Counting Flimsy Numbers via Formal Language Theory

by

Trevor William Alexander Clokie

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Let $s_2(n)$ be the sum of the digits of n when expressed in base 2. For integers n and k, Stolarsky defined n to be k-flimsy if $s_2(kn) < s_2(n)$.

In this paper, we generalize the definition of k-flimsy numbers to all bases b, and provide a method to construct a pushdown automaton recognizing the k-flimsy base-b numbers. Using the tools of context-free languages and analytic combinatorics, we use this automaton to determine precise asymptotics for the number of k-flimsy N-digit numbers in base b. Lastly, using the results we obtained, we discuss the natural densities of k-flimsy numbers in base b for all values k and b.

Our main results can be found in Theorems 2, 3, 8, and 9.

Acknowledgements

I would like to thank my supervisor Jeffrey Shallit for his guidance, mentorship, and patience. I would also like to thank Leon Witzman who helped me formulate the first working 3-flimsy binary PDA. I would also like to thank Rafael Oliveira and Eric Schost for reading this thesis and providing valuable feedback. Lastly, I would like to thank my family and friends for their endless support.

Table of Contents

List of Figures

Chapter 1

Introduction

The focus of this thesis is to study the k-flimsy numbers, which were first studied (independently) by Kátai in 1977 [15] and Stolarsky in 1980 [23] using techniques of number theory. In this thesis, we study these same numbers, but using the tools of formal language theory instead. In particular, we show that results about context-free languages provide a profound amount of insight into the flimsy numbers.

This difference in approach allows us to compute far more detailed approximations of the number of N-digit flimsy numbers than has been previously done. It also allows us to easily study the k-flimsy numbers in an arbitrary base b for any integers k and $b \geq 2$. We will, however, also discuss the limitations of this method, as while it works easily in theory, in practice the method seems too computationally demanding to obtain results for most values of k and b within a reasonable amount of time and computer memory.

In this thesis we will also explore modifying our technique to tackle related problems, such as the k-equal numbers, defined and described in Section 4.2. In general, our method provides insight into any set of natural numbers whose base-b representations form an unambiguous context-free language.

Our main results are Theorems 2, 3, 8, and 9, along with their accompanying Remarks 4 and 10. Theorems 2, 3 and 8 appeared in [10].

1.1 The k-flimsy numbers

We start by defining the function $s_b(n)$ to be the sum of the digits of an integer n, when represented in base b. For instance $s_{10}(16) = 7$ and $s_2(16) = 1$.

The representation of a number n in base b, as a string of symbols, is denoted by $(n)_b$. We extend this notation to include sets, so for a set S of numbers we define $(S)_b = \{(n)_b : n \in S\}$.

In a 1980 paper, Stolarsky[23] called an integer n *sturdy* if $s_2(n) \leq s_2(kn)$ for all positive integers k. Conversely, if there exists k such that $s_2(n) > s_2(kn)$, then n is *k-flimsy*. If n is k-flimsy for some k then n is called *flimsy*.

In this thesis, we extend the definition of k-flimsy numbers to base b. For example, 7 is 3-flimsy in base-10, since $s_{10}(7) = 7$, which is greater than $s_{10}(21) = 3$. However, 7 is not 3-flimsy in base-2, since $s_2(7) = 3$ and $s_2(21) = 3$.

With this expanded definition of flimsy numbers, we define the set of k-flimsy numbers in base-b to be

$$F_{k,b} = \{n \in \mathbb{Z}_{\geq 0} \mid s_b(n) > s_b(kn)\}.$$

We are interested in computing the density of this set.

1.1.1 Some trivial results

When k is an even number, we write $k = 2\ell$. We observe that, for a number n, we have $(kn)_2 = (\ell n)_2 \cdot 0$ where $\cdot$ is the string concatenation operator. Therefore $s_2(kn) = s_2(\ell n)+0$, so n is k-flimsy if and only if n is ℓ-flimsy. Thus it suffices to limit our search to odd k.

In general base b, we note that $s_b((bk)n) = s_b(kn)$, so it is unnecessary to consider values of k where k is a multiple of b.

When k is a power of b (that is, $k = b^m$ where m is a non-negative integer) then we have $s_b(kn) = s_b(n)$, so no numbers are b^m-flimsy in base b.

Additionally, if k is negative, then $s_b(kn) = s_b(-kn)$.

Finally, if $k = 0$ then $s_b(kn) = 0 \leq s_b(n)$ for all n, so all non-zero numbers are 0-flimsy in any base b.

1.1.2 Prior work

The analysis of the sums of binary digits of multiples was first studied in 1977 by Kátai [15] who studied the 3-flimsy binary numbers and found that as $N \to \infty$, the distribution of $s_2(3n) - s_2(n)$ over $0 \leq n < 2^N$ approaches a normal distribution with a mean of 0 and a variance of $\sqrt{N/3}$. That is, for $a \in \mathbb{Z}$, we have

$$2^{-N} \left|\{n \in \mathbb{Z} \ : \ 0 \leq n < 2^N, s_2(3n) - s_2(n) = a\}\right| \to \frac{\sqrt{3}}{\sqrt{2\pi N}} e^{-3a^2/2N}$$

as $N \to \infty$.

Following that, Stolarsky published a paper in 1980 [23] in which the terms *sturdy number* and *flimsy number* are coined. In this paper, Stolarsky proved that the number of $(N+1)$-bit binary numbers n for which $s_2(3n)-s_2(n) = a$ is asymptotic to $2^N \frac{\sqrt{3}}{\sqrt{2\pi N}} e^{-3a^2/2N}$, which can be shown to be equivalent to Kátai's result.

In a 1983 paper, Schmidt [21] examined Stolarsky's result that the number of k-flimsy integers in the interval $[2^N, 2^{N+1})$ is $2^N\left(\frac{1}{2} + o(1)\right)$ where $k = 3$. Schmidt, using Markov chains, extended this result to all odd $k \geq 3$. In a paper published the following year, Schmid [20] improved on the error term found by Schmidt, using linear algebra and spectral analysis.

A 2017 paper by Bašic [6] provides a proof that for all integers $b \geq 2$ and $k \neq b^m$, there exists a k-flimsy number n in base b. This paper also provides an algorithm to construct such a number, although the complexity of this algorithm is nontrivial to analyze.

A 2014 survey paper by Chen, Hwang, and Zacharovas [8] discusses the various approaches and results in the number theory literature regarding functions that sum the digits of numbers, like s_b. The survey includes the results listed above, except for the paper by Bašic which was not yet published. Most of these results are not highly relevant to the work presented in this thesis, but the survey is an excellent source of further reading for readers interested in related problems.

1.2 Generating series and Context-free languages

In this work, we use tools of context-free languages to obtain estimates for the size of $F_{k,b} \cap \left[b^{N-1}, b^N\right)$. That is, we determine how many k-flimsy numbers can be written in precisely N digits in base b with no leading zeros.

1.2.1 Defining context-free languages

In formal language theory, a *word* (also called a *string*) is a finite ordered sequence of symbols selected from a finite set of symbols Σ, which we call an *alphabet*. A *language* is a set of words over a given alphabet. The *length* of a word w is the number of symbols in w, and is written as $|w|$. The word of length 0, often called the *empty word*, is denoted ϵ.

A word w raised to an integer power n denotes the word w repeated n times; for instance $(mur)^2 = murmur$. Note that $w^0 = \epsilon$ for all words w. For a word w, we define $w^* = \{w^n \ : \ n \geq 0\} = \{\epsilon, w, w^2, w^3, \ldots\}$. Similarly, for an alphabet Σ we define Σ^* to be the set of all words over Σ, so $\Sigma^* = \{a_1 a_2 \cdots a_n \ : \ a_i \in \Sigma, 1 \leq i \leq n, n \geq 0\}$ [14, § 1.1].

A *context-free grammar* (or CFG) is a set of variables V and a set of production rules to generate words over an alphabet Σ. A production rule is of the form $A \to \alpha_1\alpha_2 \cdots \alpha_k$, for some k, where $\alpha_1, \alpha_2, \ldots, \alpha_k \in V \cup \Sigma$ are either variables or symbols. One variable $S \in V$ is designated to be the starting variable. One begins with a sequence S, and iteratively replaces variables A in the sequence with the right-hand side of a production rule until no variables remain. The sequence of symbols that results is a word generated by the CFG [14, § 4.2]. A *derivation* of a word w generated by a CFG is a series of production

steps, beginning with S, and ending with w. The set of words generated by a CFG G is called a *context-free language* (or CFL), which is denoted $L(G)$.

For example, the CFG consisting of variables $V = \{S, T\}$, the alphabet $\Sigma = \{a, b, c\}$, and production rules $S \to T$, $S \to aSc$, $T \to bT$, and $T \to \epsilon$ generates the language $\{a^m b^n c^m \ : \ m, n \geq 0\}$. To generate the word $aabcc$, one follows the derivation $S \Longrightarrow aSc \Longrightarrow aaScc \Longrightarrow aaTcc \Longrightarrow aabTcc \Longrightarrow aabcc$.

1.2.2 Defining automata

An *automaton* is a tool for describing a language. An automaton processes an input word one symbol at a time, from left to right, and either accepts or rejects it upon reading every symbol in the word. If an automaton M accepts a word w, then w is in the language recognized by M, which we denote $w \in L(M)$.

Informally, an automaton is a finite set of states and a set of transitions from one state to another under various conditions, such as reading an input symbol. Each state is either accepting or rejecting, and not all transitions are deterministic; in the case where multiple transitions can be taken at a given time, so long as any series of choices could lead to the automaton accepting the input word, we say that the automaton accepts the word and that the automaton is non-deterministic.

A *pushdown automaton* (or PDA) is an automaton with an unbounded stack onto which it can push symbols. Transitions of a PDA may depend on what is on top of the stack, as well as the symbol being read (or no symbol being read, which we denote by reading the empty word ϵ; we call these ϵ-transitions). Traditionally, the stack starts with one symbol on it, and the PDA accepts if the stack is empty; although a PDA may have accepting states as well. Note: when the stack is expressed horizontally, the top of the stack is traditionally denoted as the left end.

Formally, a PDA M is the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, with each term defined as follows. We define Q to be the set of states, which must be finite, and $F \subseteq Q$ is the set of accepting states (also known as final states), however for the purposes of this thesis we will always use $F = \emptyset$ and accept via empty stack. We define Σ as the alphabet of valid input symbols (and hence $L(M) \subseteq \Sigma^*$), and Γ as the alphabet of valid symbols that may be pushed onto the stack. We define δ to be the (finite) set of transitions, which is a multi-function from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$ [14, § 5.2]. We define $q_0 \in Q$ to be the start state; which is the state M is in before performing any transitions. Lastly, we define $Z \in \Gamma$ to be the start symbol, which is the only symbol on the stack before any transitions are performed.

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ that recognizes the language $\{a^m b^n c^m \ : \ m, n \geq 0\}$ can be constructed as follows: $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{Z, a\}$, $F = \emptyset$, and δ

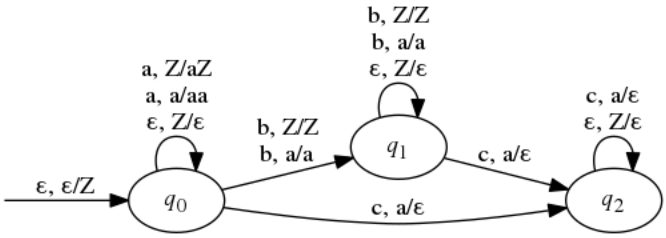

Figure 1.1: The Unambiguous PDA M recognizes the language $\{a^m b^n c^m \ : \ m, n \geq 0\}$

consists of the following transitions:

$$\begin{aligned}
\delta(q_0, a, Z) &= \{(q_0, aZ)\} & \delta(q_1, b, Z) &= \{(q_1, Z)\} \\
\delta(q_0, b, Z) &= \{(q_1, Z)\} & \delta(q_1, b, a) &= \{(q_1, a)\} \\
\delta(q_0, a, a) &= \{(q_0, aa)\} & \delta(q_1, c, a) &= \{(q_2, \epsilon)\} \\
\delta(q_0, b, a) &= \{(q_1, a)\} & \delta(q_1, \epsilon, Z) &= \{(q_1, \epsilon)\} \\
\delta(q_0, \epsilon, Z) &= \{(q_0, \epsilon)\} & \delta(q_2, c, a) &= \{(q_2, \epsilon)\} \\
\delta(q_0, c, a) &= \{(q_2, \epsilon)\} & \delta(q_2, \epsilon, Z) &= \{(q_2, \epsilon)\}
\end{aligned}$$

This, of course, is not easy for people to interpret, so to make automata more human-readable, we use *transition diagrams*, which are directed graphs displaying the states as nodes and the transitions as edges. The transition diagram for the automaton above is shown in Figure 1.1.

Lastly, a *deterministic pushdown automaton* (or DPDA) is a pushdown automaton in which there is at most one possible transition to be performed at a given step. Formally, a PDA $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ is a DPDA when for all $q \in Q, z \in \Gamma$, we have no more than one transition from (q, a, z) for all $a \in \Sigma \cup \epsilon$, and if there exists a transition from (q, ϵ, z) then there must exist no transitions from (q, a, z) for all $a \in \Sigma$ [14, pp. 112-113]. A language recognized by a DPDA is a *deterministic context-free language* (or DCFL).

1.2.3 Unambiguous context-free languages

A context-free grammar G is said to be *unambiguous* [13, p. 27] if every word in $L(G)$ has no more than one leftmost derivation in G, and a context-free language L is said to be *unambiguous* if $L = L(G)$ for an unambiguous context-free grammar G.

Similarly, we can define a PDA M to be *unambiguous* [13, p. 142] if for every word $w \in L(M)$, there is exactly one sequence of transitions by which M accepts w, stack pushes, and stack pops that leads to acceptance, either by reaching an accepting state or by emptying the stack. It is important to note that if M is an unambiguous PDA, then there exists an unambiguous CFG G for which $L(M) = L(G)$.

The unambiguous PDA is an important tool, as deterministic PDAs (which are a strict subclass of unambiguous PDAs) cannot recognize certain context-free languages that unambiguous PDAs can. For example, the language of palindromes over $\{a, b\}$ requires non-determinism to determine the midpoint of a given palindrome, so it cannot be recognized with a DPDA; but an unambiguous PDA can still recognize the language since only guessing the correct midpoint can lead to acceptance, and every palindrome has exactly one midpoint.

