

A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory

Adam Yedidia
MIT
adamy@mit.edu

Scott Aaronson
MIT
aaronson@csail.mit.edu

May 17, 2016

Abstract

Since the definition of the Busy Beaver function by Radó in 1962, an interesting open question has been the smallest value of n for which $BB(n)$ is independent of ZFC set theory. Is this n approximately 10, or closer to 1,000,000, or is it even larger? In this paper, we show that it is at most 7,910 by presenting an explicit description of a 7,910-state Turing machine Z with 1 tape and a 2-symbol alphabet that cannot be proved to run forever in ZFC (even though it presumably does), assuming ZFC is consistent. The machine is based on work of Harvey Friedman on independent statements involving order-invariant graphs. In doing so, we give the first known upper bound on the highest provable Busy Beaver number in ZFC. To create Z , we develop and use a higher-level language, Laconic, which is much more convenient than direct state manipulation. We also use Laconic to design two Turing machines, G and R , that halt if and only if there are counterexamples to Goldbach's Conjecture and the Riemann Hypothesis, respectively.

1 Introduction

1.1 Background and Motivation

Zermelo-Fraenkel set theory with the axiom of choice, more commonly known as ZFC, is an axiomatic system invented in the twentieth which has since been used as the foundation of most of modern mathematics. It encodes arithmetic by describing natural numbers as increasing sets of sets.

Like any axiomatic system capable of encoding arithmetic, ZFC is constrained by Gödel's two incompleteness theorems. The first incompleteness theorem states that if ZFC is *consistent* (it never proves both a statement and its opposite), then ZFC cannot also be *complete* (able to prove every true statement). The second incompleteness theorem states that if ZFC is consistent, then ZFC cannot prove its own consistency. Because we have built modern mathematics on top of ZFC, we can reasonably be said to have assumed ZFC's consistency. This means that we must also believe that ZFC cannot prove its own consistency. This fact carries with it certain surprising conclusions.

In particular, consider a Turing machine Z that enumerates, one after the other, each of the provable statements in ZFC. To describe how such a machine might be constructed, Z could iterate over the axioms and inference rules of ZFC, applying each in every possible way to each conclusion

or pair of conclusions that had been reached so far. We might ask Z to halt if it ever reaches a contradiction; in other words, Z will halt if and only if it finds a proof of $0 = 1$. Because this machine will enumerate *every* provable statement in ZFC, it will run forever if and only if ZFC is consistent.

It follows that Z is a Turing machine for which the question of its behavior (whether or not it halts when run indefinitely) is equivalent to the consistency of ZFC.¹ Therefore, just as ZFC cannot prove its own consistency (assuming ZFC is consistent), ZFC also cannot prove that Z will run forever. In other words, the statement, “ Z will run forever” is *independent of* ZFC.

This is interesting because, while the undecidability of the halting problem tells us that there cannot exist an algorithmic method for determining whether an *arbitrary* Turing machine loops or halts, Z is an example of a *specific* Turing machine whose behavior cannot be proven one way or the other using the foundation of modern mathematics. Mathematicians and computer scientists think of themselves as being able to determine how a given algorithm will behave if given enough time to stare at it; despite this intuition, Z is a machine whose behavior we can never prove without assuming axioms more powerful than those generally assumed in modern mathematics.

1.2 Turing Machines

There are many slightly different definitions of Turing machines. For example, some definitions allow the machine to have multiple tapes; others only allow it to have one; some allow an arbitrarily large alphabet, while others allow only two symbols, and so on. In most research regarding Turing machines, mathematicians don’t concern themselves with which of these models to use, because any one can simulate the others (usually efficiently). However, because this work is concerned with upper-bounding the exact number of states required to perform certain tasks, it’s important to define the model precisely. The model we choose here is traditional for the Busy Beaver function.

Formally, a k -state Turing machine is a 7-tuple $M = (Q, \Gamma, a, \Sigma, \delta, q_0, F)$, where:

Q is the set of k states $\{q_0, q_1, \dots, q_{k-2}, q_{k-1}\}$

$\Gamma = \{a, b\}$ is the set of *tape alphabet symbols*

a is the *blank symbol*

Σ is the set of *input symbols*

$\delta = Q \times \Gamma \rightarrow (Q \cup F) \times \Gamma \times \{L, R\}$ is the *transition function*

q_0 is the *start state*

$F = \{\text{HALT}, \text{ERROR}\}$ is the set of *halting states*.

A Turing machine’s *states* make up the Turing machine’s easily-accessible, finite memory. The Turing machine’s state is initialized to q_0 .

The *tape alphabet symbols* correspond to the symbols that can be written on the Turing machine’s infinite tape.

In this work, all Turing machines are run on the all- a input.

The *transition function* encodes the Turing machine’s behavior. It takes two inputs: the current state of the Turing machine (an element of $Q \cup F$) and the symbol read off the tape (an element of

¹While we will talk about ZFC throughout this paper, rather than simple ZF set theory, this is simply a convention. For our purposes, the Axiom of Choice is irrelevant: the consistency of ZFC is equivalent to the consistency of simple ZF set theory, [14] and ZFC and ZF prove exactly the same arithmetical statements (which include, among other things, statements about whether Turing machines halt). [23]

Γ). It outputs three instructions: what state to enter (an element of $Q \cup F$), what symbol to write onto the tape (an element of Γ) and what direction to move the head in (an element of $\{L, R\}$). A transition function specifies the entire behavior of the Turing machine in all cases.

The *start state* is the state that the Turing machine is in at initialization.

A *halting transition* is a transition to a halting state, which causes the Turing machine to halt. While having three possible halting transitions is not necessary for our purposes, being able to differentiate between different types of halting (HALT and ERROR) is useful for testing.

1.3 The Busy Beaver Function

Consider the set of all Turing machines with k states, for some positive integer k . We call a Turing machine B a *k-state Busy Beaver* if when run on the empty tape as input, B halts, and also runs for at least as many steps before halting as all other halting k -state Turing machines. [22]

In other words, a Busy Beaver is a Turing machine that runs for at least as long as all other halting Turing machines with the same number of states. Another common definition for a Busy Beaver is a Turing machine that writes as many 1's on the tape as possible; because the number of 1's written is a somewhat arbitrary measure, it is not used in this work.

The *Busy Beaver function*, written $BB(k)$, equals the number of steps it takes for a k -state Busy Beaver to halt. The Busy Beaver function has many striking properties. To begin with, it is not *computable*; in other words, there does not exist an algorithm that takes k as input and returns $BB(k)$, for arbitrary values of k . This follows directly from the undecidability of the halting problem. Suppose an algorithm existed to compute the Busy Beaver function; then given a k -state Turing machine M as input, we could compute $BB(k)$ and run M for $BB(k)$ steps. If, after $BB(k)$ steps, M had not yet halted, we could safely conclude that M would never halt. Thus, we could solve the halting problem, which we know is impossible.

By the same argument, $BB(k)$ must grow faster than any computable function. (To check this, assume that some computable function $f(k)$ grows faster than $BB(k)$, and substitute $f(k)$ for $BB(k)$ in the rest of the proof.) In particular, the Busy Beaver grows even faster than (for instance) the Ackermann function, a well-known fast-growing function.

Because finding the value of $BB(k)$ for a given k requires so much work (one must fully explore the behavior of all k -state Turing machines), few explicit values of the Busy Beaver function are known. The known values are [4, 16]:

$$BB(1) = 1$$

$$BB(2) = 6$$

$$BB(3) = 21$$

$$BB(4) = 107$$

For $BB(5)$, $BB(6)$, and $BB(7)$ only lower bounds are known [19, 8]:

$$BB(5) \geq 47,176,870$$

$$BB(6) > 7.4 \times 10^{36,534}$$

$$BB(7) > 10^{10^{10^{10^7}}}$$

Additionally, $BB(22)$ is known to be larger than Graham’s Number (a famous huge number from Ramsey theory, obtained by iterating the Ackermann function 64 times) [9]. Researchers have worked on pinning down the value of $BB(5)$ exactly, and some consider it to be possibly within reach.

Another way to discuss the Busy Beaver sequence is to say that modern mathematics has established a *lower bound* of 4 on the highest provable Busy Beaver value. In this paper, we prove the first known *upper bound* on the highest provable Busy Beaver value in ZFC; that is, we give a value of k , namely 7,910, such that the value of $BB(k)$ cannot be proven in ZFC.

Intuitively, one might expect that while no algorithm may exist to compute $BB(k)$ for *all* values of k , we could find the value of $BB(k)$ for any *specific* k using a procedure similar to the one we used to find the value of $BB(k)$ for $k \leq 4$. The reason this is not so is closely tied to the existence of a machine like the Gödelian machine Z , as described in Section 1.1. Suppose that Z has k states. Because Z ’s behavior (whether it halts or loops) cannot be proven in ZFC, it follows that the value of $BB(k)$ also can’t be proven in ZFC; if it could, then a proof would exist of Z ’s behavior in ZFC. Such a proof would consist of a *computation history* for Z , which is an explicit step-by-step description of Z ’s behavior for a certain number of steps. If Z halts, then a computation history leading up to Z ’s halting would be the entire proof; if Z loops, then a computation history that takes $BB(k)$ steps, combined with a proof of the value of $BB(k)$, would constitute a proof that Z will run forever.

In this paper we construct a machine like Z , for which a proof that Z runs forever would imply that ZFC was consistent. In doing so, we give an explicit upper bound on the highest Busy Beaver value provable in ZFC assuming the consistency of a slightly stronger set theory. Our machine, which we shall refer to as Z hereafter, contains 7,910 states. Therefore, we will never be able to prove the value of $BB(7,910)$ without assuming more powerful axioms than those of ZFC. This upper bound is presumably very far from tight, but it is a first step.

Even to achieve a state count of 7,910, we will need three nontrivial ideas: Harvey Friedman’s order-theoretic statements, *on-tape processing*, and *introspective encoding*. Without all three ideas, we found that the state count would be in the tens of thousands, hundreds of thousands, or even millions. We briefly introduce these ideas in the following subsection, and explore them in much greater detail in Section 8. The implementation of these ideas constitutes this paper’s main technical contribution.

1.4 Parsimony

In most algorithmic study, efficiency is the primary concern. In designing Z , however, parsimony is the only thing that matters. One historical analogue is the practice of “code-golfing”: a recreational pursuit adopted by some programmers in which the goal is to produce a piece of code in a given programming language, using as few characters as possible. Many examples of code-golfing can be found at [26]. The goal of designing a Turing machine with as few states as possible to accomplish a certain task, without concern for the machine’s efficiency or space usage, can be thought of as code-golfing with a particularly low-level programming language.

Part of the charm of Turing machines is that they give us a “standard reference point” for measuring complexity, unencumbered by the details of more sophisticated programming languages. Also, with Turing machines, there can be no suspicion that we engineered a programming formalism just for the purpose of code-golfing, or for making the concepts we want artificially simple to describe. This is why we prefer Turing machines as a tool for measuring complexity; not because

they are particularly special, but simply because they are so primitive that their specifics will interfere minimally with what we mean by an algorithm being “complicated.”

In this paper, we use three ideas for generating parsimonious Turing machines: Harvey Friedman’s mathematical statements, *on-tape processing*, and *introspective* Turing machines. The last of these ideas was proposed, under a different name and with some variations, by Ben-Amram and Petersen in 2002 [3]. These three ideas are explained in more detail in Subsections 3.1, 8.1, and 8.3, respectively, but we summarize them very briefly here.

The first idea is simply to use the research done by Friedman into finding simple-to-express statements that are equivalent to the consistency of various axiomatic systems. In particular, we use a statement discovered by Friedman to be equivalent to the consistency of a set theory stronger than ZFC (and whose consistency, therefore, would imply the consistency of ZFC).² [10]

The second idea, on-tape processing, is a way to encode high-level commands into a Turing machine parsimoniously. Instead of converting commands to groups of states directly, which incurs a multiplicative overhead based on how large these groups need to be, on-tape processing begins by writing the commands onto the tape, using as efficient an encoding as possible. Then, once the commands are on the tape, the commands are processed by a single group of states that understands how to interpret them.

The third idea, introspective Turing machines, is a way to write long strings onto the tape using as few states as possible. The idea is to encode information one of each state’s transitions, instead of encoding information in each state’s write field. This is advantageous because there are many choices for which state to point a transition to, but only two choices for which bit to write. Therefore, more information can be encoded in each state using this method.

1.5 Implementation Overview

To generate descriptions of Turing machines with nice mathematical properties entirely by hand is a daunting task. Rather than approach the problem directly, we created tools for generating parsimonious Turing machines while presenting an interface that is comfortably familiar to most programmers (and to us!).

We created two tools. At the top level is the Laconic programming language, whose syntax and capabilities are similar to those of most programming languages, such as Java or Python. Beneath it we created a lower-level language called Turing Machine Descriptor (TMD). TMD is quite unlike most programming languages, and is better thought of as a convenient way to describe a multi-tape, 3-symbol Turing machine plus a function stack. The style of multi-tape Turing machine used in TMD is the commonly used “one-tape-at-a-time” abstraction: only one tape at a time can be interacted with, for reading, writing, and moving the head. Laconic compiles down to a TMD program, and TMD compiles down to a description of a single-tape, 2-symbol Turing machine. This process is illustrated in Figure 1.

We recommend that programmers hoping to use our tools to generate their own encodings

²Admittedly, it’s not obvious that using Friedman’s current statements *does* decrease the state count of the Turing machines. It’s possible that one could do as well or better by directly searching for contradictions in ZFC, and indeed, recent unpublished work by Stefan O’Rear has given some evidence for that [1]. On the other hand, Friedman’s statements can be translated into code without using the apparatus of first-order logic, which arguably gives us a conceptual simplification. In addition, statements like Friedman’s seem like the most plausible path forward for *further* reductions in the state count, beyond whatever lower limit one hits when one needs to encode the ZFC axioms explicitly.

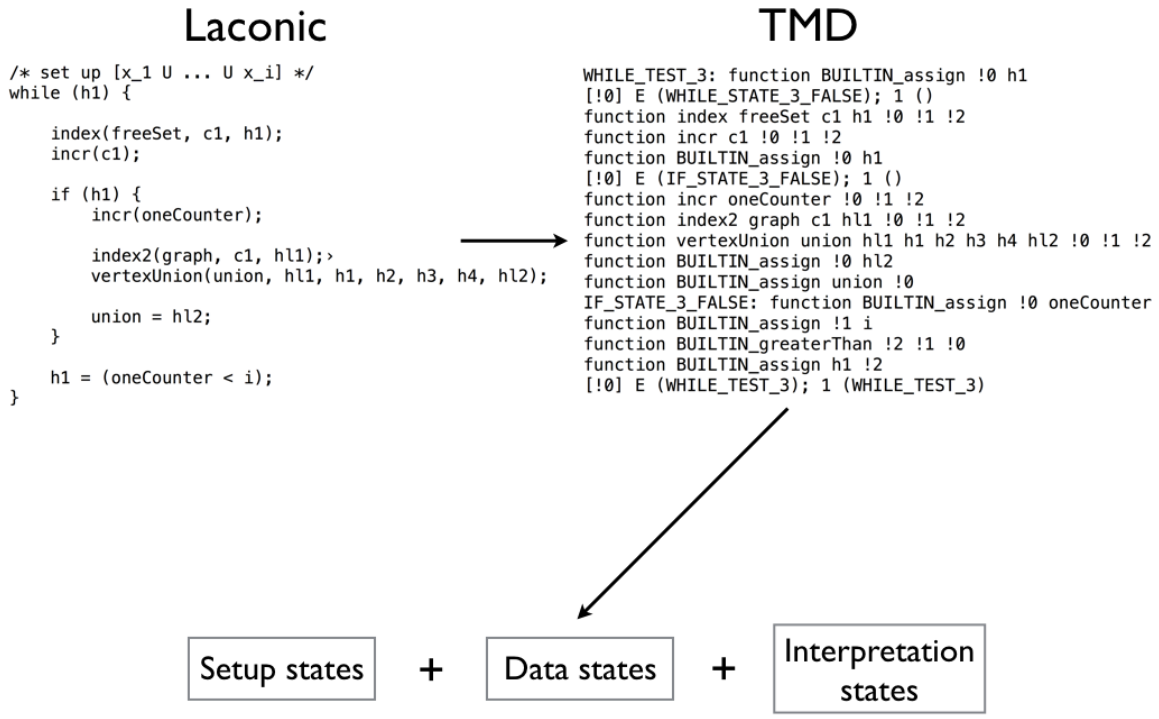


Figure 1: A visual overview of the compilation process.

of mathematical statements or algorithms as Turing machines use Laconic. Laconic’s interface is perfect for somebody hoping to write in a “traditional” language. On the other hand, if the programmer wishes to improve upon Laconic’s compilation process, writing code directly in TMD is likely to be the better option.

