Sums of Palindromes: an Approach via Automata

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

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

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

Abstract

Recently, Cilleruelo, Luca, and Baxter proved, for all bases $b \geq 5$, that every natural number is the sum of at most 3 natural numbers whose base-b representation is a palindrome. However, the cases $b = 2, 3, 4$ were left unresolved. We prove, using a decision procedure based on automata, that every natural number is the sum of at most 4 natural numbers whose base-2 representation is a palindrome. Here the constant 4 is optimal. We obtain similar results for bases 3 and 4, thus completely resolving the problem.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Mathematics of computing → Combinatorics on words, Theory of computation → Automated reasoning

Keywords and phrases Finite automaton, Nested-word Automaton, Decision Procedure, Palindrome, Additive Number Theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.54

Acknowledgements We thank Dirk Nowotka, Parthasarathy Madhusudan, and Jean-Paul Allouche for helpful discussions. We thank the creators of the ULTIMATE automaton library for their assistance. Finally, we thank the referees of STACS 2018 for their helpful comments and corrections.

1 Introduction

In this paper we develop a new method, based on automata theory, for solving problems in additive number theory. As an example of the power of our method, we are able to prove the new result that *every natural number is the sum of at most* 4 *numbers whose base*-2 *representation is a palindrome.*

Additive number theory is the study of the additive properties of integers; it has a very long and celebrated history. For example, Lagrange proved (1770) that every natural number is the sum of four squares (see, e.g., [12]). In additive number theory, a subset $S \subseteq \mathbb{N}$ is called an *additive basis of order* h if every element of $\mathbb{N}$ can be written as a sum of at most h members of S, not necessarily distinct.

Waring's problem asks for the smallest value $g(k)$ such that the k'th powers form a basis of order $g(k)$. Lagrange's theorem then shows that $g(2) = 4$. In a variation on Waring's problem, one can ask for the smallest value $G(k)$ such that every *sufficiently large* natural number is the sum of $G(k)$ k'th powers [26]. This kind of representation is called an *asymptotic additive basis* of order $G(k)$.

35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 54; pp. 54:1–54:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Quoting Nathanson [19, p. 7], *"The central problem in additive number theory is to determine if a given set of integers is a basis of finite order."*

In this paper we show how to solve this central problem for certain sets of natural numbers, using automata theory and *almost no number theory at all*.

We are concerned with numbers with particular representations in base k. For example, numbers of the form $11\cdots1$ in base k are sometimes called *repunits* [28], and special effort has been devoted to factoring such numbers, with the Mersenne numbers 2^n-1 being the most famous examples. The *Nagell-Ljunggren problem* asks for a characterization of those repunits that are integer powers (see, e.g., [24]).

Another interesting class, and the one that principally concerns us in this article, consists of those numbers whose base-k representation forms a *palindrome*: a string that reads the same forwards and backwards, like the English word `redder`, the French word `ressasser`, and the German word `reliefpfeiler`. Palindromic numbers have been studied for some time in number theory; see, for example, [7, 6], just to name two recent references.

Recently Banks initiated the study of the additive properties of palindromes, proving that every natural number is the sum of at most 49 numbers whose decimal representation is a palindrome [5]. Banks' result was then improved by Cilleruelo, Luca, and Baxter [8, 9], who proved that for all bases $b \geq 5$, every natural number is the sum of at most 3 numbers whose base-b representation is a palindrome. The proofs of Banks and Cilleruelo, Luca, and Baxter are both rather lengthy and case-based. Up to now, there have been no results proved for bases $b = 2, 3, 4$.

The long case-based solutions to the problem of representation by sums of palindromes suggests that perhaps a more automated approach might be useful. We turn to formal languages and automata theory as a suitable framework for expressing the palindrome representation problem. Since we want to make assertions about the representations of *all* natural numbers, this requires finding (a) a machine model or logical theory in which universality is decidable and (b) a variant of the additive problem of palindromes suitable for this machine model or logical theory. The first model we use is the *nested-word automaton*, a variant of the more familiar pushdown automaton. This is used to handle the case for base $b = 2$. The second model we use is the ordinary finite automaton, which we use to resolve the cases $b = 3, 4$.

Our paper is organized as follows: In Section 2 we introduce some notation and terminology, and state more precisely the problem we want to solve. In Section 3 we recall the pushdown automaton model and give an example, and we motivate our use of nested-word automata. In Section 4 we restate our problem in the framework of nested-word automata, and the proof of a bound of 4 palindromes is given in Section 5. In Section 6 we make use of finite automata to prove a bound of 3 palindromes for bases 3 and 4. The novelty of our approach involves replacing the long case-based reasoning of previous proofs with an automaton-based approach using a decision procedure. In Section 7 we discuss possible objections to our approach. Finally, in Section 8 we discuss future work.