However, not all CFLs are unambiguous. For instance, the context-free language $\{a^m b^m c^n d^n \mid m, n \geq 1\} \cup \{a^m b^n c^n d^n \mid m, n \geq 1\}$ cannot be generated by an unambiguous CFG [14, Thm 4.7, pp. 100–103]. Therefore, it cannot be recognized by an unambiguous PDA.

1.2.4 Generating series

Pushdown automata and context-free grammars have been used for a wide variety of combinatorial enumerations; see, for example, [4, 5, 2, 3]. This is typically done using generating series.

For a set S and a "weight" function $w : S \to \mathbb{Z}_{\geq 0}$, where $\mathbb{Z}_{\geq 0}$ is the set of non-negative integers, we define the *generating series* of S to be the power series

$$\Phi_S(x) = \sum_{s \in S} x^{w(s)}.$$

For example, if $S = \mathbb{Z}$ and $w(n) = n^2$ for all $n \in \mathbb{Z}$, then $\Phi_{\mathbb{Z}}(x) = 1+2x+2x^4+2x^9+\cdots$.

We define the *coefficient extraction operation* $[x^n]\Phi_S(x)$ to be coefficient corresponding to x^n in the power series $\Phi_S(x)$. For example $[x^2](x^3 - 5x^2 + 4x - 3) = -5$.

If, for all $n \geq 0$, there exist finitely many $s \in S$ such that $w(s) = n$, then it follows that

$$\Phi_S(x) = \sum_{n=0}^{\infty} |\{s \in S \mid w(s) = n\}| x^n.$$

In this case $[x^n]\Phi_S(x)$ is the number of elements in S with weight n. For this reason, analyzing a generating function Φ_S can reveal combinatorial properties of our set S.

For a language L, it is common practice to define the weight of a word to be its length. That is $w(s) = |s|$ for all $s \in S$. For the rest of this thesis, we will follow this convention, and we will not mention weight functions again.

1.2.5 The Chomsky-Schützenberger Theorem

A power series $\Phi(x)$ is called *algebraic* over a field F if there exists a finite set of polynomials $p_0(x), p_1(x), \ldots, p_n(x) \in F[x]$ such that $p_n(x)\Phi^n(x) + \cdots + p_1(x)\Phi(x) + p_0(x) = 0$.

There is a theorem of Chomsky and Schützenberger [9], proven in [16, 18], that states the following:

Theorem 1. *If L is an unambiguous context-free language, then the generating series Φ_L is algebraic over $\mathbb{Q}$.*

In particular, given an unambiguous context-free grammar G, we can easily find a system of equations that describes the generating series $\Phi_{L(G)}$. The details of this process are described in Section 2.1.

1.2.6 Gröbner bases and Buchberger's algorithm

Given a system of n linearly independent polynomial equations over n generating series $\Phi_1, \Phi_2, \ldots, \Phi_n$, one can solve for each of the series Φ_m (where $1 \leq m \leq n$) in terms of polynomial equations over Φ_1 to Φ_m. Such a solution is called a Gröbner basis [7]. In particular, we can designate any of the n generating series to be Φ_1, and thereby solve for any of the generating series without reference to any of the other $n-1$ series.

Buchberger's algorithm [7] is an algorithm to compute the Gröbner basis for a such a system. In this thesis, we will be using an implementation of Buchberger's algorithm included in the `Maple` programming language [17].

1.2.7 Analytic combinatorics

Analytic combinatorics is the field formed by combining the fields of complex analysis and combinatorial enumeration. It provides very useful methods for determining the asymptotic nature of combinatorial processes. In particular, singularity analysis provides methods for taking a formal power series that is algebraic over $\mathbb{Q}$ and determining the asymptotic behaviour of its coefficients [11, § VII. 7.1].

An implementation of this method in the `Maple` programming language is provided in a library called `algolib` [19] by former University of Waterloo professor Bruno Salvy. Specifically, the `algolib` library contains a package called `gdev` which uses the saddle-point method [11, Thm. VIII. 3] to approximate the coefficients.

Chapter 2

The Method

2.1 Outline of the method

In this section, we describe a method for determining the number of words of length N in an unambiguous CFL L. To do this, we follow the algorithm:

1. We construct an unambiguous PDA M for which $L(M) = L$.
2. We can then convert M into an equivalent CFG G, where G is unambiguous and $L(G) = L$. (Note that we can skip step 1 if we can construct G directly.) We do this using a standard technique called the "triple construction." [14, pp. 115–119] We note that performing the triple construction on an unambiguous PDA M gives us an unambiguous grammar G [13, Thm. 5.4.3, p. 151].
3. We simplify G by removing useless variables. That is, variables that do not produce any all-terminal strings, or variables that are not reachable by the start state). The algorithm to do this is provided in [14, pp. 88–90]. Additionally, we propose three rules to further simplify our grammar:
   (a) If non-starting variable A has exactly one production rule $A \to \alpha$, then since α does not contain A (since if it did, it would have been removed as a useless variable) we can replace all instances of A with α and remove A from our set of variables.
   (b) If the start variable S has exactly one production rule $S \to A$ for some variable A, then we can replace all instances of A with S.
   (c) We simplify any production rule $A \to \alpha\epsilon\beta$ to $A \to \alpha\beta$. That is, in a production where an ϵ is specified alongside a variable or terminal, we do not need to specify the ϵ, so we can remove it from our representation of the production.

4. We convert our simplified grammar G to a system of equations. To accomplish this, we replace each variable A with the generating series $A(x)$ for which the N^{th} coefficient $[x^N]A(x)$ is the number of terminal strings of length N produced by variable A. To make this work, we replace every $|$ with $+$, every empty string ϵ with 1, every terminal 0 or 1 with x, string concatenation with multiplication, and $\to$ with $=$. This technique was introduced in [9] and proven to be correct in [16, 18].

5. We solve our system of equations using Buchberger's algorithm [7] to obtain an equation for $S(x)$ in terms of x, where S corresponds to the start variable of G. We note that $S(x)$ is the generating series for L.

6. We use methods of analytic combinatorics [11, 19] to obtain estimates for the N^{th} coefficient in the generating series of L. This is, therefore, the number of words in L of length N.

2.2 An example: binary palindromes

Consider the binary palindrome language $L_P = \{w \in \{0,1\}^* \mid w = w^R\}$. Note that $L = \{\epsilon, 0, 1, 00, 11, 000, 010, 101, 111, \ldots\}$. We know that this is an unambiguous CFL, as it is produced by the grammar $S \to \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$ which is unambiguous, even though it cannot be recognized by a DPDA. We also know that L contains exactly $2^{\lceil N/2 \rceil}$ words of length N.

2.2.1 An unambiguous PDA M_P

We construct the PDA M_P (see Figure 2.1) to recognize L_P. It works by pushing input onto the stack, non-deterministically guessing where the middle of the word is, then confirming by comparing each new symbol read to the top of the stack, then popping. We note that M_P is unambiguous since each palindrome has exactly one midpoint, and if it is not correctly guessed, then M_P will not accept the word (if it guesses too early, then M_P will stop reading the input before it finishes reading the word, so it cannot accept it; if it guesses too late, then it will not be able to empty the stack). Since the only non-deterministic step is how to guess the midpoint, there is exactly one accepting sequence of transitions through M_P for each palindrome $p \in L_P$.

2.2.2 Converting M_P to an unambiguous CFG G_P

Using the triple construction, we obtain the CFG G_P, seen in Figure 2.2.

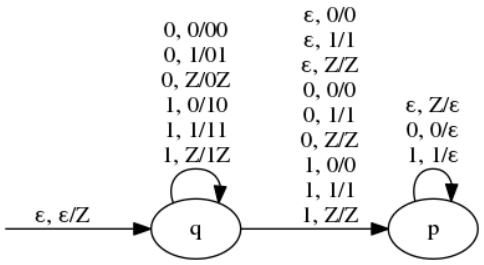

Figure 2.1: The Unambiguous PDA M_P recognizes the language of binary palindromes

$$
\begin{aligned}
S &\to (q, Z, q) \mid (q, Z, p) \\
(q, 0, q) &\to \epsilon(p, 0, q) \mid 0(q, 0, q)(q, 0, q) \mid 0(q, 0, p)(p, 0, q) \mid 0(p, 0, q) \mid 1(q, 1, q)(q, 0, q) \mid 1(q, 1, p)(p, 0, q) \mid 1(p, 0, q) \\
(q, 0, p) &\to \epsilon(p, 0, p) \mid 0(q, 0, q)(q, 0, p) \mid 0(q, 0, p)(p, 0, p) \mid 0(p, 0, p) \mid 1(q, 1, q)(q, 0, p) \mid 1(q, 1, p)(p, 0, p) \mid 1(p, 0, p) \\
(q, 1, q) &\to \epsilon(p, 1, q) \mid 0(q, 0, q)(q, 1, q) \mid 0(q, 0, p)(p, 1, q) \mid 0(p, 1, q) \mid 1(q, 1, q)(q, 1, q) \mid 1(q, 1, p)(p, 1, q) \mid 1(p, 1, q) \\
(q, 1, p) &\to \epsilon(p, 1, p) \mid 0(q, 0, q)(q, 1, p) \mid 0(q, 0, p)(p, 1, p) \mid 0(p, 1, p) \mid 1(q, 1, q)(q, 1, p) \mid 1(q, 1, p)(p, 1, p) \mid 1(p, 1, p) \\
(q, Z, q) &\to \epsilon(p, Z, q) \mid 0(q, 0, q)(q, Z, q) \mid 0(q, 0, p)(p, Z, q) \mid 0(p, Z, q) \mid 1(q, 1, q)(q, Z, q) \mid 1(q, 1, p)(p, Z, q) \mid 1(p, Z, q) \\
(q, Z, p) &\to \epsilon(p, Z, p) \mid 0(q, 0, q)(q, Z, p) \mid 0(q, 0, p)(p, Z, p) \mid 0(p, Z, p) \mid 1(q, 1, q)(q, Z, p) \mid 1(q, 1, p)(p, Z, p) \mid 1(p, Z, p) \\
(p, 0, q) & \\
(p, 0, p) &\to 0 \\
(p, 1, q) & \\
(p, 1, p) &\to 1 \\
(p, Z, q) & \\
(p, Z, p) &\to \epsilon
\end{aligned}
$$

Figure 2.2: Unambiguous CFG G_P producing the binary palindrome language L_P, produced by triple construction, unsimplified

$$
\begin{aligned}
S &\to (q,Z,p)\\
(q,0,p) &\to \epsilon(p,0,p) \mid 0(q,0,p)(p,0,p) \mid 0(p,0,p) \mid 1(q,1,p)(p,0,p) \mid 1(p,0,p)\\
(q,1,p) &\to \epsilon(p,1,p) \mid 0(q,0,p)(p,1,p) \mid 0(p,1,p) \mid 1(q,1,p)(p,1,p) \mid 1(p,1,p)\\
(q,Z,p) &\to \epsilon(p,Z,p) \mid 0(q,0,p)(p,Z,p) \mid 0(p,Z,p) \mid 1(q,1,p)(p,Z,p) \mid 1(p,Z,p)\\
(p,0,p) &\to 0\\
(p,1,p) &\to 1\\
(p,Z,p) &\to \epsilon
\end{aligned}
$$

Figure 2.3: Unambiguous CFG for L_P with no useless variables

$$
\begin{aligned}
S &\to \epsilon \mid 0 \mid 1 \mid 0A \mid 1B\\
A &\to 0 \mid 00 \mid 10 \mid 0A0 \mid 1B0\\
B &\to 1 \mid 01 \mid 11 \mid 0A1 \mid 1B1
\end{aligned}
$$

Figure 2.4: Significantly simplified unambiguous CFG G'_P for L_P

2.2.3 Tidying up the CFG G

Using the instructions described in step 3, we can identify useless variables $(q,0,q)$, $(q,1,q)$, (q,Z,q), $(p,0,q)$, $(p,1,q)$, and (p,Z,q) to obtain Figure 2.3.

Then, we can further simplify using our additional simplification rules (3a, 3b, and 3c), and rename variables $(q,0,p)$ and $(q,1,p)$ to A and B respectively, to obtain Figure 2.4.

2.2.4 Converting G to a system of equations

Following our steps, we replace each variable A with the generating series $A(x)$ for which the N^{th} coefficient $[x^N]A(x)$ is the number of terminal strings of length N produced by variable A. From this procedure we obtain the system shown in Figure 2.5.

$$
\begin{aligned}
S &= 1 + 2x + xA + xB\\
A &= x + 2x^2 + x^2A + x^2B\\
B &= x + 2x^2 + x^2A + x^2B
\end{aligned}
$$

Figure 2.5: System of equations of generating series derived from G'_P, simplified

Note that $S(x)$, where S is our start variable, corresponds to the generating series of our language L. Since G was unambiguous, this system is accurate and algebraic over $\mathbb{Q}$ [9, 18, 12].

2.2.5 Solving the system

Since the system is algebraic over $\mathbb{Q}$, we can find the Gröbner basis and solve for $S(x)$ using Buchberger's algorithm [7]. We do this using the `Maple` code below.

```
eqs := [-S + 1 + 2*x + x*A + x*B,
        -A + x + 2*x^2 + x^2*A + x^2*B,
        -B + x + 2*x^2 + x^2*A + x^2*B]:
algeq := Groebner[Basis](eqs, lexdeg([A, B], [S]))[1]:
assume(x, positive):
assume(N, integer):
f := solve(algeq, S);
```

This gives us the analytic solution $S(x) = (1 + 2x)/(1 - 2x^2)$.

To derive the asymptotic behaviour of the coefficients of the power series associated with $S(x)$, we use the methods of singularity analysis [11, § VII. 7.1]. Bruno Salvy's `gdev` package [19] does this for us. To use it, we append the following `Maple` code to the code above and run it in a folder containing the `algolib` files.

```
libname := ".",libname:
combine(equivalent(f, x, N, 8));
```

When we run this, we obtain the following asymptotic results for the number of words of length N.

$$2^{N/2} \cdot \frac{1}{2}\left(1 + \sqrt{2} + (-1)^N(1 - \sqrt{2})\right) + O\left(2^{N/2}N^{-12}\right)$$

For now we ignore the error term $O\left(2^{N/2}N^{-12}\right)$. For odd N this expression simplifies to $2^{N/2} \cdot \frac{1}{2}\left(1 + \sqrt{2} - 1 + \sqrt{2})\right) = 2^{(N+1)/2}$, and for even N it simplifies to $2^{N/2} \cdot \frac{1}{2}\left(1 + \sqrt{2} + 1 - \sqrt{2})\right) = 2^{N/2}$. In both cases, we observe that this is equal to $2^{\lceil N/2 \rceil}$, which is the correct answer, so the asymptotic is correct and the error term is zero for all positive integers N.

