

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 16, 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, [13] 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, 7]:

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

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

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

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) [8]. 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 [9]

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.

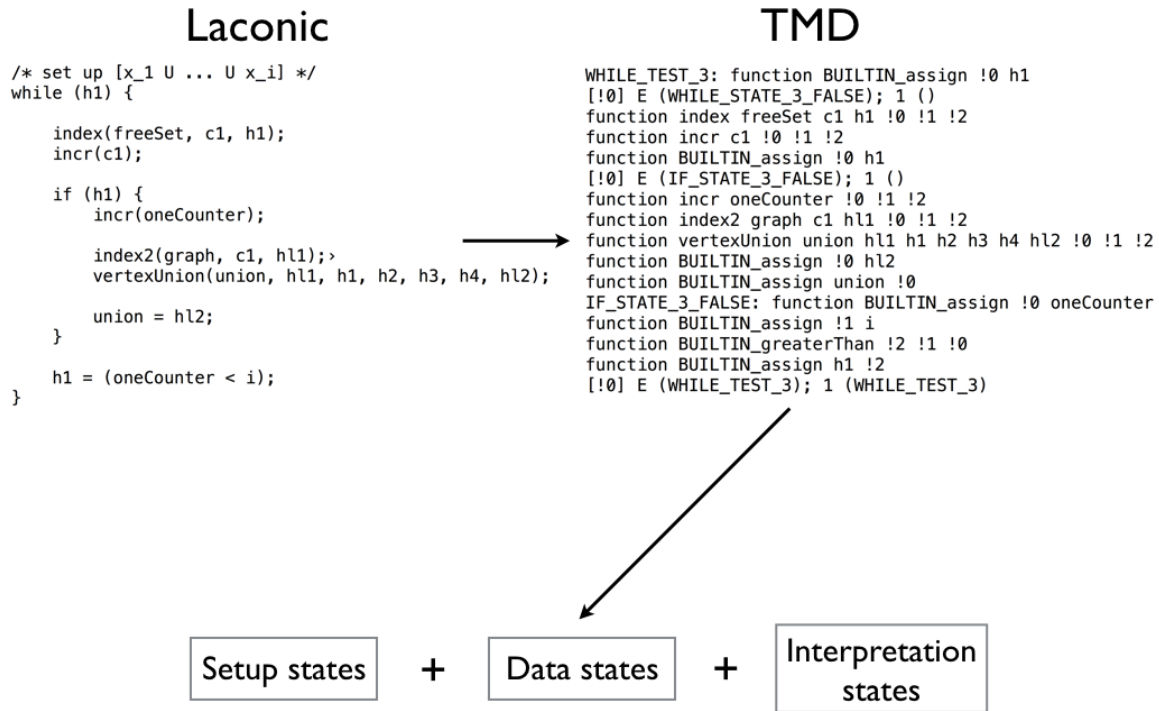


Figure 1: A visual overview of the compilation process.

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*. [6] 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 [5] 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 [14] 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 [9] 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 $[x_1 \cup \dots \cup x_i \cup \{0, \dots, n\}]^{\leq k}$. [9]*

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. [9] 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 [12].

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 [15] 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 $_(1|E)^+_$; 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.

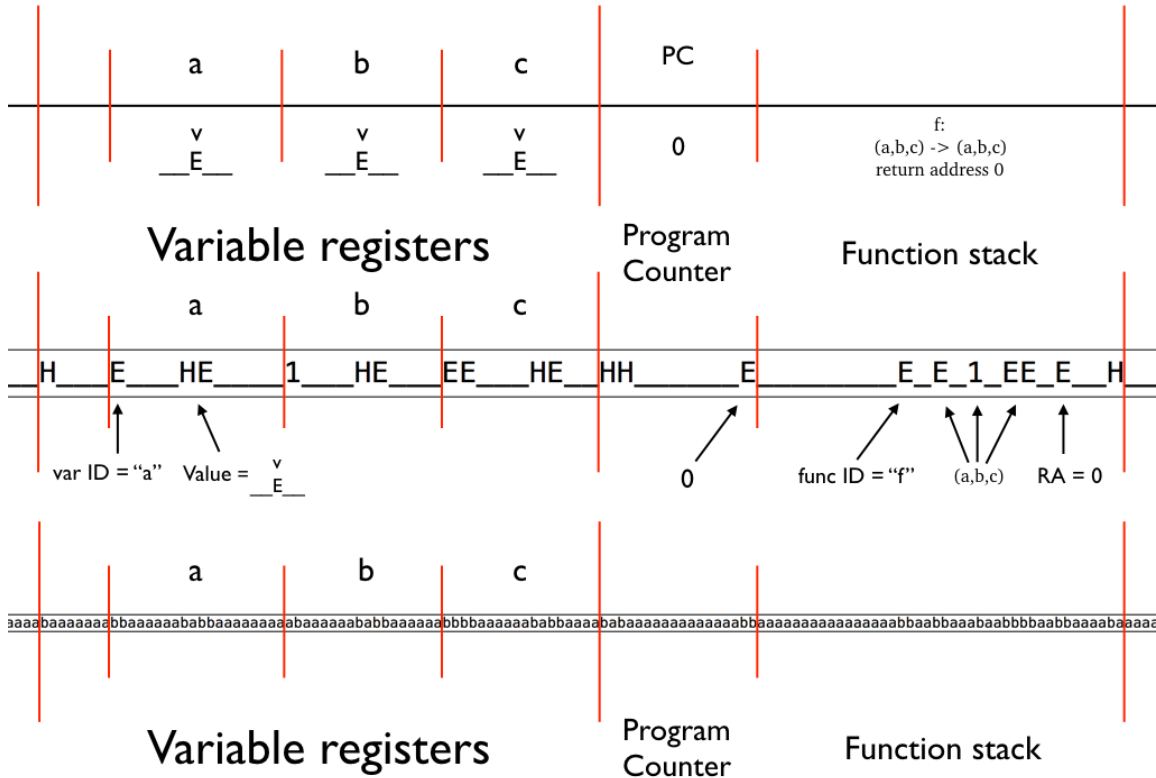


Figure 2: The state of the Turing machine tape after the initializer completes. The TMD program being expressed in Turing machine form is described in full in Appendix B. The top bar is a high-level description of what each part of the Turing machine tape represents. The middle bar is an encoding of 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.

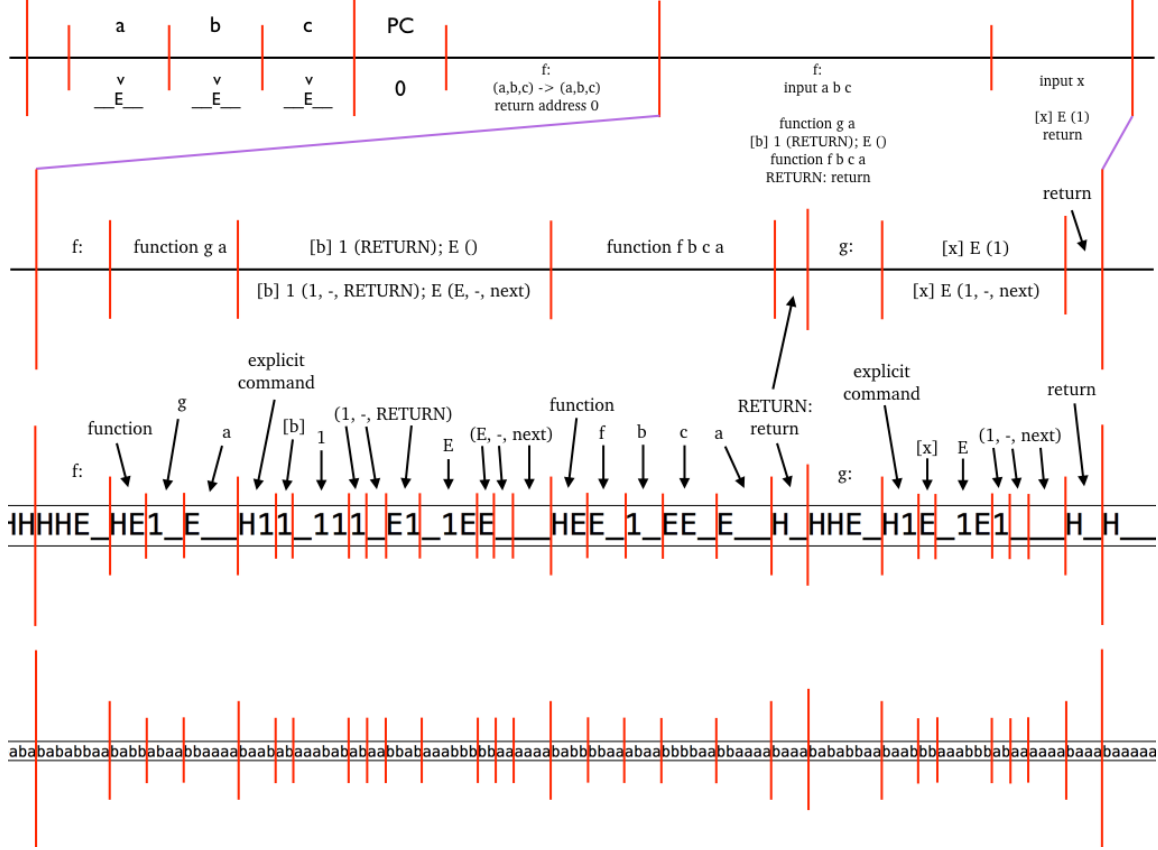


Figure 3: The state of the Turing machine tape after the printer completes. The TMD program being expressed in Turing machine form is described in full in Appendix B. The top bar is a high-level description of the entire tape; unfortunately, at this point there are so many symbols on the tape that it 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 $n_w = \left\lceil \frac{k}{w} \right\rceil$ *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.

