Proceedings* **2022**, *81*, 78. https://doi.org/10.3390/proceedings2022081078

Academic Editor: Peter Boltuc

Published: 25 March 2022

1. Introduction

The first Sudoku puzzles appeared in 1979 in an East Coast puzzle magazine [1], and despite their relative youth, there has been huge development when it comes to solving them. Multiple methods for filling the fields of Sudoku were developed. From this emerged a method which closely resembles proof systems.

A Sudoku puzzle is a 9×9 matrix (also called a grid) partially filled with numbers from 1 to 9 (later refered to as digits). Its main parts are fields (single entries in the matrix), columns, rows and subgrids—9 3×3 submatrices (There exists a simple function which values group together the fields to create subgrids (given that both coordinates start with 0): $f(i,j) = idiv3 + 3 * (jdiv3)$). According to Britannica [1], the name of Sudoku puzzles is an abbreviation of the main rule of Sudokus—"*suuji wa dokushin ni kagiru*" ("the numbers must remain single" in English). This can be formalized as:

Definition 1. *In a solved Sudoku, no digits reappear in a given row, column or subgrid, and every field is filled in.*

When solving, the player slowly narrows down the possible numbers (this resembles a logical disjunction [2]) in given fields to single digits. This is carried out using a set of solvability-preserving strategies on the grid. Two of most basic ones will be presented below (based on [3]).

The hidden single strategy (denoted as *HS*) is based around the notion that if a given digit can be placed only in one field of a given column, row or set, it certainly is the value of this field.

The naked single strategy (denoted as *NS*) can be thought of as the reverse of the first strategy. When only one digit can be placed in a field, this digit will surely be there in the solution. Because of this, it can be permanently placed there.

When trying to represent deductive reasoning, one might be inspired by proof systems in formal logic. While there are a few of those, here, only two of them will be mentioned—Hilbert systems and natural deduction.

Axiomatic systems were introduced by Hilbert [4]. These use direct reasoning—the proof is a derivation of given formula from a given axiom set, using clearly defined rules [5].

To simplify this system, Fitting [5] introduces axiom schemas. These contain symbols that denote arbitrary formulas [5], instead of propositional variables. When given an axiom scheme, $A \to (B \to A)$, one might substitute A and B with any formula without using additional proving rules.

The notion of consequence can be used to define a function $Cn : 2^{Form} \to 2^{Form}$ ($Form$ denotes a set of all sentences in propositional logic), which for any set of formulas returns a set of every derivable formula. When given a set of axioms, Ax, the set of all derivable formulas will be equal to $Cn(Ax)$ [6].

Axiomatic systems, while elegant, have little connection to how human mathematicians reason. Because of this, Gentzen [7] and Jaśkowski [8] simultaneously proposed natural deduction. This system was simplified and popularized by Fitch [9] as the method of subordinate proofs.

In contrast to axioms in Hilbert systems, here one can introduce new assumptions to proofs (which initiates new subproofs) and discharge them (ending the subproofs). Usually, reasoning in these systems is direct, but when a contradiction is reached, it is possible to use reasoning by contradiction to eliminate the faulty assumption, deducing its negation at the same time [5].

2. Materials and Methods

When examined carefully, one can find certain similarities between the processes of proof creation and Sudoku solving. Both start with a fundamental object that is transformed using certain reasoning schemes to the desired form. In this way, a Sudoku puzzle becomes a formula, while the solvability of a Sudoku can be compared to being true, and unsolvability to being false. With this being said, the solvability-preserving strategies can be compared to truth-preserving inference rules.

With this being said, it is possible to modify the Cn function to return all derivable Sudokus. Let us define a function: $Cn^S_{\{NS,HS\}} : Sudokus \to 2^{Sudokus}$ (where $Sudokus$ denotes the set of all solvable Sudoku puzzles). Such a function should return every Sudoku that is derivable using these rules. As the consequence operation was originally defined with Hilbert systems in mind, it felt natural to try implementing it. One of the main parts of this solver was creating a working consequence function, as its definition is quite theoretical. To simplify this process, the author introduces an assumption:

Assumption 1. *Hidden single and naked single rules commute with each other. This means that, regardless of the rule order, the result of applying those rules will always be the same.*

To implement the notion of consequence, the immediate consequence in Herbrandt models [10] was used. It is a function that represents using any rule once. Using this operation until the result of X is equal to $\{X\}$ yields a set which consists of all of its consequences.