2 Related Work

Gregory Chaitin raised the problem of proving a version of our result in his book *The Limits of Mathematics*. [7] He wrote:

I would like to have somebody program out Zermelo-Fraenkel set theory in my version of LISP, which is pretty close to normal LISP as far as this task is concerned, just to see how many bits of complexity mathematicians normally assume . . . If you programmed ZF, you’d get a really sharp incompleteness result. It wouldn’t say that you can get at most $H(ZF) + 15328$ bits of [Chaitin’s halting probability] Ω , it would say, perhaps, at most 96000 bits! We’d have a much more definite incompleteness theorem.

We did not program ZF set theory in LISP, but we programmed it in an even simpler language—thereby answering Chaitin’s call for an explicit number of bits to attach to the complexity of ZF set theory. (As many as required to fully describe our Turing machine—or more precisely, 157,819.)

This paper is not the first to attempt to quantify the complexity of arithmetical statements. Calude and Calude [6] define a register machine of their own design, and provide quantifications of the complexity of Legendre’s Conjecture, Fermat’s Last Theorem, Goldbach’s Conjecture, Dyson’s Conjecture, the Riemann Hypothesis, and the Four Color Theorem.³ In addition, Koza [15] and Pargellis [21] each invent instruction sets that are particularly well-suited to representing self-reproducing programs simply, and show that starting from a “primordial soup” of such instructions distributed about a large memory, along with an increasing number of program threads, a rich ecosystem of increasingly efficient self-reproducing programs start to dominate the “landscape.”

This paper differs from the previous work in two ways: firstly, it’s the first to give explicit, relatively small machines whose behavior is provably independent of the standard axioms of modern mathematics. Secondly, to our knowledge, this paper is the first concrete study of parsimony to use Turing machines themselves as the model of computation—rather than (for example) a new programming language proposed by the authors, or a complex on-tape description of Turing machines! We consider it important to use the weakest and most common model of computation for complexity comparisons across different mathematical statements. This is because the more powerful and complex the model of computation used, the more of the complexity of the algorithm can be “shunted” onto the model of computation, and the greater the potential distortion created by the choice of model. As a *reductio ad absurdum*, we could imagine a programming language that included “test the Riemann Hypothesis” and “test the consistency of ZFC” as primitive operations. By using the “weakest” model of computation that’s commonly known, we hope to avoid this pitfall and make it easier to interpret our results in a model-independent way.

Also related to the work of this paper is the famous search for the smallest universal Turing machine. Here a *universal Turing machine* is a Turing machine that can simulate any other

³Because Fermat’s Last Theorem and the Four Color Theorem have been proved, their “complexity” is now known to be 1—the minimum number of states in a Turing machine that runs forever.

Turing machine, when a description of the latter is provided on its input tape. The smallest-known universal Turing machine has only 2 states and a 3-symbol alphabet, and was found by Alex Smith [24] in 2007. From the perspective of this paper, the problem is that the known small universal Turing machines achieve their small size only at the cost of an extremely complicated description format for the input machine. I.e., most of the complexity gets “shunted” from the Turing machine itself to the input encoding format. By contrast, with small Turing machines to test $Con(ZFC)$, the Riemann Hypothesis, Goldbach’s Conjecture, etc., and which run on an initially blank tape, there’s no analogous trick for hiding the statement’s complexity.

3 A Turing Machine that Cannot Be Shown to Run Forever Using ZFC

We present a 7,910-state Turing machine whose behavior is *independent of ZFC*; it is not possible to prove that this machine halts or doesn’t halt using the axioms of ZFC, assuming that a stronger set theory is consistent. It’s therefore impossible to prove the value of $BB(7,910)$ to be any given value without assuming axioms more powerful than ZFC, assuming that ZFC is consistent.

For an explicit listing of this machine, see Appendix C.

We call this machine Z . One way to build this machine would be to start with the axioms of ZFC and apply the inference rules of first-order logic repeatedly in each possible way so as to enumerate every statement ZFC could prove, and to halt if ever a contradiction was found. While this method is conceptually simple, to actually construct such a machine would lead to a huge number of states, because it would require writing a program to manipulate the axioms of ZFC and the inference rules of first-order logic, and then compiling that program all the way down to Turing machine states.

3.1 Friedman’s Mathematical Statement

Thankfully, a simpler method exists for creating Z . Friedman [10] was able to derive a graph-theoretic statement whose truth implies the consistency of ZFC, and which will be false if ZFC is inconsistent.⁴ Here is Friedman’s statement (the notation will be explained in the rest of this section):

Statement 1. *For all $k, n, r > 0$, every order invariant graph on $[\mathbb{Q}]^{\leq k}$ has a free $\{x_1, \dots, x_r, ush(x_1), \dots, ush(x_r)\}$ of complexity $\leq (8knr)!$, each $\{x_1, \dots, x_{(8kni)!}\}$, for $i > 0$ and $(8kni)! \leq r$, reducing $[x_1 \cup \dots \cup x_i \cup \{0, \dots, n\}]^{\leq k}$. [10]*

If s is a set, the operation $(.)^{\leq k}$ refers to the set of all subsets of s with size at most k .

A graph on $[\mathbb{Q}]^{\leq k}$ is an irreflexive symmetric relation on $[\mathbb{Q}]^{\leq k}$. In other words, it can be thought of as a graph whose vertices are elements of $[\mathbb{Q}]^{\leq k}$, and whose edges are undirected, connect pairs of vertices, and never connect vertices to themselves.

A *free* set is a set such that no pair of elements in that set are connected by an edge.

⁴In fact, Friedman’s statement is equivalent to the consistency of SRP (“stationary Ramsey property”), which is a system of axioms more powerful than ZFC. Because SRP is strictly more powerful than ZFC (it in fact consists of ZFC plus some additional axioms), the consistency of SRP implies the consistency of ZFC, and the inconsistency of ZFC implies the inconsistency of SRP.

A number of *complexity* at most c refers to a number that can be written as a fraction a/b , where a and b are both integers with absolute value less than or equal to c . A set has complexity at most c if all the numbers it contains have complexity at most c .

An *order invariant graph* is a graph containing a countably infinite number of nodes. In particular, it has one node for each finite set of rational numbers. The only numbers relevant to the statement are numbers of complexity $(8knr)!$ or smaller. In every description of nodes that follows, the term *node* refers both to the object in the order invariant graph and to the set of numbers that it represents.

In an order invariant graph, two nodes (a, b) have an edge between them if and only if each other pair of nodes (c, d) that is *order equivalent* with (a, b) has an edge between them. Two pairs of nodes (a, b) and (c, d) are *order equivalent* if a and c are the same size and b and d are the same size and if for all $1 \leq i \leq |a|$ and $1 \leq j \leq |b|$, the i -th element of a is less than the j -th element of b if and only if the i -th element of c is less than the j -th element of d .

To give some trivial examples of order invariant graphs: the graph with no edges is order invariant, as is the complete graph. A less trivial example is a graph on $[\mathbb{Q}]^{\leq 2}$, in which each node corresponds to a set of two rational numbers of a given complexity, and there is an edge between two nodes if and only if their corresponding sets a and b satisfy $|a| = |b| = 2$ and $a_1 < b_1 < a_2 < b_2$. (Because edges are undirected in order invariant graphs, such an edge will exist if *either* assignment of the vertices to a and b satisfies the inequality above.)

The *ush()* function takes as input a set and returns a copy of that set with all non-negative numbers in that set incremented by 1.

For vertices x and y , $x \leq_{rlex} y$ if and only if $x = y$ or $x_{|x|-i} < y_{|y|-i}$ where i is the least integer such that $x_{|x|-i} \neq y_{|y|-i}$.⁵ (The \leq_{rlex} operation creates a lexicographic ordering over the vertices, weighting the last and largest elements of those vertices most heavily. Like with lexicographic orderings, if the two vertices are identical but one is longer, the shorter one comes first.)

Finally, a set of vertices X *reduces* a set of vertices Y if and only if for all $y \in Y$, there exists $x \in X$ such that either $x = y$ or $x \leq_{rlex} y$ and an edge exists between x and y .

3.2 Implementation Methods

To create Z , we needed to design a Turing machine that halts if Statement 1 is false, and loops if Statement 1 is true. Such a Turing Machine's behavior is necessarily independent of ZFC, because the truth or falsehood of Statement 1 is independent of ZFC, assuming the consistency of SRP. [10] SRP is an extension of ZFC by certain relatively mild large cardinal hypotheses, and is widely regarded by set theorists as consistent. For more information about SRP, see [13].

To design such a Turing machine, we wrote a Laconic program which encodes Friedman's statement, then compiled the program down to a description of a single-tape, 2-symbol Turing machine. What follows is an extremely brief description of the design of the Laconic program; for the documented Laconic code itself, along with a detailed explanation of the full compilation process, see [25].

Our Laconic program begins by looping over all non-negative values for k , n , and r . For each trio (k, n, r) , our program generates a list N of all numbers of complexity at most $(8knr)!$. These numbers represent the vertices in our putative order invariant graph. Because Laconic

⁵Friedman recommended in private communication that we use the \leq_{rlex} comparator to compare vertices, instead of comparing their maximum elements as described in his manuscript.

does not support floating-point numbers, the list is entirely composed of integers; it is a list of all numbers that can be written in the form $((8knr!)!)^{\frac{i}{j}}$, where i and j are integers satisfying $-(8knr)! \leq i \leq (8knr)!$ and $1 \leq j \leq (8knr)!$. (Note that any number that can be expressed in this form is necessarily an integer, because of the large scaling factor in front.)

After we generate N , we generate the nodes in a potential order invariant graph by adding to N all possible lists of k or fewer numbers from N . We call this list of lists V .

We iterate over all binary lists of length $|V|^2$. Any such list E represents a possible set of edges in the graph. To be more precise, we say that an edge exists between node i and node j (represented by V_i and V_j respectively) if and only if $E_{i|V|+j}$ is 1.

For any graph (V, E) , we say that it is “valid” if the following three conditions hold:

1. No node has an edge to itself.
2. If an edge exists between node i and node j , an edge also exists between node j and node i .
3. The graph has a free $\{x_1, \dots, x_r, \text{ush}(x_1), \dots, \text{ush}(x_r)\}$, each $\{x_1, \dots, x_{(8kni)!}\}$ reducing $[x_1 \cup \dots \cup x_i \cup \{0, \dots, n\}]^{\leq k}$.

For each list of nodes V , we loop over every possible binary list E , and if no pair (V, E) yields a valid graph, we halt.

When verifying the validity of a graph, checking the first two conditions is trivial, but the third merits further explanation. In order to verify that a given graph (V, E) has a free $\{x_1, \dots, x_r, \text{ush}(x_1), \dots, \text{ush}(x_r)\}$, each $\{x_1, \dots, x_{(8kni)!}\}$ reducing $[x_1 \cup \dots \cup x_i \cup \{0, \dots, n\}]^{\leq k}$, we look at every possible subset of the nodes in V . For each subset, we verify that it has length r , that $\text{ush}(x_1), \dots, \text{ush}(x_r)$ all exist in V , and for each i such that $(8kni)! \leq r$, that $\{x_1, \dots, x_{(8kni)!}\}$ reduces $[x_1 \cup \dots \cup x_i \cup \{0, \dots, n\}]^{\leq k}$. Once we have found such a subset, we know that the third condition is satisfied.

4 A Turing Machine that Encodes Goldbach’s Conjecture

We present a 4,888-state Turing machine that *encodes Goldbach’s Conjecture*; in other words, to know whether this machine halts is to know whether Goldbach’s Conjecture is true. It’s therefore impossible to prove the value of $BB(4,888)$ without simultaneously proving or disproving Goldbach’s Conjecture.⁶

Recall that Goldbach’s Conjecture is as follows:

Statement 2. Every even integer greater than 2 can be expressed as the sum of two primes.

Because Goldbach’s Conjecture is so simple to state, the Laconic program encoding the statement is also quite simple. It can be found in Appendix A. A detailed explanation of the compilation process, documentation for the Laconic language, and an explicit description of this Turing machine are available at [25].

⁶Note that our tools were primarily meant to encode complex statements into Turing machines, such as Statement 1. Because Goldbach’s Conjecture is so simple, it’s feasible in that case to make dramatically smaller Turing machines through a more direct approach. Indeed, after a preprint of this paper was circulated online, “Jared S” and “code golf addict” created Turing machines for Goldbach’s Conjecture with 47 and 31 states respectively [1], though these machines have not yet been tested in detail.

5 A Turing Machine that Encodes the Riemann Hypothesis

We present a 5,372-state Turing machine that *encodes the Riemann Hypothesis*; in other words, to know whether this machine halts is to know whether the Riemann Hypothesis is true. An explicit description of this machine can be found at [25]

The Riemann Hypothesis is traditionally stated as follows:

Statement 3. The Riemann zeta function has its zeros only at the negative even integers and the complex numbers with real part $1/2$.

5.1 Equivalent Statement

Instead of encoding the Riemann zeta function into a Laconic program, it is simpler to use the following statement, which was shown by Lagarias [5] to be equivalent to the Riemann Hypothesis:

Statement 4. For all integers $n \geq 1$,

$$\left(\left(\sum_{k \leq \delta(n)} \frac{1}{k} \right) - \frac{n^2}{2} \right)^2 < 36n^3$$

The function $\delta(n)$ used in Statement 4 is defined as follows:

$$\begin{aligned} \eta(j) &= p \text{ if } j = p^k, p \text{ is prime, } k \text{ is a positive integer} \\ \eta(j) &= 1 \text{ otherwise} \\ \delta(x) &= \prod_{n < x} \prod_{j \leq n} \eta(j) \end{aligned}$$

5.2 Implementation Methods

Statement 4 is equivalent to the following statement, which involves only positive integers⁷:

$$l(n) < r(n)$$

for all positive integers n , where

$$\begin{aligned} l(n) &= a(n)^2 + b(n)^2 \\ r(n) &= 36n^3(\delta(n)!)^2 + 2a(n)b(n) \end{aligned}$$

$$\begin{aligned} a(n) &= \sum_{k \leq \delta(n)} \frac{\delta(n)!}{k} \\ b(n) &= \frac{n^2 \delta(n)!}{2} \end{aligned}$$

To check the Riemann Hypothesis, our program computes $a(n)$, $b(n)$, $l(n)$, and $r(n)$, in that order, for each possible value of n . If $l(n) \geq r(n)$, our program halts.

⁷Although it is not immediately obvious, $\frac{\delta(n)!}{k}$ is necessarily an integer for all $k \leq \delta(n)$, and $\frac{\delta(n)!}{2}$ is an integer for all $n > 1$.

6 Laconic

Laconic is a programming language designed to be both user-friendly and easy to compile down to parsimonious Turing machine descriptions.

Laconic is a strongly-typed language that supports recursive functions. Laconic compiles to an intermediate language called TMD. TMD programs are spread across multiple files and grouped into directories. TMD directories are meant to represent sequences of commands that could be given to a multi-tape, 3-symbol Turing machine, using the Turing machine abstraction that allows the machine to read and write from one head at a time.

For an example of a Laconic program, see Appendix A. For an illustration of the compilation process, see Figure 1.