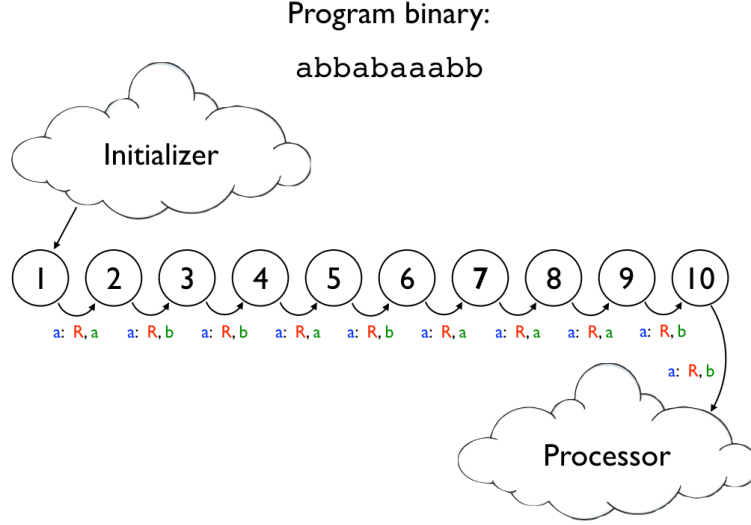


Figure 4: A naïve implementation of the printer. In this example, the hypothetical program is ten bits long, and the printer uses ten states, one for each bit. In the diagram, the blue symbol is the symbol that is read on a transition, the red letter indicates the direction the head moves, and the green 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)

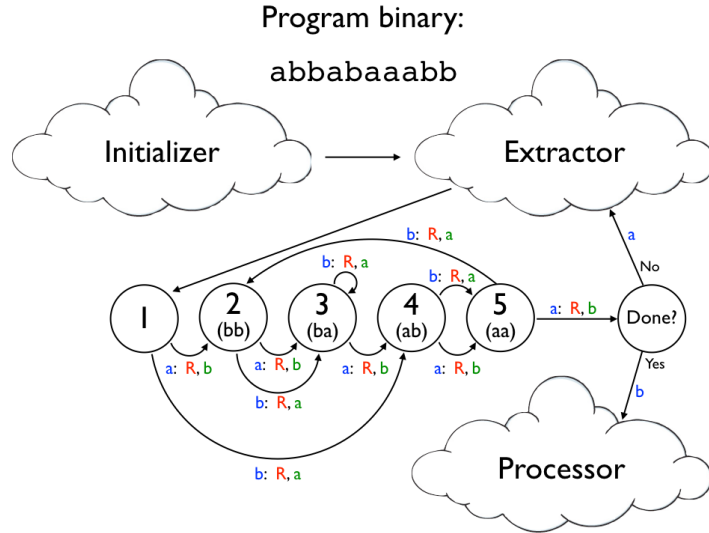


Figure 5: An introspective implementation of the printer. In this example, the hypothetical program is $k = 10$ bits long, and so the word size must be 2 (since $w = 2$ is the largest w such that $w2^w \leq 10$). There are therefore $n_w = \left\lceil \frac{k}{w} \right\rceil = 5$ data states, each encoding 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] 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.
- [6] Chaitin, G. “The Limits of Mathematics.” pp. 79. 1994.
- [7] 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
- [8] Deedlit11, Wythagoras. “Okay, more Turing machines.” 2013. http://googology.wikia.com/wiki/User_blog:Deedlit11/Okay,_more_Turing_machines
- [9] Friedman, H. “Order Invariant Graphs and Finite Incompleteness.” <https://u.osu.edu/friedman.8/files/2014/01/FIiniteSeqInc062214a-v9w7q4.pdf>
- [10] Personal communications with H. Friedman.
- [11] Friedman, H. “Order Theoretic Equations, Maximality, and Incompleteness.” June 7, 2014. <http://u.osu.edu/friedman.8/foundational-adventures/downloadable-manuscripts#78>.
- [12] Friedman H. “The Upper Shift Kernel Theorems.” October 9, 2010. <https://u.osu.edu/friedman.8/files/2014/01/KernStruThm100910-1lu0b8v.pdf>
- [13] 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.
- [14] 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.
- [15] Lagarias, J. “An Elementary Problem Equivalent to the Riemann Hypothesis.” The American Mathematical Monthly, Vol. 109, No. 6, pp. 534-543 (2002)
- [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

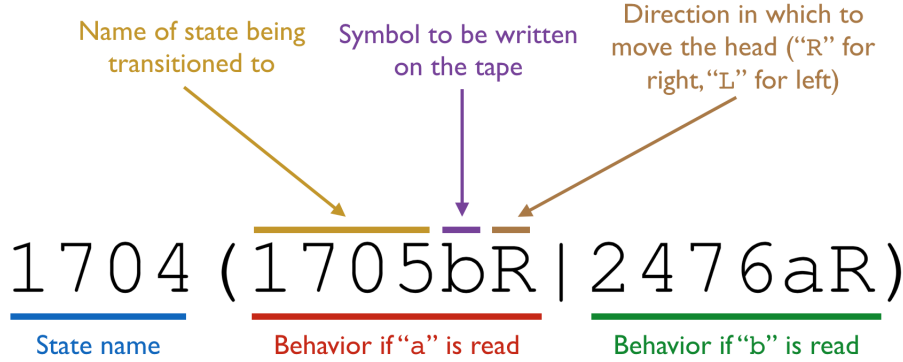


Figure 6: This figure explains how to read a description of a single state. Note that "ERROR-" or "HALT-" denote transitions to the ERROR or HALT states, respectively (no further information is provided because what symbol is written and which direction the head moves are at that point irrelevant).