The main problem of this method was its incompleteness. A complete version of this method certainly exists, as there are a finite number of Sudoku puzzles; therefore, it is possible to create a finite number of rules that would match unfinished puzzles with their finished counterparts. This method is not useful, because such algorithm would simply be inefficient. While smaller complete strategy set can exist, the author could not find any work regarding them, and instead proposes using another tool from the proof–theoretic toolbox.

Let us recall Definition 1. It is easy to deduce Theorem 1 from it.

Theorem 1. *A Sudoku is solvable if it is possible to place every digit once in every row, column and subgrid.*

Based on this, we can infer Corollary 1, which introduces the concept of contradiction to this system (and because of this, the unsolvability of a Sudoku will be denoted as $\perp$).

Corollary 1. *If there exists a field that can't be filled with any digit, then the Sudoku is unsolvable.*

With contradiction comes the possibility of reasoning by contradiction. Such reasoning is quite popular in natural deduction systems. Similarly, one might imagine an inference schema—"if placing a given digit into a field yields a contradiction (an unsolvable Sudoku), then the digit is not in the field". This mechanism is introduced into the Hilbert-inspired solver as a negation rule [5].

It is worth mentioning that the negation rule is quite risky in a proof search. When used amiss, the assumption introduction might inflate the computational complexity, as it can test the correctness of any value in any field. Because of this, the negation rule is not introduced via the consequence operation.

There is a finite number of possible digits in every field. These can be searched through using the negation rule and the consequence operation. As one might imagine, this makes this proof search a breadth-first tree search. As a heuristic, this search prioritizes fields with a lower number of possible digits.

3. Results

The software was tested against two Sudoku puzzle sets. The experiment was carried out using an Intel Core i9-10850 processor with the clock speed set to 4.8 GHz. In both sets, every Sudoku was solved 100 times to minimize the measurement error.

The first set [11] consisted of 50 easy Sudokus. The software was able to solve them with mean solving time of 43.853 ms, and a standard deviation of 89.282 ms.

The second set [12] consisted of 46 hard Sudokus, and again the presented algorithm was able to solve all of them. The average solving time was 64.611 ms. The standard deviation in this set was 146.752 ms. Higher solving times are understandable when the rise in difficulty is taken into account.

4. Discussion

It is important to note that the performance of other algorithms will not be discussed in this section, because of the differences in both hardware and software used.

Peter Norvig [13] proposes the same algorithm, which he develops based on the concept of constraint programming. Conceptually, his algorithm starts with a possible solution space consisting of all Sudokus. He then uses different strategies to constrict this solution space. When nothing else is possible, Norvig searches through the solution space using—what this work would call—assumption creation and retraction. The duality of this algorithm will be discussed later; however, Norvig reassures the completeness of this algorithm.

Another work which presents this algorithm is [14]. The authors use rewriting logic to create a similar set of inference rules with branching, which in turn creates a system that resembles tableaux methods. In the author's opinion, this work, while coming from logic, still should be viewed as a different approach. The isomorphism presented here is a case for treating Sudoku as a logic; therefore, it is a proof–theoretic solution of the problem.

A similar case can be made for [2]. The authors encode every Sudoku as a conjunctive normal form and then use a series of SAT inference techniques (these bear resemblance to the negations rules presented here) to find a solution to the presented Sudoku.

The fact that multiple perspectives led to a similar mechanism might be used to draw important conclusions. First of all, there is the difference between Peter Norvig's constraints and this work's inference rules. These, while being opposite to themselves

(one disallows solutions and the other infers them), might be considered as two sides of the same coin. Similar to the rules of generative grammars [15], from which these algorithms do not deviate too far, the mechanisms of Sudoku seem to both match and generate possible solutions. This is quite understandable given the connection of logic and constraint programming [16].