7 TMD

TMD is a programming language designed to help the user describe the behavior of a multi-tape, 3-symbol Turing machine with a function stack. Each tape is infinite in one direction and supports three symbols: `_`, `1`, and `E`. The blank symbol is `_`: that is, `_` is the only symbol that can appear on the tape an infinite number of times. The tape must always have the form $_{?}(1|E)^+_{\infty}$; in other words, each tape must always contain a string of `1`'s and `E`'s of size at least 1, possibly preceded by a `_` symbol, and necessarily followed by an infinite number of copies of the `_` symbol.

What is the purpose of having a language like TMD as an intermediary between Laconic and a description of a single-tape machine? The concept of tapes in a multi-tape Turing machine and the concept of variables in standard imperative programming languages map to one another very nicely. The idea of the Laconic-to-TMD compiler is to encode the value of each variable on one tape. Then, each Laconic command that manipulates the value of one or more variables compiles down to a TMD function call that manipulates the tapes that correspond to those variables appropriately.

As an example, consider the following Laconic command:

```
a=b*c;
```

This Laconic command assigns the value of `b*c` to the variable `a`. It compiles down to the following TMD function call:

```
function BUILTIN_multiply a b c
```

This function call will result in `BUILTIN_multiply` being run on the three tapes `a`, `b`, and `c`. This will cause the symbols on tape `a` to take on a representation of an integer whose value is equal to `bc`.

In turn, the TMD code compiles directly to a string of bits that are written onto the tape at the start of the Turing machine's execution.

A TMD directory consists of three types of files:

1. The `functions` file. This file contains a list of the names of all the functions used by the TMD program. The top function in the file is pushed onto the stack at initialization. When this top function returns, the Turing machine halts.

2. The `initvar` file. This file contains the non-blank symbols that start in each register (or tape) at initialization.
3. Any files used to describe TMD functions. These files all end in a `.tfn` extension and only have any relevance to the compiled program if they show up in the functions file.

8 Compilation and Processing

There are two ways to think about the layout of the tape symbols: with a 4-symbol alphabet (`{_, 1, H, E}`, blank symbol `_`), and with a 2-symbol alphabet (`{a, b}`, blank symbol `a`). The 2-symbol alphabet version is the one that's ultimately used for the results in this paper, since we advertised a Turing machine that used only two symbols. However, in nearly all parts of the Turing machine, the 2-symbol version of the machine is a direct translation of the 4-symbol version, according to the following mapping:

```

_ ↔ aa
1 ↔ ab
H ↔ ba
E ↔ bb

```

The sections that follow sometimes refer to the `ERROR` state. Transitions to the `ERROR` state should never be taken under any circumstances, and are useful for debugging purposes.

8.1 Concept

A directory of TMD functions is converted at compilation time to a string of bits to be written onto the tape, along with other states designed to interpret these bits. The resulting Turing machine has three main components, or *submachines*:

1. The *initializer* sets up the basic structure of the variable registers and the function stack.
2. The *printer* writes down the binary string that corresponds to the compiled TMD code.
3. The *processor* interprets the compiled binary, modifying the variable registers and the function stack as necessary.

The Turing machine's control flow proceeds from the initializer to the the printer to the interpreter. In other words, initializer states point only to initializer states or to printer states, printer states point only to printer states or to interpreter states, and interpreter states point only to interpreter states or the `HALT` state.

This division of labor, while seemingly straightforward, actually constitutes an important idea. The problem of the compiler is to convert a higher-level representation—a machine with many tapes, a larger alphabet, and a function stack—to the lower-level representation of a machine with a single tape, a 2-symbol alphabet and no function stack. The immediately obvious solution, and the one taught in every computability theory class as a proof of the equivalence of different kinds of Turing machines, is to have every “state” in the higher-level machine compile down to many states in the lower-level machine.

While simple, this approach is suboptimal in terms of the number of states. As is nearly always true when designing systems to be parsimonious, the clue that improvement is possible lies in the presence of repetition. Each state transition in the higher-level machine is converted to a group of lower-level states with the same basic structure. Why not instead explain how to perform this conversion exactly once, and then apply the conversion many times?

This idea is at the core of the division of labor described previously. We begin by writing a description of the higher-level machine onto the tape, and then “run” the higher-level machine by reading what is on the tape with a set of states that understands how to interpret the encoded higher-level machine. We refer to this idea as *on-tape processing*.

In this paper, we use TMD as the representation of the higher-level machine.⁸ The printer writes the TMD program onto the tape, and the processor executes it. As a result of using this scheme, we incur a constant *additive* overhead—we have to include the processor in our final Turing machine—but we avoid the constant *multiplicative* overhead required for the naïve scheme.

Is this additive overhead small enough to be worth it? We found that it is. Our implementation of the processor requires 3,860 states. (See Section 8.5 for a detailed breakdown of the state cost by submachine.) In contrast to this additive overhead of 3,860, the naïve approach incurs a large multiplicative overhead that depends in part on how many states must be used to represent each higher-level state transition, and in part on how efficient an encoding scheme can be devised for the on-tape approach. The following table compares the performance of on-tape processing to the performance of an implementation that used the naïve approach. The comparison is shown for three kinds of machines: a machine that halts if and only if Goldbach’s Conjecture is false, a machine that halts if and only if the Riemann Hypothesis is false, and a machine whose behavior is independent of ZFC.

Program	States (Naïve)	States (On-Tape Processing)
Goldbach	7,902	4,888
Riemann	36,146	5,372
ZFC	340,943	7,910

As can be seen from this table, on-tape interpretation results in huge gains, particularly in large and complex programs.

The subsections that follow describe each of the three submachines—the initializer, the printer, and the processor—in greater detail.

8.2 The Initializer

The initializer starts by writing a counter onto the tape which encodes how many registers there will be in the program. Using the value in that counter, it creates each register, with demarcation patterns between registers, and unique identifiers for each register. Each register’s value begins with the pattern of non-blank symbols laid out in the `initvar` file. The initializer also creates the program counter, which starts at 0, and the function stack, which starts out with only a single function call to the top function in the `functions` file.

⁸Note that instead of TMD, the on-tape processing scheme could be used for any language, assuming the designer provides both a processor and an encoding for that language. We chose TMD because it made the interpreter easy to write, but other minimalist languages, like Unlambda [17], Brainf*ck [20], or Iota and Jot [2], might be good candidates for parsimonious designs, with the additional advantage of being already known to some programmers! Thanks to Luke Schaeffer for this point.

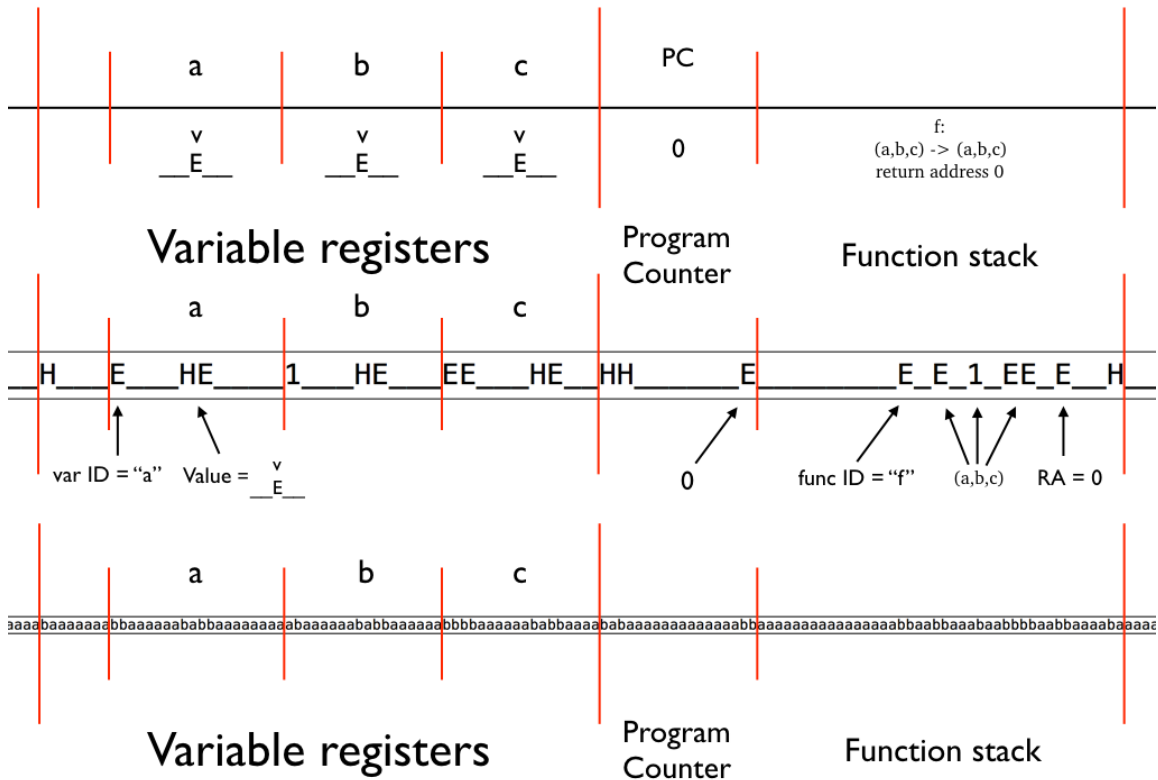


Figure 2: The state of the Turing machine tape after the initializer completes. The TMD program being expressed in Turing machine form is described in full in Appendix B. The top bar is a high-level description of what each part of the Turing machine tape represents. The middle bar is an encoding of the tape in the standard 4-symbol alphabet; the bottom bar is simply the translation of that tape into the 2-symbol alphabet. For a more detailed explanation of how to interpret the tape patterns, see [25].

Figure 2 is a detailed diagram describing the tape’s state when the initializer passes control to the printer.

8.3 The Printer

8.3.1 Specification

The printer writes down a long binary string which encodes the entirety of the TMD program onto the tape.

Figure 3 shows the tape’s state when the printer passes control to the processor.

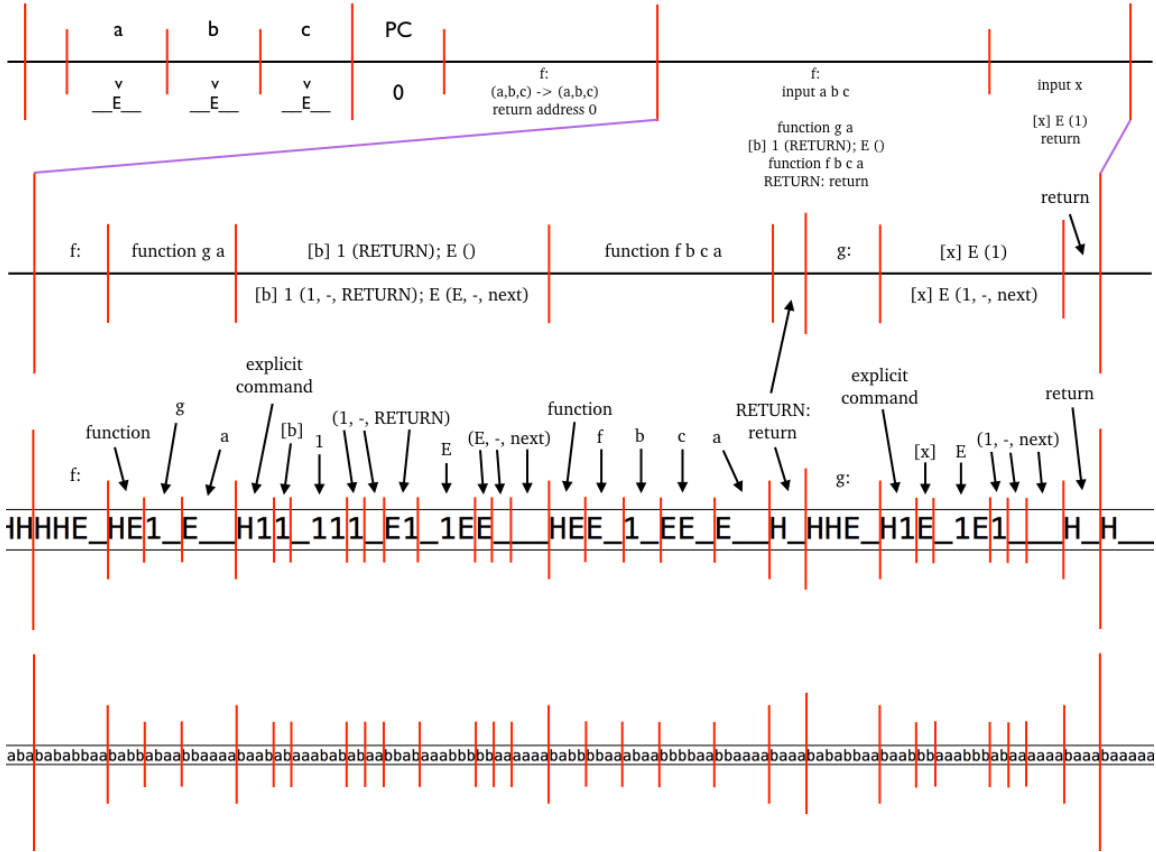


Figure 3: The state of the Turing machine tape after the printer completes. The TMD program being expressed in Turing machine form is described in full in Appendix B. The top bar is a high-level description of the entire tape; unfortunately, at this point there are so many symbols on the tape that it is impossible to see everything at once. For a detailed view of the first two-thirds of the tape (registers, program counter, and stack), see Figure 2. The bottom three bars show a zoomed-in view of the program binary. From the top, the second bar gives a high-level description of what each part of the program binary means; the third bar gives the direct correspondence between 4-symbol alphabet symbols on the tape and their meaning in TMD; the fourth and final bar gives the translation of the third bar into the 2-symbol alphabet. For a more detailed explanation of the encoding of TMD into tape symbols, see [25].

8.3.2 Introspection

Writing down a long binary string onto a Turing machine tape in a parsimonious fashion is not as straightforward as it might initially appear. The first idea that comes to mind is simply to use one state per symbol, with each state pointing to the next, as shown in Figure 4.

On closer examination, however, this approach is quite wasteful for all but the smallest binary files. Every **a** transition points to the next state in the sequence, and none of the **b** transitions are used at all! Indeed, the only information-bearing part of the state is the single bit contained in the choice of which symbol to write. But in theory, far more information than that could be encoded in each state. In a machine with n states, each state could contain $2(\log_2(n) + 1)$ bits of information, because each of its two transitions could point to any of the n states, and write either an **a** or a **b** onto the tape. Of course, this is only in theory; in practice, to extract the information contained in therefore Turing machine’s states and translate it into bits on the tape is nontrivial.

We will use a scheme originally conceived by Ben-Amram and Petersen [3] and refined further and suggested to us by Luke Schaeffer. It does not achieve the optimal theoretical encoding described above, but is relatively simple to implement and understand, and is within a factor of 2 of optimal for large binary strings. Schaeffer named Turing machines that use this idea *introspective*.

Introspection works as follows. If the binary string contains k bits, then let w be the *word size*. The word size w takes the largest value it can such that $w2^w \leq k$. We can split the binary string into $n_w = \lceil \frac{k}{w} \rceil$ words of w bits each (we can pad the last word with the blank symbol). In our scheme, each word in the bit-string is represented by a *data state*. Each data state points to the state representing the next word in the sequence for its **a** transition, but which state the **b** transition points to encodes the next word. Every **b** transition points to one of the last 2^w data states, thereby encoding w bits of information.

Of course, the encoding is useless until we specify how to extract the encoded bit-string from the data states. The extraction scheme works as follows. To query the i^{th} data state for the bits it encodes, we run the data states on the string $\mathbf{a}^{i-1}\mathbf{b}\mathbf{a}^\infty$ (a string of $i - 1$ **a**’s followed by a **b** in the i^{th} position). After running the data states on that string, what remains on the tape is the string $\mathbf{b}^{i-1}\mathbf{a}\mathbf{b}^r\mathbf{a}^\infty$, assuming that the i^{th} data state pointed to the r^{th} -to-last data state. Thus, what we’re left with is essentially a unary encoding of the “value” of the word in binary. Thus, the job of the extractor is to set up a binary counter which removes one **b** at a time and increments the counter appropriately. Then, afterward, the extractor reverts the tape back to the form $\mathbf{a}^i\mathbf{b}\mathbf{a}^\infty$, shifts all symbols on the tape over by w bits, and repeats the process. Finally, when the state beyond the last data state sees a **b** on the tape, we know that the process has completed, and we can pass control to the processor. Figure 5 shows the whole procedure.