2 The sum-of-palindromes problem

We first introduce some notation and terminology.

The natural numbers are $\mathbb{N} = \{0, 1, 2, \ldots\}$. If n is a natural number, then by $(n)_k$ we mean the string (or word) representing n in base k, with no leading zeroes, starting with the most significant digit. Thus, for example, $(43)_2 = 101011$. The alphabet Σ_k is defined to be $\{0, 1, \ldots, k-1\}$; by Σ_k^* we mean the set of all finite strings over Σ_k. If $x \in \Sigma_\ell^*$ for some

ℓ, then by $[x]_k$ we mean the integer represented by the string x, considered as if it were a number in base k, with the most significant digit at the left. That is, if $x = a_1 a_2 \cdots a_n$, then $[x]_k = \sum_{1 \le i \le n} a_i k^{n-i}$. For example, $[135]_2 = 15$.

If x is a string, then x^i denotes the string $\overbrace{xx\cdots x}^{i}$, and x^R denotes the reverse of x. Thus, for example, $(\texttt{ma})^2 = \texttt{mama}$, and $(\texttt{drawer})^R = \texttt{reward}$. If $x = x^R$, then x is said to be a *palindrome*.

We are interested in integers whose base-k representations are palindromes. In this article, we routinely abuse terminology by calling such an integer a *base-k palindrome*. In the case where $k = 2$, we also call such an integer a *binary palindrome*. The first few binary palindromes are

$$0, 1, 3, 5, 7, 9, 15, 17, 21, 27, 31, 33, 45, 51, 63, \ldots;$$

these form sequence A006995 in the *On-Line Encyclopedia of Integer Sequences* (OEIS).

If $k^{n-1} \le r < k^n$ for $n \ge 1$, we say that r is an *n-bit integer* in base k. If k is unspecified, we assume that $k = 2$. Note that the first bit of an n-bit integer is always nonzero. The *length* of an integer r satisfying $k^{n-1} \le r < k^n$ is defined to be n; alternatively, the length of r is $1 + \lfloor \log_k r \rfloor$.

Our goal is to find a constant c such that every natural number is the sum of at most c binary palindromes. To the best of our knowledge, no such bound has been proved up to now. In Sections 4 and 5 we describe how we used a decision procedure for nested-word automata to prove the following result:

▶ **Theorem 1.** *For all $n \ge 8$, every n-bit odd integer is either a binary palindrome itself, or the sum of three binary palindromes*
(a) *of lengths n, $n-2$, and $n-3$; or*
(b) *of lengths $n-1$, $n-2$, and $n-3$.*

As a corollary, we get our main result:

▶ **Corollary 2.** *Every natural number N is the sum of at most 4 binary palindromes.*

Proof. It is a routine computation to verify the result for $N < 128$.

Now suppose $N \ge 128$. Let N be an n-bit integer; then $n \ge 8$. If N is odd, then Theorem 1 states that N is the sum of at most 3 binary palindromes. Otherwise, N is even.

If $N = 2^{n-1}$, then it is the sum of $2^{n-1} - 1$ and 1, both of which are palindromes.

Otherwise, $N - 1$ is also an n-bit odd integer. Use Theorem 1 to find a representation for $N - 1$ as the sum of at most 3 binary palindromes, and then add the palindrome 1 to get a representation for N. ◀

▶ Remark. We note that the bound 4 is optimal since, for example, the number 176 is not the sum of three or fewer binary palindromes.

Sequence A261678 in the OEIS lists those even numbers that are not the sum of two binary palindromes. Sequence A261680 gives the number of distinct representations as the sum of four binary palindromes.

3 Finding an appropriate computational model

To find a suitable model for proving Theorem 1, we turn to formal languages and automata. We seek some class of automata with the following property: for each k, there is an automaton which, given a natural number n as input, accepts the input if and only if n can be expressed

as the sum of k palindromes. Furthermore, we would like the problem of universality ("Does the automaton accept every possible input?") to be decidable in our chosen model. By constructing the appropriate automaton and checking whether it is universal, we could then determine whether every number n can be expressed as the sum of k palindromes.

