Say No to Case Analysis: Automating the Drudgery of Case-Based Proofs

Jeffrey Shallit(✉)

School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
shallit@uwaterloo.ca

Abstract. I present an argument that long, tedious proofs requiring a human to check many cases should be replaced by an algorithm, so a computer can do the work instead.

Keywords: Decision procedure · Automata · Case-based proof · Algorithm

1 Introduction

My talk can be briefly summarized as follows: *Long, tedious proofs that require a human to check many cases should be replaced by an algorithm, so a computer can do the work instead.*

Doing so offers a number of advantages:

- An algorithm replaces valuable human time with what a computer does best: tedious examination of a large number of cases.
- Implementing an algorithm allows one to test whether all cases have in fact been considered, and correct any errors in the analysis.
- An algorithm is often more general than the specific problem at hand, and can easily be modified to explore generalizations of the original problem.
- If a conjecture can be phrased in a logical language that is algorithmically decidable, then one can use a decision procedure instead of a case-based proof.
- By combining a decision procedure with heuristics, one can algorithmically "guess" possible solutions to a problem, and then prove the guess is correct. So one can "guess" the correct routes to a proof, and then complete it.

Furthermore, automata and formal languages provide a framework that can replace case analysis for a diverse set of problems.

2 Why We Need Cases: Some Things Are True for No Good Reason

As mathematicians and theoretical computer scientists, we are conditioned to believe that most of the true statements we are interested in have proofs. For

S. Maneth (Ed.): CIAA 2021, LNCS 12803, pp. 14–24, 2021.
https://doi.org/10.1007/978-3-030-79121-6_2

hundreds of years, nearly everyone believed that Fermat's "Last Theorem" was true and that a proof would be found someday. And our intuition was confirmed when Andrew Wiles succeeded in finding a proof.

Yet we know from Gödel that, in a sufficiently powerful consistent formal system, there are true statements that have no proof in the system. Furthermore, some of these assertions will be very simple to state (at least conceptually), such as "This statement has no proof in Peano arithmetic". But there may be other, more "natural" true statements that are simple, lack obvious self-reference, and still have no proof. Here is a possible example:

> *Numbers that are exact powers of two are 2, 4, 8, 16, 32, 64, 128 and so on. Numbers that are exact powers of five are 5, 25, 125, 625 and so on. Given any number such as 131072 (which happens to be a power of two), the reverse of it is 270131, with the same digits taken in the opposite order. Now my statement is: it never happens that the reverse of a power of two is a power of five.*

– Freeman Dyson [9]

Dyson's conjecture is plausible because of some "randomness" in the decimal digits of powers, together with the lack of small counterexamples. But it is that very "randomness" that makes it hard to find a route to a proof.

We are also conditioned to believe that the true statements we are interested in not only have proofs, but also have *simple* proofs, if only we are clever enough to find them. Consider, for example, the attraction of "proofs from the Book"—an Erdős fantasy that there exists a celestial Book containing *the* optimal proofs for all important theorems [1].

While it is certainly desirable to find short proofs that give insight into a problem, we also know that in any sufficiently powerful consistent system there are true statements that are provable, but whose shortest proof is astronomically long in comparison to the length of the statement.[1] We might say these are statements that are *true, but for no good reason.*

So, a priori, we should not be at all surprised that some simple statements like the Four-Color Conjecture (4CC) might end up having no simple proof. The original proof of 4CC by Appel and Haken [2] involved finding an "unavoidable set of reducible configurations", reducing the problem to checking 1,834 individual cases by a computer. To date there is still no really simple proof of this theorem.