How much have we gained by using introspection for encoding the program binary, instead of the naïve approach? It depends on how large the program binary is. Using introspection incurs an $O(\log k)$ *additive* overhead, because we have to include the extractor in our machine. (Our implementation of the extractor takes $10w + 17$ states. It’s possible to build a constant-size extractor, but it’s not worth it for our value of w) In return, we save a *multiplicative* factor of w (which scales with $\log k$) on the number of data states needed.

This is plainly not worth it for the 10-bit example binary shown in Figs. 4 and 5. For that binary, we require 69 additional states for the extractor in order to save 5 data states. For real programs, however, it is worth it, as can be seen from the following table.

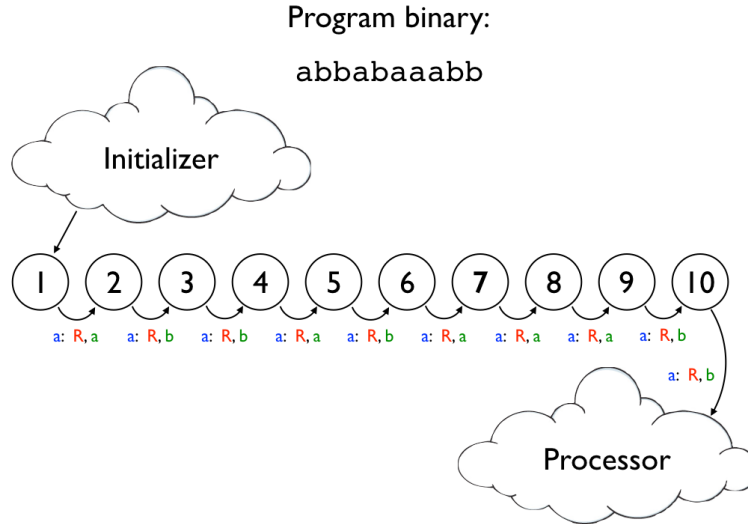


Figure 4: A naïve implementation of the printer. In this example, the hypothetical program is ten bits long, and the printer uses ten states, one for each bit. In the diagram, the blue symbol is the symbol that is read on a transition, the red letter indicates the direction the head moves, and the green symbol indicates the symbol that it written. Note the lack of transitions on reading a **b**; this is because in this implementation, the printer will only ever read the blank symbol, which is **a**, since the head is always proceeding to untouched parts of the tape. It therefore makes no difference what behavior the Turing machine adopts upon reading a **b** in states 1-10 (and therefore **b** transitions are presumed to lead to the ERROR state)

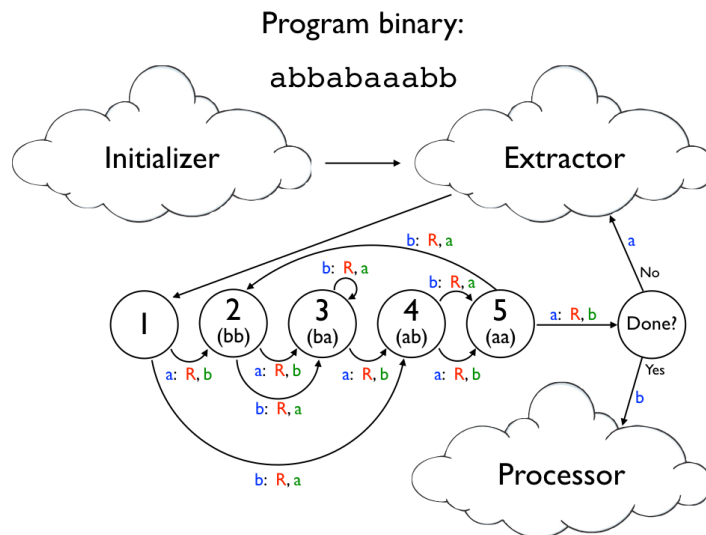


Figure 5: An introspective implementation of the printer. In this example, the hypothetical program is $k = 10$ bits long, and so the word size must be 2 (since $w = 2$ is the largest w such that $w2^w \leq 10$). There are therefore $n_w = \lceil \frac{k}{w} \rceil = 5$ data states, each encoding two bits. The **b** transitions carry the information about the encoding; note that each one only points to one of the last four data states. The last four data states have in parentheses what word we mean to encode if we point to them.

Program	Binary Size	w	n_w	Extractor Size	States (Naïve)	States (Introspective)
Example TMD	116	4	29	57	116	86
Goldbach	4,964	9	552	107	4,964	659
Riemann	9,532	10	1,024	117	9,532	1,141
ZFC	38,864	11	3,534	127	38,864	3,661

One minor detail concerns the numbers presented for the Riemann program. Ordinarily, with a binary of size 9,532, we would opt to split the program into 1,060 words of 9 bits each plus a 107-state extractor, since 9 is the greatest w such that $w2^w < 9,532$. But because 9,532 is so close to the “magic number” 10,240, it’s actually more parsimonious to pad the program with copies of the blank symbol until it’s 10,240 bits long, and split it into 1,024 words of 10 bits each plus a 117-state extractor.

8.4 The Processor

The processor’s job is to interpret the code written onto the tape and modify the variable registers and function stack accordingly. The processor does this by the following sequence of steps:

START:

1. Find the function call at the top of the stack. Mark the function f in the code whose ID matches that of the top function call.
2. Read the current program counter. Mark the line of code l in f whose line number matches the program counter.
3. Read l . Depending on what type of command l is, carry out one of the following three lists of tasks.

IF l IS AN EXPLICIT TAPE COMMAND:

1. Read the variable name off l . Index the variable name into the list of variables in the top function on the stack. This list of variables corresponds to the mapping between the function’s local variables and the register names.
2. Match the indexed variable to its corresponding register r . Mark r . Read the symbol s_r to the right of the head marker in that register.
3. Travel back to l , remembering the value of s_r using states. Find and mark the reaction x corresponding to the symbol. See what symbol s_w should be written in response to reading s_r .
4. Travel back to r , remembering the value of s_w using states. Replace s_r with s_w .
5. Travel back to x . See which direction d the head should move in response to reading s_r .
6. Travel back to r , remembering the value of d using states. Move the head marker accordingly.

7. Travel back to x . See if a jump is specified. If a jump is specified, copy the jump address onto the program counter. Otherwise, increment the program counter by 1.
8. Go back to START.

IF l IS A FUNCTION CALL:

1. Write the function's name to the top of the stack.
2. For each variable in the function call, index the variable name into the list of variables in the top function on the stack. This list of variables corresponds to the mapping between the function's local variables and the register names. Push the corresponding register names in the order that they correspond to the variables in the function call.
3. Copy the current program counter to the return address of the newborn function call at the top of the stack.
4. Replace the current program counter with 0 (meaning "read the first line of code").
5. Go back to START.

IF l IS A RETURN STATEMENT:

1. Replace the current program counter with f 's return address.
2. Increment the program counter by 1.
3. Erase the call to f from the top of the stack.
4. Check if the stack is now empty. If so, halt.
5. Go back to START.

8.5 Cost Analysis

It's worthwhile to analyze the relative contributions of the initializer, the printer, and the processor to the machine's final state count. The following table lists the number of states in each submachine for each of the four different TMD programs under discussion.

Program	Initializer	Printer	Processor	Total
Example TMD	349	86	3,860	4,295
Goldbach	369	659	3,860	4,888
Riemann	371	1,141	3,860	5,372
ZFC	389	3,661	3,860	7,910

As can be seen from this table, the processor makes the largest contribution to all four programs. Improving the processor, therefore, is probably the best approach for improving upon the bounds we present. Equally clear, however, is that for programs more complicated than the ones presented here, the cost of the printer will grow almost linearly but the cost of the processor will stay the same. The cost of the initializer grows very slightly with the complexity of programs because of the need to initialize additional registers.

Improving the printer, and with it the TMD and Laconic languages, is probably the best approach for reducing state count for very large and complex programs.

9 Future Work

This paper still leaves a three orders-of-magnitude gap between the smallest n , namely 7,910, for which $BB(n)$ is known to be independent of ZF set theory, and the largest n , namely 4, for which $BB(n)$ is known to be determinable. We regard it as a fascinating problem to pin down the truth here: for example, is it conceivable that $BB(10)$ or even $BB(6)$ might be independent of ZF? If so, that would arguably force a qualitative change in our understanding of the Gödel incompleteness phenomenon—showing that incompleteness from strong set theories rears its head for much simpler arithmetical questions than had previously been known.

A more immediate question is how much further Z 's state count can be reduced. We are optimistic about the possibility of further reductions. For example, one could adapt the processor-printer model to use a better language than TMD. Ideally, one wants a language whose processor contains fewer states than TMD's, and whose typical programs are *also* shorter than TMD programs. A few ideas have been proposed for this [1], many of them related in some way to lambda calculus.

Other future work might involve further use of our Laconic language to upper-bound the 'complexities' of mathematical statements and algorithms, in as standardized and model-independent a way as possible. Perhaps Laconic could be used to measure the complexity of other well-known conjectures, or even to compare different algorithms for solving the same problem to each other (e.g., to try to quantify the notion that an insertion sort is simpler than a merge sort)!

10 Acknowledgements

We thank Prof. Harvey Friedman for having done the crucial theoretical work that made this project feasible. Prof. Friedman was endlessly available over email, and provided us with detailed clarifications when we needed them.

We thank Luke Schaeffer for his early help, as well as his help designing introspective Turing machines.

We thank Alex Arkhipov for introducing us to the term "code golfing."

We thank the commenters on Scott Aaronson's blog [1] for their ideas and suggestions.

Supported by an Alan T. Waterman Award from the National Science Foundation, under grant no. 1249349.

References

- [1] Aaronson, S. "The 8000th Busy Beaver number eludes ZF set theory: new paper by Adam Yedidia and me." May 3, 2016. <http://www.scottaaronson.com/blog/?p=2725#comments> [Scott Aaronson publicized a preprint of our results on his blog, and many of his readers offered helpful comments and suggestions for future improvements.]
- [2] Barker, C. "Iota and Jot: the Simplest Languages?" <http://semarch.linguistics.fas.nyu.edu/barler/Iota/> [A website describing the Iota and Jot programming languages]
- [3] Ben-Amram, A., Petersen, H. "Improved Bounds for Functions Related to Busy Beavers" *Theory of Computing Systems* 35, 1-11 (2002)

- [4] Brady, A.H. "Solution of the Non-computable 'Busy Beaver' game for $k = 4$." Abstracts for: ACM Computer Science Conference (Washington, DC, February 18-20, 1975), p. 27, ACM, 1975.
- [5] Browder, F. "Mathematical Developments Arising from Hilbert Problems." American Mathematical Society. Volume 28, Part 1.
- [6] Calude, C., Calude, E. "Evaluating the Complexity of Mathematical Problems: Part 1," "Evaluating the Complexity of Mathematical Problems: Part 2." Complex Systems 18, pp. 387-401. 2010.
- [7] Chaitin, G. "The Limits of Mathematics." pp. 79. 1994.
- [8] Cloudy176, Wythagoras. "A good bound for S(7)?" 2014. http://googology.wikia.com/wiki/User_blog:Wythagoras/A_good_bound_for_S%287%29%3F
- [9] Deedlit11, Wythagoras. "Okay, more Turing machines." 2013. http://googology.wikia.com/wiki/User_blog:Deedlit11/Okay,_more_Turing_machines
- [10] Friedman, H. "Order Invariant Graphs and Finite Incompleteness." <https://u.osu.edu/friedman.8/files/2014/01/FIniteSeqInc062214a-v9w7q4.pdf>
- [11] Personal communications with H. Friedman.
- [12] Friedman, H. "Order Theoretic Equations, Maximality, and Incompleteness." June 7, 2014. <http://u.osu.edu/friedman.8/foundational-adventures/downloadable-manuscripts#78>.
- [13] Friedman H. "The Upper Shift Kernel Theorems." October 9, 2010. <https://u.osu.edu/friedman.8/files/2014/01/KernStruThm100910-11u0b8v.pdf>
- [14] Gödel, K. "The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory." Published in 1940 by the Princeton University Press. Annals of Mathematics Studies.
- [15] Koza, J. "Spontaneous Emergence of Self-Replicating and Evolutionarily Self-Improving Computer Programs." in Artificial Life III (SFI Studies in the Sciences of Complexity, vol. XVII), C. G. Langton, Ed. Reading, MA: Addison-Wesley. pp. 225-262. 1994.
- [16] Lin, S., Rado, T. "Computer Studies of Turing Machine Problems." Published in Journal of the ACM, Volume 12, Issue 2, April 1965. Pages 196-212.
- [17] Madore, D. "The Unlambda Programming Language." <http://www.madore.org/~david/programs/unlambda/>
[A website describing the Unlambda programming language]
- [18] Marxen, H., Buntrock, J. "Attacking the Busy Beaver 5." Bull EATCS, Vol. 40, pp. 247-251. 1990.
- [19] Marxen, H. <http://www.drb.insel.de/~heiner/BB/>
[A list of the known busy beaver values]

- [20] Müller, U. “Brainfuck.” <http://www.muppetlabs.com/~breadbox/bf/>
[A website describing the Brainf*ck programming language]
- [21] Pargellis, A. “The Spontaneous Generation of Digital ‘Life.’” *Physica D*, 91, 86-96. 1996.
- [22] Rado, T. “On Non-Computable Functions.” *Bell System Technical Journal*, 41: 3. May 1962 pp 877-884.
- [23] Schoenfield, J. “The Problem of Predicativity.” *Essays on the foundations of mathematics*, Y. Bar-Hillel et al., eds., pp. 132-142. 1961.
- [24] Smith, A. “Universality of Wolfram’s 2, 3 Turing Machine.” Submitted for the Wolfram 2, 3 Turing Machine Research Prize. <http://www.wolframscience.com/prizes/tm23/TM23Proof.pdf>
- [25] Yedidia, A. <https://github.com/adamyedidia/parsimony>
[A link to a GitHub repository containing all programs and Turing machines related to this paper, with accompanying documentation.]
- [26] <http://codegolf.stackexchange.com/>
[A place where programmers go for recreational code golfing]

Appendices

A Example Laconic Program: Goldbach’s Conjecture

The following is an example Laconic program, which compiles down to the Turing machine *G* mentioned in Section 4 (which halts if and only Goldbach’s Conjecture is false).

```
func zero(x) {
    x = 0;
    return;
}

func one(x) {
    x = 1;
    return;
}

func incr(x) {
    x = x + 1;
    return;
}

/* Computes x modulo y */
func modulus(x, y, out) {
    out = x;

    while (out >= y) {
        out = out - y;
    }

    return;
}
```

```

func assignXtoYminusX(x, y) {
    x = y - x;
    return;
}

/* Figures out if x is prime, and puts the output in y */
/* Does not modify x, modifies y */
func isPrime(x, h, y) {
    if (x == 1) {
        zero(y);
        return;
    }

    y = 2;

    while (x > y) {
        modulus(x, y, h);

        if (h == 0) {
            zero(y);
            return;
        }
        incr(y);
    }

    return;
}

int evenNumber;
int primeCounter;
int isThisOnePrime;
int foundSum;
int h;

evenNumber = 2;
one(foundSum);

while (foundSum) {
    zero(foundSum);
    evenNumber = evenNumber + 2;
    one(primeCounter);

    while (primeCounter < evenNumber) {
        isPrime(primeCounter, h, isThisOnePrime);

        if (isThisOnePrime) {
            assignXtoYminusX(primeCounter, evenNumber);
            isPrime(primeCounter, h, isThisOnePrime);
            assignXtoYminusX(primeCounter, evenNumber);

            if (isThisOnePrime) {
                print evenNumber;
                print primeCounter;

                one(foundSum);
            }
        }

        incr(primeCounter);
    }
}

halt;

```

For detailed documentation of the Laconic programming language, see [25]. To find this file

specifically, navigate to `parsimony/src/laconic/laconic_files/goldbach.lac` at [25].