Palindromes suggest considering the model of pushdown automaton (PDA), since it is well-known that this class of machines, equipped with a stack, can accept the palindrome language $\texttt{PAL} = \{x \in \Sigma^* \ : \ x = x^R\}$ over any fixed alphabet Σ. A tentative approach is as follows: create a PDA M that, on input n expressed in base 2, uses nondeterminism to "guess" the k summands and verify that (a) every summand is a palindrome, and (2) they sum to the input n. We would then use a decision procedure for universality to determine whether M accepts all of its inputs. However, two problems immediately arise.

The first problem is that universality is recursively unsolvable for nondeterministic PDAs (see, e.g.,[15, Thm. 8.11, p. 203]), so even if the automaton M existed, there would be no algorithm guaranteed to check universality.

The second problem involves checking that the guessed summands are palindromes. One can imagine guessing the summands in parallel, or in series. If we try to check them in parallel, this seems to correspond to the recognition of a language which is not a CFL (i.e., a context-free language, the class of languages recognized by nondeterministic PDAs). Specifically, we encounter the following obstacle:

▶ **Theorem 3.** *The set of strings L over the alphabet $\Sigma \times (\Sigma \cup \{\#\})$, where the first "track" is a palindrome and the second "track" is another, possibly shorter, palindrome, padded on the right with # signs, is not a CFL.*

Proof. Assume that it is. Consider L intersected with the regular language

$$[1,1]^+[1,0][1,1]^+[0,1][1,\#]^+,$$

and call the result L'. We use Ogden's lemma [20] to show L' is not a CFL.

Let n be the constant in Ogden's lemma, and choose z to be the string where the first track is $(1^{2n}01^{2n})$ and the second track is $(1^n01^n\#^{2n})$. Mark the compound symbols $[1,\#]$. Then every factorization $z = uvwxy$ with vx nonempty must have at least one $[1,\#]$ in v or x. If it is in v, then the only choice for x is also $[1,\#]$, so pumping gives a non-palindrome on the first track. If it is in x then v can be $[1,1]^i$ or contain $[1,0]$ or $[0,1]$. If the latter, pumping twice gives a string not in L' because there is more than one 0 on one of the two tracks. If the former, pumping twice gives a string with the second track not a palindrome. This contradiction shows that L', and hence L, is not a context-free language. ◀

So, using a pushdown automaton, we cannot check whether two arbitrary strings of wildly unequal lengths, presented in parallel, are both palindromes.

If the summands were presented serially, we could check whether each summand individually is a palindrome, using the stack, but doing so destroys our copy of the summand, and so we cannot add them all up and compare them to the input. In fact, we cannot add serial summands in any case, because we have

▶ **Theorem 4.** *The language*

$$L = \{(m)_2\#(n)_2\#(m+n)_2 \ : \ m,n \geq 0\}$$

is not a CFL.

Proof. Assume L is a CFL and intersect with the regular language $1^+01^+\#1^+\#1^+0$, obtaining L'. We claim that

$$L' = \{1^a01^b\#1^c\#1^d0 \ : \ b = c \text{ and } a + b = d\}.$$

This amounts to the claim, easily verified, that the only solutions to the equation $2^{a+b+1} - 2^b - 1 + 2^c - 1 = 2^{d+1} - 2$ are $b = c$ and $a + b = d$. Then, if n is the constant in Ogden's lemma, starting with the string $z = 1^n01^n\#1^n\#1^{2n}0$, an easy argument proves that L' is not a CFL, and hence neither is L. ◀

So, using a pushdown automaton, we cannot handle summands presented in series, either.

These issues lead us to restrict our attention to representations as sums of palindromes of the same (or similar) lengths. More precisely, we consider the following variant of the additive problem of palindromes: for a length l and number of summands k, given a natural number n as input, is n the sum of k palindromes all of length exactly l? Since the palindromes are all of the same length, a stack would allow us to guess and verify them in parallel. To tackle this problem, we need a model which is both (1) powerful enough to handle our new variant, and (2) restricted enough that universality is decidable. We find such a model in the class of *nested-word automata*, described in the next section.

4 Restating the problem in the language of nested-word automata

Nested-word automata (NWAs) were popularized by Alur and Madhusudan [1, 2], although essentially the same model was discussed previously by Mehlhorn [18], von Braunmühl and Verbeek [27], and Dymond [10]. They are closely related to a restricted variant of pushdown automata called visibly-pushdown automata (VPAs). Under linear encodings, NWAs recognize the same class of languages as VPAs, namely the visibly-pushdown languages [1, 2]. We only briefly describe their functionality here. For other theoretical aspects of nested-word and visibly-pushdown automata, see [16, 22, 11, 23, 21]. The definition of NWAs provided here modifies the standard definition by borrowing some aspects of VPAs. We use this definition because that is what is used by the ULTIMATE program analysis framework [14, 13], which is the software tool we use to prove our results.