2.3 Another example: the Dyck language

The language of Dyck words [1, p. 333] is the language of balanced parentheses. (Note that all Dyck words have even length.) It is recognized by the unambiguous CFG $S \to \epsilon \mid (S)S$. From this we follow the same steps as above (without the PDA part) compute the system of one equation: $S = 1 + x^2S^2$.

```
eqs := [-S + 1 + x^2 * S^2]:
algeq := Groebner[Basis](eqs, lexdeg([S]))[1]:
assume(x, positive):
assume(N, integer):
additionally(N, even):
f := solve(algeq, S)[2]:
libname := ".",libname:
simplify(combine(equivalent(f, x, N, 1)));
```

This code outputs $2^{N+3/2}N^{-3/2}\pi^{-1/2} + O\left(2^N N^{-5/2}\right)$. Now, it is known that the number of length-N Dyck words, where N is even, is $C_{N/2}$ where C_n is the n^{th} Catalan number[1, pp. 333–338]. The Catalan numbers are exactly given by the expression $C_n = \frac{1}{n+1}\binom{2n}{n}$. It is a well-known result by Stirling that the Catalan numbers are asymptotic to $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$ [11, p. 38] so $C_{N/2} \sim \frac{2^N \cdot 2^{3/2}}{N^{3/2}\sqrt{\pi}}$ which is exactly what we find in our `Maple` output (when we drop the error term).

Chapter 3

Solving for k-Flimsy Binary Numbers

Note: much of this chapter is taken verbatim from [10].

3.1 Overview

We can study k-flimsy binary numbers using the method described in Chapter 2. In this section, we will walk through the analysis of k-flimsy numbers in base 2. In particular, we obtain our main results:

Theorem 2. *The number of* 3*-flimsy numbers in base 2 in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(\frac{1}{4} - cN^{-1/2} + O\left(N^{-3/2}\right) \right), \tag{3.1}$$

where $c = \frac{7\sqrt{6}}{24\sqrt{\pi}} \doteq 0.4030765$.

and

Theorem 3. *The number of* 5*-flimsy numbers in base 2 in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(\frac{1}{4} - cN^{-1/2} + O\left(N^{-3/2}\right) \right), \tag{3.2}$$

where $c = \frac{3\sqrt{5}}{8\sqrt{\pi}} \doteq 0.4730874$.

We note that the integers in the interval $[2^{N-1}, 2^N)$ are the integers that require exactly N bits to write in binary (with no leading zeros), so Theorem 2 serves as a proof of of Stolarsky's result [23] that $|F_{3,2} \cap [2^{N-1}, 2^N)| \sim \frac{1}{4}2^N$ as $N \to \infty$.

Remark 4. What makes our result stronger is that we are able to provide more accurate estimates because, with our technique, we can compute more terms in the asymptotic expansion. In particular, we find that the number of 3-flimsy binary numbers in the interval $[2^{N-1}, 2^N)$ is

$$2^N \left(\frac{1}{4} + \frac{\sqrt{6}}{\sqrt{\pi}} \left(-\frac{7}{24} N^{-1/2} + \frac{13}{72} N^{-3/2} - \frac{17}{64} N^{-5/2} + \frac{3365}{13824} N^{-7/2} + \cdots \right) \right)$$

and the number of 5-flimsy binary numbers in the interval $[2^{N-1}, 2^N)$ is

$$2^N \left(\frac{1}{4} + \frac{\sqrt{5}}{\sqrt{\pi}} \left(-\frac{3}{8} N^{-1/2} + \frac{799}{1920} N^{-3/2} - \frac{16623}{20480} N^{-5/2} + \frac{7343297}{4915200} N^{-7/2} + \cdots \right) \right).$$

3.2 Building the PDA for $(F_{k,2})_2^R$

The general idea is as follows: we create a PDA accepting the base-2 representation of k-flimsy numbers n, read from least-to-most significant bit. We use the stack of the PDA to record the absolute value of $s_2(n) - s_2(kn)$, and we use the state to record both the carry needed when multiplying input by k, and the sign of $s_2(n) - s_2(kn)$ (since the stack cannot have negatively many counters). We accept the input if the carry is 0, the sign of $s_2(n) - s_2(kn)$ is positive, and the stack has at least one counter.

Our PDA is designed to begin its computation with a special start symbol, `Z`, on top of the stack, and if the input is accepted, to end its computation when the stack becomes empty. The symbol `Z` exists solely as an indicator that the stack is otherwise empty, and is popped if and only if the input is accepted.

The sketch above is not quite sufficient because of two technical issues. First, (a) in some cases this approach requires reading extra leading zeroes since kn has more bits than n (which, because we are representing numbers starting with the least significant digit first, would be at the end of the input) and (b) we must have that the leading bit of the input is 1, to avoid incorrectly counting smaller numbers as having n bits (for example, the input `11010` should be ignored since it is equal to the input `1101` and we do not want to count it as both a 4-bit and 5-bit number).

To handle both these issues, we slightly modify the construction in several ways. First, if the state has a negative sign, then the stack holds $|y|_1 - |x|_1$ counters (for which we use the stack symbol `X`), where x is the input seen so far and y is the $|x|$ least significant bits of $k(x)_2^R$. On the other hand, if the state has a positive sign, then the stack holds $|x|_1 - |y|_1 - 1$ counters. By reducing the stack height of the positive states, we get that the PDA will always be in the the negative state when $|x|_1 = |y|_1$.

Second, to simulate the needed leading zeroes required to handle the carry, without actually reading them, we use a special series of $\log_2 k$ states to pop counters from the stack; thus guaranteeing that the stack has a sufficient number of counters.

Finally, we have a special state called `END` used to empty the stack when acceptance is detected. We need this, as opposed to a final state, because applying the "triple-construction" to a PDA M creates a CFG G for which $L(M) = L(G)$, where all words in $L(M)$ are accepted by M via empty stack, not via final states. The total number of states is therefore $2k + \lfloor \log_2 k \rfloor$.

To be specific, we define the PDA $M_k = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ such that M_k recognizes the language $(F_{k,2})_2^R$ unambiguously. The states Q contain $2k$ states of the form (sign, carry) where sign $\in \{+, -\}$ and carry $\in \{0, 1, \ldots, \mathrm{k}-1\}$, as well as the `END` state and $\log_2(k) - 1$ intermediate states to pop counters off the stack between a (sign, carry) state and `END`. The input alphabet $\Sigma = \{0, 1\}$, and the stack alphabet $\Gamma = \{\mathtt{X}, \mathtt{Z}\}$. The start state $q_0 = (-, 0)$, the initial stack symbol $Z = \mathtt{Z}$, and the set of final states $F = \emptyset$ because this PDA accepts when the stack is empty so that we can use the "triple-construction."

The transition function δ is described as follows: for (sign, carry) state (s, c), upon reading input symbol a, we simulate the changes in the values of n and kn as follows. To update n, we concatenate a as the new most significant digit; and to update kn, we simulate concatenating $ka + c$. Since $ka + c$ may not fit in a single digit, we compute the new carry $c' = \lfloor \frac{ka+c}{2} \rfloor$, and the stack height must change by $\Delta h = a - (ka + c \bmod 2)$. Depending on Δh and c we have the following transitions between (sign,carry) states:

- If $\Delta h = 0$ then the top of the stack must not change, and we have new transitions $((s, c'), \gamma) \in \delta((s, c), a, \gamma)$ for all $\gamma \in \Gamma$.
- If $s = +$ and $\Delta h = 1$ then we must push a counter onto the stack, so we have the new transitions $((+, c'), \mathtt{X}\gamma) \in \delta((+, c), a, \gamma)$ for all $\gamma \in \Gamma$.
- If $s = -$ and $\Delta h = -1$ then we must push a counter onto the stack, so we have the new transitions $((-, c'), \mathtt{X}\gamma) \in \delta((-, c), a, \gamma)$ for all $\gamma \in \Gamma$.
- If $s = +$ and $\Delta h = -1$ then we must pop a counter off the stack if possible, or else switch the sign of the carry, so we have the new transitions $((+, c'), \epsilon) \in \delta((+, c), a, \mathtt{X})$ and $((-, c'), \mathtt{Z}) \in \delta((+, c), a, \mathtt{Z})$.
- If $s = -$ and $\Delta h = 1$ then we must pop a counter off the stack if possible, or else switch the sign of the carry, so we have the new transitions $((-, c'), \epsilon) \in \delta((-, c), a, \mathtt{X})$ and $((-, c'), \mathtt{Z}) \in \delta((-, c), a, \mathtt{Z})$.

Now we must account for transitions to the `END` state. First, since we wish to accept upon arrival at the `END` state, and we accept via the empty stack, we want to pop all stack symbols off the empty stack, so we have transitions $\delta(\mathtt{END}, \epsilon, \gamma) = (\mathtt{END}, \epsilon)$ for all $\gamma \in \Gamma$.

Second, we must determine the transitions from the (sign, carry) states to `END`. Since we do not wish to accept any values with a leading 0, and we read the most significant bit last, we add a non-deterministic transition upon reading a 1 from any (sign, carry) state for which reading a 1 results in the positive sign (so $s_2(y) > s_2(x)$) and the stack is high enough that reading the remaining digits of the carry would not flip that sign; put another way, we add a transition if after reading $\mathtt{1} \cdot \mathtt{0}^{\log_2 \mathtt{k}}$ the PDA would arrive in the state (+,0). This non-deterministic transition must arrive in the `END` state after popping $s_2(k + c) - 1$ counters off the stack. Since the model of PDA with which we're working does not allow a transition to pop multiple symbols off the stack, we must add $\log_2(k) - 1$ intermediate states that ϵ-transition to `END` while popping counters off the stack. Now we are done.

For example, the PDA M_3 is depicted in Figure 3.1.

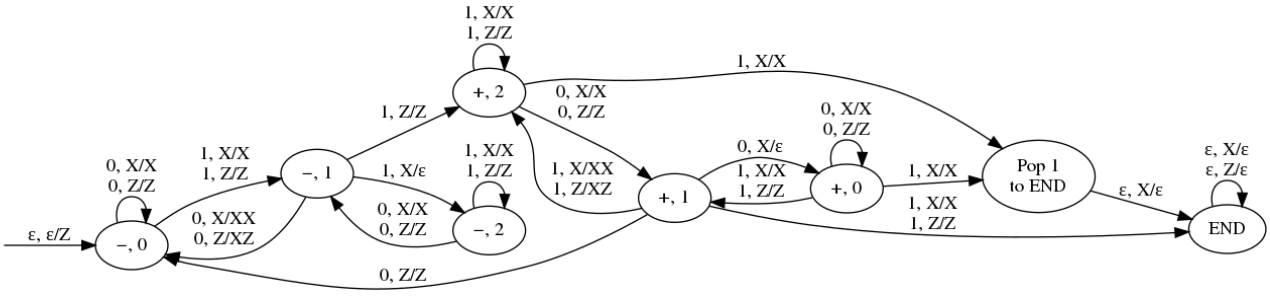

Figure 3.1: The PDA M_3 recognizes the language $(F_{3,2})_2^R$ unambiguously.

One crucially important property of our construction is that our PDA M_k is *unambiguous*. We know this because of the following:

1. All transitions between (sign, carry) states are deterministic and read input.
2. For M_k to accept a word, it must arrive at the `END` state, in order to pop the `Z` off the stack.
3. The only non-deterministic transitions are not followed by any transitions that read any more symbols, so in essence it is using non-determinism to decide when to stop reading the input. For each word in the language, this can only happen in one place.

Thus for each word in $(F_{k,2})_2^R$, there is exactly one accepting path through M_k, so M_k is unambiguous.

3.3 Converting the PDA to an equivalent CFG

We can convert M_k to an equivalent context-free grammar G_k using the triple construction. This gives us a grammar G_k with $O(k^2)$ variables and $O(k^3)$ productions.

Since M_k is an unambiguous PDA, we know that G_k must be an unambiguous CFG [13, Thm. 5.4.3, p. 151].

3.3.1 Simplifying the CFG

The resulting CFG G_k is likely to contain useless variables and productions, so we simplify the grammar using the steps outlined in Step 3 of Section 2.1. Because these changes do not affect any useful productions, the simplified grammar is also unambiguous. When we cleaning G_3 using this procedure, we obtain the following grammar G_3':

$$
\begin{aligned}
S &\to 1F \mid 0S & A &\to 1E \mid 0A \\
B &\to 1G \mid 0B & C &\to 1H \mid 1 \mid 0C \\
D &\to 1I \mid 0D & E &\to 1 \mid 0AJ \\
F &\to 1N \mid 0AK & G &\to 1LB \mid 0 \\
H &\to 1M \mid 1LC \mid 1 & I &\to 1M \mid 1LD \mid 1 \mid 0S \\
J &\to 1J \mid 0E & K &\to 1K \mid 0F \\
L &\to 1L \mid 0G & M &\to 1M \mid 1 \mid 0H \\
N &\to 1N \mid 0I
\end{aligned}
$$

3.4 Converting the CFG to a system of equations

This transformation was discussed in [9] and proven in [16, 18]. It suffices to replace, in each set of productions $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_i$ of a grammar G, each terminal symbol by the indeterminate x, each $\mid$ symbol by a plus sign, and the $\to$ with an equals sign.

Performing this transformation on G_3' gives us the following system of equations:

$$
\begin{aligned}
S &= xF + xS & A &= xE + xA \\
B &= xG + xB & C &= xH + x + xC \\
D &= xI + xD & E &= x + xAJ \\
F &= xN + xAK & G &= xLB + x \\
H &= xM + xLC + x & I &= xM + xLD + x + xS \\
J &= xJ + xE & K &= xK + xF \\
L &= xL + xG & M &= xM + x + xH \\
N &= xN + xI
\end{aligned}
$$

3.5 Solving the system

We can now solve the resulting system of equations for S, obtaining an algebraic equation for which S is the root. The main tool is Gröbner bases, for which a helpful package already exists in `Maple`.