B Example TMD Program

The following is an example TMD directory, which compiles down to a binary string to be written on a Turing machine tape. It's the example used in illustrations throughout this paper, most notably in the example compilation shown in Figs. 2 and 3. The program calls itself recursively three times until the starting symbol on each tape, `E`, is replaced with a `1`, at which point the program halts.

This TMD directory is called `example_tmd_dir`, and contains four files: `f.tmd`, `g.tmd`, `initvar`, and `functions`.

```
f.tmd:
input a b c

// Recursively writes a 1 on every tape.

function g a
[b] 1 (RETURN); E ()
function f b c a
RETURN: return

g.tmd:
input x

// Writes a 1 on the input tape.

[x] E (1)
return

functions:
f
g

initvar:
E
```

For detailed documentation of the TMD programming language, see [25]. To find this directory specifically, navigate to `parsimony/src/tmd/tmd_dirs/example_tmd_dir/` at [25].

C Explicit Listing of Z

We present below an explicit listing of Z . For a more easily readable version of Z , complete with descriptive state names, see [25].

We ran this Turing machine for 10,000,000,000 steps (more than half a day on our simulators) and within that time it did not halt. We note, however, that Z was designed for parsimony rather than efficiency, and that this “experiment” is of little consequence! We similarly ran a Turing machine designed to test the conjecture that all perfect squares are less than 5, and it ran for 2,895,083,899 steps (a couple hours on our simulator) before it found the counterexample 9 and halted.

Figure 6 explains how to interpret the description shown below. In addition, note the following:

A description of a single state in Z

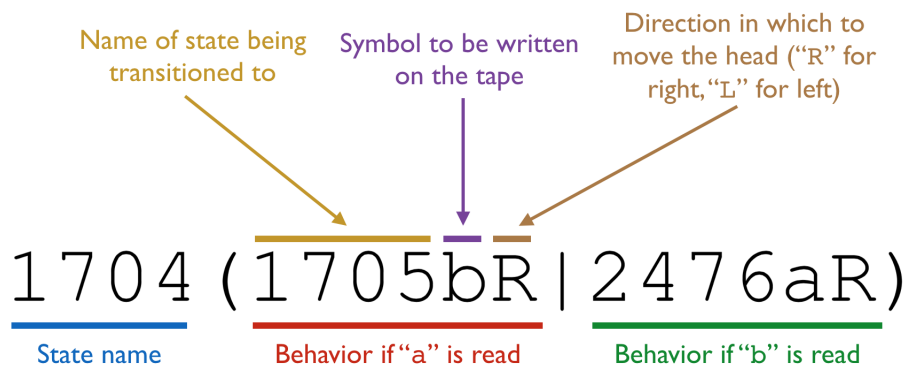


Figure 6: This figure explains how to read a description of a single state. Note that “ERROR–” or “HALT–” denote transitions to the ERROR or HALT states, respectively (no further information is provided because what symbol is written and which direction the head moves are at that point irrelevant).

1. The tape has a 2-symbol alphabet, with tape symbols {a, b} and blank symbol a (in other words, a is the only symbol that can appear an infinite times on the tape).
2. The start state of Z is 0000.
3. Z will never transition to the ERROR state. Any transition to the ERROR state could be replaced by a transition to any other state (including HALT) and the Turing machine’s behavior would remain identical.
4. Z consists only one transition to the HALT state, out of state 7862.

0000(0001b ERROR-)	0001(0004b ERROR-)	0002(0003b ERROR-)	0003(0012aR 0012bR)	0004(0005aR ERROR-)	0005(0006bR ERROR-)	0006(0007bR ERROR-)	0007(0008bR ERROR-)	0008(0009bR ERROR-)	0009(0010bR ERROR-)
0010(0011bR ERROR-)	0011(0002bR ERROR-)	0012(0013aR ERROR-)	0013(0014aR 0014bR)	0014(0015aR ERROR-)	0015(0007aR 0007bR)	0016(0017bR ERROR-)	0017(0018bR ERROR-)	0018(0019aR ERROR-)	0019(0020aR 0020bR)
0020(0021aR 0021bR)	0021(0022aR 0022bR)	0022(0023aR 0023bR)	0023(0024aR 0024bR)	0024(0025aR ERROR-)	0025(0007aR 0007bR)	0026(0027aR 0027bR)	0027(0028aR 0030bL)	0028(0029aL 0029bL)	0029(0026aL 0026bL)
0030(0031aL 0031bL)	0031(0026aL 0026bL)	0032(0033aL 0033bL)	0033(0034aL 0034bL)	0034(0037aL 0037bL)	0035(0036aL 0036bL)	0036(0025aL 0025bL)	0037(0038aR 0041bR)	0038(0044aR 0039bL)	0039(0040bR 0040bR)
0040(0049aL 0049bL)	0041(ERROR- 0042bL)	0042(0043aL 0043aL)	0043(0037aL 0037bL)	0044(0045aR 0045bR)	0045(ERROR- 0046aL)	0046(0047aR 0047aR)	0047(0049aL 0049bL)	0048(0071aR ERROR-)	0049(0050aR 0051bR)
0050(0049aR 0049bR)	0051(0052aR 0049bR)	0052(0053aR 0054bR)	0053(0052aR 0052bR)	0054(0055aL 0052bR)	0055(0056aR 0056aR)	0056(0012aR 0012bR)	0057(0058aR ERROR-)	0058(0059aR 0059bR)	0059(0060aR 0059bR)
0060(0061aR 0061bR)	0061(0062aR ERROR-)	0062(0063aR 0063bR)	0063(0064aR ERROR-)	0064(0065aR 0065bR)	0065(0066aR 0066bR)	0066(0015aR 0016bR)	0067(0068bR ERROR-)	0068(0069bR ERROR-)	0069(0070bL ERROR-)
0070(0026aL 0026bL)	0071(0072aR 0075bR)	0072(0071aR 0073bL)	0073(0074aR 0074bR)	0074(0075aR 0075bL)	0075(ERROR- 0076bL)	0076(0077aR 0077bR)	0077(0078aL 0078bL)	0078(0079aR 0082bR)	0079(0080aL 0078bR)
0080(0081bR 0081bR)	0081(0083aR 0083bR)	0082(0078aR 0078bR)	0083(0084aR 0088bR)	0084(0085aR 0083bR)	0085(0086aL 0083bR)	0086(0087aL 0087bL)	0087(0088aL 0088bL)	0088(0089aR 0094bR)	0089(0090aL 0092bL)
0090(0091aR 0091bR)	0091(0093aL 0093bL)	0092(0093aL 0093bL)	0093(0093aR 0098bL)	0094(0095aL 0097bL)	0095(0096aL 0096bL)	0096(0088aL 0088bL)	0097(0098aL 0098bL)	0098(0088aL 0088bL)	0099(0100aR 0105bR)
0100(0101aL 0103bL)	0101(0102aL 0102bL)	0102(0099aL 0099bL)	0103(0104aR 0104bR)	0104(0105aL 0110bL)	0105(0106aL 0108bL)	0106(0107aR 0107aR)	0107(0139aL 0139bL)	0108(0109aR 0109bR)	0109(0110aL 0110bL)
0110(0111aR 0116bR)	0111(0112bL 0114bL)	0112(0113bL 0113bL)	0113(0130aL 0130bL)	0114(0115bL 0115bL)	0115(0110aL 0110bL)	0116(ERROR- 0137bL)	0117(0118aR 0118aR)	0118(0119aL 0119bL)	0119(0120aR 0125bR)
0120(0121aL 0123bL)	0121(0122aR 0122bR)	0122(0130aL 0130bL)	0123(0124aR 0124bL)	0124(0111aR 0119bL)	0125(0126aL 0126bL)	0126(0127aL 0127bL)	0127(0119aL 0119bL)	0128(0129aL 0129bL)	0129(0119aL 0119bL)
0130(0131aR 0136bR)	0131(0132aL 0134bL)	0132(0133aL 0133bL)	0133(0130aL 0130bL)	0134(0135aR 0135bR)	0135(0088aL 0088bL)	0136(ERROR- 0137bL)	0137(0138aR 0138bR)	0138(0088aL 0088bL)	0139(0140aR 0143bR)
0140(0139aR 0141bL)	0141(0142aR 0142bR)	0142(0146aL 0146bL)	0143(ERROR- 0144bL)	0144(0145aR 0145bR)	0145(0146aL 0146bL)	0146(0147aR 0150bR)	0147(0148aL 0148bR)	0148(0149aR 0149bR)	0149(0071aL 0071bL)
0160(0157aL 0157bL)	0161(0162aR 0162bR)	0162(0166aL 0166bL)	0163(0197aR 0164bL)	0164(0155aR 0155bR)	0165(0156bL ERROR-)	0166(0167aR 0172bR)	0167(0168aL 0170bL)	0168(0169bL 0169bL)	0169(0170aL 0177bL)
0170(0171aL 0171bL)	0171(0166aL 0166bL)	0172(0173aL 0175bL)	0173(0174aL 0174bL)	0174(0166aL 0166bL)	0175(0176aL 0176bL)	0176(0166aL 0166bL)	0177(0178aR 0183bR)	0178(0179aL 0181bL)	0179(0180aL 0180bL)
0180(0177aL 0177bL)	0181(0182aR 0182bR)	0182(0186aL 0186bL)	0183(ERROR- 0184bL)	0184(0185aR 0185bR)	0185(0186aL 0186bL)	0186(0187aR 0192bR)	0187(0188aL 0190bL)	0188(0189aR 0189bR)	0189(0157aL 0157bL)
0190(0191aL 0191bL)	0191(0186aL 0186bL)	0192(0193aL 0195bL)	0193(0194aL 0194bL)	0194(0186aL 0186bL)	0195(0196aL 0196bL)	0196(0186aL 0186bL)	0197(0198aR 0199bR)	0198(0197aR 0197bR)	0199(0200aR 0197bR)
0200(0201aR 0204bR)	0201(0202aL 0197bR)	0202(0203bR 0203bR)	0203(0205aR 0205bR)	0204(0200aR 0197bR)	0205(0206aR ERROR-)	0206(0207aR 0207bR)	0207(0208aR ERROR-)	0208(0209aR 0209bR)	0209(0210aR ERROR-)
0210(0211aR 0211bR)	0211(0212bR ERROR-)	0212(0213bR ERROR-)	0213(0214aR ERROR-)	0214(0223aR 0233bR)	0215(0216bR ERROR-)	0216(0217bR ERROR-)	0217(0218bL ERROR-)	0218(0263aL 0263bL)	0219(0220aR ERROR-)
0220(0221aR 0221bR)	0221(0222aR ERROR-)	0222(0223aR 0223bR)	0223(0224aR ERROR-)	0224(0225aR 0225bR)	0225(0226aR ERROR-)	0226(0227aR 0227bR)	0227(0228aR ERROR-)	0228(0229aR 0229bR)	0229(0230aR ERROR-)
0230(0231aR 0231bR)	0231(0232aR ERROR-)	0232(0211aR 0211bR)	0233(0234bR ERROR-)	0234(0237bR ERROR-)	0235(0236bR ERROR-)	0236(0245aR 0245bR)	0237(0238aR ERROR-)	0238(0239bR ERROR-)	0239(0240bR ERROR-)
0240(0241bR ERROR-)	0241(0242aR ERROR-)	0242(0243bR ERROR-)	0243(0244aR ERROR-)	0244(0238bR ERROR-)	0245(0246aR ERROR-)	0246(0247aR ERROR-)	0247(0248aR ERROR-)	0248(0249aR 0249bR)	0249(0250aR ERROR-)
0250(0251aR 0251bR)	0251(0252aR ERROR-)	0252(0253aR 0253bR)	0253(0254aR ERROR-)	0254(0255aR 0255bR)	0255(0256aR ERROR-)	0256(0257aR 0257bR)	0257(0258aR ERROR-)	0258(0259aR 0259bR)	0259(0260aR ERROR-)
0260(0261aR 0261bR)	0261(0262aR ERROR-)	0262(0215aL 0215bL)	0263(0264aR 0269bR)	0264(0265aL 0267bL)	0265(0266aL 0266bL)	0266(0263aL 0263bL)	0267(0268aL 0268bL)	0268(0263aL 0263bL)	0269(0270aL 0272bL)
0270(0271aL 0271bL)	0271(0272aL 0274bL)	0272(0273aL 0273bL)	0273(0265aL 0263bL)	0274(0275aR 0275bR)	0275(0291aL 0276bL)	0276(0277aR 0277bR)	0277(0288aL 0288bL)	0278(ERROR- 0278bR)	0279(0280aL 0280bL)
0280(0274aL 0274bL)	0281(0282aR 0285bR)	0282(ERROR- 0283aL)	0283(0284aR 0284aR)	0284(0288aL 0288bL)	0285(0286aL ERROR-)	0286(0287aR 0287aR)	0287(0300aR 0300bR)	0288(0289aR 0290bR)	0289(0288aR 0288bR)
0290(0291aR 0289bR)	0291(0292aR 0293bR)	0292(0291aR 0291bR)	0293(0294aL 0291bR)	0294(0295aR 0295aR)	0295(0296aR 0296bR)	0296(0297bR ERROR-)	0297(0298bR ERROR-)	0298(0299bL ERROR-)	0299(0263aL 0263bL)
0300(0301aR 0302bR)	0301(0301aR 0303bR)	0302(0301aR 0303bR)	0303(0304aR 0305bR)	0304(0306aR 0306bR)	0305(0306aR 0306bR)	0306(0307bR ERROR-)	0307(0308bR ERROR-)	0308(0310aR ERROR-)	0309(0309aR 0309bR)
0310(0311aL 0309bR)	0311(0312aL 0312bL)	0312(0313aL 0313bL)	0313(0314aR 0317bR)	0314(0338aR 0315bL)	0315(0316aL 0316bL)	0316(0313aL 0313bL)	0317(0338aR 0318bL)	0318(0319aR 0319aR)	0319(0360aL 0360bL)
0320(0321aR 0324bR)	0321(0320aR 0322aL)	0322(0323aR 0323aR)	0323(0329aR 0329bR)	0324(0325aL 0327aL)	0325(0326aR 0326aR)	0326(0347aR 0347bR)	0327(0328aR 0328aR)	0328(0338aR 0338bR)	0329(0330aR 0333bR)
0330(0331bL 0329bR)	0331(0332aR 0332aR)	0332(0332aR 0329bR)	0333(0334aL 0336bL)	0334(0335aR 0335aR)	0335(0347aR 0347bR)	0336(0337aR 0337aR)	0337(0338aR 0338bR)	0338(0339aR 0344bR)	0339(0340bL 0342bL)
0340(0341bR 0341bR)	0341(0320aR 0320bR)	0342(0343bR 0343bR)	0343(0329aR 0329bR)	0344(0345bL 0338bR)	0345(0346bR 0346bR)	0346(0347aR 0347bR)	0347(0348bL ERROR-)	0348(0349aL 0349bL)	0349(0350aR 0353bR)
0350(0351aL 0353bL)	0351(0352aL 0352bL)	0352(0349aL 0349bL)	0353(0354aR 0354bR)	0354(0313aL 0313bL)	0355(0356aL 0356bL)	0356(0357aR 0357aR)	0357(0371aR 0371bR)	0358(0359aL 0359bL)	0359(0349aL 0349bL)
0360(0361aR 0360bR)	0361(0362aL 0364bL)	0362(0363aL 0363bL)	0363(0311aL 0311bL)	0364(0365aL 0365bL)	0365(0366aL 0366bL)	0366(0367aR 0367aR)	0367(0368aR 0368bR)	0368(0371aR 0371bR)	0369(0370aL 0370bL)
0370(0360aL 0360bL)	0371(0372aR 0375bR)	0372(0373aL 0373bL)	0373(0374bR 0374bR)	0374(0380aR 0380bR)	0375(0376aL 0371bL)	0376(0377aR 0377aR)	0377(0378aR 0378bR)	0378(0379bR 0379bR)	0379(0380bR 0379bR)
0380(0381aR ERROR-)	0381(0382aR 0382bR)	0382(0383aR 0384bR)	0383(0384aR 0384bR)	0384(0385bR ERROR-)	0385(0389aR 0389bR)	0386(0387aR 0389bR)	0387(0386aR 0386bR)	0388(0389aR 0389bR)	0389(0390aR ERROR-)
0390(0391bL ERROR-)	0391(0392aR ERROR-)	0392(0393aL 0392aL)	0393(0393aR 0394bR)	0394(0405bR ERROR-)	0395(0396bR 2559aR)	0396(0397bR 3077aR)	0397(0398bR 2293aR)	0398(0399bR 2637aR)	0399(0410bR 3253aR)
0400(0401bR 2589aR)	0401(0402bR 3610aR)	0402(0403bR 2065aR)	0403(0404bR 2265aR)	0404(0405bR 1939aR)	0405(0406bR 3411aR)	0406(0407bR 2579aR)	0407(0408bR 2234aR)	0408(0409bR 2070aR)	0409(0410bR 2234aR)
0410(0411bR 2967aR)	0411(0412bR 3275aR)	0412(0413bR 3002aR)	0413(0414bR 3002aR)	0414(0415bR 3737aR)	0415(0416bR 1978aR)	0416(0417bR 2569aR)	0417(0418bR 3188aR)	0418(0419bR 2440aR)	0419(0420bR 2654aR)
0420(0421bR 2517aR)	0421(0422bR 2520aR)	0422(0423bR 2520aR)	0423(0424bR 3082aR)	0424(0425bR 2645aR)	0425(0426bR 3203aR)	0426(0427bR 2637aR)	0427(0428bR 3293aR)	0428(0429bR 2458aR)	0429(0430bR 2259aR)