The applications of this work can be summed up using a famous quote—"Just a spoonful of sugar helps the medicine go down" [17]. Proof theory, in the author's opinion, can be thought of as quite an inaccessible part of science. Using analogies similar to the one presented here might make it easier for students to understand the topic and its problems. A great example might be the concept of completeness and soundness, which are much easier to introduce using down-to-earth examples, such as Sudoku puzzles.

While using more advanced techniques of proof search might yield a better solver, there are other topics worth exploring in the intersection of formal proofs and puzzles. Assumption 1, and all of the rules in the system, require formal proofs. One might also come back to the presented Hilbert system and try creating a complete version. Furthermore, as Hilbert systems are used to create true formulas from axioms, one might create a calculus which allows the creation of new puzzles from the solutions. (This would be a trivial task without the constraint of only one solution, but might produce interesting results with this restriction.) These might be considered Sudoku axioms, as they are solvable by definition. To define these solutions, one might use axiom schemas [5] of solved puzzles, defined based on [18]. A final suggestion would be comparing solvability to satisfiability [6] and searching for tautology puzzles, which would resolve for every possible digit substitution. The axiom set would not change, as all Sudoku solutions are solvable for every possible substitution by definition.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data generated in this study and the tested Sudoku examples can be found here: https://github.com/PogromcaPapai/Teresa/tree/measure/data (accessed on 25 October 2021). The same repository contains the implementation of the algorithm.

Acknowledgments: The author would like to thank Aleksander Kiryk for his remarks which inspired some parts of the discussion.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Sudoku. Available online: https://www.britannica.com/topic/sudoku (accessed on 31 May 2021).
2. Lynce, I.; Ouaknine, J. Sudoku as a SAT Problem. *ISAIM* **2006**, *11*, 6–13.
3. Sudoku9x9.com. Sudoku Solving Techniques. Available online: http://www.sudoku9x9.com/sudoku_solving_techniques_9x9.html (accessed on 31 May 2021).
4. Van Heijenoort, J. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*; Harvard University Press: Cambridge, MA, USA, 2002.
5. Fitting, M. *First-Order Logic and Automated Theorem Proving*; Springer Science & Business Media: New York, NY, USA, 2012.
6. Batóg, T. *Podstawy Logiki*; Wydawnictwo Naukowe im. Adama Mickiewicza w Poznaniu: Poznań, Poland, 1986; pp. 65–67.
7. Gentzen, G. Untersuchungen über das logische Schließen. *Math. Z.* **1934**, *39*, 405–431. [CrossRef]
8. Jaśkowski, S. *On the Rules of Suppositions in Formal Logic*; Studia Logica: Warszawa, Poland, 1934.
9. Fitch, F.B. *Symbolic Logic: An Introduction*; Ronald Press Co.: New York, NY, USA, 1952.
10. Garcez, A.S.D.; Broda, K.B.; Gabbay, D.M. *Neural-Symbolic Learning Systems: Foundations and Applications*; Springer Science & Business Media: London, UK, 2012.
11. Solving Every Sudoku Puzzle. Available online: https://github.com/dimitri/sudoku (accessed on 31 May 2021).
12. Mantere, T.; Koljonen, J. Solving and analyzing Sudokus with cultural algorithms. In Proceedings of the 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–6 June 2008; pp. 4053–4060.
13. Solving Every Sudoku Puzzle. Available online: http://norvig.com/sudoku.html (accessed on 31 May 2021).

14. Santos-García, G.; Palomino, M. Solving Sudoku puzzles with rewriting rules. *Electron. Notes Theor. Comput. Sci.* **2007**, *176*, 79–93. [CrossRef]
15. Sipser, M. Introduction to the Theory of Computation. *ACM Sigact News* **1996**, *27*, 27–29. [CrossRef]
16. Hooker, J.N. Logic, optimization, and constraint programming. *INFORMS J. Comput.* **2002**, *14*, 295–321. [CrossRef]
17. Stevenson, R. *Mary Poppins*; Buena Vista Distribution Company: Burbank, CA, USA, 1964.
18. Felgenhauer, B.; Jarvis, F. Enumerating Possible Sudoku Grids. 2005, *preprint*.