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 contains only one transition to the HALT state, out of state 7862.

```

0000(0001bR|ERROR-) 0001(0004bR|ERROR-) 0002(0003bR|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(0017bR|ERROR-) 0016(0017bR|ERROR-) 0017(0018bR|ERROR-) 0018(0019aR|ERROR-) 0019(0020aR|0020bR)
0020(0021aR|ERROR-) 0021(0022aR|0022bR) 0022(0023aR|ERROR-) 0023(0024aR|0024bR) 0024(0025aR|ERROR-) 0025(0027aR|0027bR) 0026(0027aR|0032bR) 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(0026aL|0026bL) 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|ERROR-)
0060(0061aR|0061bR) 0061(0062aR|ERROR-) 0062(0063aR|0063bR) 0063(0064aR|ERROR-) 0064(0065aR|0065bR) 0065(0066aR|ERROR-) 0066(0016aR|0016bR) 0067(0068bR|ERROR-) 0068(0069bR|ERROR-) 0069(0070bL|ERROR-)
0070(0062aL|0062bL) 0071(0072aR|0075bR) 0072(0071aR|0073bL) 0073(0074aR|0074bR) 0074(0075aL|0075bL) 0075(ERROR-|0076bL) 0076(0077aR|0077bR) 0077(0078aL|0078bL) 0078(0079aR|0079bR) 0079(0080aL|0078bR)
0080(0081bR|0081bR) 0081(0083aR|0083bR) 0082(0078aR|0078bR) 0083(0084aR|0085bR) 0084(0085aR|0083bR) 0085(0086aL|0083bR) 0086(0087aL|0087bL) 0087(0088aL|0088bL) 0088(0089aR|0094bR) 0089(0090aL|0092bL)
0090(0091aR|0091bR) 0091(0099aL|0099bL) 0092(0099aL|0099bL) 0093(0098aR|0098bL) 0094(0099aL|0097bL) 0095(0099aL|0096bL) 0096(0088aL|0088bL) 0097(0098aL|0098bL) 0098(0088aL|0088bL) 0099(0100aR|0103bR)
0100(0101aL|0103bL) 0101(0102aL|0102bL) 0102(0099aL|0099bL) 0103(0104aR|0104bR) 0104(0101aL|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-|0117bL) 0117(0118aR|0118aR) 0118(0119aL|0119bL) 0119(0120aR|0125bR)
0120(0121aL|0123bL) 0121(0122aR|0122bR) 0122(0130aL|0130bL) 0123(0124aL|0124bL) 0124(0111aR|0111bL) 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)
0150(0151aL|0146bR) 0151(0152aR|0152bR) 0152(0153aR|0153bR) 0153(0154aR|ERROR-) 0154(0155aR|0155bR) 0155(0156bL|ERROR-) 0156(0157aL|0157bL) 0157(0158aR|0163bR) 0158(0159aL|0161bL) 0159(0160aL|0160bL)
0160(0157aL|0157bL) 0161(0162aR|0162bR) 0162(0166aL|0166bL) 0163(0197aR|0164bL) 0164(0165aR|0165bR) 0165(0166aL|0166bL) 0166(0167aR|0172bR) 0167(0168aL|0170bL) 0168(0169aL|0169bL) 0169(0177aL|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(0188aL|0188bL) 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(0219aR|0219bR) 0211(0212bR|ERROR-) 0212(0213bR|ERROR-) 0213(0214aR|ERROR-) 0214(0223aR|0223bR) 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(0236aR|ERROR-) 0236(0245aR|0245bR) 0237(0238aR|ERROR-) 0238(0239bR|ERROR-) 0239(0240bR|ERROR-)
0240(0241bR|ERROR-) 0241(0242bR|ERROR-) 0242(0243bR|ERROR-) 0243(0244aR|ERROR-) 0244(0238bR|ERROR-) 0245(0246aR|ERROR-) 0246(0247aR|0247bR) 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(0215aR|0215bR) 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(0274aL|0274bL) 0272(0273aL|0273bL) 0273(0265aL|0263bL) 0274(0275aR|0275bR) 0275(0281aL|0276bL) 0276(0277aR|0277bR) 0277(0288aL|0288bL) 0278(ERROR-|0279bL) 0279(0280aL|0280bL)
0280(0274aL|0274bL) 0281(0282aR|0285bR) 0282(ERROR-|0283aL) 0283(0284aR|0284bR) 0284(0288aL|0288bL) 0285(0286aL|ERROR-) 0286(0287aR|0287aR) 0287(0300aR|0300bR) 0288(0288aR|0288bR) 0289(0288aR|0288bR)
0290(0311aR|0289bR) 0291(0292aR|0293bR) 0292(0291aR|0291bR) 0293(0294aL|0291bR) 0294(0295aR|0295aR) 0295(0296aR|0296bL) 0296(0297aR|ERROR-) 0297(0298bR|ERROR-) 0298(0299bL|ERROR-) 0299(0263aL|0263bL)
0300(0301aR|0302bR) 0301(0303aR|0303bR) 0302(0304aR|0303bR) 0303(0304aR|0303bR) 0304(0305aR|0306bR) 0305(0306aR|0306bR) 0306(0307bR|ERROR-) 0307(0308aR|0308bR) 0308(0309aR|0309bR) 0309(0310aR|0309bR)
0310(0311aL|0309bR) 0311(0312aL|0312bL) 0312(0313aL|0313bL) 0313(0314aR|0317bR) 0314(0315aR|0315bL) 0315(0316bL|0316bL) 0316(0313aL|0313bL) 0317(0338aR|0318bL) 0318(0339aR|0319aR) 0319(0319aR|0319bR)
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(0320aR|0320bR) 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|0355bR)
0350(0351aL|0355bL) 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(0313aL|0313bL) 0364(0365aL|0365bL) 0365(0360aL|0360bL) 0366(0367aR|0367aR) 0367(0368aR|0368bR) 0368(0371aR|0371bR) 0369(0370aL|0370bL)
0370(0360aL|0360bL) 0371(0372aR|0372bR) 0372(0373aL|0371bR) 0373(0374bR|0374bR) 0374(0385aR|0385bR) 0375(0376aL|0371bR) 0376(0377aR|0377aR) 0377(0378aR|0378bR) 0378(0379bR|ERROR-) 0379(0380aR|0380bR)
0380(0381aR|ERROR-) 0381(0382aR|0382bR) 0382(0383aR|ERROR-) 0383(0384aR|0384bR) 0384(0385bR|ERROR-) 0385(0389aR|0389bR) 0386(0387aR|0389bR) 0387(0306aR|0306bR) 0388(0306aR|0306bR) 0389(0390aR|ERROR-)
0390(0391bL|ERROR-) 0391(0392aR|ERROR-) 0392(0393aL|0392aL) 0393(0393aL|0394aR) 0394(0405aR|ERROR-) 0395(0395bR|2559aR) 0396(0397bR|0377aR) 0397(0398bR|2269aR) 0398(0399bR|2637aR) 0399(0400bR|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|2067aR) 0411(0412bR|2587aR) 0412(0413bR|3002aR) 0413(0414bR|3002aR) 0414(0415bR|3737aR) 0415(0416bR|1978aR) 0416(0417bR|2569aR) 0417(0418bR|3185aR) 0418(0419bR|2440aR) 0419(0420bR|2259aR)
0420(0421bR|2617aR) 0421(0422bR|3757aR) 0422(0423bR|2520aR) 0423(0424bR|3838aR) 0424(0425bR|2658aR) 0425(0426bR|3023aR) 0426(0427bR|2637aR) 0427(0428bR|2637aR) 0428(0429bR|2637aR) 0429(0430bR|2637aR)