Using the code given in Appendix B.1.1, we find the following quadratic equation for S in the case $k = 3$.

$$x(2x-1)^2(x+1)(2x^2-x+1)S(x)^2+(2x-1)(x-1)^2(x+1)(2x^2-x+1)S(x)+x^4(x^2-x+1) = 0.$$

Solving this quadratic for S gives

$$S(x) = \frac{-(x-1)^2(x+1)(2x^2-x+1) + \sqrt{-(x-1)(2x-1)(2x^2-x+1)(x^3+x^2-x+1)^2}}{2x(2x-1)(x+1)(2x^2-x+1)}.$$

Since the grammar G_3' is unambiguous, the formal power series $S(x)$ is the census generating function for the set $(F_{3,2})_2^R$. In particular, this means that $[x^N]S(x) = |F_{3,2} \cap [2^{N-1}, 2^N)|$, or in other words, the coefficient of x^N in $S(x)$ is the number k-flimsy numbers in $[2^{N-1}, 2^N)$.

3.6 Asymptotics

Finally, we use Bruno Salvy's `gdev` package [19] to perform Flajolet-Sedgewick-style asymptotic analysis [11, § VII. 7.1] to determine an asymptotic formula for the N^{th} coefficient of the power series expansion for $S(x)$. When we run this on our formula for $S(x)$, using the code found in Appendix B.1.1, we get our desired result.

This completes our discussion of the proof of Theorem 2.

Corollary 5. *The number of 3-flimsy numbers $< 2^N$ is $2^{N-1} - O(2^N N^{-1/2})$.*

Proof. For any real number $a > 0$ we have

$$\begin{aligned} 2^N N^{-a} \le \sum_{1 \le n \le N} 2^n n^{-a} &\le \sum_{1 \le n \le N/2} 2^n n^{-a} + \sum_{N/2 < n \le N} 2^n n^{-a} \\ &\le \sum_{1 \le n \le N/2} 2^n + (N/2)^{-a} \sum_{N/2 < n \le N} 2^n \\ &\le 2^{N/2+1} + (N/2)^{-a} 2^{N+1}. \end{aligned}$$

Summing (3.1) and applying the inequalities above gives the desired result. □

Chapter 4

Generalizations

4.1 Results for other values of k in binary

4.1.1 Results for $k = 5$

Using the same construction as before, we can build the unambiguous PDA M_5 for 5-flimsy binary numbers, which is shown in Figure 4.1.

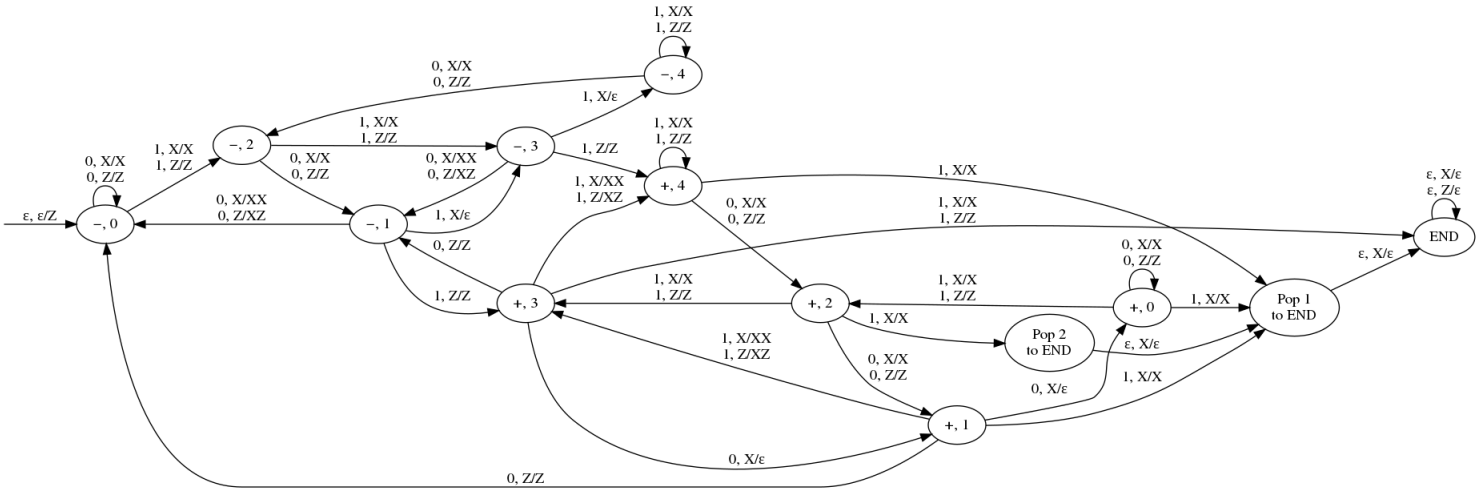

Figure 4.1: The PDA M_5 recognizes the language $(F_{5,2})_2^R$ unambiguously.

Converting M_5 into the unambiguous CFG G_5 and simplifying it produces the grammar

G_5' as follows:

$$
\begin{aligned}
S &\to 1V_1 \mid 0S & V_1 &\to 1V_{29} \mid 0V_{27} \\
V_2 &\to 1V_{36} \mid 1 \mid 0V_2 & V_3 &\to 1V_3 \mid 0V_{12} \\
V_4 &\to 1V_4 \mid 1 \mid 0V_{36} & V_5 &\to 1V_{10}V_{37} \mid 1V_{20}V_2 \mid 1V_{24} \mid 1V_4 \mid 1 \\
V_6 &\to 1V_{28} \mid 0V_6 & V_7 &\to 1V_{10}V_{21} \mid 1V_{20}V_{11} \mid 0 \\
V_8 &\to 1V_7V_8 \mid 1V_{22}V_{25} & V_9 &\to 1V_7V_9 \mid 1V_{22}V_{19} \mid 1V_5 \mid 0S \\
V_{10} &\to 1V_{10} \mid 0V_{23} & V_{11} &\to 1V_{23} \mid 0V_{11} \\
V_{12} &\to 1V_{14} \mid 0V_9 & V_{13} &\to 1V_{13} \mid 0V_{39} \\
V_{14} &\to 1V_{10}V_9 \mid 1V_{20}V_{19} \mid 1V_4 \mid 1 \mid 0V_{27} & V_{15} &\to 1V_7V_{15} \mid 1V_{22}V_{26} \mid 0 \\
V_{16} &\to 1V_{10}V_8 \mid 1V_{20}V_{25} & V_{17} &\to 0V_6V_{33} \mid 0V_{18}V_{34} \\
V_{18} &\to 1V_{39} \mid 0V_{18} & V_{19} &\to 1V_{12} \mid 0V_{19} \\
V_{20} &\to 1V_{20} \mid 0V_{31} & V_{21} &\to 1V_7V_{21} \mid 1V_{22}V_{11} \\
V_{22} &\to 1V_{10}V_{15} \mid 1V_{20}V_{26} & V_{23} &\to 1V_7 \mid 0V_{21} \\
V_{24} &\to 1V_{24} \mid 0V_{35} & V_{25} &\to 1V_{35} \mid 0V_{25} \\
V_{26} &\to 1V_{31} \mid 0V_{26} & V_{27} &\to 1V_{14} \mid 0V_6V_{30} \mid 0V_{18}V_{29} \\
V_{28} &\to 1V_{34} \mid 0V_{17} & V_{29} &\to 1V_3 \mid 0V_{17}V_{30} \mid 0V_{32}V_{29} \\
V_{30} &\to 1V_{30} \mid 0V_1 & V_{31} &\to 1V_{22} \mid 0V_{15} \\
V_{32} &\to 1 \mid 0V_6V_{13} \mid 0V_{18}V_{38} & V_{33} &\to 1V_{33} \mid 0V_{28} \\
V_{34} &\to 1 \mid 0V_{17}V_{33} \mid 0V_{32}V_{34} & V_{35} &\to 1V_{16} \mid 1 \mid 0V_8 \\
V_{36} &\to 1V_5 \mid 0V_{37} & V_{37} &\to 1V_7V_{37} \mid 1V_{22}V_2 \mid 1V_{16} \mid 1V_5 \mid 1 \\
V_{38} &\to 0V_{17}V_{13} \mid 0V_{32}V_{38} & V_{39} &\to 1V_{38} \mid 0V_{32}
\end{aligned}
$$

This yields the following system of equations:

$$
\begin{aligned}
S &= xV_1 + xS & V_1 &= xV_{29} + xV_{27} \\
V_2 &= xV_{36} + x + xV_2 & V_3 &= xV_3 + xV_{12} \\
V_4 &= xV_4 + x + xV_{36} & V_5 &= xV_{10}V_{37} + xV_{20}V_2 + xV_{24} + xV_4 + x \\
V_6 &= xV_{28} + xV_6 & V_7 &= xV_{10}V_{21} + xV_{20}V_{11} + x \\
V_8 &= xV_7V_8 + xV_{22}V_{25} & V_9 &= xV_7V_9 + xV_{22}V_{19} + xV_5 + xS \\
V_{10} &= xV_{10} + xV_{23} & V_{11} &= xV_{23} + xV_{11} \\
V_{12} &= xV_{14} + xV_9 & V_{13} &= xV_{13} + xV_{39} \\
V_{14} &= xV_{10}V_9 + xV_{20}V_{19} + xV_4 + x + xV_{27} & V_{15} &= xV_7V_{15} + xV_{22}V_{26} + x \\
V_{16} &= xV_{10}V_8 + xV_{20}V_{25} & V_{17} &= xV_6V_{33} + xV_{18}V_{34} \\
V_{18} &= xV_{39} + xV_{18} & V_{19} &= xV_{12} + xV_{19} \\
V_{20} &= xV_{20} + xV_{31} & V_{21} &= xV_7V_{21} + xV_{22}V_{11} \\
V_{22} &= xV_{10}V_{15} + xV_{20}V_{26} & V_{23} &= xV_7 + xV_{21} \\
V_{24} &= xV_{24} + xV_{35} & V_{25} &= xV_{35} + xV_{25} \\
V_{26} &= xV_{31} + xV_{26} & V_{27} &= xV_{14} + xV_6V_{30} + xV_{18}V_{29} \\
V_{28} &= xV_{34} + xV_{17} & V_{29} &= xV_3 + xV_{17}V_{30} + xV_{32}V_{29} \\
V_{30} &= xV_{30} + xV_1 & V_{31} &= xV_{22} + xV_{15} \\
V_{32} &= x + xV_6V_{13} + xV_{18}V_{38} & V_{33} &= xV_{33} + xV_{28} \\
V_{34} &= x + xV_{17}V_{33} + xV_{32}V_{34} & V_{35} &= xV_{16} + x + xV_8 \\
V_{36} &= xV_5 + xV_{37} & V_{37} &= xV_7V_{37} + xV_{22}V_2 + xV_{16} + xV_5 + x \\
V_{38} &= xV_{17}V_{13} + xV_{32}V_{38} & V_{39} &= xV_{38} + xV_{32}
\end{aligned}
$$

Solving this system using our method in `Maple` takes 20.60 seconds and uses 179.8 MB of RAM on a server with an Intel Xeon E5-2697 2.60GHz v3 CPU. The solution gives us an algebraic equation of $S(x)$ and x that is sextic in terms of S with coefficients that are polynomials on x of degree up to 26. It can be seen in Appendix B.1.2.

Finally, we obtain our desired asymptotic result: the number of 5-flimsy binary numbers in the interval $[2^{N-1}, 2^N)$ is

$$2^N \left(\frac{1}{4} + \frac{\sqrt{5}}{\sqrt{\pi}} \left(-\frac{3}{8} N^{-1/2} + \frac{799}{1920} N^{-3/2} - \frac{16623}{20480} N^{-5/2} + \frac{7343297}{4915200} N^{-7/2} + \cdots \right) \right)$$

(as stated previously in Remark 4) thus proving Theorem 3.

4.1.2 Results for $k = 7$

Once again, we compute the unambiguous PDA M_7 that recognizes $(F_{7,2})_2^R$, shown in Figure 4.2.

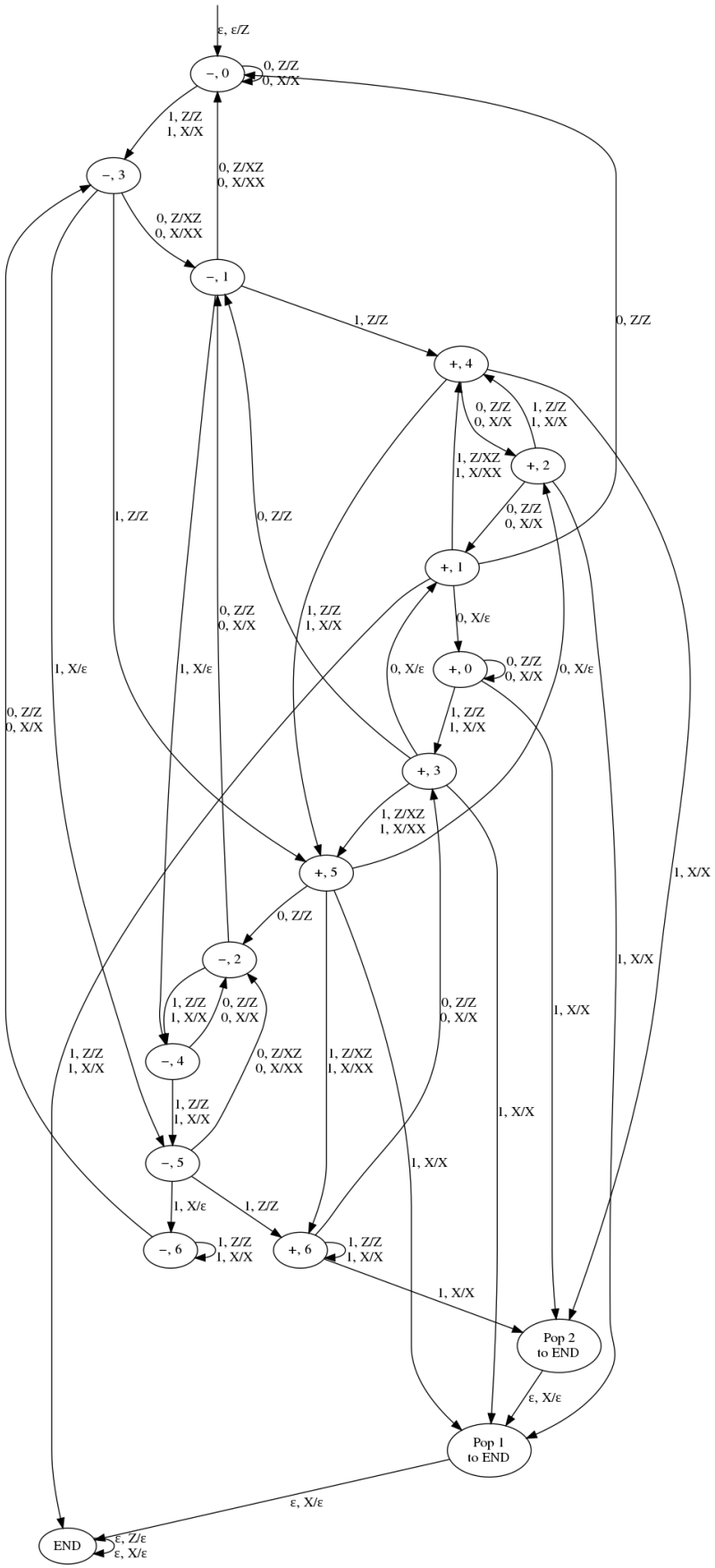

Figure 4.2: The PDA M_7 recognizes the language $(F_{7,2})_2^R$ unambiguously.