The input alphabet of an NWA is partitioned into three sets: a *call alphabet*, an *internal alphabet*, and a *return alphabet*. An NWA has a stack, but has more restricted access to it than PDAs do. If the input symbol read is from the call alphabet, the NWA pushes its current state onto the stack, and then performs a transition, based only on the current state and input symbol read. If an input symbol is from the internal alphabet, the NWA cannot access the stack in any way. If the input symbol read is from the return alphabet, the NWA pops the state at the top of the stack, and then performs a transition based on three pieces of information: the current state, the popped state, and the input state read. An NWA accepts if the state it terminates in is an accepting state.

As an example, Figure 1 illustrates a nested-word automaton accepting the language $\{0^n12^n \ : \ n \geq 1\}$. Here the call alphabet is $\{0\}$, the internal alphabet is $\{1\}$, and the return alphabet is $\{2\}$.

The first 0 pushes q_0 onto the stack. Each of the $n - 1$ subsequent 0s push q_1 onto the stack. When the machine reads the 1, it goes to state q_2 without accessing the stack. So long as the machine reads 2s and q_1 is on the stack, we stay in the non-accepting state q_2. Reading a 2 with q_0 on the top of the stack takes us to the only accepting state, q_3.

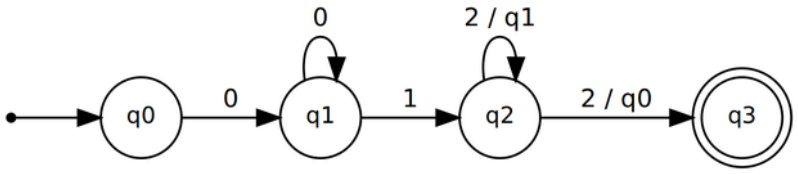

Figure 1 A nested-word automaton for the language $\{0^n 1 2^n \ : \ n \geq 1\}$.

Nondeterministic NWAs are a good machine model for our problem, because nondeterminism allows "guessing" the palindromes that might sum to the input, and the stack allows us to "verify" that they are indeed palindromes. Deterministic NWAs are as expressive as nondeterministic NWAs, and the class of languages they accept is closed under the operations of union, complement and intersection. Finally, testing emptiness, universality, and language inclusion are all decidable problems for NWAs [1, 2].

For a nondeterministic NWA of n states, the corresponding determinized machine has at most $2^{\Theta(n^2)}$ states, and there are examples for which this bound is attained. This very rapid explosion in state complexity potentially could make decision problems, such as language inclusion, infeasible in practice. Fortunately, we did not run into determinized machines with more than 40000 states in proving our results. Most of the algorithms invoked to prove our results run in under a minute.

We now discuss the general construction of the NWAs that check whether inputs are sums of binary palindromes. We partition the input alphabet into the call alphabet $\{a, b\}$, the internal alphabet $\{c, d\}$, and the return alphabet $\{e, f\}$. The symbols a, c, and e correspond to 0, while b, d, and f correspond to 1. The input string is fed to the machine starting with the *least significant digit*. We provide the NWA with input strings whose first half is entirely made of call symbols, and second half is entirely made of return symbols. Internal symbols are used to create a divider between the halves (for the case of odd-length inputs).

The idea behind the NWA is to nondeterministically guess all possible summands when reading the first half of the input string. The guessed summands are characterized by the states pushed onto the stack. The machine then checks if the guessed summands can produce the input bits in the second half of the string. The machine keeps track of any carries in the current state.

Following the above construction, we implemented our NWAs in the ULTIMATE program analysis framework. As an experimental spot check on the correctness of our implementations, we also built an NWA-simulator, and ran simulations of the machines on various types of inputs, which we then checked against experimental results.

For instance, we built the machine that accepts representations of integers that can be expressed as the sum of 2 binary palindromes. We then simulated this machine on every integer from 513 to 1024, and checked that it only accepts those integers that we experimentally confirmed as being the sums of 2 binary palindromes. This did not prove our results, but served as a sanity check.

The general procedure to prove our results is to build an NWA `PalSum` accepting only those inputs that it verifies as being appropriate sums of palindromes, as well as an NWA `SyntaxChecker` accepting all valid representations. We then run the decision algorithms

for language inclusion, language emptiness, etc. on `PalSum` and `SyntaxChecker` as needed. To do this, we used the Automata Library toolchain of the ULTIMATE program analysis framework.