0430 (0431br) 2587aR 0431 (0432br) 3002aR 0432 (0433br) 3737aR 0433 (0434br) 1978aR 0434 (0435br) 2569aR 0435 (0436br) 3185aR 0436 (0437br) 2440aR 0437 (0438br) 1265aR 0438 (0439br) 2617aR 0439 (0440br) 3753aR
0440 (0441br) 2820aR 0441 (0442br) 3838aR 0442 (0443br) 2651aR 0443 (0444br) 3293aR 0444 (0445br) 2637aR 0445 (0446br) 3293aR 0446 (0447br) 2858aR 0447 (0448br) 3283aR 0448 (0449br) 2820aR 0449 (0450br) 3224aR
0450 (0451br) 2012aR 0451 (0452br) 3183aR 0452 (0453br) 1892aR 0453 (0454br) 3929aR 0454 (0455br) 2637aR 0455 (0456br) 3088aR 0456 (0457br) 2038aR 0457 (0458br) 3678aR 0458 (0459br) 2402aR 0459 (0460br) 3782aR
0470 (0471br) 2441aR 0471 (0472br) 2250aR 0472 (0473br) 2651aR 0473 (0474br) 2269aR 0474 (0475br) 2887aR 0475 (0476br) 3002aR 0476 (0477br) 2828aR 0477 (0478br) 2186aR 0478 (0479br) 2788aR 0479 (0480br) 3728aR
0480 (0481br) 2651aR 0481 (0482br) 3434aR 0482 (0483br) 2865aR 0483 (0484br) 2208aR 0484 (0485br) 2837aR 0485 (0486br) 3404aR 0486 (0487br) 2865aR 0487 (0488br) 2186aR 0488 (0489br) 2458aR 0489 (0490br) 3272aR
0490 (0491br) 2954aR 0491 (0492br) 3434aR 0492 (0493br) 2865aR 0493 (0494br) 2208aR 0494 (0495br) 2837aR 0495 (0496br) 3404aR 0496 (0497br) 2865aR 0497 (0498br) 2186aR 0498 (0499br) 2458aR 0499 (0500br) 3728aR
0500 (0501br) 2789aR 0501 (0502br) 3006aR 0502 (0503br) 2657aR 0503 (0504br) 3294aR 0504 (0505br) 2073aR 0505 (0506br) 3223aR 0506 (0507br) 2507aR 0507 (0508br) 2226aR 0508 (0509br) 2859aR 0509 (0510br) 1978aR
0510 (0511br) 2441aR 0511 (0512br) 3434aR 0512 (0513br) 2865aR 0513 (0514br) 2208aR 0514 (0515br) 2837aR 0515 (0516br) 3404aR 0516 (0517br) 2865aR 0517 (0518br) 2186aR 0518 (0519br) 2458aR 0519 (0520br) 3728aR
0520 (0521br) 2954aR 0521 (0522br) 1969aR 0522 (0523br) 3394aR 0523 (0524br) 2933aR 0524 (0525br) 2073aR 0525 (0526br) 3223aR 0526 (0527br) 2507aR 0527 (0528br) 2226aR 0528 (0529br) 2859aR 0529 (0530br) 1978aR
0530 (0531br) 2449aR 0531 (0532br) 3394aR 0532 (0533br) 2659aR 0533 (0534br) 3249aR 0534 (0535br) 2437aR 0535 (0536br) 3208aR 0536 (0537br) 3207aR 0537 (0538br) 1979aR 0538 (0539br) 2659aR 0539 (0540br) 2910aR
0540 (0541br) 3598aR 0541 (0542br) 2388aR 0542 (0543br) 3437aR 0543 (0544br) 2858aR 0544 (0545br) 2659aR 0545 (0546br) 3207aR 0546 (0547br) 2858aR 0547 (0548br) 1979aR 0548 (0549br) 3193aR 0549 (0550br) 3474aR
0550 (0551br) 2657aR 0551 (0552br) 3394aR 0552 (0553br) 2659aR 0553 (0554br) 2933aR 0554 (0555br) 2073aR 0555 (0556br) 3208aR 0556 (0557br) 3033aR 0557 (0558br) 3297aR 0558 (0559br) 2659aR 0559 (0560br) 2920aR
0560 (0561br) 3429aR 0561 (0562br) 2911aR 0562 (0563br) 2665aR 0563 (0564br) 3190aR 0564 (0565br) 2698aR 0565 (0566br) 2282aR 0566 (0567br) 3077aR 0567 (0568br) 1979aR 0568 (0569br) 2708aR 0569 (0570br) 3992aR
0570 (0571br) 2908aR 0571 (0572br) 2536aR 0572 (0573br) 2823aR 0573 (0574br) 1889aR 0574 (0575br) 2807aR 0575 (0576br) 3208aR 0576 (0577br) 2054aR 0577 (0578br) 1966aR 0578 (0579br) 3434aR 0579 (0580br) 3588aR
0580 (0581br) 2939aR 0581 (0582br) 3757aR 0582 (0583br) 3769aR 0583 (0584br) 2997aR 0584 (0585br) 2063aR 0585 (0586br) 3278aR 0586 (0587br) 2893aR 0587 (0588br) 2921aR 0588 (0589br) 2648aR 0589 (0590br) 2962aR
0590 (0591br) 3463aR 0591 (0592br) 3001aR 0592 (0593br) 2625aR 0593 (0594br) 3289aR 0594 (0595br) 2858aR 0595 (0596br) 3278aR 0596 (0597br) 3089aR 0597 (0598br) 3487aR 0598 (0599br) 2659aR 0599 (0600br) 2270aR
0600 (0601br) 3143aR 0601 (0602br) 1193aR 0602 (0603br) 1931aR 0603 (0604br) 3306aR 0604 (0605br) 2127aR 0605 (0606br) 3306aR 0606 (0607br) 2158aR 0607 (0608br) 1931aR 0608 (0609br) 3434aR 0609 (0610br) 3454aR
0610 (0611br) 2659aR 0611 (0612br) 2270aR 0612 (0613br) 2593aR 0613 (0614br) 1981aR 0614 (0615br) 2071aR 0615 (0616br) 3234aR 0616 (0617br) 2068aR 0617 (0618br) 2659aR 0618 (0619br) 2659aR 0619 (0620br) 2269aR
0620 (0621br) 2639aR 0621 (0622br) 1962aR 0622 (0623br) 3429aR 0623 (0624br) 3248aR 0624 (0625br) 2894aR 0625 (0626br) 2882aR 0626 (0627br) 1916aR 0627 (0628br) 2226aR 0628 (0629br) 2440aR 0629 (0630br) 2397aR
0630 (0631br) 2619aR 0631 (0632br) 2464aR 0632 (0633br) 3595aR 0633 (0634br) 3489aR 0634 (0635br) 2437aR 0635 (0636br) 3208aR 0636 (0637br) 2859aR 0637 (0638br) 3002aR 0638 (0639br) 2529aR 0639 (0640br) 2244aR
0640 (0641br) 2065aR 0641 (0642br) 2259aR 0642 (0643br) 2619aR 0643 (0644br) 2269aR 0644 (0645br) 2839aR 0645 (0646br) 3429aR 0646 (0647br) 2158aR 0647 (0648br) 2226aR 0648 (0649br) 2540aR 0649 (0650br) 2935aR
0650 (0651br) 2913aR 0651 (0652br) 3374aR 0652 (0653br) 3463aR 0653 (0654br) 3359aR 0654 (0655br) 2443aR 0655 (0656br) 3677aR 0656 (0657br) 2939aR 0657 (0658br) 3437aR 0658 (0659br) 2659aR 0659 (0660br) 3394aR
0660 (0661br) 3164aR 0661 (0662br) 1902aR 0662 (0663br) 2939aR 0663 (0664br) 3437aR 0664 (0665br) 2858aR 0665 (0666br) 3394aR 0666 (0667br) 2020aR 0667 (0668br) 2920aR 0668 (0669br) 1939aR 0669 (0670br) 3282aR
0670 (0671br) 2857aR 0671 (0672br) 2265aR 0672 (0673br) 2587aR 0673 (0674br) 2234aR 0674 (0675br) 2441aR 0675 (0676br) 3255aR 0676 (0677br) 3425aR 0677 (0678br) 3354aR 0678 (0679br) 3678aR 0679 (0680br) 2567aR
0680 (0681br) 2857aR 0681 (0682br) 2344aR 0682 (0683br) 2657aR 0683 (0684br) 2944aR 0684 (0685br) 2073aR 0685 (0686br) 3208aR 0686 (0687br) 1915aR 0687 (0688br) 3600aR 0688 (0689br) 2115aR 0689 (0690br) 3730aR
0700 (0701br) 1893aR 0701 (0702br) 3854aR 0702 (0703br) 3580aR 0703 (0704br) 2906aR 0704 (0705br) 2906aR 0705 (0706br) 3028aR 0706 (0707br) 2780aR 0707 (0708br) 3374aR 0708 (0709br) 3069aR 0709 (0710br) 3023aR
0720 (0721br) 2966aR 0721 (0722br) 2151aR 0722 (0723br) 2629aR 0723 (0724br) 2370aR 0724 (0725br) 2699aR 0725 (0726br) 3859aR 0726 (0727br) 3113aR 0727 (0728br) 2780aR 0728 (0729br) 3069aR 0729 (0730br) 2966aR
0730 (0731br) 2329aR 0731 (0732br) 1887aR 0732 (0733br) 2867aR 0733 (0734br) 3674aR 0734 (0735br) 3589aR 0735 (0736br) 3284aR 0736 (0737br) 2632aR 0737 (0738br) 3374aR 0738 (0739br) 3804aR 0739 (0740br) 3849aR
0740 (0741br) 2777aR 0741 (0742br) 2777aR 0742 (0743br) 2777aR 0743 (0744br) 2777aR 0744 (0745br) 2777aR 0745 (0746br) 2777aR 0746 (0747br) 2777aR 0747 (0748br) 2777aR 0748 (0749br) 2777aR 0749 (0750br) 2777aR
0760 (0761br) 2115aR 0761 (0762br) 3696aR 0762 (0763br) 1937aR 0763 (0764br) 3394aR 0764 (0765br) 2577aR 0765 (0766br) 3616aR 0766 (0767br) 2577aR 0767 (0768br) 3104aR 0768 (0769br) 2115aR 0769 (0770br) 3440aR
0770 (0771br) 2854aR 0771 (0772br) 2339aR 0772 (0773br) 2444aR 0773 (0774br) 3296aR 0774 (0775br) 3161aR 0775 (0776br) 3377aR 0776 (0777br) 2870aR 0777 (0778br) 2497aR 0778 (0779br) 2524aR 0779 (0780br) 3394aR
0780 (0781br) 2708aR 0781 (0782br) 3600aR 0782 (0783br) 2944aR 0783 (0784br) 3296aR 0784 (0785br) 2780aR 0785 (0786br) 3377aR 0786 (0787br) 2896aR 0787 (0788br) 2457aR 0788 (0789br) 2524aR 0789 (0790br) 3394aR
0790 (0791br) 1942aR 0791 (0792br) 3022aR 0792 (0793br) 3160aR 0793 (0794br) 3767aR 0794 (0795br) 3475aR 0795 (0796br) 2377aR 0796 (0797br) 2896aR 0797 (0798br) 2457aR 0798 (0799br) 2524aR 0799 (0800br) 3394aR
0800 (0801br) 2791aR 0801 (0802br) 3838aR 0802 (0803br) 1939aR 0803 (0804br) 3703aR 0804 (0805br) 3047aR 0805 (0806br) 2651aR 0806 (0807br) 2637aR 0807 (0808br) 3296aR 0808 (0809br) 2128aR 0809 (0810br) 2547aR
0810 (0811br) 2908aR 0811 (0812br) 2908aR 0812 (0813br) 3595aR 0813 (0814br) 3489aR 0814 (0815br) 2437aR 0815 (0816br) 3208aR 0816 (0817br) 2859aR 0817 (0818br) 3002aR 0818 (0819br) 2529aR 0819 (0820br) 2244aR
0820 (0821br) 1925aR 0821 (0822br) 3834aR 0822 (0823br) 3745aR 0823 (0824br) 1932aR 0824 (0825br) 2107aR 0825 (0826br) 3511aR 0826 (0827br) 2637aR 0827 (0828br) 3296aR 0828 (0829br) 2128aR 0829 (0830br) 2254aR
0830 (0831br) 3765aR 0831 (0832br) 2394aR 0832 (0833br) 3151aR 0833 (0834br) 3374aR 0834 (0835br) 2916aR 0835 (0836br) 2912aR 0836 (0837br) 3335aR 0837 (0838br) 3893aR 0838 (0839br) 2131aR 0839 (0840br) 3412aR
0840 (0841br) 1939aR 0841 (0842br) 2908aR 0842 (0843br) 2908aR 0843 (0844br) 3595aR 0844 (0845br) 2437aR 0845 (0846br) 3208aR 0846 (0847br) 2859aR 0847 (0848br) 3002aR 0848 (0849br) 2529aR 0849 (0850br) 2244aR
0850 (0851br) 2839aR 0851 (0852br) 3017aR 0852 (0853br) 2820aR 0853 (0854br) 3354aR 0854 (0855br) 2407aR 0855 (0856br) 3442aR 0856 (0857br) 3603aR 0857 (0858br) 3401aR 0858 (0859br) 2820aR 0859 (0860br) 3125aR
0860 (0861br) 2710aR 0861 (0862br) 2216aR 0862 (0863br) 3144aR 0863 (0864br) 2877aR 0864 (0865br) 2443aR 0865 (0866br) 3394aR 0866 (0867br) 2533aR 0867 (0868br) 3474aR 0868 (0869br) 3474aR 0869 (0870br) 2266aR
0870 (0871br) 2869aR 0871 (0872br) 2337aR 0872 (0873br) 3595aR 0873 (0874br) 3489aR 0874 (0875br) 2437aR 0875 (0876br) 3208aR 0876 (0877br) 2859aR 0877 (0878br) 3002aR 0878 (0879br) 2529aR 0879 (0880br) 2244aR
0880 (0881br) 2867aR 0881 (0882br) 3760aR 0882 (0883br) 2065aR 0883 (0884br) 2877aR 0884 (0885br) 2833aR 0885 (0886br) 3894aR 0886 (0887br) 2633aR 0887 (0888br) 3248aR 0888 (0889br) 1891aR 0889 (0890br) 3525aR
0890 (0891br) 2125aR 0891 (0892br) 2549aR 0892 (0893br) 2105aR 0893 (0894br) 3373aR 0894 (0895br) 2626aR 0895 (0896br) 3859aR 0896 (0897br) 3113aR 0897 (0898br) 1964aR 0898 (0899br) 3161aR 0899 (0900br) 2998aR
0900 (0901br) 2966aR 0901 (0902br) 2297aR 0902 (0903br) 2053aR 0903 (0904br) 3284aR 0904 (0905br) 2637aR 0905 (0906br) 3208aR 0906 (0907br) 3113aR 0907 (0908br) 1964aR 0908 (0909br) 3161aR 0909 (0910br) 2998aR
0910 (0911br) 2708aR 0911 (0912br) 2003aR 0912 (0913br) 1921aR 0913 (0914br) 2318aR 0914 (0915br) 2926aR 0915 (0916br) 3589aR 0916 (0917br) 3045aR 0917 (0918br) 3446aR 0918 (0919br) 3739aR 0919 (0920br) 3186aR
0920 (0921br) 3790aR 0921 (0922br) 2297aR 0922 (0923br) 2053aR 0923 (0924br) 3284aR 0924 (0925br) 2637aR 0925 (0926br) 3859aR 0926 (0927br) 3113aR 0927 (0928br) 1964aR 0928 (0929br) 3161aR 0929 (0930br) 2998aR
0930 (0931br) 3191aR 0931 (0932br) 2966aR 0932 (0933br) 3595aR 0933 (0934br) 3489aR 0934 (0935br) 2437aR 0935 (0936br) 3208aR 0936 (0937br) 2859aR 0937 (0938br) 3002aR 0938 (0939br) 2529aR 0939 (0940br) 2244aR
0940 (0941br) 2060aR 0941 (0942br) 3295aR 0942 (0943br) 2620aR 0943 (0944br) 3838aR 0944 (0945br) 2035aR 0945 (0946br) 3018aR 0946 (0947br) 2632aR 0947 (0948br) 3374aR 0948 (0949br) 3804aR 0949 (0950br) 3926aR
0950 (0951br) 3141aR 0951 (0952br) 2369aR 0952 (0953br) 2556aR 0953 (0954br) 2375aR 0954 (0955br) 3213aR 0955 (0956br) 3875aR 0956 (0957br) 2859aR 0957 (0958br) 3429aR 0958 (0959br) 2659aR 0959 (0960br) 3439aR
0960 (0961br) 2125aR 0961 (0962br) 2908aR 0962 (0963br) 2908aR 0963 (0964br) 3595aR 0964 (0965br) 2437aR 0965 (0966br) 3208aR 0966 (0967br) 2859aR 0967 (0968br) 3002aR 0968 (0969br) 2529aR 0969 (0970br) 2244aR
0970 (0971br) 2532aR 0971 (0972br) 1910aR 0972 (0973br) 2660aR 0973 (0974br) 3276aR 0974 (0975br) 3071aR 0975 (0976br) 3859aR 0976 (0977br) 3113aR 0977 (0978br) 1926aR 0978 (0979br) 2659aR 0979 (0980br) 3599aR
0980 (0981br) 2532aR 0981 (0982br) 1982aR 0982 (0983br) 3737aR 0983 (0984br) 1983aR 0984 (0985br) 1983aR 0985 (0986br) 1983aR 0986 (0987br) 1983aR 0987 (0988br) 1983aR 0988 (0989br) 1983aR 0989 (0990br) 1983aR
0990 (0991br) 2134aR 0991 (0992br) 3589aR 0992 (0993br) 3737aR 0993 (0994br) 1983aR 0994 (0995br) 1983aR 0995 (0996br) 1983aR 0996 (0997br) 1983aR 0997 (0998br) 1983aR 0998 (0999br) 1983aR 0999 (1000br) 1983aR
1000 (1001br) 3862aR 1001 (1002br) 3583aR 1002 (1003br) 3474aR 1003 (1004br) 3447aR 1004 (1005br) 2120aR 1005 (1006br) 3447aR 1006 (1007br) 3756aR 1007 (1008br) 2540aR 1008 (1009br) 2627aR 1009 (1010br) 3509aR
1010 (1011br) 2524aR 1011 (1012br) 2339aR 1012 (1013br) 2094aR 1013 (1014br) 3022aR 1014 (1015br) 3790aR 1015 (1016br) 2297aR 1016 (1017br) 2088aR 1017 (1018br) 2741aR 1018 (1019br) 1942aR 1019 (1020br) 3282aR
1020 (1021br) 3595aR 1021 (1022br) 3595aR 1022 (1023br) 3595aR 1023 (1024br) 3595aR 1024 (1025br) 3595aR 1025 (1026br) 3595aR 1026 (1027br) 3595aR 1027 (1028br) 3595aR 1028 (1029br) 3595aR 1029 (1030br) 3595aR
1030 (1031br) 337aR 1031 (1032br) 3545aR 1032 (1033br) 3770aR 1033 (1034br) 3022aR 1034 (1035br) 2115aR 1035 (1036br) 1997aR 1036 (1037br) 2998aR 1037 (1038br) 2761aR 1038 (1039br) 3603aR 1039 (1040br) 3017aR
1040 (1041br) 2127aR 1041 (1042br) 337aR 1042 (1043br) 2449aR 1043 (1044br) 1888aR 1044 (1045br) 3635aR 1045 (1046br) 3186aR 1046 (1047br) 2777aR 1047 (1048br) 3893aR 1048 (1049br) 2068aR 1049 (1050br) 2742aR
1050 (1051br) 2908aR 1051 (1052br) 2908aR 1052 (1053br) 3595aR 1053 (1054br) 3489aR 1054 (1055br) 2437aR 1055 (1056br) 3208aR 1056 (1057br) 2859aR 1057 (1058br) 3002aR 1058 (1059br) 2529aR 1059 (1060br) 2244aR
1060 (1061br) 2105aR 1061 (1062br) 3054aR 1062 (1063br) 3161aR 1063 (1064br) 1988aR 1064 (1065br) 2666aR 1065 (1066br) 3894aR 1066 (1067br) 2616aR 1067 (1068br) 3189aR 1068 (1069br) 2659aR 1069 (1070br) 3616aR
1070 (1071br) 2020aR 1071 (1072br) 3314aR 1072 (1073br) 3745aR 1073 (1074br) 2996aR 1074 (1075br) 1077aR 1075 (1076br) 2314aR 1076 (1077br) 3548aR 1077 (1078br) 2894aR 1078 (1079br) 2854aR 1079 (1080br) 3750aR
1080 (1081br) 3859aR 1081 (1082br) 3859aR 1082 (1083br) 3859aR 1083 (1084br) 3859aR 1084 (1085br) 3859aR 1085 (1086br) 3859aR 1086 (1087br) 3859aR 1087 (1088br) 3859aR 1088 (1089br) 3859aR 1089 (1090br) 3859aR
1090 (1091br) 2053aR 1091 (1092br) 3859aR 1092 (1093br) 3113aR 1093 (1094br) 3767aR 1094 (1095br) 3603aR 1095 (1096br) 1902aR 1096 (1097br) 2966aR 1097 (1098br) 2966aR 1098 (1099br) 2966aR 1099 (1100br) 2053aR
1100 (1101br) 3205aR 1101 (1102br) 3796aR 1102 (1103br) 3238aR 1103 (1104br) 1887aR 1104 (1105br) 2867aR 1105 (1106br) 3760aR 1106 (1107br) 3431aR 1107 (1108br) 2373aR 1108 (1109br) 2115aR 1109 (1110br) 3866aR
1110 (1111br) 3474aR 1111 (1112br) 2896aR 1112 (1113br) 2478aR 1113 (1114br) 2916aR 1114 (1115br) 2867aR 1115 (1116br) 2959aR 1116 (1117br) 2881aR 1117 (1118br) 2984aR 1118 (1119br) 2659aR 1119 (1120br) 3277aR
1120 (