The resulting CFG simplifies to 70 variables, listed in Appendix A.1.1. We attempted to solve the system of 70 equations on 70 variables in `Maple` on the same Intel Xeon E5-2697 2.60GHz v3 CPU server we used for $k = 5$ using the code found in Appendix B.1.3. `Maple` took 12 days (1,058,356 seconds) and required over 31 GB of RAM. This algebraic solution is a polynomial in S (where S is the generating series of $(F_{7,2})_2^R$) of order 20, with coefficients of polynomials x of order up to 186, themselves with integer coefficients as large as 31 decimal digits. However, after three weeks of computing and nearly 400 GB of RAM usage, the server could not give `Maple` any more memory and stopped the program before it could find any asymptotics of the coefficients of $S(x)$. So we could not determine the asymptotic behaviour of $F_{7,2}$ using this method.

4.2 The k-equal numbers

Our method so far has been used to compute $s_2(n) - s_2(kn) > 0$. However, it can easily be modified to compute other conditions, such as $s_2(n) - s_2(kn) < a$ or $s_2(n) - s_2(kn) = a$ for $a \in \mathbb{Z}$. We will now examine $s_2(n) - s_2(kn) = 0$.

Stolarsky showed that the number of integers $n \in [2^N, 2^{N+1})$ with $s_2(3n) - s_2(n) = a$ is asymptotic to $\sqrt{\frac{3}{2\pi N}} 2^N e^{-3a^2/2N}$ for $|a| \leq (N-1)^{2/3-\varepsilon}$ where $\varepsilon > 0$, [23].

Naturally $a = 0$ satisfies the criterion on a, so it follows that the number of solutions to $s_2(3n) = s_2(n)$ for $n \in [2^{N-1}, 2^N)$ is asymptotic to $2^{N-1}\sqrt{\frac{3}{2\pi N}}$. We confirm this result using our method, and expand it to $k = 5$ as well.

We define an integer n to be *k-equal* in base b if $s_b(n) = s_b(kn)$. We denote the set of such numbers as $E_{k,b} = \{n \ : \ s_b(n) = s_b(kn)\}$.

To modify our construction of M_k to recognize the language of k-equal binary numbers instead of the k-flimsy numbers, we want the automaton to keep track of $s_2(kn) - s_2(n)$ as before, but now we want the automaton to accept when $s_2(kn) - s_2(n) = 0$ instead of when $s_2(kn) - s_2(n) < 0$. To do this we modify the acceptance condition (that is, the transitions to `END`); but do not modify the (sign, carry) states, nor the transitions between them.

We want to find all the states from which reading $1 \cdot 0^{\lfloor \log_2(k) \rfloor}$ results in being in state $(-, 0)$ with an empty stack, and keep track of the necessary height of the stack to reach that goal. To do this, we perform a search, starting with the final state (and empty stack), and reading the input $1 \cdot 0^{\lfloor \log_2(k) \rfloor}$ in reverse order (following transitions from destination to origin). Then, for each state q and stack height h that can lead to this goal, we add a chain of $h+1$ transitions from q to `END` upon reading a 1 that pop exactly h counters off the stack. (Note that `END` cannot pop counters off the stack in this case, so $\delta(\texttt{END}, \epsilon, \texttt{X}) = \emptyset$.) The resulting PDA is unambiguous for the same reasons that the k-flimsy PDA is. Figures 4.3 and 4.4 show the k-equal PDAs for $k = 3$ and 5, respectively. As with the 7-flimsy numbers, the computation for the 7-equal numbers required too many resources.

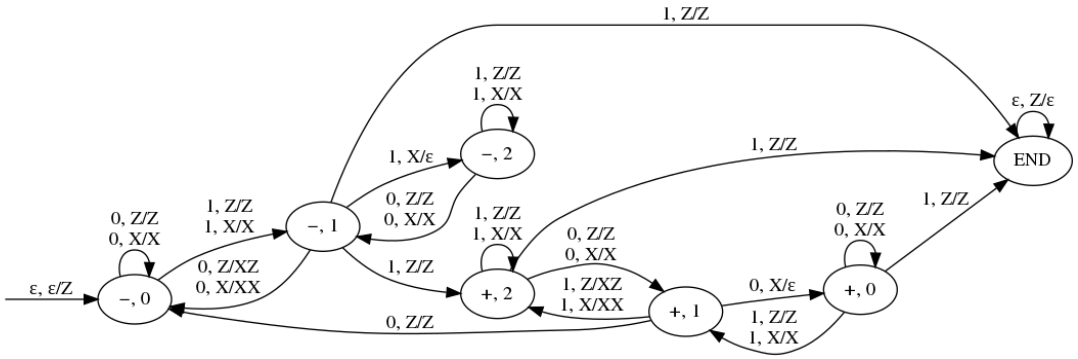

Figure 4.3: This PDA recognizes the language $(E_{3,2})_2^R$ unambiguously.

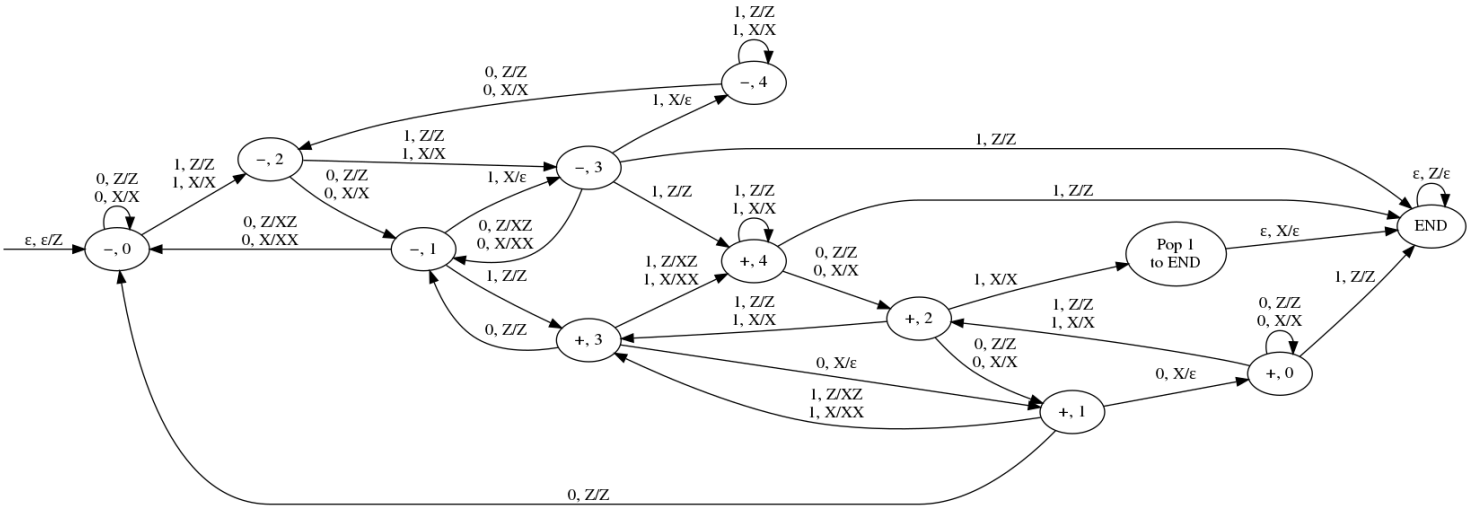

Figure 4.4: This PDA recognizes the language $(E_{5,2})_2^R$ unambiguously.

Once the unambiguous PDAs are constructed, we can use our method to compute the number of length-N k-equal binary numbers, as before. The grammars and `Maple` code can be viewed in Appendices A and B respectively.

Theorem 6. *The number of* 3*-equal numbers in base 2 in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(cN^{-1/2} + O\left(N^{-3/2}\right)\right), \tag{4.1}$$

where $c = \frac{\sqrt{3}}{2\sqrt{2\pi}} \doteq 0.3454941$.

More specifically, we compute the number of N-bit 3-equal binary numbers to be

$$2^N \frac{\sqrt{6}}{\sqrt{\pi}} \left(\frac{1}{4} N^{-1/2} - \frac{1}{6} N^{-3/2} + \frac{29}{96} N^{-5/2} - \frac{515}{2304} N^{-7/2} + \cdots \right).$$

This aligns exactly with Stolarsky's result, [23] but provides more accurate approximations.

Theorem 7. *The number of* 5*-equal numbers in base 2 in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left(cN^{-1/2} + O\left(N^{-3/2}\right)\right), \tag{4.2}$$

where $c = \frac{\sqrt{5}}{4\sqrt{\pi}} \doteq 0.3153916$.

More specifically, we compute the number of N-bit 5-equal binary numbers to be

$$2^N \frac{\sqrt{5}}{\sqrt{\pi}} \left(\frac{1}{4}N^{-1/2} - \frac{361}{960}N^{-3/2} + \frac{46993}{51200}N^{-5/2} - \frac{2995327}{2457600}N^{-7/2} + \cdots\right).$$

4.3 The k-flimsy numbers in base b

Now, our approach works in all bases $b \geq 2$. However, to construct our PDA we must modify the algorithm in Section 3.2 to accommodate this, because the stack may need to account for more than one push/pop at a time. (Note: this depends on the implementation of the model PDA; however, the Hopcroft-Ullman triple construction assumes that PDA transitions do not pop more than one element from the stack at a time, so we must do this in order to follow the same method when converting our PDA into a CFG.)

For our PDA $M_{k,b}$, we must have three kinds of states. We must have the same kinds of states as before; plus another set of intermediate states for multiple pushes or pops between two (sign, carry) states. These states are simple: they each have one epsilon transition that pushes one item onto the stack, or pops one item off the stack, forming a chain of ϵ-transitions linking two (sign, carry) states in order to accomplish multiple pushes/pops. This is necessary because the standard formulation of a PDA cannot pop more than one item off the stack in a single transition (we chose to perform multiple pops in the same way for simplicity). Multi-pop transitions must also connect to their equivalent multi-push transitions if there are no more counters to pop; for an example of this behaviour, see Figure 4.5.

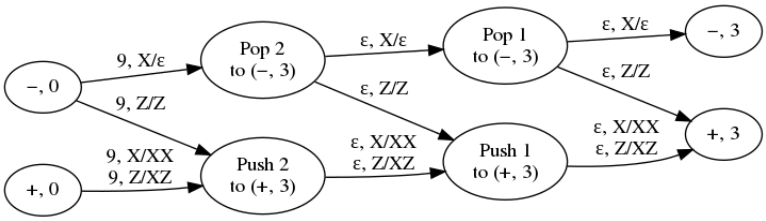

Figure 4.5: Transitions in 4-flimsy base-10 PDA, on reading a 9 with a carry of 0.

Additionally, when constructing these PDAs in higher bases, we must consider transitions from (−, carry) states to `END`, because it is possible for the PDA to transition from

a state $(-, c_1)$ to a $(+, c_2)$ state with multiple pops and pushes, and end with multiple counters on the stack. In the case that such a transition results in more at least $s_b(c_2)$ counters on the stack, and the symbol read is not a 0, there should also be a transition (or chain of transitions) from $(-, c_1)$ to the `END` upon reading this symbol. This is opposed to what happens in base 2, where all transitions to `END` must start from a $(+, c)$ state. Note that because transitions from a $(-, c)$ state to `END` require popping all the counters off the stack and then some, here we must pop at most a maximum number of counters off the stack, instead of popping at least a minimum number of counters. To do this, one can either use a separate `END` state, or reuse the same `END` state but via a path that requires emptying the stack completely first (that is, popping off the base symbol `Z`). For the sake of simplicity, we use the latter in this thesis.

4.3.1 Some results

Using this method, we are able to build PDAs that recognizes the languages $(F_{2,3})_3^R$ and $(F_{2,4})_4^R$. That is, the 2-flimsy numbers in base 3 and base 4. These PDAs can be seen in figures 4.6 and 4.7 respectively.

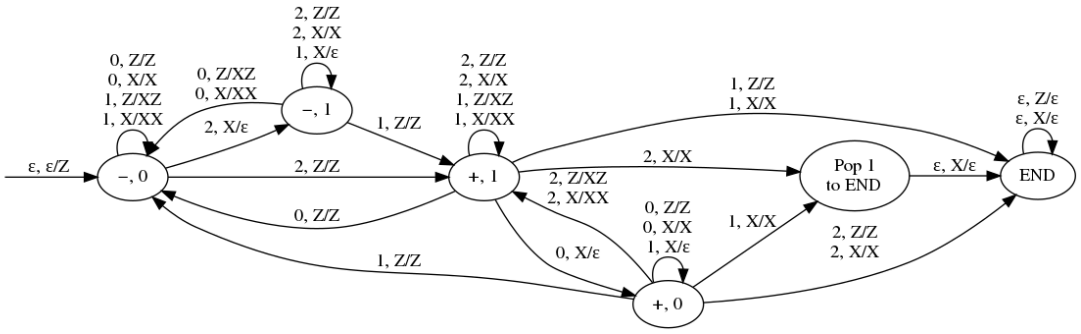

Figure 4.6: This PDA recognizes the language $(F_{2,3})_3^R$ unambiguously.