This is an automata theory conference, so let's look at an example from automata theory. Suppose we conjecture that all strings satisfy some property. If this property can be represented by an NFA $M = (Q, \Sigma, \delta, q_0, F)$, then this conjecture becomes the *universality problem*: does M recognize Σ^*? Unfortunately, the universality problem for NFA's is PSPACE-complete [18], so probably there is no efficient algorithm to check universality. Even worse, we may not

[1] An example is "This statement has no proof in Peano arithmetic with less than 10^{100} symbols.".

even be able to check a possible counterexample in polynomial time, since there are $O(n)$-state NFA's where the shortest string *not* accepted is of length $\geq 2^n$. An example is provided by the language

$$L_n = \{0, 1, \#\}^* - \{ [0]\#[1]\# \cdots \#[2^n - 1] \},$$

where $[a]$ is the binary representation of a, padded on the left to make it n bits long. It is not hard to construct an NFA M_n for L_n that has $O(n)$ states, while the shortest (and only) string M_n does not accept is clearly of length $(n+1)2^n - 1$.

Hence short conjectures about universality, represented by NFA's, might have exponentially long counterexamples, and we might need to examine exponentially many cases to rule them out.

3 Let a Computer Do the Work

In a classic paper of Entringer, Jackson, and Schatz [10], the authors proved that every binary word containing no squares xx with $|x| \geq 2$ is of length ≤ 18. They do so by a case-based analysis that is displayed in a large diagram that takes up an entire page of their paper.

But why do this? One can check each case tediously by hand, but do we get any real insight this way? And after doing so, does the reader feel sure that every case has been covered?

Instead, one can recognize this as a classic avoidance problem that can be solved almost trivially with breadth-first or depth first search. Let P be a set of patterns one wants to avoid over some alphabet Σ. Construct a (potentially) infinite tree T, with nodes labeled by Σ^*. The root is labeled with the empty string ϵ. If a node is labeled x and does not end in a pattern in P, then its children are xa for $a \in \Sigma$. Otherwise the node is a leaf. Then T is finite iff the set P cannot be avoided. Furthermore, if the node at greatest depth is xa for $a \in \Sigma$, then x is a longest word avoiding P.

By reporting statistics obtained by breadth-first or depth-first search, one can provide enough information that anyone else can easily check the results with a simple program. For the Entringer et al. problem, one can provide the number of leaves in the tree (478) and the leaves at greatest depth, which are

$$0100110001110011010, 0100110001110011011,$$

and their binary complements.

4 Algorithmic Case Analysis Prevents Errors

One of the advantages of automating case-based proofs is increased certainty in the correctness of the proof. Once all the cases have been expressed algorithmically, one can then test a large number of randomly-chosen examples (or try to exercise all paths in the case analysis) to make sure all cases have been covered.

As an example, consider a recent theorem by Cilleruelo and Luca [5]: for every integer base $b \geq 5$, every natural number is the sum of three natural numbers whose base-b representations are palindromes. Their 30-page proof required examining a very large number of cases (one case was labeled IV.5.v.b), and would be rather challenging to verify. As it turns out, however, the initial proof had some small, easily-repaired flaws that were only discovered when the case analysis was programmed up in Python by Baxter [6].

5 Replacing a Large Number of Cases with a General Argument

Returning to the sum-of-palindromes problem, Cilleruelo et al. were not able to handle the case of bases $b = 2, 3, 4$ in their analysis. I wondered if a more general approach might work to solve this problem. We want to show that every integer can be represented as a sum of numbers with a certain easily-describable base-b representation. If we can use some flavor of automata to check these representations, then this becomes a universality problem for nondeterministic machines: for every natural number N, we "guess" a representation as a sum of palindromes, and then check it. Even though universality problems are hard in general, we might "luck out" and get one that runs in a reasonable length of time.

We were able to solve the sum-of-palindromes problem for the remaining cases $b = 2, 3, 4$ using two different approaches:

– "guess" a representation of N as a sum of terms and use a visibly-pushdown automaton to verify that the guessed representations are palindromes;
– "guess" only the first half of the representations of the terms to be summed with an NFA, and then verify that the full representations sum to a "folded" version of the representation of N.

Using these ideas, we were able to prove

Theorem 1. *For base* 2*, every natural number is the sum of four palindromes. For bases* 3 *and* 4*, every natural number is the sum of three palindromes.*

For the details, see [21,22].