We have provided links to the proof scripts used to establish all of our results. To run these proof scripts, simply copy the contents of the script into `https://monteverdi.informatik.uni-freiburg.de/tomcat/Website/?ui=int&tool=automata_library` and click "execute". Some of the larger proof scripts might time out on the web client, but can be run to completion by downloading the ULTIMATE Automata Library toolchain and executing them on a local machine.

5 Proving Theorem 1

In this section, we discuss construction of the appropriate nested-word automaton in more detail.

Proof of Theorem 1. We build three separate automata. The first, `palChecker`, has 9 states, and simply checks whether the input number is a binary palindrome. The second, `palChecker2`, has 771 states, and checks whether an input number of length n can be expressed as the sum of three binary palindromes of lengths n, $n-2$, and $n-3$. The third machine, `palChecker3`, has 1539 states, and checks whether an input number of length n can be expressed as the sum of three binary palindromes of lengths $n-1$, $n-2$, and $n-3$. We then determinize these three machines, and take their union, to get a single deterministic NWA, `FinalAut`, with 36194 states. We then run the command `FinalAut = shrinkNwa(FinalAut);` to reduce this to a deterministic NWA of only 106 states.

The language of valid inputs to our automata is given by

$$L = \{\{a,b\}^n\{c,d\}^m\{e,f\}^n \ : \ 0 \le m \le 1,\ n \ge 4\}.$$

We only detail the mechanism of `palChecker3` here. Let p, q and r be the binary palindromes representing the guessed $(n-1)$-length summand, $(n-2)$-length summand and $(n-3)$-length summand respectively. The states of `palChecker3` include 1536 t-states that are 10-tuples. We label these states $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$, where $0 \le g \le 2$, while all other coordinates are either 0 or 1. The g-coordinate indicates the current carry, and can be as large as 2. The x, y and z coordinates indicate whether we are going to guess 0 or 1 for the next guesses of p, q and r respectively. The remaining coordinates serve as "memory" to help us manage the differences in lengths of the guessed summands. The k-coordinate records the most recent guess for p. We have l_1 and l_2 record the two most recent q guesses, with l_1 being the most recent one, and we have m_1, m_2 and m_3 record the three most recent r-guesses, with m_1 being the most recent one, then m_2, and m_3 being the guess we made three steps ago. We also have three s-states labeled s_0, s_1 and s_2, representing carries of 0, 1 and 2 respectively. These states process the second half of the input string.

The initial state of the machine is $(0, 1, 1, 1, 0, 0, 0, 0, 0, 0)$ since we start with no carry, must guess 1 for our first guess of a valid binary palindrome, and all "previous" guesses are 0. A t-state has an outgoing transition on either a or b, but not both. If $g + x + y + z$ produces an output bit of 0, it takes a transition on a, else it takes a transition on b. The destination states are all six states of the form $(g', x', y', z', x, y, l_1, z, m_1, m_2)$, where g' is the carry resulting from $g + x + y + z$, and x', y', z' can be either 0 or 1. Note that we "update" the remembered states by forgetting k, l_2 and m_3, and saving x, y and z.

The s-states only have transitions on the return symbols e and f. When we read these symbols, we pop a t-state off the stack. If state s_i pops the state $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$

off the stack, its transition depends on the addition of $i + k + l_2 + m_3$. If this addition produces a carry of j, then s_i can take a transition to s_j on e if the output bit produced is 0, and on f otherwise. By reading the k, l_2 and m_3 values, we correctly realign p, q and r, correcting for their different lengths. This also ensures that we supply 0s for the last three guesses of r, the last two guesses of q and the last guess of p. The only accepting state is s_0.

It remains to describe how we transition from t-states to s-states. This transition happens when we are halfway done reading the input. If the length of the input is odd, we read a c or a d at the halfway point. We only allow certain t-states to have outgoing transitions on c and d. Specifically, we require the state's coordinates to satisfy $k = x$, $l_2 = y$, $m_1 = m_2$ and $m_3 = z$. These conditions are required for p, q and r to be palindromes. We transition to state s_i if the carry produced by adding $g + x + y + z$ is i, and we label the transition c if the output bit is 0, and d otherwise.

If the length of the input is even, then our t-states take a return transition on e or f. Once again, we restrict the t-states that can take return transitions. We require the state's coordinates to satisfy $l_1 = l_2$ and $m_1 = m_3$ to ensure our guessed summands are palindromes. Let the current state be $(g, x, y, z, k, l_1, l_1, m_1, m_2, m_1)$, and the state at the top of the stack be $(g', x', y', z', k', l'_1, l'_2, m'_1, m'_2, m'_3)$. We can take a transition to s_i if the sum $g+k'+l'_2+m'_3$ produces a carry of i, and we label the transition e if the output bit is 0, and f otherwise.