3490 (3491br|3748ar) 3491 (3492br|3184ar) 3492 (3493br|2582ar) 3493 (3494br|3230ar) 3494 (3495br|3431ar) 3495 (3496br|2934ar) 3496 (3497br|2646ar) 3497 (3498br|1961ar) 3498 (3499br|2593ar) 3499 (3500br|3296ar)
3500 (3501br|2698ar) 3501 (3502br|2229ar) 3502 (3503br|2107ar) 3503 (3504br|3744ar) 3504 (3505br|2439ar) 3505 (3506br|3180ar) 3506 (3507br|2969ar) 3507 (3508br|1911ar) 3508 (3509br|2051ar) 3509 (3510br|2253ar)
3510 (3511br|2861ar) 3511 (3512br|2780ar) 3512 (3513br|3066ar) 3513 (3514br|2798ar) 3514 (3515br|2155ar) 3515 (3516br|3194ar) 3516 (3517br|2916ar) 3517 (3518br|2519ar) 3518 (3519br|3019ar) 3519 (3520br|1937ar)
3520 (3521br|2693ar) 3521 (3522br|2780ar) 3522 (3523br|2780ar) 3523 (3524br|3799ar) 3524 (3525br|2155ar) 3525 (3526br|3180ar) 3526 (3527br|2916ar) 3527 (3528br|1911ar) 3528 (3529br|2051ar) 3529 (3530br|2930ar)
3530 (3531br|2443ar) 3531 (3532br|3273ar) 3532 (3533br|2113ar) 3533 (3534br|3744ar) 3534 (3535br|2817ar) 3535 (3536br|3194ar) 3536 (3537br|2905ar) 3537 (3538br|3180ar) 3538 (3539br|2852ar) 3539 (3540br|3742ar)
3540 (3541br|3544ar) 3541 (3542br|3273ar) 3542 (3543br|2113ar) 3543 (3544br|3744ar) 3544 (3545br|2817ar) 3545 (3546br|3194ar) 3546 (3547br|2905ar) 3547 (3548br|3180ar) 3548 (3549br|2852ar) 3549 (3550br|3742ar)
3550 (3551br|2617ar) 3551 (3552br|3079ar) 3552 (3553br|3079ar) 3553 (3554br|2946ar) 3554 (3555br|2884ar) 3555 (3556br|1968ar) 3556 (3557br|2959ar) 3557 (3558br|3180ar) 3558 (3559br|2852ar) 3559 (3560br|2253ar)
3560 (3561br|2587ar) 3561 (3562br|3394ar) 3562 (3563br|2779ar) 3563 (3564br|3742ar) 3564 (3565br|3779ar) 3565 (3566br|2244ar) 3566 (3567br|2663ar) 3567 (3568br|2158ar) 3568 (3569br|3739ar) 3569 (3570br|3738ar)
3570 (3571br|3035ar) 3571 (3572br|3562ar) 3572 (3573br|3745ar) 3573 (3574br|1903ar) 3574 (3575br|3194ar) 3575 (3576br|3194ar) 3576 (3577br|2916ar) 3577 (3578br|3561ar) 3578 (3579br|1929ar) 3579 (3580br|3561ar)
3580 (3581br|1927ar) 3581 (3582br|3838ar) 3582 (3583br|2940ar) 3583 (3584br|1975ar) 3584 (3585br|3067ar) 3585 (3586br|3438ar) 3586 (3587br|2916ar) 3587 (3588br|1952ar) 3588 (3589br|3291ar) 3589 (3600br|3354ar)
3600 (3601br|3035ar) 3601 (3602br|3745ar) 3602 (3603br|3745ar) 3603 (3604br|1903ar) 3604 (3605br|3194ar) 3605 (3606br|3194ar) 3606 (3607br|2916ar) 3607 (3608br|3561ar) 3608 (3609br|1929ar) 3609 (3610br|3274ar)
3610 (3611br|2587ar) 3611 (3612br|2370ar) 3612 (3613br|2652ar) 3613 (3614br|1975ar) 3614 (3615br|3067ar) 3615 (3616br|3438ar) 3616 (3617br|2916ar) 3617 (3618br|1952ar) 3618 (3619br|3291ar) 3619 (3620br|2237ar)
3620 (3621br|2859ar) 3621 (3622br|3438ar) 3622 (3623br|3478ar) 3623 (3624br|2161ar) 3624 (3625br|2669ar) 3625 (3626br|3001ar) 3626 (3627br|2639ar) 3627 (3628br|1968ar) 3628 (3629br|1929ar) 3629 (3630br|2934ar)
3630 (3631br|2949ar) 3631 (3632br|2292ar) 3632 (3633br|2292ar) 3633 (3634br|3218ar) 3634 (3635br|2186ar) 3635 (3636br|2186ar) 3636 (3637br|3748ar) 3637 (3638br|3397ar) 3638 (3639br|2549ar) 3639 (3640br|2394ar)
3640 (3641br|1929ar) 3641 (3642br|2529ar) 3642 (3643br|2023ar) 3643 (3644br|3218ar) 3644 (3645br|3218ar) 3645 (3646br|2234ar) 3646 (3647br|3748ar) 3647 (3648br|1951ar) 3648 (3649br|1929ar) 3649 (3650br|2197ar)
3650 (3651br|2659ar) 3651 (3652br|3422ar) 3652 (3653br|2940ar) 3653 (3654br|1968ar) 3654 (3655br|2820ar) 3655 (3656br|3994ar) 3656 (3657br|2436ar) 3657 (3658br|3027ar) 3658 (3659br|2790ar) 3659 (3660br|3190ar)
3660 (3661br|2519ar) 3661 (3662br|3322ar) 3662 (3663br|2940ar) 3663 (3664br|1968ar) 3664 (3665br|2820ar) 3665 (3666br|3994ar) 3666 (3667br|2798ar) 3667 (3668br|1903ar) 3668 (3669br|2584ar) 3669 (3670br|2228ar)
3670 (3671br|3035ar) 3671 (3672br|3278ar) 3672 (3673br|2919ar) 3673 (3674br|1967ar) 3674 (3675br|2820ar) 3675 (3676br|3994ar) 3676 (3677br|2436ar) 3677 (3678br|3027ar) 3678 (3679br|2790ar) 3679 (3680br|2282ar)
3680 (3681br|2857ar) 3681 (3682br|3186ar) 3682 (3683br|2789ar) 3683 (3684br|2237ar) 3684 (3685br|2789ar) 3685 (3686br|1903ar) 3686 (3687br|2021ar) 3687 (3688br|2226ar) 3688 (3689br|1942ar) 3689 (3690br|1965ar)
3690 (3691br|2197ar) 3691 (3692br|1981ar) 3692 (3693br|1981ar) 3693 (3694br|3434ar) 3694 (3695br|2940ar) 3695 (3696br|3434ar) 3696 (3697br|3603ar) 3697 (3698br|2617ar) 3698 (3699br|2617ar) 3699 (3700br|3002ar)
3700 (3701br|2637ar) 3701 (3702br|1981ar) 3702 (3703br|1981ar) 3703 (3704br|3434ar) 3704 (3705br|2789ar) 3705 (3706br|1903ar) 3706 (3707br|2617ar) 3707 (3708br|2226ar) 3708 (3709br|3598ar) 3709 (3710br|1911ar)
3710 (3711br|3081ar) 3711 (3712br|2238ar) 3712 (3713br|2859ar) 3713 (3714br|1978ar) 3714 (3715br|2073ar) 3715 (3716br|3397ar) 3716 (3717br|2023ar) 3717 (3718br|1981ar) 3718 (3719br|2067ar) 3719 (3720br|3411ar)
3720 (3721br|2057ar) 3721 (3722br|1977ar) 3722 (3723br|2640ar) 3723 (3724br|3546ar) 3724 (3725br|3748ar) 3725 (3726br|1969ar) 3726 (3727br|2441ar) 3727 (3728br|3294ar) 3728 (3729br|1892ar) 3729 (3730br|2934ar)
3730 (3731br|3591ar) 3731 (3732br|3883ar) 3732 (3733br|2649ar) 3733 (3734br|3422ar) 3734 (3735br|3748ar) 3735 (3736br|2302ar) 3736 (3737br|3144ar) 3737 (3738br|2987ar) 3738 (3739br|2710ar) 3739 (3740br|3613ar)
3740 (3741br|3041ar) 3741 (3742br|3446ar) 3742 (3743br|2652ar) 3743 (3744br|3266ar) 3744 (3745br|2441ar) 3745 (3746br|3535ar) 3746 (3747br|2620ar) 3747 (3748br|1905ar) 3748 (3749br|2537ar) 3749 (3750br|3838ar)
3750 (3751br|2917ar) 3751 (3752br|3394ar) 3752 (3753br|2660ar) 3753 (3754br|3027ar) 3754 (3755br|1899ar) 3755 (3756br|3487ar) 3756 (3757br|1929ar) 3757 (3758br|3397ar) 3758 (3759br|2449ar) 3759 (3760br|2394ar)
3760 (3761br|2820ar) 3761 (3762br|3833ar) 3762 (3763br|2652ar) 3763 (3764br|3266ar) 3764 (3765br|2441ar) 3765 (3766br|3438ar) 3766 (3767br|2436ar) 3767 (3768br|3054ar) 3768 (3769br|2790ar) 3769 (3770br|2912ar)
3770 (3771br|2663ar) 3771 (3772br|2266ar) 3772 (3773br|2949ar) 3773 (3774br|3006ar) 3774 (3775br|2073ar) 3775 (3776br|3667ar) 3776 (3777br|1931ar) 3777 (3778br|3838ar) 3778 (3779br|2951ar) 3779 (3780br|2370ar)
3780 (3781br|2978ar) 3781 (3782br|3567ar) 3782 (3783br|2292ar) 3783 (3784br|3292ar) 3784 (3785br|1929ar) 3785 (3786br|3194ar) 3786 (3787br|2916ar) 3787 (3788br|3561ar) 3788 (3789br|1929ar) 3789 (3790br|3294ar)
3790 (3791br|2652ar) 3791 (3792br|2252ar) 3792 (3793br|2587ar) 3793 (3794br|3703ar) 3794 (3795br|3045ar) 3795 (3796br|3001ar) 3796 (3797br|2660ar) 3797 (3798br|3838ar) 3798 (3799br|2861ar) 3799 (3800br|2196ar)
3800 (3801br|2584ar) 3801 (3802br|2281ar) 3802 (3803br|2076ar) 3803 (3804br) 3804 (3805br|2653ar) 3805 (3806br|3532ar) 3806 (3807br|2533ar) 3807 (3808br|2294ar) 3808 (3809br|3069ar) 3809 (3810br|3423ar)
3810 (3811br|2949ar) 3811 (3812br|2292ar) 3812 (3813br|2292ar) 3813 (3814br|3218ar) 3814 (3815br|2186ar) 3815 (3816br|2186ar) 3816 (3817br|3748ar) 3817 (3818br|3397ar) 3818 (3819br|2549ar) 3819 (3820br|2394ar)
3820 (3821br|3748ar) 3821 (3822br|1910ar) 3822 (3823br|3747ar) 3823 (3824br|1981ar) 3824 (3825br|2652ar) 3825 (3826br|3882ar) 3826 (3827br|2917ar) 3827 (3828br|2294ar) 3828 (3829br|2065ar) 3829 (3830br|1939ar)
3830 (3831br|2819ar) 3831 (3832br|3760ar) 3832 (3833br|3798ar) 3833 (3834br|2230ar) 3834 (3835br|2652ar) 3835 (3836br|3651ar) 3836 (3837br|2715ar) 3837 (3838br|2294ar) 3838 (3839br|2959ar) 3839 (3840br|3290ar)