Furthermore, now that we have the idea that computational models such as visibly-pushdown automata and NFA's can be used this way, it suggests a large number of related problems that are easily solved. For example, instead of palindromes, we could consider sums of *generalized palindromes*: these are numbers, like 1100, that have palindromic base-k representations if one allows insertion of leading zeroes. We also obtained results about sums of generalized palindromes with only minor modifications.

Or we could look at "squares" instead of palindromes: these are numbers whose base-b representation consists of two consecutive identical blocks. For this, see [17].

Let's look at another example: words avoiding various sets of palindromes. Let x be a finite or infinite word. The set of all of its factors (that is, contiguous blocks appearing in x) is written $\mathrm{Fac}(x)$, and the set of its factors that are palindromes is written $\mathrm{PalFac}(x)$. In [12], we proved the following result:

Theorem 2. *Let S be a finite set of palindromes over an alphabet Σ. Then the language*

$$C_\Sigma(S) := \{x \in \Sigma^* \ : \ \mathrm{PalFac}(x) \subseteq S\}$$

is regular.

Proof. Let ℓ be the length of the longest palindrome in S. We claim that $\overline{C_\Sigma(S)} = L$, where

$$L = \bigcup_{t \in P_{\leq \ell+2} - S} \Sigma^* \, t \, \Sigma^*.$$

$\overline{C_\Sigma(S)} \subseteq L$: If $x \in \overline{C_\Sigma(S)}$, then x must have some palindromic factor y such that $y \notin S$. If $|y| \leq \ell + 2$, then $y \in P_{\leq \ell+2} - S$. If $|y| > \ell + 2$, we can write $y = uvu^R$ for some palindrome v such that $|v| \in \{\ell+1, \ell+2\}$. Hence x has the palindromic factor v and $v \in P_{\leq \ell+2} - S$. In both cases $x \in L$.

$L \subseteq \overline{C_\Sigma(S)}$: Let $x \in L$. Then $x \in \Sigma^* \, t \, \Sigma^*$ for some $t \in P_{\leq \ell+2} - S$. Hence x has a palindromic factor outside the set S and so $x \notin C_\Sigma(S)$.

Thus we have written $\overline{C_\Sigma(S)}$ as the finite union of regular languages, and so $C_\Sigma(S)$ is also regular. ■

Not only does this theorem show that the language of words avoiding palindromes is regular, it also gives a method to actually construct a DFA recognizing the language of all such words. With this theorem, then, we can replace much of the case analysis in [11,23] with a calculation based on automata. As an example of the power of the method, we just mention one result from [12]:

Theorem 3. *The sequence $(e_{2,5}(n))_{n \geq 0}$ counting the number of binary words of length n containing no palindromes of length > 5 satisfies the recurrence*

$$e_{2,5}(n) = 3e_{2,5}(n-6) + 2e_{2,5}(n-7) + 2e_{2,5}(n-8) + 2e_{2,5}(n-9) + e_{2,5}(n-10)$$

for $n \geq 20$. Asymptotically $e_{2,5}(n) \sim c\alpha^n$ where $\alpha \doteq 1.36927381628918060784\cdots$ is the positive real zero of the equation $X^{10} - 3X^4 - 2X^3 - 2X^2 - 2X - 1$, and $c = 9.8315779\cdots$.

6 Heuristics Plus Algorithms Can Create Proofs

One of the most useful examples of these ideas is the following: use heuristics to find possible routes to a proof, and then use an algorithm to complete the proof itself.

Consider the following problem: choose a finite set of unary operations on languages, such as $S = \{$ Kleene closure, complement $\}$. Start with a language L, and apply the operations of S to L as many times as you like, and in any order. (This is the *orbit* of L under the set S.) How many different languages can you get?

For the particular S above, the answer is 14; this is a version of the Kuratowski 14-theorem from topology.

We can then try different sets of operations. In 2012, we proved the following result [4].