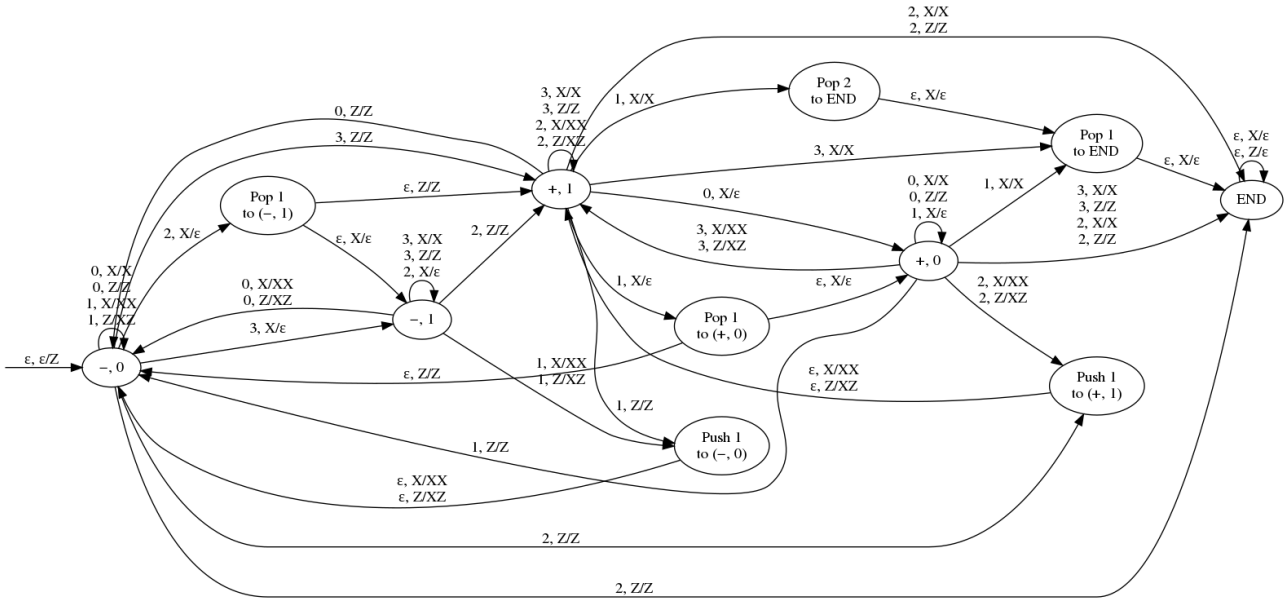

Figure 4.7: This PDA recognizes the language $(F_{2,4})_4^R$ unambiguously.

By applying our method, we obtain the following results.

Theorem 8. *The number of 2-flimsy numbers in base 3 in the interval* $[3^{N-1}, 3^N)$ *is*

$$3^N \left(\frac{1}{3} - cN^{-1/2} + O\left(N^{-3/2}\right) \right), \tag{4.3}$$

where $c = \frac{1}{\sqrt{3\pi}} \doteq 0.32573501$.

Theorem 9. *The number of 2-flimsy numbers in base 4 in the interval* $[4^{N-1}, 4^N)$ *is*

$$4^N \left(\frac{3}{8} - cN^{-1/2} + O\left(N^{-3/2}\right) \right), \tag{4.4}$$

where $c = \frac{3}{8\sqrt{\pi}} \doteq 0.211571094$.

Remark 10. More specifically, we compute that the number of 2-flimsy numbers in base 3 in the interval $[3^{N-1}, 3^N)$ is

$$3^N \left(\frac{1}{3} + \frac{\sqrt{3}}{\sqrt{\pi}} \left(-\frac{1}{3} N^{-1/2} + \frac{1}{48} N^{-3/2} - \frac{13}{1536} N^{-5/2} + \frac{65}{24576} N^{-7/2} + \cdots \right) \right)$$

Similarly we compute that the number of 2-flimsy numbers in base 4 in the interval $[4^{N-1}, 4^N)$ is

$$4^N \left(\frac{3}{8} + \frac{1}{\sqrt{\pi}} \left(-\frac{3}{8} N^{-1/2} + \frac{7}{64} N^{-3/2} + \frac{21}{1024} N^{-5/2} + \frac{85}{8192} N^{-7/2} + \cdots \right) \right)$$

Unfortunately, as with the 7-flimsy binary numbers (see subsection 4.1.2), we were unable to apply this method to obtain results for the 4-flimsy ternary numbers, the 3-flimsy base-4 numbers, or the 2-flimsy base-5 numbers due to the amount of time and memory required to compute these results via this method.

4.3.2 Our conjectures

We have shown that the proportions of N-digit numbers in bases 3 and 4 that are 2-flimsy approaches $\frac{1}{2}$ as $N \to \infty$. This is interesting, because it shows that not only binary numbers are k-flimsy with density $\frac{1}{2}$. It also shows that, for $k \neq b^m$ for all integers $m \geq 0$, it is not required that $\gcd(b, k) = 1$ for this pattern to be observed. Indeed, Bašic proved in [6] that $k \neq b^m$ implies that a k-flimsy number in base b exists. From these results, we make the following conjecture.

Conjecture 11.

$$\lim_{N\to\infty} \frac{\left|F_{b,k} \cap [b^{N-1}, b^N)\right|}{b^N - b^{N-1}} = \begin{cases} 0, & k = b^m, m \in \mathbb{Z}; \\ \frac{1}{2}, & \text{otherwise.} \end{cases}$$

We note that this conjecture was proven for $b = 2$ in [21], but remains to be proven for values of $b \geq 3$. Moreover, it is likely that the natural density of $F_{b,k}$ is $\frac{1}{2}$ for $k \neq b^m$; which we formally state as follows.

Conjecture 12.

$$\lim_{n\to\infty} \frac{|F_{b,k} \cap \{1, 2, \ldots, n\}|}{n} = \begin{cases} 0, & k = b^m, m \in \mathbb{Z}; \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

We note that Conjecture 12 is strictly stronger than Conjecture 11. That is, the proposition that half of the N-digit numbers are k-flimsy does not directly imply that the density converges. If, say, all the integers in the interval $[b^{N-1}, \frac{1}{2}b^{N-1}(b+1))$ were not k-flimsy, while all the integers in the interval $[\frac{1}{2}b^{N-1}(b+1), b^N)$ were k-flimsy, then the natural density of $F_{k,b}$ would oscillate between $\frac{1}{4}$ and $\frac{1}{2}$ and never converge. We present heuristic arguments (not a rigorous proof) for these conjectures in the following section.

4.3.3 A heuristic argument regarding density

Fix $b \geq 2$ and $k \neq b^m$ for all m. Using our procedure, build the automaton $M_{k,b}$ that recognizes the language $(F_{k,b})_b^R$. Recall that the stack height must change by Δh, where

$\Delta h = a - (ka + c \bmod b)$ upon reading symbol a from a carry c. Since $0 \le a \le b-1$ we observe that $1 - b \le \Delta h \le b - 1$.

At a given point in reading the input, the carry can be any value $0 \le c \le k-1$, which combined with the modulo operator in the definition of Δh provides a leveling effect; so Δh should be positive or negative with roughly equal probability. It follows that $M_{k,b}$ contains as many transitions from (−, carry) states to (+, carry) states as transitions from (+, carry) states to (−, carry) states. That is, the value of $s_b(n) - s_b(kn)$ increases and decreases with equal probability as digits of n are read.

Because of this, reading an arbitrary N-digit number is akin to a random walk through $M_{k,b}$, and the odds of that walk ending in a (+, carry) state approaches $\frac{1}{2}$.

Now, $M_{k,b}$ does not simply simply accept when reading the last symbol transitions to a (+, carry) state, but rather, when reading the last symbol followed by $(\log_b(k) + 1)$ 0's would end in the state $(+, 0)$; that is the simulation that $M_{k,b}$ performs when determining the transitions to the `END` state. Recall that this is for the purpose of simulating reading the $|(kn)_b| - |(n)_b|$ most significant digits of kn. In simple terms, we must simulate reading the leftover carry after reading the last symbol.

However, reading the leftover carry c would only affect the stack by $s_b(c) \le (b-1)^{\log_b(k)}$, which is a constant. So for sufficiently large values of N, where the variance of the stack height after reading N digits is considerably greater than $(b-1)^{\log_b(k)}$, reading the leftover carry should not alter whether or not most N-digit numbers are accepted by $M_{k,b}$, so ending in a (+, carry) state following an N-step random walk through $M_{k,b}$ is a good approximation of whether or not the walk corresponds to an N-digit k-flimsy number in base b. Thus we conclude that approximately half of all N-digit base-b numbers should be k-flimsy, as required for Conjecture 11.

Now, since the most significant digits affect the height stack no more than any other digits of the input, we expect to see k-flimsy numbers distributed nearly uniformly among the integers of $[b^{N-1}, b^N)$ as $N \to \infty$. Thus, we expect that the natural density should converge to $\frac{1}{2}$, as required for Conjecture 12.

Chapter 5

Next Steps

5.1 Open problems

5.1.1 Efficient solutions

Due to the computational demands of solving Gröbner bases and computing the asymptotic behaviour of their solutions, we were unable to compute any nontrivial results except for the asymptotics of $F_{3,2}$, $F_{5,2}$, $F_{2,3}$, and $F_{2,4}$. Should more efficient methods be found, or less general solutions that apply to these cases, we may be able to compute more values in the future. However, such a task is well beyond the scope of this thesis.

5.1.2 Alternative CFG constructions

Perhaps an alternative method of creating a CFG that generates $(F_{b,k})_b^R$ would be helpful in solving the solution more quickly. Notably, Sipser [22] provides an alternative method to convert a PDA to a CFG. However, since the CFGs we obtained were simplified dramatically as described in step 3 outlined in Section 2.1, it is not clear if using an alternate construction that creates fewer variables or productions would actually result in a CFG with fewer variables/productions when simplified.

Additionally, the number of variables does not correlate directly to the complexity of the system. For instance, solving the Gröbner basis for $F_{5,2}$ involved a system of 40 equations (which can be seen in Appendix B.1.2) corresponding to a CFG with 40 variables and was solved in under 30 seconds using less than 200 MB of memory, while the Gröbner basis for $F_{2,5}$ involved a system of just 21 equations (which can be seen in Appendix B.4.1), but did not finish in 21 days of computation with 59 GB of memory. Since the underlying language being represented (namely $(F_{k,b})_b^R$) is the same, I believe that finding an alternative CFG construction is an unlikely means of improving our results.

5.2 CFLpy software

The software used for this thesis is provided in the public repository of Python code:

https://git.uwaterloo.ca/Flimsy/CFLpy

The software repository provides the following functionality:

1. Create a PDA (that accepts by empty stack)
2. Test whether or not a PDA accepts a given word
3. Create a transition diagram for a PDA as a GraphViz file
4. Create a CFG
5. Simplify the CFG using the the procedure described in step 3 of Section 2.1
6. Convert a PDA to a CFG using the Hopcroft-Ullman triple construction
7. Generate the first n words for a given integer n, from shortest to longest, generated by a CFG
8. Print the CFG in a readable fashion
9. Print the `Maple` code for determining the asymptotic behaviour of the generating series of the language $L(G)$ for a CFG G

Additionally, the repository includes a factory for creating PDAs, with various sample PDAs described in this thesis (for example, $\{a^m b^n c^m \ : \ m, n \geq 0\}$, the Dyck language, and the palindrome language) as well as functions for generating k-flimsy PDAs in any base b, and k-equal binary PDAs. For more details on how to use the software, see the `README` file included in the repository.

References

[1] J. Arndt. *Matters Computational*. Springer-Verlag, 2011.

[2] A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns: enumerative aspects. In S. T. Klein et al., editors, *LATA 2018*, volume 10792 of *Lecture Notes in Computer Science*, pages 195–206. Springer-Verlag, 2018.

[3] A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns, the vectorial kernel method, and generating functions for pushdown automata. *Algorithmica*, 82:386–428, 2020.

[4] C. Banderier and M. Drmota. Coefficients of algebraic functions: formulae and asymptotics. In *FPSAC 2013*, volume AS of *DMTCS Proc.*, pages 1065–1076. 2013.

[5] C. Banderier and M. Drmota. Formulae and asymptotics for coefficients of algebraic functions. *Combin. Prob. Comput.*, 24:1–53, 2015.

[6] B. Bašić. The existence of n-flimsy numbers in a given base. *Ramanujan J.*, 43:359–369, 2017.

[7] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD thesis, University of Innsbruck, Institute for Mathematics, 1965.

[8] L. H. Y. Chen, H.-K. Hwang, and V. Zacharovas. Distribution of the sum-of-digits function of random integers: a survey. *Prob. Surveys*, 11:177–236, 2014.

[9] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North Holland, Amsterdam, 1963.

[10] Trevor Clokie, Thomas F. Lidbetter, Antonio J. Molina Lovett, Jeffrey Shallit, and Leon Witzman. Computational Fun with Sturdy and Flimsy Numbers. In *10th International Conference on Fun with Algorithms (FUN 2021)*, volume 157 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[11] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[12] H. Gruber, J. Lee, and J. Shallit. Handbook of automata vol. 1.

[13] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

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

[15] I. Kátai. Change of the sum of digits by multiplication. *Acta Sci. Math. (Szeged)*, 39:319–328, 1977.

[16] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, 1986.

[17] Maplesoft. Algorithms used by groebner[basis]. https://www.maplesoft.com/support/help/Maple/view.aspx?path=Groebner/Basis_algorithms. Accessed: 2020-02-23.

[18] A. Panholzer. Gröbner bases and the defining polynomial of a context-free grammar generating function. *J. Automata, Languages, and Combinatorics*, 10:79–97, 2005.

[19] B. Salvy. gdev package of algolib version 17.0. Available at http://algo.inria.fr/libraries/, 2013.

[20] J. Schmid. The joint distribution of the binary digits of integer multiples. *Acta Arith.*, 43:391–415, 1984.

[21] W. M. Schmidt. The joint distributions of the digits of certain integer s-tuples. In P. Erdős, editor, *Studies in Pure Mathematics to the Memory of Paul Turán*, pages 605–622. Birkhäuser, 1983.

[22] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[23] K. B. Stolarsky. Integers whose multiples have anomalous digital frequencies. *Acta Arith.*, 38:117–128, 1980/81.