3840 (3841br|2819ar) 3841 (3842br|3760ar) 3842 (3843br|3798ar) 3843 (3844br|2230ar) 3844 (3845br|2652ar) 3845 (3846br|3651ar) 3846 (3847br|2715ar) 3847 (3848br|2294ar) 3848 (3849br|2959ar) 3849 (3850br|1967ar)
3850 (3851br|2870ar) 3851 (3852br|2498ar) 3852 (3853br|2019ar) 3853 (3854br|3239ar) 3854 (3855br|2052ar) 3855 (3856br|3859ar) 3856 (3857br|3154ar) 3857 (3858br|2142ar) 3858 (3859br|3149ar) 3859 (3860br|2226ar)
3860 (3861br|3790ar) 3861 (3862br|2301ar) 3862 (3863br|2532ar) 3863 (3864br|1968ar) 3864 (3865br|2820ar) 3865 (3866br|3651ar) 3866 (3867br|2721ar) 3867 (3868br|2975ar) 3868 (3869br|2874ar) 3869 (3870br|3264ar)
3870 (3871br|3035ar) 3871 (3872br|3562ar) 3872 (3873br|3745ar) 3873 (3874br|1903ar) 3874 (3875br|3194ar) 3875 (3876br|3194ar) 3876 (3877br|2916ar) 3877 (3878br|3561ar) 3878 (3879br|1929ar) 3879 (3880br|3561ar)
3880 (3881br|3149ar) 3881 (3882br|2262ar) 3882 (3883br|3790ar) 3883 (3884br|2297ar) 3884 (3885br|2789ar) 3885 (3886br|1903ar) 3886 (3887br|2021ar) 3887 (3888br|2226ar) 3888 (3889br|1939ar) 3889 (3890br|3017ar)
3890 (3891br|2792ar) 3891 (3892br|2742ar) 3892 (3893br|2396ar) 3893 (3894br|2292ar) 3894 (3895br|2125ar) 3895 (3896br|2974ar) 3896 (3897br|2966ar) 3897 (3898br|3018ar) 3898 (3899br|2055ar) 3899 (3900br|3540ar)
3900 (3901br|3035ar) 3901 (3902br|3562ar) 3902 (3903br|3745ar) 3903 (3904br|1903ar) 3904 (3905br|3194ar) 3905 (3906br|3194ar) 3906 (3907br|2916ar) 3907 (3908br|3561ar) 3908 (3909br|1929ar) 3909 (3910br|3274ar)
3910 (3911br|2710ar) 3911 (3912br|3278ar) 3912 (3913br|2652ar) 3913 (3914br|1975ar) 3914 (3915br|3067ar) 3915 (3916br|3438ar) 3916 (3917br|2916ar) 3917 (3918br|1952ar) 3918 (3919br|3291ar) 3919 (3920br|3854ar)
3920 (3921br|3429ar) 3921 (3922br|2259ar) 3922 (3923br|2012ar) 3923 (3924br|2370ar) 3924 (3925br|2533ar) 3925 (3926br|3795ar) 3926 (3927br|1892ar) 3927 (3928br|2598ar) 3928 (3929br|3928ar) 3929 (3930br|3046ar)
3930 (3931br|2949ar) 3931 (3932br|2292ar) 3932 (3933br|2292ar) 3933 (3934br|3218ar) 3934 (3935br|2186ar) 3935 (3936br|2186ar) 3936 (3937br|3748ar) 3937 (3938br|3397ar) 3938 (3939br|2549ar) 3939 (3940br|2394ar)
3940 (3941br|1929ar) 3941 (3942br|3943ar) 3942 (3943br|3943ar) 3943 (3944br|3943ar) 3944 (3945br|3943ar) 3945 (3946br|3943ar) 3946 (3947br|3943ar) 3947 (3948br|3943ar) 3948 (3949br|3943ar) 3949 (3950br|3943ar)
3950 (3951br|3950ar) 3951 (3952ar) 3952 (3953ar) 3953 (3954ar) 3954 (3955ar) 3955 (3956br) 3956 (3957ar) 3957 (3958br) 3958 (3959ar) 3959 (3960br)
3960 (3961br|3960ar) 3961 (3962ar) 3962 (3963ar) 3963 (3964ar) 3964 (3965ar) 3965 (3966br) 3966 (3967ar) 3967 (3968br) 3968 (3969ar) 3969 (3970br)
3970 (3971br|3971ar) 3971 (3972ar) 3972 (3973ar) 3973 (3974ar) 3974 (3975ar) 3975 (3976br) 3976 (3977ar) 3977 (3978br) 3978 (3979ar) 3979 (3980ar)
3980 (3981br|3980ar) 3981 (3982ar) 3982 (3983ar) 3983 (3984ar) 3984 (3985ar) 3985 (3986br) 3986 (3987ar) 3987 (3988br) 3988 (3989ar) 3989 (3990ar)
3990 (3991br|3990ar) 3991 (3992ar) 3992 (3993ar) 3993 (3994ar) 3994 (3995ar) 3995 (3996br) 3996 (3997ar) 3997 (3998br) 3998 (3999ar) 3999 (4000br)
4000 (4001br|4000ar) 4001 (4002ar) 4002 (4003ar) 4003 (4004ar) 4004 (4005ar) 4005 (4006br) 4006 (4007ar) 4007 (4008br) 4008 (4009ar) 4009 (4010ar)
4010 (4011br|4010ar) 4011 (4012ar) 4012 (4013ar) 4013 (4014ar) 4014 (4015ar) 4015 (4016br) 4016 (4017ar) 4017 (4018br) 4018 (4019ar) 4019 (4020ar)
4020 (4021br|4020ar) 4021 (4022ar) 4022 (4023ar) 4023 (4024ar) 4024 (4025ar) 4025 (4026br) 4026 (4027ar) 4027 (4028br) 4028 (4029ar) 4029 (4030ar)
4030 (4031br|4030ar) 4031 (4032ar) 4032 (4033ar) 4033 (4034ar) 4034 (4035ar) 4035 (4036br) 4036 (4037ar) 4037 (4038br) 4038 (4039ar) 4039 (4040ar)
4040 (4041br|4040ar) 4041 (4042ar) 4042 (4043ar) 4043 (4044ar) 4044 (4045ar) 4045 (4046br) 4046 (4047ar) 4047 (4048br) 4048 (4049ar) 4049 (4050ar)
4050 (4051br|4050ar) 4051 (4052ar) 4052 (4053ar) 4053 (4054ar) 4054 (4055ar) 4055 (4056br) 4056 (4057ar) 4057 (4058br) 4058 (4059ar) 4059 (4060ar)
4060 (4061br|4060ar) 4061 (4062ar) 4062 (4063ar) 4063 (4064ar) 4064 (4065ar) 4065 (4066br) 4066 (4067ar) 4067 (4068br) 4068 (4069ar) 4069 (4070ar)
4070 (4071br|4070ar) 4071 (4072ar) 4072 (4073ar) 4073 (4074ar) 4074 (4075ar) 4075 (4076br) 4076 (4077ar) 4077 (4078ar) 4078 (4079ar) 4079 (4080ar)
4080 (4081br|4080ar) 4081 (4082ar) 4082 (4083ar) 4083 (4084ar) 4084 (4085ar) 4085 (4086br) 4086 (4087ar) 4087 (4088br) 4088 (4089ar) 4089 (4090ar)
4090 (4091br|4090ar) 4091 (4092ar) 4092 (4093ar) 4093 (4094ar) 4094 (4095ar) 4095 (4096br) 4096 (4097ar) 4097 (4098br) 4098 (4099ar) 4099 (4100ar)
4100 (4101br|4100ar) 4101 (4102ar) 4102 (4103ar) 4103 (4104ar) 4104 (4105ar) 4105 (4106br) 4106 (4107ar) 4107 (4108br) 4108 (4109ar) 4109 (4110ar)
4110 (4111br|4110ar) 4111 (4112ar) 4112 (4113ar) 4113 (4114ar) 4114 (4115ar) 4115 (4116br) 4116 (4117ar) 4117 (4118br) 4118 (4119ar) 4119 (4120ar)
4120 (4121br|4120ar) 4121 (4122ar) 4122 (4123ar) 4123 (4124ar) 4124 (4125ar) 4125 (4126br) 4126 (4127ar) 4127 (4128br) 4128 (4129ar) 4129 (4130ar)
4130 (4131br|4130ar) 4131 (4132ar) 4132 (4133ar) 4133 (4134ar) 4134 (4135ar) 4135 (4136br) 4136 (4137ar) 4137 (4138br) 4138 (4139ar) 4139 (4140ar)
4140 (4141br|4140ar) 4141 (4142ar) 4142 (4143ar) 4143 (4144ar) 4144 (4145ar) 4145 (4146br) 4146 (4147ar) 4147 (4148br) 4148 (4149ar) 4149 (4150ar)
4150 (4151br|4150ar) 4151 (4152ar) 4152 (4153ar) 4153 (4154ar) 4154 (4155ar) 4155 (4156br) 4156 (4157ar) 4157 (4158br) 4158 (4159ar) 4159 (4160ar)
4160 (4161br|4160ar) 4161 (4162ar) 4162 (4163ar) 4163 (4164ar) 4164 (4165ar) 4165 (4166br) 4166 (4167ar) 4167 (4168br) 4168 (4169ar) 4169 (4170ar)
4170 (4171br|4170ar) 4171 (4172ar) 4172 (4173ar) 4173 (4174ar) 4174 (4175ar) 4175 (4176br) 4176 (4177ar) 4177 (4178br) 4178 (4179ar) 4179 (4180ar)
4180 (4181br|4180ar) 4181 (4182ar) 4182 (4183ar) 4183 (4184ar) 4184 (4185ar) 4185 (4186br) 4186 (4187ar) 4187 (4188br) 4188 (4189ar) 4189 (4190ar)
4190 (4191br|4190ar) 4191 (4192ar) 4192 (4193ar) 4193 (4194ar) 4194 (4195ar) 4195 (4196br) 4196 (4197ar) 4197 (4198br) 4198 (4199ar) 4199 (4200ar)
4200 (4201br|4200ar) 4201 (4202ar) 4202 (4203ar) 4203 (4204ar) 4204 (4205ar) 4205 (4206br) 4206 (4207ar) 4207 (4208br) 4208 (4209ar) 4209 (4210ar)
4210 (4211br|4210ar) 4211 (4212ar) 4212 (4213ar) 4213 (4214ar) 4214 (4215ar) 4215 (4216br) 4216 (4217ar) 4217 (4218ar) 4218 (4219ar) 4219 (4220ar)
4220 (4221br|4220ar) 4221 (4222ar) 4222 (4223ar) 4223 (4224ar) 4224 (4225ar) 4225 (4226br) 4226 (4227ar) 4227 (4228br) 4228 (4229ar) 4229 (4230ar)
4230 (4231br|4230ar) 4231 (4232ar) 4232 (4233ar) 4233 (4234ar) 4234 (4235ar) 4235 (4236br) 4236 (4237ar) 4237 (4238br) 4238 (4239ar) 4239 (4240ar)
4240 (4241br|4240ar) 4241 (4242ar) 4242 (4243ar) 4243 (4244ar) 4244 (4245ar) 4245 (4246br) 4246 (4247ar) 4247 (4248br) 4248 (4249ar) 4249 (4250ar)
4250 (4251br|4250ar) 4251 (4252ar) 4252 (4253ar) 4253 (4254ar) 4254 (4255ar) 4255 (4256br) 4256 (4257ar) 4257 (4258br) 4258 (4259ar) 4259 (4260ar)
4260 (4261br|4260ar) 4261 (4262ar) 4262 (4263ar) 4263 (4264ar) 4264 (4265ar) 4265 (4266br) 4266 (4267ar) 4267 (4268br) 4268 (4269ar) 4269 (4270ar)
4270 (4271br|4270ar) 4271 (4272ar) 4272 (4273ar) 4273 (4274ar) 4274 (4275ar) 4275 (4276br) 4276 (4277ar) 4277 (4278br) 4278 (4279ar) 4279 (4280ar)
4280 (4281br|4280ar) 4281 (4282ar) 4282 (4283ar) 4283 (4284ar) 4284 (4285ar) 4285 (4286br) 4286 (4287ar) 4287 (4288br)