```

“

1960 (1961bR) 2637aR 1961 (1962bR) 2930aR 1962 (1963bR) 2075aR 1963 (1964bR) 3850aR 1964 (1965bR) 2105aR 1965 (1966bR) 3744aR 1966 (1967bR) 2584aR 1967 (1968bR) 2409aR 1968 (1969bR) 2085aR 1969 (1970bR) 2266aR
1970 (1971bR) 3465aR 1971 (1972bR) 3306aR 1972 (1973bR) 3080aR 1973 (1974bR) 2654aR 1974 (1975bR) 3429aR 1975 (1976bR) 1952aR 1976 (1977bR) 2659aR 1977 (1978bR) 2167aR 1978 (1979bR) 3081aR 1979 (1980bR) 3360aR
1980 (1981bR) 3137aR 1981 (1982bR) 2639aR 1982 (1983bR) 1913aR 1983 (1984bR) 3461aR 1984 (1985bR) 3717aR 1985 (1986bR) 3759aR 1986 (1987bR) 3081aR 1987 (1988bR) 3620aR 1988 (1989bR) 3620aR
1990 (1991bR) 3193aR 1991 (1992bR) 2162aR 1992 (1993bR) 2639aR 1993 (1994bR) 3478aR 1994 (1995bR) 3022aR 1995 (1996bR) 2651aR 1996 (1997bR) 2085aR 1997 (1998bR) 2085aR 1998 (1999bR) 2353aR
2000 (2001bR) 2696aR 2001 (2002bR) 2351aR 2002 (2003bR) 2059aR 2003 (2004bR) 2286aR 2004 (2005bR) 3748aR 2005 (2006bR) 3748aR 2006 (2007bR) 2859aR 2007 (2008bR) 3488aR 2008 (2009bR) 2637aR 2009 (2010bR) 3060aR
2010 (2011bR) 2696aR 2011 (2012bR) 2351aR 2012 (2013bR) 2059aR 2013 (2014bR) 2286aR 2014 (2015bR) 3748aR 2015 (2016bR) 3748aR 2016 (2017bR) 2859aR 2017 (2018bR) 3488aR 2018 (2019bR) 2637aR 2019 (2020bR) 3060aR
2020 (2021bR) 2696aR 2021 (2022bR) 2351aR 2022 (2023bR) 2059aR 2023 (2024bR) 2286aR 2024 (2025bR) 3748aR 2025 (2026bR) 3748aR 2026 (2027bR) 2859aR 2027 (2028bR) 3488aR 2028 (2029bR) 2637aR 2029 (2030bR) 2166aR
2030 (2031bR) 2859aR 2031 (2032bR) 2286aR 2032 (2033bR) 2059aR 2033 (2034bR) 2286aR 2034 (2035bR) 2059aR 2035 (2036bR) 1977aR 2036 (2037bR) 1939aR 2037 (2038bR) 3411aR 2038 (2039bR) 2436aR 2039 (2040bR) 1910aR
2040 (2041bR) 2859aR 2041 (2042bR) 2286aR 2042 (2043bR) 2059aR 2043 (2044bR) 2286aR 2044 (2045bR) 3748aR 2045 (2046bR) 3748aR 2046 (2047bR) 2859aR 2047 (2048bR) 3488aR 2048 (2049bR) 2637aR 2049 (2050bR) 3060aR
2050 (2051bR) 2696aR 2051 (2052bR) 2351aR 2052 (2053bR) 2059aR 2053 (2054bR) 2286aR 2054 (2055bR) 2084aR 2055 (2056bR) 1977aR 2056 (2057bR) 1939aR 2057 (2058bR) 2083aR 2058 (2059bR) 2436aR 2059 (2060bR) 2287aR
2060 (2061bR) 2833aR 2061 (2062bR) 2761aR 2062 (2063bR) 2593aR 2063 (2064bR) 3504aR 2064 (2065bR) 3291aR 2065 (2066bR) 3679aR 2066 (2067bR) 2627aR 2067 (2068bR) 3511aR 2068 (2069bR) 3756aR 2069 (2070bR) 2482aR
2070 (2071bR) 3661aR 2071 (2072bR) 3287aR 2072 (2073bR) 2897aR 2073 (2074bR) 3859aR 2074 (2075bR) 3274aR 2075 (2076bR) 3759aR 2076 (2077bR) 2105aR 2077 (2078bR) 3906aR 2078 (2079bR) 2565aR 2079 (2080bR) 3849aR
2080 (2081bR) 2820aR 2081 (2082bR) 3017aR 2082 (2083bR) 2893aR 2083 (2084bR) 3247aR 2084 (2085bR) 2870aR 2085 (2086bR) 2497aR 2086 (2087bR) 2436aR 2087 (2088bR) 2083aR 2088 (2089bR) 2324aR 2089 (2090bR) 3249aR
2090 (2091bR) 2699aR 2091 (2092bR) 2761aR 2092 (2093bR) 2075aR 2093 (2094bR) 3838aR 2094 (2095bR) 3658aR 2095 (2096bR) 3854aR 2096 (2097bR) 2780aR 2097 (2098bR) 1882aR 2098 (2099bR) 2654aR 2099 (2100bR) 2410aR
2100 (2101bR) 3617aR 2101 (2102bR) 2754aR 2102 (2103bR) 271aR 2103 (2104bR) 2258aR 2104 (2105bR) 2628aR 2105 (2106bR) 2286aR 2106 (2107bR) 2577aR 2107 (2108bR) 2334aR 2108 (2109bR) 3592aR 2109 (2110bR) 3546aR
2110 (2111bR) 2085aR 2111 (2112bR) 2282aR 2112 (2113bR) 2585aR 2113 (2114bR) 3440aR 2114 (2115bR) 3163aR 2115 (2116bR) 1994aR 2116 (2117bR) 3091aR 2117 (2118bR) 3423aR 2118 (2119bR) 2070aR 2119 (2120bR) 1910aR
2120 (2121bR) 2821aR 2121 (2122bR) 1978aR 2122 (2123bR) 2867aR 2123 (2124bR) 3354aR 2124 (2125bR) 3568aR 2125 (2126bR) 3277aR 2126 (2127bR) 2857aR 2127 (2128bR) 2727aR 2128 (2129bR) 2685aR 2129 (2130bR) 3440aR
2130 (2131bR) 3163aR 2131 (2132bR) 3091aR 2132 (2133bR) 3091aR 2133 (2134bR) 347aR 2134 (2135bR) 201aR 2135 (2136bR) 3253aR 2136 (2137bR) 2076aR 2137 (2138bR) 3249aR 2138 (2139bR) 2870aR 2139 (2140bR) 2398aR
2140 (2141bR) 2708aR 2141 (2142bR) 3550aR 2142 (2143bR) 2021aR 2143 (2144bR) 3447aR 2144 (2145bR) 3756aR 2145 (2146bR) 2394aR 2146 (2147bR) 3152aR 2147 (2148bR) 1923aR 2148 (2149bR) 3446aR
2150 (2151bR) 3083aR 2151 (2152bR) 2286aR 2152 (2153bR) 3432aR 2153 (2154bR) 2480aR 2154 (2155bR) 2084aR 2155 (2156bR) 3613aR 2156 (2157bR) 2780aR 2157 (2158bR) 2944aR 2158 (2159bR) 2646aR 2159 (2160bR) 1898aR
2160 (2161bR) 3612aR 2161 (2162bR) 1908aR 2162 (2163bR) 2083aR 2163 (2164bR) 3680aR 2164 (2165bR) 3271aR 2165 (2166bR) 3263aR 2166 (2167bR) 1931aR 2167 (2168bR) 3702aR 2168 (2169bR) 2660aR 2169 (2170bR) 2152aR