Appendix A

Flimsy Grammars

A.1 Base 2

A.1.1 7-flimsy base-2 grammar

$S \to 0S \mid 1V_{63}$
$V_1 \to 1V_{60} \mid 1V_{18} \mid 1V_{36}V_{50} \mid 1V_{22}V_{64} \mid 1V_5V_{17} \mid 1$
$V_2 \to 1V_{57}V_{20} \mid 1V_{32}V_{21} \mid 1V_{28}V_{49}$
$V_3 \to 0V_{30}V_{12} \mid 0V_{40}V_{69} \mid 0V_{44}V_3 \mid 1$
$V_4 \to 0V_{49} \mid 1V_{28}$
$V_5 \to 0V_2 \mid 1V_5$
$V_6 \to 0V_6 \mid 1V_{61}$
$V_7 \to 0V_7 \mid 1V_{58} \mid 1$
$V_8 \to 0V_8 \mid 1V_{16}$
$V_9 \to 0V_8V_{10} \mid 0V_6V_{65} \mid 0V_{11}V_{48} \mid 1V_{37}$
$V_{10} \to 0V_{54} \mid 1V_{48}$
$V_{11} \to 0V_{11} \mid 1V_{25}$
$V_{12} \to 0V_{40} \mid 1V_3$
$V_{13} \to 0V_{47} \mid 1V_{57}$
$V_{14} \to 1V_1 \mid 1V_{41} \mid 1V_{57}V_{50} \mid 1V_{32}V_{64} \mid 1V_{28}V_{17} \mid 1$
$V_{15} \to 0V_{19} \mid 1V_{35}$
$V_{16} \to 0V_{67}V_{23} \mid 0V_{42}V_{56} \mid 0V_{59}V_{27}$
$V_{17} \to 0V_{50} \mid 1V_{45} \mid 1$
$V_{18} \to 0V_{58} \mid 1V_{18} \mid 1$
$V_{19} \to 0 \mid 1V_{13}V_{19} \mid 1V_{35}V_{55} \mid 1V_4V_{15}$
$V_{20} \to 1V_{13}V_{20} \mid 1V_{35}V_{21} \mid 1V_4V_{49}$
$V_{21} \to 0V_{21} \mid 1V_2$
$V_{22} \to 0V_{51} \mid 1V_{22}$
$V_{23} \to 0V_{30} \mid 1V_{27}$
$V_{24} \to 0V_{30}V_{46} \mid 0V_{40}V_{33} \mid 0V_{44}V_{24}$
$V_{25} \to 0V_{67}V_{46} \mid 0V_{42}V_{33} \mid 0V_{59}V_{24} \mid 1$
$V_{26} \to 1V_{13}V_{26} \mid 1V_{35}V_7 \mid 1V_4V_{62}$
$V_{27} \to 0V_{30}V_{23} \mid 0V_{40}V_{56} \mid 0V_{44}V_{27}$
$V_{28} \to 0 \mid 1V_{36}V_{20} \mid 1V_{22}V_{21} \mid 1V_5V_{49}$
$V_{29} \to 0V_{29} \mid 1V_{34}$
$V_{30} \to 0V_{67} \mid 1V_{23}$
$V_{31} \to 0V_{54} \mid 1V_{60} \mid 1V_{36}V_{38} \mid 1V_{22}V_{52} \mid 1V_5V_{43}$
$V_{32} \to 1V_{36}V_{19} \mid 1V_{22}V_{55} \mid 1V_5V_{15}$
$V_{33} \to 0V_{25} \mid 1V_{33}$
$V_{34} \to 0 \mid 1V_{57}V_{39} \mid 1V_{32}V_{29} \mid 1V_{28}V_{47}$
$V_{35} \to 0V_{15} \mid 1V_{32}$
$V_{36} \to 0V_{34} \mid 1V_{36}$
$V_{37} \to 0V_{43} \mid 1V_{31}$
$V_{38} \to 0S \mid 1V_{45} \mid 1V_{13}V_{38} \mid 1V_{35}V_{52} \mid 1V_4V_{43} \mid 1$
$V_{39} \to 1V_{13}V_{39} \mid 1V_{35}V_{29} \mid 1V_4V_{47}$

$V_{40} \to 0V_{42} \mid 1V_{12}$
$V_{41} \to 1V_{36}V_{26} \mid 1V_{22}V_{7} \mid 1V_{5}V_{62}$
$V_{42} \to 0V_{8}V_{12} \mid 0V_{6}V_{69} \mid 0V_{11}V_{3}$
$V_{43} \to 0V_{38} \mid 1V_{37}$
$V_{44} \to 0V_{59} \mid 1V_{46}$
$V_{45} \to 0V_{17} \mid 1V_{1}$
$V_{46} \to 0V_{44} \mid 1V_{24}$
$V_{47} \to 0V_{39} \mid 1V_{13}$
$V_{48} \to 0V_{30}V_{10} \mid 0V_{40}V_{65} \mid 0V_{44}V_{48} \mid 1V_{53}$
$V_{49} \to 0V_{20} \mid 1V_{4}$
$V_{50} \to 1V_{45} \mid 1V_{66} \mid 1V_{13}V_{50} \mid 1V_{35}V_{64} \mid 1V_{4}V_{17} \mid 1$
$V_{51} \to 1V_{57}V_{19} \mid 1V_{32}V_{55} \mid 1V_{28}V_{15}$
$V_{52} \to 0V_{52} \mid 1V_{68}$
$V_{53} \to 0V_{68} \mid 1V_{53}$
$V_{54} \to 0V_{9} \mid 1V_{10}$
$V_{55} \to 0V_{55} \mid 1V_{51}$
$V_{56} \to 0V_{16} \mid 1V_{56}$
$V_{57} \to 1V_{36}V_{39} \mid 1V_{22}V_{29} \mid 1V_{5}V_{47}$
$V_{58} \to 1V_{57}V_{26} \mid 1V_{32}V_{7} \mid 1V_{28}V_{62}$
$V_{59} \to 0V_{8}V_{46} \mid 0V_{6}V_{33} \mid 0V_{11}V_{24}$
$V_{60} \to 0V_{14} \mid 1V_{60}$
$V_{61} \to 0V_{67}V_{12} \mid 0V_{42}V_{69} \mid 0V_{59}V_{3}$
$V_{62} \to 0V_{26} \mid 1V_{66}$
$V_{63} \to 0V_{67}V_{10} \mid 0V_{42}V_{65} \mid 0V_{59}V_{48} \mid 1V_{31}$
$V_{64} \to 0V_{64} \mid 1V_{14}$
$V_{65} \to 0V_{63} \mid 1V_{65}$
$V_{66} \to 0V_{62} \mid 1V_{41} \mid 1$
$V_{67} \to 0V_{8}V_{23} \mid 0V_{6}V_{56} \mid 0V_{11}V_{27} \mid 1$
$V_{68} \to 0V_{9} \mid 1V_{1} \mid 1V_{57}V_{38} \mid 1V_{32}V_{52} \mid 1V_{28}V_{43}$
$V_{69} \to 0V_{61} \mid 1V_{69}$

Appendix B

Maple code

B.1 Base 2

B.1.1 3-flimsy base-2

```
eqs := [-S + x*F + x*S ,
        -A + x*E + x*A,
        -B + x*G + x*B,
        -C + x*H + x + x*C,
        -D + x*_I + x*D,
        -E + x + x*A*J,
        -F + x*N + x*A*K,
        -G + x*L*B + x,
        -H + x*M + x*L*C + x,
        -_I + x*M + x*L*D + x + x*S,
        -J + x*J + x*E,
        -K + x*K + x*F,
        -L + x*L + x*G,
        -M + x*M + x + x*H,
        -N + x*N + x*_I]:
Groebner[Basis](eqs, lexdeg([A,B,C,D,E,F,G,H,_I,J,K,L,M,N], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[1]:
libname := ".",libname:
combine(equivalent(ps, x, n, 5));
```

B.1.2 5-flimsy base-2

```
eqs := [-S + x*V_1 + x*S,
        -V_1 + x*V_29 + x*V_27,
        -V_2 + x*V_36 + x + x*V_2,
        -V_3 + x*V_3 + x*V_12,
        -V_4 + x*V_4 + x + x*V_36,
        -V_5 + x*V_10*V_37 + x*V_20*V_2 + x*V_24 + x*V_4 + x,
        -V_6 + x*V_28 + x*V_6,
        -V_7 + x*V_10*V_21 + x*V_20*V_11 + x,
        -V_8 + x*V_7*V_8 + x*V_22*V_25,
        -V_9 + x*V_7*V_9 + x*V_22*V_19 + x*V_5 + x*S,
        -V_10 + x*V_10 + x*V_23,
        -V_11 + x*V_23 + x*V_11,
        -V_12 + x*V_14 + x*V_9,
        -V_13 + x*V_13 + x*V_39,
        -V_14 + x*V_10*V_9 + x*V_20*V_19 + x*V_4 + x + x*V_27,
        -V_15 + x*V_7*V_15 + x*V_22*V_26 + x,
        -V_16 + x*V_10*V_8 + x*V_20*V_25,
        -V_17 + x*V_6*V_33 + x*V_18*V_34,
        -V_18 + x*V_39 + x*V_18,
        -V_19 + x*V_12 + x*V_19,
        -V_20 + x*V_20 + x*V_31,
        -V_21 + x*V_7*V_21 + x*V_22*V_11,
        -V_22 + x*V_10*V_15 + x*V_20*V_26,
        -V_23 + x*V_7 + x*V_21,
        -V_24 + x*V_24 + x*V_35,
        -V_25 + x*V_35 + x*V_25,
        -V_26 + x*V_31 + x*V_26,
        -V_27 + x*V_14 + x*V_6*V_30 + x*V_18*V_29,
        -V_28 + x*V_34 + x*V_17,
        -V_29 + x*V_3 + x*V_17*V_30 + x*V_32*V_29,
        -V_30 + x*V_30 + x*V_1,
        -V_31 + x*V_22 + x*V_15,
        -V_32 + x + x*V_6*V_13 + x*V_18*V_38,
        -V_33 + x*V_33 + x*V_28,
        -V_34 + x + x*V_17*V_33 + x*V_32*V_34,
        -V_35 + x*V_16 + x + x*V_8,
        -V_36 + x*V_5 + x*V_37,
        -V_37 + x*V_7*V_37 + x*V_22*V_2 + x*V_16 + x*V_5 + x,
        -V_38 + x*V_17*V_13 + x*V_32*V_38,
        -V_39 + x*V_38 + x*V_32]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
```

```
  V_11, V_12, V_13, V_14, V_15, V_16, V_17, V_18, V_19, V_20, V_21, V_22, V_23,
  V_24, V_25, V_26, V_27, V_28, V_29, V_30, V_31, V_32, V_33, V_34, V_35, V_36,
  V_37, V_38, V_39], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[3]:
libname := ".",libname:
combine(equivalent(ps, x, n, 5));
```

B.1.3 7-flimsy base-2

```
eqs := [-S + x*S + x*V_63
-V_1 + x*V_60 + x*V_18 + x*V_36 V_50 + x*V_22 V_64 + x*V_5 V_17 + x
-V_2 + x*V_57 V_20 + x*V_32 V_21 + x*V_28 V_49
-V_3 + x*V_30 V_12 + x*V_40 V_69 + x*V_44 V_3 + x
-V_4 + x*V_49 + x*V_28
-V_5 + x*V_2 + x*V_5
-V_6 + x*V_6 + x*V_61
-V_7 + x*V_7 + x*V_58 + x
-V_8 + x*V_8 + x*V_16
-V_9 + x*V_8 V_10 + x*V_6 V_65 + x*V_11 V_48 + x*V_37
-V_10 + x*V_54 + x*V_48
-V_11 + x*V_11 + x*V_25
-V_12 + x*V_40 + x*V_3
-V_13 + x*V_47 + x*V_57
-V_14 + x*V_1 + x*V_41 + x*V_57 V_50 + x*V_32 V_64 + x*V_28 V_17 + x
-V_15 + x*V_19 + x*V_35
-V_16 + x*V_67 V_23 + x*V_42 V_56 + x*V_59 V_27
-V_17 + x*V_50 + x*V_45 + x
-V_18 + x*V_58 + x*V_18 + x
-V_19 + x + x*V_13 V_19 + x*V_35 V_55 + x*V_4 V_15
-V_20 + x*V_13 V_20 + x*V_35 V_21 + x*V_4 V_49
-V_21 + x*V_21 + x*V_2
-V_22 + x*V_51 + x*V_22
-V_23 + x*V_30 + x*V_27
-V_24 + x*V_30 V_46 + x*V_40 V_33 + x*V_44 V_24
-V_25 + x*V_67 V_46 + x*V_42 V_33 + x*V_59 V_24 + x
-V_26 + x*V_13 V_26 + x*V_35 V_7 + x*V_4 V_62
-V_27 + x*V_30 V_23 + x*V_40 V_56 + x*V_44 V_27
-V_28 + x + x*V_36 V_20 + x*V_22 V_21 + x*V_5 V_49
-V_29 + x*V_29 + x*V_34
```

```
-V_30 + x*V_67 + x*V_23
-V_31 + x*V_54 + x*V_60 + x*V_36 V_38 + x*V_22 V_52 + x*V_5 V_43
-V_32 + x*V_36 V_19 + x*V_22 V_55 + x*V_5 V_15
-V_33 + x*V_25 + x*V_33
-V_34 + x + x*V_57 V_39 + x*V_32 V_29 + x*V_28 V_47
-V_35 + x*V_15 + x*V_32
-V_36 + x*V_34 + x*V_36
-V_37 + x*V_43 + x*V_31
-V_38 + x*S + x*V_45 + x*V_13 V_38 + x*V_35 V_52 + x*V_4 V_43 + x
-V_39 + x*V_13 V_39 + x*V_35 V_29 + x*V_4 V_47
-V_40 + x*V_42 + x*V_12
-V_41 + x*V_36 V_26 + x*V_22 V_7 + x*V_5 V_62
-V_42 + x*V_8 V_12 + x*V_6 V_69 + x*V_11 V_3
-V_43 + x*V_38 + x*V_37
-V_44 + x*V_59 + x*V_46
-V_45 + x*V_17 + x*V_1
-V_46 + x*V_44 + x*V_24
-V_47 + x*V_39 + x*V_13
-V_48 + x*V_30 V_10 + x*V_40 V_65 + x*V_44 V_48 + x*V_53
-V_49 + x*V_20 + x*V_4
-V_50 + x*V_45 + x*V_66 + x*V_13 V_50 + x*V_35 V_64 + x*V_4 V_17 + x
-V_51 + x*V_57 V_19 + x*V_32 V_55 + x*V_28 V_15
-V_52 + x*V_52 + x*V_68
-V_53 + x*V_68 + x*V_53
-V_54 + x*V_9 + x*V_10
-V_55 + x*V_55 + x*V_51
-V_56 + x*V_16 + x*V_56
-V_57 + x*V_36 V_39 + x*V_22 V_29 + x*V_5 V_47
-V_58 + x*V_57 V_26 + x*V_32 V_7 + x*V_28 V_62
-V_59 + x*V_8 V_46 + x*V_6 V_33 + x*V_11 V_24
-V_60 + x*V_14 + x*V_60
-V_61 + x*V_67 V_12 + x*V_42 V_69 + x*V_59 V_3
-V_62 + x*V_26 + x*V_66
-V_63 + x*V_67 V_10 + x*V_42 V_65 + x*V_59 V_48 + x*V_31
-V_64 + x*V_64 + x*V_14
-V_65 + x*V_63 + x*V_65
-V_66 + x*V_62 + x*V_41 + x
-V_67 + x*V_8 V_23 + x*V_6 V_56 + x*V_11 V_27 + x
-V_68 + x*V_9 + x*V_1 + x*V_57 V_38 + x*V_32 V_52 + x*V_28 V_43
-V_69 + x*V_61 + x*V_69]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
  V_11, V_12, V_13, V_14, V_15, V_16, V_17, V_18, V_19, V_20, V_21, V_22, V_23,
```

```
  V_24, V_25, V_26, V_27, V_28, V_29, V_30, V_31, V_32, V_33, V_34, V_35, V_36,
  V_37, V_38, V_39, V_40, V_41, V_42, V_43, V_44, V_45, V_46, V_47, V_48, V_49,
  V_50, V_51, V_52, V_53, V_54, V_55, V_56, V_57, V_58, V_59, V_60, V_61, V_62,
  V_63, V_64, V_65, V_66, V_67, V_68, V_69], [S])):
algeq := %[1]:
f := solve(algeq, S):
ps := f[2]:
libname := ".",libname:
combine(equivalent(ps,x,n,1));
combine(equivalent(ps,x,n,2));
combine(equivalent(ps,x,n,3));
```