The structure and behavior of `palChecker2` is very similar. One difference is that there is no need for a k-coordinate in the t-states since the longest summand guessed is of the same length as the input.

The complete script executing this proof is over 750000 lines long. Since these automata are very large, we wrote two C++ programs to generate them. Both the proof script, and the programs generating them can be found at `https://cs.uwaterloo.ca/~shallit/papers.html`. It is worth noting that a t-state labeled as $(g, x, y, z, k, l_1, l_2, m_1, m_2, m_3)$ in this report is labeled $q\_g\_xyz\_k\_l_1l_2\_m_1m_2m_3$ in the proof script. Also, ULTIMATE does not currently have a union operation for NWAs, so we work around this by using De Morgan's laws for complement and intersection. ◀

6 Bases 3 and 4

In this section we prove analogous results for bases 3 and 4. We show that every natural number is the sum of at most three base-3 palindromes, and at most three base-4 palindromes. Because the NWAs needed became too large for us to manipulate effectively, we use a modified approach using nondeterministic finite automata to prove these results. This approach was suggested to us by Dirk Nowotka and Parthasarathy Madhusudan, independently.

Our result for base 3 is as follows:

▶ **Theorem 5.** *For all $n \geq 9$, every integer whose base-3 representation is of length n is the sum of*

(a) *three base-3 palindromes of lengths n, $n-1$, and $n-2$; or*
(b) *three base-3 palindromes of lengths n, $n-2$, and $n-3$; or*
(c) *three base-3 palindromes of lengths $n-1$, $n-2$, and $n-3$; or*
(d) *two base-3 palindromes of lengths $n-1$ and $n-2$.*

Proof. We represent the input in a "folded" manner over the input alphabet $\Sigma_3 \cup (\Sigma_3 \times \Sigma_3)$, where $\Sigma_k = \{0, 1, \ldots, k-1\}$, giving the machine 2 letters at a time from opposite ends. This way we can directly guess our summands without having need of a stack at all. We align the input along the length-$n-2$ summand by providing the first 2 letters of the input separately.

If $(N)_3 = a_{2i+1}a_{2i}\cdots a_0$, we represent N as the word

$$a_{2i+1}a_{2i}[a_{2i-1}, a_0][a_{2i-2}, a_1]\cdots[a_i, a_{i-1}].$$

Odd-length inputs leave a trailing unfolded letter at the end of their input. If $(N)_3 = a_{2i+2}a_{2i+1}\cdots a_0$, we represent N as the word

$$a_{2i+2}a_{2i+1}[a_{2i}, a_0][a_{2i-1}, a_1]\cdots[a_{i+1}, a_{i-1}]a_i.$$

We need to simultaneously carry out addition on both ends. In order to do this we need to keep track of two carries. On the lower end, we track the "incoming" carry at the start of an addition, as we did in the proofs using NWAs. On the higher end, however, we track the expected "outgoing" carry.

To illustrate how our machines work, we consider an NFA accepting length-n inputs that are the sum of 4 base-3 palindromes, one each of lengths n, $n-1$, $n-2$ and $n-3$. Although this is not a case in our theorem, each of the four cases in our theorem can be obtained from this machine by striking out one or more of the guessed summands.

Recall that we aligned our input along the length-$(n-2)$ summand by providing the 2 most significant letters in an unfolded manner. This means that our guesses for the length-n summand will be "off-by-two": when we make a guess at the higher end of the length-n palindromic summand, its appearance at the lower end is 2 steps away. We hence need to record the last 2 guesses at the higher end of the length-n summand in our state. Similarly, we need to record the most recent higher guess of the length-$(n-1)$ summand, since it is off by one. The length-$(n-2)$ summand is perfectly aligned, and hence nothing needs to be recorded. The length-$(n-3)$ summand has the opposite problem of the length-$(n-1)$ input. Its lower guess only appears at the higher end one step later, and so we save the most recent guess at the lower end.

Thus, in this machine, we keep track of 6 pieces of information:

- c_1, the carry we are expected to produce on the higher end,
- c_2, the carry we have entering the lower end,
- x_1 and x_2, the most recent higher guesses of the length-n summand,
- y, the most recent higher guess of the length-$(n-1)$ summand, and
- z, the most recent lower guess of the length-$(n-3)$ summand,

Consider a state $(c_1, c_2, x_1, x_2, y, z)$. Let $i, j, k, l \in [0, 2]$ be our next guesses for the four summands of lengths n, $n-1$, $n-2$ and $n-3$ respectively. Also, let α be our guess for the next incoming carry on the higher end. Let the result of adding $i+j+k+z+\alpha$ be a value $0 \le p_1 < 3$ and a carry of q_1. Let the result of adding $x+y+k+l+x_2$ be a value $0 \le p_2 < 3$ and a carry of q_2. We must have $q_1 = c_1$. If this condition is met, we add a transition from this state to $(\alpha, q_2, x_2, i, j, l)$, and label the transition $[p_1, p_2]$.