Theorem 4. *For the set of eight operations*

$$S = \{\,\mathrm{Kleene closure, positive closure, complement, prefix, suffix,}\\ \mathrm{factor, subword, and reverse}\,\}$$

the size of the orbit of every language is at most 5676*.*

The simple idea behind the proof is that certain finite sequences of composed operations generate the same language as shorter sequences. For example, if k denotes Kleene closure and c denotes complement, then $kckckck$ has the same effect as kck. By generating an extensive list of identities like $kckckck \equiv kck$, we can do a breadth-first search over the tree of all sequences of operations, demonstrating that there is a finite set of sequences that covers all possibilities.

But which identities are true? Here is where heuristics can help us. We can model all languages with the class of regular languages. To find an identity, we can apply one list of operations to some randomly-generated set of regular languages and compare it to the result of some other list. If the results agree everywhere, we have a candidate identity we can try to prove.

When implemented, our procedure generated dozens of identities, most of which had trivial proofs. Once we had these identities, we used the breadth-first search to prove that the size of the orbit was finite. I'd be very surprised if there is a simple proof of Theorem 4.

7 Decision Procedures

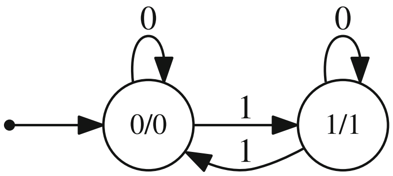
Let us continue with the theme of the previous section. The best possible example of what I'm talking about involves a *decision procedure*. If the statement you're trying to prove can be phrased in a logical theory that is recursively decidable (an algorithm exists to find proofs of all true statements), you can replace a case-based proof with running the decision procedure.

One domain where this has been very successful is the combinatorics of automatic sequences. (For us, "sequence" is synonymous with "infinite word"). A sequence $(s_n)_{n\geq 0}$ over a finite alphabet is *automatic* if, roughly speaking, there is a deterministic finite automaton with output (DFAO) that, on input the representation of the natural number n in some form, ends in a state with output s_n. A typical example of the kind of representation we are talking about is base-2 representation. For automatic sequences, thanks to Büchi and others (see [3]) there is a decision procedure to answer questions about these sequences that are phrased in first-order logic.

Let's look at a specific example. The *Thue-Morse sequence*

$$\mathbf{t} = (t_n)_{n\geq 0} = 0110100110010110\cdots$$

is an automatic sequence and is generated by the following very simple automaton. Here the label a/b on a state means that the state is numbered a and the output associated with the state is b.



A word x has period $p \geq 1$ if $x[i] = x[i+p]$ for all indices i that make sense. Currie and Saari [7] proved that $\mathbf{t}$ has a factor of least period p for all integers $p \geq 1$. Their proof required 3 lemmas, 6 cases, and 3 pages.

However, their claim can be phrased in a certain logical system that is algorithmically decidable, and there is a decision procedure for it. This procedure has been implemented in the `Walnut` theorem prover [19] written by Hamoon Mousavi, and so we can enter the commands

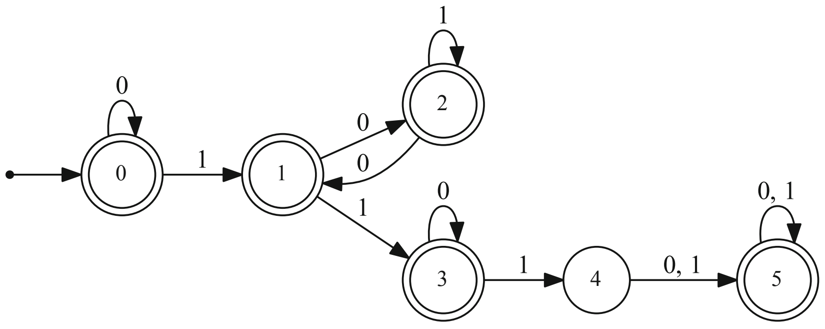
```
def tmperi "(p>0) & (p<=n) & Aj (j>=i & j+p<i+n) => T[j]=T[j+p]":
def tmlper "$tmperi(i,n,p) & (Aq (q>=1 & q<p) => ~$tmperi(i,n,q))":
eval currie_conj "Ap (p>=1) => Ei,n (n>=1) & $tmlper(i,n,p)":
```

which returns `TRUE` in a matter of .062 s of CPU time. Here `tmperi` asserts that $\mathbf{t}[i..i+n-1]$ has period p, and `tmlper` asserts that the least period of $\mathbf{t}[i..i+n-1]$ is p.