2170 (2171bR) 2617aR 2171 (2172bR) 3838aR 2172 (2173bR) 3045aR 2173 (2174bR) 2260aR 2174 (2175bR) 2023aR 2175 (2176bR) 1981aR 2176 (2177bR) 2820aR 2177 (2178bR) 2834aR 2178 (2179bR) 3012aR 2179 (2180bR) 3002aR
2180 (2181bR) 3620aR 2181 (2182bR) 2162aR 2182 (2183bR) 2617aR 2183 (2184bR) 3850aR 2184 (2185bR) 2105aR 2185 (2186bR) 3790aR 2186 (2187bR) 2838aR 2187 (2188bR) 1969aR 2188 (2189bR) 2593aR 2189 (2190bR) 3859aR
2190 (2191bR) 3115aR 2191 (2192bR) 2207aR 2192 (2193bR) 2085aR 2193 (2194bR) 3360aR 2194 (2195bR) 2888aR 2195 (2196bR) 3440aR 2196 (2197bR) 3207aR 2197 (2198bR) 3112aR 2198 (2199bR) 3332aR 2199 (2200bR) 2933aR
2200 (2201bR) 2627aR 2201 (2202bR) 3905aR 2202 (2203bR) 2452aR 2203 (2204bR) 3748aR 2204 (2205bR) 3593aR 2205 (2206bR) 2770aR 2206 (2207bR) 2780aR 2207 (2208bR) 2324aR 2208 (2209bR) 3249aR 2209 (2210bR) 3679aR
2210 (2211bR) 2451aR 2211 (2212bR) 3411aR 2212 (2213bR) 2009aR 2213 (2214bR) 3514aR 2214 (2215bR) 3467aR 2215 (2216bR) 3761aR 2216 (2217bR) 2075aR 2217 (2218bR) 3854aR 2218 (2219bR) 2654aR 2219 (2220bR) 2230aR
2220 (2221bR) 2599aR 2221 (2222bR) 3833aR 2222 (2223bR) 2663aR 2223 (2224bR) 3290aR 2224 (2225bR) 2952aR 2225 (2226bR) 3186aR 2226 (2227bR) 2864aR 2227 (2228bR) 2254aR 2228 (2229bR) 2917aR 2229 (2230bR) 2260aR
2230 (2231bR) 2577aR 2231 (2232bR) 3358aR 2232 (2233bR) 2632aR 2233 (2234bR) 2818aR 2234 (2235bR) 3196aR 2235 (2236bR) 3761aR 2236 (2237bR) 3612aR 2237 (2238bR) 2651aR 2238 (2239bR) 2721aR 2239 (2240bR) 3247aR
2240 (2241bR) 1929aR 2241 (2242bR) 2481aR 2242 (2243bR) 2444aR 2243 (2244bR) 3566aR 2244 (2245bR) 2659aR 2245 (2246bR) 3422aR 2246 (2247bR) 2940aR 2247 (2248bR) 3567aR 2248 (2249bR) 1927aR 2249 (2250bR) 3838aR
2250 (2251bR) 2820aR 2251 (2252bR) 3593aR 2252 (2253bR) 2059aR 2253 (2254bR) 1969aR 2254 (2255bR) 2454aR 2255 (2256bR) 2253aR 2256 (2257bR) 2659aR 2257 (2258bR) 2659aR 2258 (2259bR) 2535aR 2259 (2260bR) 3022aR
2260 (2261bR) 2960aR 2261 (2262bR) 2222aR 2262 (2263bR) 2053aR 2263 (2264bR) 1969aR 2264 (2265bR) 2454aR 2265 (2266bR) 3422aR 2266 (2267bR) 2940aR 2267 (2268bR) 3567aR 2268 (2269bR) 1927aR 2269 (2270bR) 3838aR
2270 (2271bR) 2917aR 2271 (2272bR) 1969aR 2272 (2273bR) 2441aR 2273 (2274bR) 3283aR 2274 (2275bR) 2858aR 2275 (2276bR) 3950aR 2276 (2277bR) 3745aR 2277 (2278bR) 2075aR 2278 (2279bR) 2021aR 2279 (2280bR) 1978aR
2280 (2281bR) 3083aR 2281 (2282bR) 2761aR 2282 (2283bR) 2593aR 2283 (2284bR) 3504aR 2284 (2285bR) 3291aR 2285 (2286bR) 3679aR 2286 (2287bR) 2627aR 2287 (2288bR) 3511aR 2288 (2289bR) 3756aR 2289 (2290bR) 2482aR
2290 (2291bR) 3661aR 2291 (2292bR) 3287aR 2292 (2293bR) 2897aR 2293 (2294bR) 3859aR 2294 (2295bR) 3274aR 2295 (2296bR) 3759aR 2296 (2297bR) 2105aR 2297 (2298bR) 2083aR 2298 (2299bR) 2324aR 2299 (2300bR) 301aR
2300 (2301bR) 2699aR 2301 (2302bR) 1969aR 2302 (2303bR) 1929aR 2303 (2304bR) 2998aR 2304 (2305bR) 2646aR 2305 (2306bR) 2251aR 2306 (2307bR) 2651aR 2307 (2308bR) 3838aR 2308 (2309bR) 2916aR 2309 (2310bR) 3027aR
2310 (2311bR) 2587aR 2311 (2312bR) 2754aR 2312 (2313bR) 271aR 2313 (2314bR) 2258aR 2314 (2315bR) 2628aR 2315 (2316bR) 2286aR 2316 (2317bR) 2577aR 2317 (2318bR) 3354aR 2318 (2319bR) 3592aR 2319 (2320bR) 3546aR
2320 (2321bR) 3748aR 2321 (2322bR) 1903aR 2322 (2323bR) 2637aR 2323 (2324bR) 3440aR 2324 (2325bR) 2021aR 2325 (2326bR) 3550aR 2326 (2327bR) 2637aR 2327 (2328bR) 2423aR 2328 (2329bR) 2916aR 2329 (2330bR) 3567aR
2330 (2331bR) 1927aR 2331 (2332bR) 2307aR 2332 (2333bR) 2393aR 2333 (2334bR) 3397aR 2334 (2335bR) 2480aR 2335 (2336bR) 1904aR 2336 (2337bR) 3207aR 2337 (2338bR) 3294aR 2338 (2339bR) 2646aR 2339 (2340bR) 2266aR
2340 (2341bR) 2116aR 2341 (2342bR) 3550aR 2342 (2343bR) 2021aR 2343 (2344bR) 3447aR 2344 (2345bR) 3756aR 2345 (2346bR) 2394aR 2346 (2347bR) 3152aR 2347 (2348bR) 1923aR 2348 (2349bR) 3446aR
2350 (2351bR) 3081aR 2351 (2352bR) 2286aR 2352 (2353bR) 3432aR 2353 (2354bR) 2480aR 2354 (2355bR) 2084aR 2355 (2356bR) 1977aR 2356 (2357bR) 2115aR 2357 (2358bR) 3434aR 2358 (2359bR) 3737aR 2359 (2360bR) 1967aR
2360 (2361bR) 2444aR 2361 (2362bR) 2761aR 2362 (2363bR) 2116aR 2363 (2364bR) 1905aR 2364 (2365bR) 3509aR 2365 (2366bR) 3509aR 2366 (2367bR) 2649aR 2367 (2368bR) 3427aR 2368 (2369bR) 2944aR 2369 (2370bR) 1968aR