The initial state is $(0, 0, 0, 0, 0, 0)$. We expand the alphabet to include special variables for the first 3 symbols of the input string. This is to ensure that we always guess a 1 or a 2 for the first (and last) positions of our summands.

The acceptance conditions depend on whether $(N)_3$ is of even or odd length. If a state $(c_1, c_2, x_1, x_2, y, z)$ satisfies $c_1 = c_2$ and $x_1 = x_2$, we set it as an accepting states. A run can only terminate in one of these states if $(N)_3$ is of even length. We accept since we are confident that our guessed summands are palindromes (the condition $x_1 = x_2$ ensures our length-n summand is palindromic), and since the last outgoing carry on the lower end is the expected first incoming carry on the higher end (enforced by $c_1 = c_2$).

We also have a special symbol to indicate the trailing symbol of an input for which $(N)_3$ is of odd length. We add transitions from our states to a special accepting state, q_{acc}, if

we read this special symbol. Consider a state $(c_1, c_2, x_1, x_2, y, z)$, and let $0 \leq k < 3$ be our middle guess for the $n-2$ summand. Let the result of adding $x_1 + y + k + z + c_2$ be a value $0 \leq p < 3$ and a carry of q. If $q = c_1$, we add a transition on p from our state to q_{acc}.

We wrote a C++ program generating a single NFA with 4 parts, one for each case of the theorem. After minimizing, the machine has 378 states. We then built a second NFA that accepts folded representations of $(N)_3$ such that the unfolded length of $(N)_3$ is greater than 8. We then use ULTIMATE to assert that the language accepted by the second NFA is included in that accepted by the first. All these operations run in under a minute.

We tested this machine by experimentally calculating which values of $243 \leq N \leq 1000$ could be written as the sum of palindromes satisfying one of our 4 conditions. We then asserted that for all the folded representations of $243 \leq N \leq 1000$, our machine accepts these values which we experimentally calculated, and rejects all others. ◀

We also have the following result for base 4:

▶ **Theorem 6.** *For all $n \geq 7$, every integer whose base-4 representation is of length n is the sum of*

(a) *exactly one palindrome each of lengths $n-1$, $n-2$, and $n-3$; or*

(b) *exactly one palindrome each of lengths n, $n-2$, and $n-3$.*

Proof. The NFA we build is very similar to the machine described for the base-3 proof. Indeed, the generator used is the same as the one for the base-3 proof, except that its input base is 4, and the only machines it generates are for the two cases of this theorem. The minimized machine has 478 states. ◀

This, together with the results previously obtained by Cilleruelo, Luca, and Baxter, completes the additive theory of palindromes for all integer bases $b \geq 2$.

7 Objections to this kind of proof

A proof based on computer calculations, like the one we have presented here, is occasionally criticized because it cannot easily be verified by hand, and because it relies on software that has not been formally proved. These kinds of criticisms are not new; they date at least to the 1970's, in response to the celebrated proof of the four-color theorem by Appel and Haken [3, 4]. See, for example, Tymoczko [25].

We answer this criticism in several ways. First, it is not reasonable to expect that every result of interest to mathematicians will have short and simple proofs. There may well be, for example, easily-stated results for which the shortest proof possible in a given axiom system is longer than any human mathematician could verify in their lifetime, even if every waking hour were devoted to checking it. For these kinds of results, an automated checker may be our only hope. There are many results for which the only proof currently known is computational.

Second, while short proofs can easily be checked by hand, what guarantee is there that any very long case-based proof — whether constructed by humans or computers — can always be certified by human checkers with a high degree of confidence? There is always the potential that some case has been overlooked. Indeed, the original proof of the four-color theorem by Appel and Haken apparently overlooked some cases. Similarly, the original proof by Cilleruelo and Luca on sums of palindromes [8] had some minor flaws that became apparent once their method was implemented as a `python` program; these were later corrected in [9].

Third, confidence in the correctness of the results can be improved by providing code that others may check. Transparency is essential. To this end, we have provided our code for the nested-word automata, and the reader can easily run this code on the software we referenced.