A factor is said to be *bordered* if it begins and ends with the same word in a nontrivial way, like the English word `entanglement`. If it is not bordered, we call it *unbordered*. Currie and Saari [7] were also interested in determining all lengths of unbordered factors in $\mathbf{t}$. They proved that $\mathbf{t}$ has a length-n unbordered factor if $n \not\equiv 1 \pmod 6$, but were unable to find a necessary condition. We can do this with `Walnut` by writing

```
def tmfactoreq "At (t<n) => T[i+t]=T[j+t]":
def tmbord "(m>=1) & (m<n) & At (t<m) => $tmfactoreq(i,(i+n)-m,m)":
def tmunbordlength "Ei Am ~$tmbord(i,m,n)":
```

Running this in `Walnut` produces the following automaton, which recognizes the base-2 representation of all n for which $\mathbf{t}$ has a length-n unbordered factor:

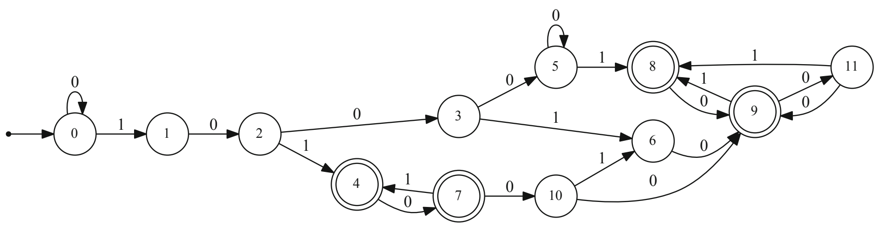


By inspection, we get the following theorem:

Theorem 5. *The Thue-Morse sequence* $\mathbf{t}$ *has an unbordered factor of length* n *if and only if* $(n)_2 \notin 1(01^*0)^*10^*1$.

Finally, let's look at one more problem from additive number theory. The *upper Wythoff set* $U = \{2, 5, 7, 10, 13, \ldots\}$ is defined to be $\{\lfloor \alpha^2 n \rfloor : n \geq 1\}$, where $\alpha = (1 + \sqrt{5})/2$ is the golden ratio. Recently Kawsumarng et al. [16] studied the sumset $U + U = \{x + y : x, y \in U\}$. Using a case-based argument, they constructed a rather complicated description of this set, noting that it "has some kinds of fractal and palindromic patterns".

However, it turns out that the assertion $n \in U + U$ is first-order expressible in a decidable logical theory; this allows us to give a complete description of $U+U$ as the set of natural numbers whose Fibonacci representation[2] is recognized by the following automaton:



Here no explicit breakdown into cases was necessary; instead, the decision procedure "automatically" constructs the automaton from a description of U. The fact that this automaton has so many states and a complicated structure partially explains why the set $U + U$ is difficult to describe explicitly. See [24].

In the next two subsections I mention some other examples of this approach that don't quite rise to the status of a decision procedure, but are still enormously useful.

[2] The Fibonacci representation of a natural number n is a finite binary string $a_1 a_2 \cdots a_t$ such that $n = \sum_{1 \leq i \leq t} a_i F_{t+2-i}$, and $a_i a_{i+1} = 0$ for $1 \leq i < t$.