2370 (2371bR) 3587aR 2371 (2372bR) 3838aR 2372 (2373bR) 3045aR 2373 (2374bR) 2260aR 2374 (2375bR) 2023aR 2375 (2376bR) 1981aR 2376 (2377bR) 2820aR 2377 (2378bR) 2834aR 2378 (2379bR) 3012aR 2379 (2380bR) 3002aR
2380 (2381bR) 2940aR 2381 (2382bR) 2254aR 2382 (2383bR) 2821aR 2383 (2384bR) 1978aR 2384 (2385bR) 2857aR 2385 (2386bR) 307aR 2386 (2387bR) 2436aR 2387 (2388bR) 3275aR 2388 (2389bR) 2606aR 2389 (2390bR) 3651aR
2390 (2391bR) 2722aR 2391 (2392bR) 3489aR 2392 (2393bR) 1928aR 2393 (2394bR) 2480aR 2394 (2395bR) 1937aR 2395 (2396bR) 2295aR 2396 (2397bR) 2617aR 2397 (2398bR) 3554aR 2398 (2399bR) 3777aR 2399 (2400bR) 2249aR
2400 (2401bR) 2650aR 2401 (2402bR) 2619aR 2402 (2403bR) 2222aR 2403 (2404bR) 3290aR 2404 (2405bR) 2888aR 2405 (2406bR) 3440aR 2406 (2407bR) 3207aR 2407 (2408bR) 3112aR 2408 (2409bR) 3332aR 2409 (2410bR) 2933aR
2410 (2411bR) 2065aR 2411 (2412bR) 3027aR 2412 (2413bR) 2619aR 2413 (2414bR) 3694aR 2414 (2415bR) 2105aR 2415 (2416bR) 3189aR 2416 (2417bR) 2859aR 2417 (2418bR) 2436aR 2418 (2419bR) 3428aR 2419 (2420bR) 2976aR
2420 (2421bR) 2056aR 2421 (2422bR) 2489aR 2422 (2423bR) 2057aR 2423 (2424bR) 3027aR 2424 (2425bR) 2021aR 2425 (2426bR) 3422aR 2426 (2427bR) 3745aR 2427 (2428bR) 3664aR 2428 (2429bR) 3428aR 2429 (2430bR) 2956aR
2430 (2431bR) 3320aR 2431 (2432bR) 2663aR 2432 (2433bR) 2663aR 2433 (2434bR) 2009aR 2434 (2435bR) 2044aR 2435 (2436bR) 3761aR 2436 (2437bR) 2075aR 2437 (2438bR) 3854aR 2438 (2439bR) 2654aR 2439 (2440bR) 3592aR
2440 (2441bR) 1929aR 2441 (2442bR) 2222aR 2442 (2443bR) 2070aR 2443 (2444bR) 2429aR 2444 (2445bR) 2619aR 2445 (2446bR) 3853aR 2446 (2447bR) 2065aR 2447 (2448bR) 2996aR 2448 (2449bR) 2115aR 2449 (2450bR) 3905aR
2450 (2451bR) 2399aR 2451 (2452bR) 2296aR 2452 (2453bR) 2498aR 2453 (2454bR) 2483aR 2454 (2455bR) 3018aR 2455 (2456bR) 2837aR 2456 (2457bR) 2458aR 2457 (2458bR) 3564aR 2458 (2459bR) 2790aR 2459 (2460bR) 3625aR
2460 (2461bR) 2617aR 2461 (2462bR) 3679aR 2462 (2463bR) 2021aR 2463 (2464bR) 3283aR 2464 (2465bR) 2617aR 2465 (2466bR) 3422aR 2466 (2467bR) 2940aR 2467 (2468bR) 3567aR 2468 (2469bR) 1927aR 2469 (2470bR) 3838aR
2470 (2471bR) 2917aR 2471 (2472bR) 1978aR 2472 (2473bR) 2441aR 2473 (2474bR) 3283aR 2474 (2475bR) 2858aR 2475 (2476bR) 3950aR 2476 (2477bR) 2859aR 2477 (2478bR) 2075aR 2478 (2479bR) 2021aR 2479 (2480bR) 1978aR
2480 (2481bR) 3083aR 2481 (2482bR) 2761aR 2482 (2483bR) 2593aR 2483 (2484bR) 3504aR 2484 (2485bR) 3291aR 2485 (2486bR) 2627aR 2486 (2487bR) 3511aR 2487 (2488bR) 3756aR 2488 (2489bR) 2482aR 2489 (2490bR) 2457aR
2490 (2491bR) 3661aR 2491 (2492bR) 3287aR 2492 (2493bR) 2897aR 2493 (2494bR) 3859aR 2494 (2495bR) 3274aR 2495 (2496bR) 3759aR 2496 (2497bR) 2105aR 2497 (2498bR) 2083aR 2498 (2499bR) 2324aR 2499 (2500bR) 301aR
2500 (2501bR) 2699aR 2501 (2502bR) 1969aR 2502 (2503bR) 1929aR 2503 (2504bR) 2998aR 2504 (2505bR) 2646aR 2505 (2506bR) 2251aR 2506 (2507bR) 2651aR 2507 (2508bR) 3838aR 2508 (2509bR) 2916aR 2509 (2510bR) 2266aR
2510 (2511bR) 2579aR 2511 (2512bR) 3411aR 2512 (2513bR) 2617aR 2513 (2514bR) 2480aR 2514 (2515bR) 2084aR 2515 (2516bR) 2480aR 2516 (2517bR) 2857aR 2517 (2518bR) 3077aR 2518 (2519bR) 2651aR 2519 (2520bR) 3278aR
2520 (2521bR) 2577aR 2521 (2522bR) 3593aR 2522 (2523bR) 2059aR 2523 (2524bR) 1969aR 2524 (2525bR) 2454aR 2525 (2526bR) 2253aR 2526 (2527bR) 2659aR 2527 (2528bR) 2659aR 2528 (2529bR) 2535aR 2529 (2530bR) 3022aR
2530 (2531bR) 2579aR 2531 (2532bR) 3166aR 2532 (2533bR) 2637aR 2533 (2534bR) 2116aR 2534 (2535bR) 2669aR 2535 (2536bR) 2939aR 2536 (2537bR) 3077aR 2537 (2538bR) 3074aR 2538 (2539bR) 3651aR 2539 (2540bR) 2482aR
2540 (2541bR) 3429aR 2541 (2542bR) 3566aR 2542 (2543bR) 3431aR 2543 (2544bR) 1975aR 2544 (2545bR) 3089aR 2545 (2546bR) 2429aR 2546 (2547bR) 2867aR 2547 (2548bR) 1973aR 2548 (2549bR) 2113aR 2549 (2550bR) 3278aR
2550 (2551bR) 2833aR 2551 (2552bR) 2761aR 2552 (2553bR) 2593aR 2553 (2554bR) 3504aR 2554 (2555bR) 3291aR 2555 (2556bR) 3679aR 2556 (2557bR) 2627aR 2557 (2558bR) 3511aR 2558 (2559bR) 3756aR 2559 (2560bR) 2482aR
2560 (2561bR) 3661aR 2561 (2562bR) 3287aR 2562 (2563bR) 2897aR 2563 (2564bR) 3504aR 2564 (2565bR) 3291aR 2565 (2566bR) 2928aR 2566 (2567bR) 2627aR 2567 (2568bR) 3511aR 2568 (2569bR) 3756aR 2569 (2570bR) 2482aR
2570 (2571bR) 2883aR 2571 (2572bR) 3679aR 2572 (2573bR) 3452aR 2573 (2574bR) 2820aR 2574 (2575bR) 3163aR 2575 (2576bR) 2226aR 2576 (2577bR) 1931aR 2577 (2578bR) 2449aR 2578 (2579bR) 2780aR 257