8 Future work

We can use the same sorts of ideas to attack other problems in additive number theory. For example, we can obtain results about "generalized palindromes" (allowing an arbitrary number of leading zeroes), "anti-palindromes" (of the form $x\overline{x^R}$), "generalized anti-palindromes", and so forth.

In a recent paper [17], we use the same kinds of techniques to prove an analogue of Lagrange's theorem for binary "squares" (those numbers whose representation in base 2 consists of two consecutive identical blocks).

References

1. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
2. Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.
3. K. Appel and W. Haken. Every planar map is four colorable. I. Discharging. *Illinois J. Math.*, 21:429–490, 1977.
4. K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. II. Reducibility. *Illinois J. Math.*, 21:491–567, 1977.
5. W. D. Banks. Every natural number is the sum of forty-nine palindromes. *INTEGERS — Electronic J. Combinat. Number Theory*, 16, 2016. #A3.
6. W. D. Banks and I. E. Shparlinski. Average value of the Euler function on binary palindromes. *Bull. Pol. Acad. Sci. Math.*, 54:95–101, 2006. doi:10.4064/ba54-2-1.
7. William D. Banks and Igor E. Shparlinski. Prime divisors of palindromes. *Periodica Mathematica Hungarica*, 51(1):1–10, 2005. doi:10.1007/s10998-005-0016-6.
8. J. Cilleruelo and F. Luca. Every positive integer is a sum of three palindromes. Preprint available at https://arxiv.org/abs/1602.06208v1, 2016.
9. J. Cilleruelo, F. Luca, and L. Baxter. Every positive integer is a sum of three palindromes. *Math. Comp.*, 2017. doi:10.1090/mcom/3221.
10. Patrick W. Dymond. Input-driven languages are in log n depth. *Inf. Process. Lett.*, 26(5):247–250, 1988. doi:10.1016/0020-0190(88)90148-2.
11. Yo-Sub Han and Kai Salomaa. Nondeterministic state complexity of nested word automata. *Theor. Comput. Sci.*, 410(30-32):2961–2971, 2009. doi:10.1016/j.tcs.2009.01.004.
12. G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 1985.
13. Matthias Heizmann, Daniel Dietsch, Marius Greitschus, Jan Leike, Betim Musa, Claus Schätzle, and Andreas Podelski. Ultimate automizer with two-track proofs - (competition contribution). In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 950–953. Springer, 2016. doi:10.1007/978-3-662-49674-9_68.

14 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2013. doi:10.1007/978-3-642-39799-8_2.

15 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

16 Salvatore La Torre, Margherita Napoli, and Mimmo Parente. On the membership problem for visibly pushdown languages. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006.*, volume 4218 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2006. doi:10.1007/11901914_10.

17 P. Madhusudan, D. Nowotka, A. Rajasekaran, and J. Shallit. Lagrange's theorem for binary squares. Preprint available at https://arxiv.org/abs/1710.04247, 2017.

18 Kurt Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.

19 M. B. Nathanson. *Additive Number Theory: The Classical Bases*. Springer-Verlag, 1996.

20 William F. Ogden. A helpful result for proving inherent ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968. doi:10.1007/BF01694004.

21 Alexander Okhotin and Kai Salomaa. State complexity of operations on input-driven pushdown automata. *J. Comput. Syst. Sci.*, 86:207–228, 2017. doi:10.1016/j.jcss.2017.02.001.

22 Xiaoxue Piao and Kai Salomaa. Operational state complexity of nested word automata. *Theor. Comput. Sci.*, 410(35):3290–3302, 2009. doi:10.1016/j.tcs.2009.05.002.

23 Kai Salomaa. Limitations of lower bound methods for deterministic nested word automata. *Inf. Comput.*, 209(3):580–589, 2011. doi:10.1016/j.ic.2010.11.021.

24 T. N. Shorey. On the equation $z^q = (x^n - 1)/(x - 1)$. *Indag. Math.*, 48:345–351, 1986. doi:10.1016/1385-7258(86)90020-X.

25 T. Tymoczko. The four-color problem and its philosophical significance. *J. Philosophy*, 76(2):57–83, 1979.

26 R. C. Vaughan and T. Wooley. Waring's problem: a survey. In M. A. Bennett, B. C. Berndt, N. Boston, H. G. Diamond, A. J. Hildebrand, and W. Philipp, editors, *Number Theory for the Millennium. III*, pages 301–340. A. K. Peters, 2002.

27 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In Marek Karpinski, editor, *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1983. doi:10.1007/3-540-12689-9_92.

28 S. Yates. The mystique of repunits. *Math. Mag.*, 51:22–28, 1978.