7.1 SAT Solvers

The *boolean Pythagorean triples problem* is the following: are there infinite binary words $\mathbf{a} = a_1 a_2 \cdots$ with the property that if $i^2 + j^2 = k^2$, then $a_i = a_j = a_k$ never holds? This was finally resolved negatively by Heule, Kullmann, and Marek [15], who proved that the longest such word is of length 7824. The really interesting thing about their proof is how it was achieved: they coded the avoidance conditions as a SAT instance and then applied a general-purpose tool—a SAT solver—to check if this instance is satisfiable. Even though, as is well-known, SAT is an NP-complete problem, modern SAT solvers can often determine if particular instances are satisfiable or not, even if they have thousands of variables and clauses.

For another interesting application of SAT solvers, see [14].

7.2 The W-Z Method

The W-Z method (developed by Gosper [13] and Wilf, Zeilberger, and Petkovšek [20]) is a decision procedure that allows verification of general combinatorial identities involving polynomials, exponentials, binomial coefficients, and similar quantities. It has been implemented in Maple, and hence automatically proving identities like

$$\sum_{-n \leq k \leq n} (-1)^k \binom{2n}{n+k}^3 = \frac{(3n)!}{n!^3}$$

is now almost trivial [25].

8 Heuristics Plus Decision Procedures Provide Proofs

Finally, we can combine the ideas of depth-first or breadth-first search over a space with a decision procedure to (a) figure out a good candidate for a solution and then (b) prove it is correct.

As an example, let's return to automatic sequences. In 1965, Dean [8] studied the *Dean words*: squarefree words over $\{x, y, x^{-1}, y^{-1}\}$ that are not reducible (that is, there are no occurrences of $xx^{-1}, x^{-1}x, yy^{-1}, y^{-1}y$) [8]. Let us use the coding $0 \leftrightarrow x$, $1 \leftrightarrow y$, $2 \leftrightarrow x^{-1}$, $3 \leftrightarrow y^{-1}$. We can use breadth-first search to find a candidate for an infinite Dean word that is automatic.

When implemented, breadth-first search quickly converges on the sequence

$$0121032101230321 \cdots,$$

which (using the Myhill-Nerode theorem) we can guess as the fixed point of the morphism

$$0 \rightarrow 01, \ 1 \rightarrow 21, \ 2 \rightarrow 03, \ 3 \rightarrow 23.$$

Now the decision procedure kicks in. We make a DFAO for this sequence and store it under the name `DE.txt` in the `Word Automata` library of `Walnut`.

Then we carry out the following commands:

```
eval dean1 "Ei,n (n>=1) & At (t<n) => DE[i+t]=DE[i+n+t]":
# check if there's a square
eval dean02 "Ei DE[i]=@0 & DE[i+1]=@2":
eval dean20 "Ei DE[i]=@2 & DE[i+1]=@0":
eval dean13 "Ei DE[i]=@1 & DE[i+1]=@3":
eval dean31 "Ei DE[i]=@3 & DE[i+1]=@1":
# check for existence of factors 02, 20, 13, 31
```

All of these return `FALSE`, so this word is a Dean word. We have thus proved the existence of Dean words with essentially no human intervention.

9 Objections

– You've replaced a case-based proof with an algorithm, but how do you know the algorithm is correct?
  Answer: Sometimes an implementation will be much simpler than the record of the cases it examines, so it will actually be *easier* to verify the program than the case-based argument.
  In other cases, the algorithm can produce a *certificate* that another, simpler program can easily verify.
  Finally, in addition to formal correctness, there is also empirical correctness.[3] With a program in hand, we can test it on a wide variety of different inputs to look for oversights and omissions.
– Running a program provides no insight as to *why* a result is true.
  Answer: Sometimes, as I've argued above, there just *won't be* a simple reason why a result is true. In situations like this, it's better just to accept the result and move on.
– Some of the decision procedures you've talked about have astronomical worst-case running times.
  Answer: Don't pay much attention to the worst-case running time of decision procedures! They often run in a reasonable length of time for the instances we are interested in.