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|2684ar) 3501 (3502br|2229ar) 3502 (3503br|2107ar) 3503 (3504br|3744ar) 3504 (3505br|3498ar) 3505 (3506br|3189ar) 3506 (3507br|2916ar) 3507 (3508br|1911ar) 3508 (3509br|3091ar) 3509 (3510br|2253ar) 3510 (3511br|2861ar) 3511 (3512br|3591ar) 3512 (3513br|2535ar) 3513 (3514br|3044ar) 3514 (3515br|3258ar) 3515 (3516br|2353ar) 3516 (3517br|1978ar) 3517 (3518br|2053ar) 3518 (3519br|2053ar) 3519 (3520br|2230ar) 3520 (3521br|3006ar) 3521 (3522br|2780ar) 3522 (3523br|2780ar) 3523 (3524br|3799ar) 3524 (3525br|2118ar) 3525 (3526br|3140ar) 3526 (3527br|2916ar) 3527 (3528br|1911ar) 3528 (3529br|2053ar) 3529 (3530br|2253ar) 3530 (3531br|2443ar) 3531 (3532br|3273ar) 3532 (3533br|2113ar) 3533 (3534br|3744ar) 3534 (3535br|3498ar) 3535 (3536br|3189ar) 3536 (3537br|2916ar) 3537 (3538br|3140ar) 3538 (3539br|2859ar) 3539 (3540br|2724ar) 3540 (3541br|3544ar) 3541 (3542br|3442ar) 3542 (3543br|3444ar) 3543 (3544br|3444ar) 3544 (3545br|3444ar) 3545 (3546br|3444ar) 3546 (3547br|3444ar) 3547 (3548br|3444ar) 3548 (3549br|3444ar) 3549 (3550br|3444ar) 3550 (3551br|2617ar) 3551 (3552br|3079ar) 3552 (3553br|3079ar) 3553 (3554br|2946ar) 3554 (3555br|2884ar) 3555 (3556br|2884ar) 3556 (3557br|2439ar) 3557 (3558br|3140ar) 3558 (3559br|2859ar) 3559 (3560br|3728ar) 3560 (3561br|2587ar) 3561 (3562br|3394ar) 3562 (3563br|2779ar) 3563 (3564br|3742ar) 3564 (3565br|3742ar) 3565 (3566br|2444ar) 3566 (3567br|2663ar) 3567 (3568br|3258ar) 3568 (3569br|3739ar) 3569 (3570br|3738ar) 3570 (3571br|3571ar) 3571 (3572br|3748ar) 3572 (3573br|3748ar) 3573 (3574br|3748ar) 3574 (3575br|3748ar) 3575 (3576br|3748ar) 3576 (3577br|3748ar) 3577 (3578br|3748ar) 3578 (3579br|3748ar) 3579 (3580br|3748ar) 3580 (3581br|2108ar) 3581 (3582br|3562ar) 3582 (3583br|2940ar) 3583 (3584br|1951ar) 3584 (3585br|1951ar) 3585 (3586br|1951ar) 3586 (3587br|1951ar) 3587 (3588br|1951ar) 3588 (3589br|1951ar) 3589 (3590br|1951ar) 3590 (3591br|1927ar) 3591 (3592br|3838ar) 3592 (3593br|2940ar) 3593 (3594br|1975ar) 3594 (3595br|3067ar) 3595 (3596br|3438ar) 3596 (3597br|2916ar) 3597 (3598br|1952ar) 3598 (3599br|3291ar) 3599 (3600br|3354ar) 3600 (3601br|3035ar) 3601 (3602br|3748ar) 3602 (3603br|3748ar) 3603 (3604br|1903ar) 3604 (3605br|2125ar) 3605 (3606br|3233ar) 3606 (3607br|2600ar) 3607 (3608br|3561ar) 3608 (3609br|1929ar) 3609 (3610br|3257ar) 3610 (3611br|2587ar) 3611 (3612br|2037ar) 3612 (3613br|2652ar) 3613 (3614br|1968ar) 3614 (3615br|3196ar) 3615 (3616br|3077ar) 3616 (3617br|3077ar) 3617 (3618br|1957ar) 3618 (3619br|3291ar) 3619 (3620br|2237ar) 3620 (3621br|2859ar) 3621 (3622br|3434ar) 3622 (3623br|3478ar) 3623 (3624br|2161ar) 3624 (3625br|2689ar) 3625 (3626br|3001ar) 3626 (3627br|2639ar) 3627 (3628br|1968ar) 3628 (3629br|1929ar) 3629 (3630br|2934ar) 3630 (3631br|2949ar) 3631 (3632br|2243ar) 3632 (3633br|2600ar) 3633 (3634br|2481ar) 3634 (3635br|2438ar) 3635 (3636br|2438ar) 3636 (3637br|2438ar) 3637 (3638br|2227ar) 3638 (3639br|2071ar) 3639 (3640br|3028ar) 3640 (3641br|1924ar) 3641 (3642br|2929ar) 3642 (3643br|2023ar) 3643 (3644br|3287ar) 3644 (3645br|3748ar) 3645 (3646br|2438ar) 3646 (3647br|3748ar) 3647 (3648br|1951ar) 3648 (3649br|2459ar) 3649 (3650br|2137ar) 3650 (3651br|2659ar) 3651 (3652br|3422ar) 3652 (3653br|2940ar) 3653 (3654br|1968ar) 3654 (3655br|2884ar) 3655 (3656br|3394ar) 3656 (3657br|2438ar) 3657 (3658br|3027ar) 3658 (3659br|2780ar) 3659 (3660br|3190ar) 3660 (3661br|2618ar) 3661 (3662br|3422ar) 3662 (3663br|2940ar) 3663 (3664br|1968ar) 3664 (3665br|3203ar) 3665 (3666br|3272ar) 3666 (3667br|2789ar) 3667 (3668br|1903ar) 3668 (3669br|2071ar) 3669 (3670br|2251ar) 3670 (3671br|3035ar) 3671 (3672br|3272ar) 3672 (3673br|2919ar) 3673 (3674br|1997ar) 3674 (3675br|2820ar) 3675 (3676br|2993ar) 3676 (3677br|2057ar) 3677 (3678br|3252ar) 3678 (3679br|2859ar) 3679 (3680br|2292ar) 3680 (3681br|2857ar) 3681 (3682br|3186ar) 3682 (3683br|2789ar) 3683 (3684br|2237ar) 3684 (3685br|2780ar) 3685 (3686br|1903ar) 3686 (3687br|2021ar) 3687 (3688br|2226ar) 3688 (3689br|1942ar) 3689 (3690br|1965ar) 3690 (3691br|2108ar) 3691 (3692br|3698ar) 3692 (3693br|1929ar) 3693 (3694br|2161ar) 3694 (3695br|2852ar) 3695 (3696br|2852ar) 3696 (3697br|2692ar) 3697 (3698br|2929ar) 3698 (3699br|2057ar) 3699 (3700br|3002ar) 3700 (3701br|2637ar) 3701 (3702br|1981ar) 3702 (3703br|2651ar) 3703 (3704br|3434ar) 3704 (3705br|3068ar) 3705 (3706br|3068ar) 3706 (3707br|2617ar) 3707 (3708br|2481ar) 3708 (3709br|2934ar) 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|2949ar) 3725 (3726br|1969ar) 3726 (3727br|2441ar) 3727 (3728br|3294ar) 3728 (3729br|1892ar) 3729 (3730br|2934ar) 3730 (3731br|3591ar) 3731 (3732br|3838ar) 3732 (3733br|2649ar) 3733 (3734br|3422ar) 3734 (3735br|3748ar) 3735 (3736br|3748ar) 3736 (3737br|3748ar) 3737 (3738br|3748ar) 3738 (3739br|3748ar) 3739 (3740br|3838ar) 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|2593ar) 3749 (3750br|3838ar) 3750 (3751br|2917ar) 3751 (3752br|3594ar) 3752 (3753br|2660ar) 3753 (3754br|3027ar) 3754 (3755br|1899ar) 3755 (3756br|3487ar) 3756 (3757br|1929ar) 3757 (3758br|3397ar) 3758 (3759br|2449ar) 3759 (3760br|2239ar) 3760 (3761br|2820ar) 3761 (3762br|3833ar) 3762 (3763br|2396ar) 3763 (3764br|3221ar) 3764 (3765br|2858ar) 3765 (3766br|3439ar) 3766 (3767br|2443ar) 3767 (3768br|3354ar) 3768 (3769br|3737ar) 3769 (3770br|2912ar) 3770 (3771br|2663ar) 3771 (3772br|2266ar) 3772 (3773br|2949ar) 3773 (3774br|3006ar) 3774 (3775br|2884ar) 3775 (3776br|3667ar) 3776 (3777br|1931ar) 3777 (3778br|3838ar) 3778 (3779br|2951ar) 3779 (3780br|2370ar) 3780 (3781br|2057ar) 3781 (3782br|3549ar) 3782 (3783br|2949ar) 3783 (3784br|3748ar) 3784 (3785br|3748ar) 3785 (3786br|3748ar) 3786 (3787br|3748ar) 3787 (3788br|3748ar) 3788 (3789br|3748ar) 3789 (3790br|3748ar) 3790 (3791br|2652ar) 3791 (3792br|2259ar) 3792 (3793br|2587ar) 3793 (3794br|3703ar) 3794 (3795br|3048ar) 3795 (3796br|3001ar) 3796 (3797br|2618ar) 3797 (3798br|1961ar) 3798 (3799br|2851ar) 3799 (3800br|2160ar) 3800 (3801br|2584ar) 3801 (3802br|2881ar) 3802 (3803br|2076ar) 3803 (3804br|1905ar) 3804 (3805br|2652ar) 3805 (3806br|3562ar) 3806 (3807br|2533ar) 3807 (3808br|2229ar) 3808 (3809br|3098ar) 3809 (3810br|3423ar) 3810 (3811br|2059ar) 3811 (3812br|3591ar) 3812 (3813br|2535ar) 3813 (3814br|3044ar) 3814 (3815br|2353ar) 3815 (3816br|1978ar) 3816 (3817br|2053ar) 3817 (3818br|2053ar) 3818 (3819br|2053ar) 3819 (3820br|2230ar) 3820 (3821br|3748ar) 3821 (3822br|2780ar) 3822 (3823br|2780ar) 3823 (3824br|3799ar) 3824 (3825br|2118ar) 3825 (3826br|3140ar) 3826 (3827br|2916ar) 3827 (3828br|1911ar) 3828 (3829br|2053ar) 3829 (3830br|2253ar) 3830 (3831br|2819ar) 3831 (3832br|3273ar) 3832 (3833br|2113ar) 3833 (3834br|3744ar) 3834 (3835br|3498ar) 3835 (3836br|3189ar) 3836 (3837br|2916ar) 3837 (3838br|3140ar) 3838 (3839br|2859ar) 3839 (3840br|3728ar) 3840 (3841br|2587ar) 3841 (3842br|3394ar) 3842 (3843br|2779ar) 3843 (3844br|3742ar) 3844 (3845br|3742ar) 3845 (3846br|2444ar) 3846 (3847br|2663ar) 