References

1. Aigner, M., Ziegler, G.M.: Proofs from THE BOOK, 5th edn. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44205-0
2. Appel, K., Haken, W.: Every Planar Map is Four-Colorable. Contemporary Mathematics, vol. 98. American Mathematical Society (1989)
3. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belg. Math. Soc. **1**,191–238 (1994), corrigendum, Bull. Belg. Math. Soc. **1** (1994), 577

[3] For example, "Beware of bugs in the above code; I have only proved it correct, not tried it."– Donald Knuth.

4. Charlier, E., Domaratzki, M., Harju, T., Shallit, J.: Composition and orbits of language operations: finiteness and upper bounds. Int. J. Comput. Math. **90**, 1171–1196 (2013)
5. Cilleruelo, J., Luca, F.: Every positive integer is a sum of three palindromes (2016). preprint available at https://arxiv.org/abs/1602.06208v1
6. Cilleruelo, J., Luca, F., Baxter, L.: Every positive integer is a sum of three palindromes. Math. Comp. **87**, 3023–3055 (2018)
7. Currie, J.D., Saari, K.: Least periods of factors of infinite words. RAIRO Info. Theor. Appl. **43**, 165–178 (2009)
8. Dean, R.A.: A sequence without repeats on x, x^{-1}, y, y^{-1}. Amer. Math. Monthly **72**, 383–385 (1965)
9. Dyson, F.: What do you believe is true even though you cannot prove it? (2005). https://www.edge.org/response-detail/11675
10. Entringer, R.C., Jackson, D.E., Schatz, J.A.: On nonrepetitive sequences. J. Combin. Theory Ser. A **16**, 159–164 (1974)
11. Fici, G., Zamboni, L.Q.: On the least number of palindromes contained in an infinite word. Theoret. Comput. Sci. **481**, 1–8 (2013)
12. Fleischer, L., Shallit, J.: Automata, palindromes, and reversed subwords (2020). Manuscript under submission
13. Gosper Jr., R.W.: Decision procedure for indefinite hypergeometric summation. Proc. Natl. Acad. Sci. U.S.A. **75**, 40–42 (1978)
14. Heule, M.: Schur number five. In: Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), pp. 6598–6606. AAAI Press (2018)
15. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean Pythagorean triples problem via cube-and-conquer. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 228–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_15
16. Kawsumarng, S., Khemaratchatakumthorn, T., Noppakaew, P., Pongsriiam, P.: Sumsets associated with Wythoff sequences and Fibonacci numbers. Period. Math. Hung. **82**(1), 98–113 (2020). https://doi.org/10.1007/s10998-020-00343-0
17. Madhusudan, P., Nowotka, D., Rajasekaran, A., Shallit, J.: Lagrange's theorem for binary squares. In: Potapov, I., Spirakis, P., Worrell, J. (eds.) 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), pp. 18:1–18:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2018)
18. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Symposium on Switching and Automata Theory (SWAT), pp. 125–129. IEEE Computer Society (1972)
19. Mousavi, H.: Automatic theorem proving in Walnut (2016). arxiv preprint. http://arxiv.org/abs/1603.06017
20. Petkovšek, M., Wilf, H., Zeilberger, D.: $A = B$. A. K. Peters (1996)
21. Rajasekaran, A., Shallit, J., Smith, T.: Additive number theory via automata theory. Theor. Comput. Syst. **64**, 542–567 (2020)
22. Rajasekaran, A., Smith, T., Shallit, J.: Sums of palindromes: an approach via automata. In: Niedermeier, R., Vallée, B. (eds.) 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018), pp. 54:1–54:12. Leibniz International Proceedings in Informatics, Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2018)
23. Rampersad, N., Shallit, J.: Words avoiding reversed subwords. J. Combin. Math. Combin. Comput. **54**, 157–164 (2005)
24. Shallit, J.: Sumsets of Wythoff sequences, Fibonacci representation, andbeyond (2021). Period. Math. Hung., to appear
25. Tefera, A.: What is a Wilf-Zeilberger pair? Not. Am. Math. Soc. **57**, 508–509 (2010)