3847 (3848br|3258ar) 3848 (3849br|3739ar) 3849 (3850br|3738ar) 3850 (3851br|2108ar) 3851 (3852br|3562ar) 3852 (3853br|2940ar) 3853 (3854br|1951ar) 3854 (3855br|1951ar) 3855 (3856br|1951ar) 3856 (3857br|1951ar) 3857 (3858br|1951ar) 3858 (3859br|1951ar) 3859 (3860br|1951ar) 3860 (3861br|1927ar) 3861 (3862br|3838ar) 3862 (3863br|2940ar) 3863 (3864br|1975ar) 3864 (3865br|3067ar) 3865 (3866br|3438ar) 3866 (3867br|2916ar) 3867 (3868br|1952ar) 3868 (3869br|3291ar) 3869 (3870br|3354ar) 3870 (3871br|3035ar) 3871 (3872br|3748ar) 3872 (3873br|3748ar) 3873 (3874br|1903ar) 3874 (3875br|2125ar) 3875 (3876br|3233ar) 3876 (3877br|2600ar) 3877 (3878br|3561ar) 3878 (3879br|1929ar) 3879 (3880br|3257ar) 3880 (3881br|2587ar) 3881 (3882br|3562ar) 3882 (3883br|2940ar) 3883 (3884br|1951ar) 3884 (3885br|1951ar) 3885 (3886br|1951ar) 3886 (3887br|1951ar) 3887 (3888br|1951ar) 3888 (3889br|1951ar) 3889 (3890br|1951ar) 3890 (3891br|1927ar) 3891 (3892br|3838ar) 3892 (3893br|2940ar) 3893 (3894br|1975ar) 3894 (3895br|3067ar) 3895 (3896br|3438ar) 3896 (3897br|2916ar) 3897 (3898br|1952ar) 3898 (3899br|3291ar) 3899 (3900br|3354ar) 3900 (3901br|3035ar) 3901 (3902br|3748ar) 3902 (3903br|3748ar) 3903 (3904br|1903ar) 3904 (3905br|2125ar) 3905 (3906br|3233ar) 3906 (3907br|2600ar) 3907 (3908br|3561ar) 3908 (3909br|1929ar) 3909 (3910br|3257ar) 3910 (3911br|2587ar) 3911 (3912br|2037ar) 3912 (3913br|2652ar) 3913 (3914br|1968ar) 3914 (3915br|3196ar) 3915 (3916br|3077ar) 3916 (3917br|3077ar) 3917 (3918br|1957ar) 3918 (3919br|3291ar) 3919 (3920br|2237ar) 3920 (3921br|2859ar) 3921 (3922br|3434ar) 3922 (3923br|3478ar) 3923 (3924br|2161ar) 3924 (3925br|2689ar) 3925 (3926br|3001ar) 3926 (3927br|2639ar) 3927 (3928br|1968ar) 3928 (3929br|1929ar) 3929 (3930br|2934ar) 3930 (3931br|2949ar) 3931 (3932br|2243ar) 3932 (3933br|2600ar) 3933 (3934br|2481ar) 3934 (3935br|2438ar) 3935 (3936br|2438ar) 3936 (3937br|2438ar) 3937 (3938br|2227ar) 3938 (3939br|2071ar) 3939 (3940br|3028ar) 3940 (3941br|1924ar) 3941 (3942br|2929ar) 3942 (3943br|2023ar) 3943 (3944br|3287ar) 3944 (3945br|3748ar) 3945 (3946br|2438ar) 3946 (3947br|3748ar) 3947 (3948br|1951ar) 3948 (3949br|2459ar) 3949 (3950br|2137ar) 3950 (3951br|2659ar) 3951 (3952br|3422ar) 3952 (3953br|2940ar) 3953 (3954br|1968ar) 3954 (3955br|2884ar) 3955 (3956br|3394ar) 3956 (3957br|2438ar) 3957 (3958br|3027ar) 3958 (3959br|2780ar) 3959 (3960br|3190ar) 3960 (3961br|2618ar) 3961 (3962br|3422ar) 3962 (3963br|2940ar) 3963 (3964br|1968ar) 3964 (3965br|3203ar) 3965 (3966br|3272ar) 3966 (3967br|2789ar) 3967 (3968br|1903ar) 3968 (3969br|2071ar) 3969 (3970br|2251ar) 3970 (3971br|3035ar) 3971 (3972br|3272ar) 3972 (3973br|2919ar) 3973 (3974br|1997ar) 3974 (3975br|2820ar) 3975 (3976br|2993ar) 3976 (3977br|2057ar) 3977 (3978br|3252ar) 3978 (3979br|2859ar) 3979 (3980br|2292ar) 3980 (3981br|2857ar) 3981 (3982br|3186ar) 3982 (3983br|2789ar) 3983 (3984br|2237ar) 3984 (3985br|2780ar) 3985 (3986br|1903ar) 3986 (3987br|2021ar) 3987 (3988br|2226ar) 3988 (3989br|1942ar) 3989 (3990br|1965ar) 3990 (3991br|2108ar) 3991 (3992br|3698ar) 3992 (3993br|1929ar) 3993 (3994br|2161ar) 3994 (3995br|2852ar) 3995 (3996br|2852ar) 3996 (3997br|2692ar) 3997 (3998br|2929ar) 3998 (3999br|2057ar) 3999 (4000br|3002ar) 4000 (4001br|2637ar) 4001 (4002br|1981ar) 4002 (4003br|2651ar) 4003 (4004br|3434ar) 4004 (4005br|3068ar) 4005 (4006br|3068ar) 4006 (4007br|2617ar) 4007 (4008br|2481ar) 4008 (4009br|2934ar) 4009 (4010br|1911ar) 4010 (4011br|3081ar) 4011 (4012br|2238ar) 4012 (4013br|2859ar) 4013 (4014br|1978ar) 4014 (4015br|2073ar) 4015 (4016br|3397ar) 4016 (4017br|2023ar) 4017 (4018br|1981ar) 4018 (4019br|2067ar) 4019 (4020br|3411ar) 4020 (4021br|2057ar) 4021 (4022br|1977ar) 4022 (4023br|2640ar) 4023 (4024br|3546ar) 4024 (4025br|2949ar) 4025 (4026br|1969ar) 4026 (4027br|2441ar) 4027 (4028br|3294ar) 4028 (4029br|1892ar) 4029 (4030br|2934ar) 4030 (4031br|2934ar) 4031 (4032br|3838ar) 4032 (4033br|2649ar) 4033 (4034br|3422ar) 4034 (4035br|3748ar) 4035 (4036br|3748ar) 4036 (4037br|3748ar) 4037 (4038br|3748ar) 4038 (4039br|3748ar) 4039 (4040br|3748ar) 4040 (4041br|3041ar) 4041 (4042br|3446ar) 4042 (4043br|2652ar) 4043 (4044br|3266ar) 4044 (4045br|2441ar) 4045 (4046br|3535ar) 4046 (4047br|2620ar) 4047 (4048br|1905ar) 4048 (4049br|2593ar) 4049 (4050br|3838ar) 4050 (4051br|2917ar) 4051 (4052br|3594ar) 4052 (4053br|2660ar) 4053 (4054br|3027ar) 4054 (4055br|1899ar) 4055 (4056br|3487ar) 4056 (4057br|1929ar) 4057 (4058br|3397ar) 4058 (4059br|2449ar) 4059 (4060br|2239ar) 4060 (4061br|2820ar) 4061 (4062br|3833ar) 4062 (4063br|2396ar) 4063 (4064br|3221ar) 4064 (4065br|2858ar) 4065 (4066br|3439ar) 4066 (4067br|2443ar) 4067 (4068br|3354ar) 4068 (4069br|3737ar) 4069 (4070br|2912ar) 4070 (4071br|2663ar) 4071 (4072br|2266ar) 4072 (4073br|2949ar) 4073 (4074br|3006ar) 4074 (4075br|2884ar) 4075 (4076br|3667ar) 4076 (4077br|1931ar) 4077 (4078br|3838ar) 4078 (4079br|2951ar) 4079 (4080br|2370ar) 4080 (4081br|2057ar) 4081 (4082br|3549ar) 4082 (4083br|2949ar) 4083 (4084br|3748ar) 4084 (4085br|3748ar) 4085 (4086br|3748ar) 4086 (4087br|3748ar) 4087 (4088br|3748ar) 4088 (4089br|3748ar) 4089 (4090br|3748ar) 4090 (4091br|2652ar) 4091 (4092br|2259ar) 4092 (4093br|2587ar) 4093 (4094br|3703ar) 4094 (4095br|3048ar) 4095 (4096br|3001ar) 4096 (4097br|2618ar) 4097 (4098br|1961ar) 4098 (4099br|2851ar) 4099 (4100br|2160ar) 4100 (4101br|2584ar) 4101 (4102br|2881ar) 4102 (4103br|2076ar) 4103 (4104br|1905ar) 4104 (4105br|2652ar) 4105 (4106br|3562ar) 4106 (4107br|2533ar) 4107 (4108br|2229ar) 4108 (4109br|3098ar) 4109 (4110br|3423ar) 4110 (4111br|2059ar) 4111 (4112br|3591ar) 4112 (4113br|2535ar) 4113 (4114br|3044ar) 4114 (4115br|2353ar) 4115 (4116br|1978ar) 4116 (4117br|2053ar) 4117 (4118br|2053ar) 4118 (4119br|2053ar) 4119 (4120br|2230ar) 4120 (4121br|3748ar) 4121 (4122br|2780ar) 4122 (4123br|2780ar) 4123 (4124br|3799ar) 4124 (4125br|2118ar) 4125 (4126br|3140ar) 4126 (4127br|2916ar) 4127 (4128br|1911ar) 4128 (4129br|2053ar) 4129 (4130br|2253ar) 4130 (4131br|2819ar) 4131 (4132br|3273ar) 4132 (4133br|2113ar) 4133 (4134br|3744ar) 4134 (4135br|3498ar) 4135 (4136br|3189ar) 4136 (4137br|2916ar) 4137 (4138br|3140ar) 4138 (4139br|2859ar) 4139 (4140br|3728ar) 4140 (4141br|2587ar) 4141 (4142br|3394ar) 4142 (4143br|2779ar) 4143 (4144br|3742ar) 4144 (4145br|3742ar) 4145 (4146br|2444ar) 4146 (4147br|2663ar) 4147 (4148br|3258ar) 4148 (4149br|3739ar) 4149 (4150br|3738ar) 4150 (4151br|2108ar) 4151 (4152br|3562ar) 4152 (4153br|2940ar) 4153 (4154br|1951ar) 4154 (4155br|1951ar) 4155 (4156br|1951ar) 4156 (4157br|1951ar) 4157 (4158br|1951ar) 4158 (4159br|1951ar) 4159 (4160br|1951ar) 4160 (4161br|1927ar) 4161 (4162br|3838ar) 4162 (4163br|2940ar) 4163 (4164br|1975ar) 4164 (4165br|3067ar) 4165 (4166br|3438ar) 4166 (4167br|2916ar) 4167 (4168br|1952ar) 4168 (4169br|3291ar) 4169 (4170br|3354ar) 4170 (4171br|3035ar) 4171 (4172br|3748ar) 4172 (4173br|3748ar) 4173 (4174br|1903ar) 4174 (4175br|2125ar) 4175 (4176br|3233ar) 4176 (4177

30

31