B.2 Base 3

B.2.1 2-flimsy base-2

```
eqs := [-S + x*V_7*V_4 + x*S + x*V_1,
        -V_1 + x*V_3*V_6 + x*V_9 + x + x*S + x*V_1,
        -V_2 + x + x*V_2 + x*V_3*V_2 + x*V_9 + x,
        -V_3 + x*V_3*V_8 + x + x*V_3,
        -V_4 + x*V_1 + x*V_7*V_4 + x*V_4,
        -V_5 + x + x*V_7*V_5 + x*V_5,
        -V_6 + x*S + x*V_6 + x*V_3*V_6 + x*V_9 + x,
        -V_7 + x*V_7*V_5 + x*V_7 + x,
        -V_8 + x + x*V_8 + x*V_3*V_8,
        -V_9 + x*V_3*V_2 + x*V_9 + x + x*V_9 + x]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[1]:
libname := ".",libname:
combine(equivalent(ps, x, n, 5));
```

B.2.2 4-flimsy base-2

```
eqs := [-S + x*V_34 + x*S + x*V_22,
-V_1 + x*V_1 + x*V_11 + x*V_23,
-V_2 + V_43*V_8 + V_6*V_19,
-V_3 + V_43*V_48 + V_6*V_29,
```

```
-V_4 + x*V_43*V_25 + x*V_36 + x*V_6*V_32 + x + x*V_13 + x*V_18*V_25
     + x*V_33 + x*V_26*V_32,
-V_5 + x*V_5 + x*V_38 + x*V_45 + x,
-V_6 + x*V_6 + x*V_19 + x*V_26,
-V_7 + x*V_17*V_15 + x*V_49 + x*V_10 + x*V_2*V_7 + x,
-V_8 + x*V_26 + x*V_8 + x*V_19,
-V_9 + V_46*V_37 + V_24*V_41,
-V_10 + V_43*V_15 + V_5 + V_36 + V_6*V_7,
-V_11 + x*V_46*V_23 + x*V_27 + x*V_24*V_1 + x*V_12*V_23 + x*V_30 + x*V_11*V_1,
-V_12 + x*V_46*V_37 + x*V_24*V_41 + x*V_12*V_37 + x*V_11*V_41,
-V_13 + V_46*V_34 + V_27*V_47 + V_24*V_44,
-V_14 + x*V_9*V_14 + x*V_35*V_16,
-V_15 + x*V_33 + x*V_15 + x*V_7 + x,
-V_16 + x*V_16 + x*V_30 + x*V_14,
-V_17 + V_43*V_40 + V_42 + V_6*V_28,
-V_18 + x*V_43*V_40 + x*V_42 + x*V_6*V_28 + x*V_18*V_40 + x*V_21 + x*V_26*V_28,
-V_19 + x*V_17*V_8 + x*V_2*V_19 + x,
-V_20 + V_43*V_25 + V_36 + V_42*S + V_6*V_32,
-V_21 + x*V_43*V_48 + x*V_6*V_29 + x + x*V_18*V_48 + x*V_26*V_29,
-V_22 + x*V_20 + x + x*V_46*V_34 + x*V_27*V_47 + x*V_24*V_44
      + x*V_12*V_34 + x*V_30*V_47 + x*V_11*V_44,
-V_23 + x + x*V_9*V_23 + x*V_39 + x*V_35*V_1,
-V_24 + x*V_23 + x*V_24 + x*V_11,
-V_25 + x*V_4 + x*V_25 + x*V_32,
-V_26 + x*V_43*V_8 + x*V_6*V_19 + x*V_18*V_8 + x*V_26*V_19,
-V_27 + x*V_14 + x*V_27 + x*V_30,
-V_28 + x*V_17*V_40 + x*V_3 + x*V_2*V_28 + x,
-V_29 + x*V_17*V_48 + x*V_2*V_29,
-V_30 + x + x*V_46*V_14 + x*V_24*V_16 + x*V_12*V_14 + x*V_11*V_16,
-V_31 + x*V_45 + x + x*V_31 + x*V_38,
-V_32 + x*V_17*V_25 + x*V_10 + x*V_3*S + x*V_2*V_32 + x + x*S + x*V_22,
-V_33 + x*V_43*V_15 + x*V_5 + x*V_36 + x*V_6*V_7 + x + x*V_18*V_15
      + x*V_45 + x*V_33 + x*V_26*V_7 + x,
-V_34 + x*V_47 + x*V_9*V_34 + x*V_39*V_47 + x*V_35*V_44 + x*V_4,
-V_35 + V_46*V_23 + V_27 + V_24*V_1,
-V_36 + x*V_36 + x + x*V_7 + x*V_33,
-V_37 + x*V_9*V_37 + x*V_35*V_41 + x,
-V_38 + x*V_17*V_31 + x*V_2*V_38 + x,
-V_39 + V_46*V_14 + V_24*V_16,
-V_40 + x*V_18 + x*V_40 + x*V_28,
-V_41 + x*V_41 + x*V_12 + x*V_37,
-V_42 + x*V_42 + x*V_29 + x*V_21,
```

```
-V_43 + x*V_43 + x*V_28 + x*V_18,
-V_44 + x*V_44 + x*V_22 + x*V_34,
-V_45 + x*V_43*V_31 + x*V_6*V_38 + x*V_18*V_31 + x*V_26*V_38,
-V_46 + x*V_37 + x*V_46 + x*V_12,
-V_47 + x*V_47 + x*V_32 + x*V_4,
-V_48 + x*V_21 + x*V_48 + x*V_29,
-V_49 + V_43*V_31 + V_6*V_38]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
  V_11, V_12, V_13, V_14, V_15, V_16, V_17, V_18, V_19, V_20, V_21, V_22, V_23,
  V_24, V_25, V_26, V_27, V_28, V_29, V_30, V_31, V_32, V_33, V_34, V_35, V_36,
  V_37, V_38, V_39, V_40, V_41, V_42, V_43, V_44, V_45, V_46, V_47, V_48, V_49],
  [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[1]:
libname := ".",libname:
combine(equivalent(ps, x, n, 1));
combine(equivalent(ps, x, n, 2));
combine(equivalent(ps, x, n, 3));
```

B.3 Base 4

B.3.1 2-flimsy base-4

```
eqs := [-S + x*S + x*V_3 + x + x*V_17*V_10 + x*V_11*V_4 + x*V_4,
-V_1 + V_17*V_10 + V_11*V_4,
-V_2 + x + x*V_2,
-V_3 + V_2*S + V_8*V_16 + V_7,
-V_4 + x*S + x*V_2*S + x*V_8*V_16 + x*V_7 + x + x*V_1 + x*V_4,
-V_5 + x*V_5 + x*V_12*V_5 + x + x*V_2 + x*V_8*V_5,
-V_6 + x*V_6 + x*V_12*V_6 + x*V_9 + x + x + x*V_8*V_6 + x*V_13 + x*V_7 + x,
-V_7 + x*V_8*V_6 + x*V_13 + x*V_7 + x + x*V_7 + x,
-V_8 + x + x*V_2 + x*V_8*V_5 + x*V_8,
-V_9 + V_8*V_6 + V_13 + V_7,
-V_10 + x*V_17*V_10 + x*V_11*V_4 + x*V_4 + x*V_14*V_10 + x*V_10,
-V_11 + x*V_11 + x,
-V_12 + V_2 + V_8*V_5,
-V_13 + x + x*V_13,
-V_14 + V_17*V_15 + V_11,
-V_15 + x*V_17*V_15 + x*V_11 + x + x*V_14*V_15 + x*V_15,
```

```
-V_16 + x*V_16 + x*V_12*V_16 + x*V_9 + x + x*S + x*V_2*S + x*V_8*V_16 + x*V_7 + x,
-V_17 + x*V_17 + x*V_17*V_15 + x*V_11 + x]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
    V_11, V_12, V_13, V_14, V_15, V_16, V_17], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[2]:
libname := ".",libname:
combine(equivalent(ps, x, n, 5));
```

B.3.2 3-flimsy base-4

```
eqs := [ -S + x*S + x*V_1 + x*V_32 + x*V_34*V_32 + x*V_29*V_36 + x*V_8*V_35,
-V_1 + V_13 + V_24*V_5 + V_19*V_15,
-V_2 + x*V_2 + x + x*V_12 + x*V_34*V_12 + x*V_8*V_23,
-V_3 + x*V_3 + x*V_14 + x*V_34*V_14 + x*V_29 + x*V_8*V_17,
-V_4 + V_24*V_30 + V_19*V_6,
-V_5 + x*V_5 + x*V_25 + x*V_10*V_5 + x*V_4*V_15 + x + x*V_15 + x*V_28,
-V_6 + x*V_24*V_30 + x*V_19*V_6 + x + x*V_16*V_30 + x*V_6*V_6,
-V_7 + x*V_7 + x*V_39 + x*V_25 + x*V_10*V_7 + x*V_4*V_20 + x + x*V_20 + x,
-V_8 + V_37*V_14 + V_2 + V_3*V_17,
-V_9 + x*V_9 + x*V_10*V_9 + x*V_4*V_27 + x*V_27 + x,
-V_10 + V_24*V_33 + V_19*V_16 + V_26,
-V_11 + V_24*V_21 + V_19*V_38,
-V_12 + x*V_37*V_12 + x*V_3*V_23 + x*V_22*V_12 + x*V_14*V_23,
-V_13 + x*V_13 + x*V_39 + x*V_25 + x*V_10*V_7 + x*V_4*V_20 + x + x*V_20 + x,
-V_14 + x*V_37*V_14 + x*V_2 + x*V_3*V_17 + x + x*V_22*V_14 + x*V_12 + x*V_14*V_17,
-V_15 + x*S + x*V_13 + x*V_24*V_5 + x*V_19*V_15 + x*V_32 + x*V_20 + x*V_16*V_5
      + x*V_6*V_15 + x,
-V_16 + x + x*V_24*V_33 + x*V_19*V_16 + x*V_26 + x*V_16*V_33 + x*V_6*V_16 + x*V_38,
-V_17 + x*V_34*V_14 + x*V_29 + x*V_8*V_17 + x*V_17 + x*V_14,
-V_18 + x*V_18 + x + x*V_10*V_9 + x*V_4*V_27 + x*V_27,
-V_19 + x*V_19 + x*V_10*V_30 + x*V_4*V_6 + x*V_6,
-V_20 + x*V_18 + x*V_13 + x*V_24*V_7 + x*V_19*V_20 + x + x*V_27 + x*V_20
      + x*V_16*V_7 + x*V_6*V_20 + x,
-V_21 + x*V_21 + x*V_10*V_21 + x*V_4*V_38 + x*V_38 + x,
-V_22 + x*V_37*V_22 + x*V_3*V_31 + x*V_22*V_22 + x*V_14*V_31 + x,
-V_23 + x*V_34*V_12 + x*V_8*V_23 + x*V_23 + x + x*V_12,
-V_24 + x*V_24 + x*V_10*V_33 + x*V_4*V_16 + x*V_11 + x*V_16,
-V_25 + V_18 + V_13 + V_24*V_7 + V_19*V_20,
-V_26 + x + x*V_26 + x*V_10*V_21 + x*V_4*V_38 + x*V_38,
```

```
-V_27 + x*V_24*V_9 + x*V_19*V_27 + x + x*V_16*V_9 + x*V_6*V_27,
-V_28 + V_37*V_32 + V_2*V_36 + V_3*V_35,
-V_29 + V_37*V_12 + V_3*V_23,
-V_30 + x*V_30 + x*V_10*V_30 + x*V_4*V_6 + x*V_6,
-V_31 + x*V_34*V_22 + x*V_8*V_31 + x*V_31 + x*V_22,
-V_32 + x*V_37*V_32 + x*V_2*V_36 + x*V_3*V_35 + x*V_36 + x*V_22*V_32 + x*V_12*V_36
      + x*V_14*V_35 + x*V_15,
-V_33 + x*V_33 + x*V_10*V_33 + x*V_4*V_16 + x*V_11 + x*V_16,
-V_34 + V_37*V_22 + V_3*V_31,
-V_35 + x*V_34*V_32 + x*V_29*V_36 + x*V_8*V_35 + x*V_35 + x*V_1 + x*V_32,
-V_36 + x*V_28 + x*V_36 + x*V_25 + x*V_10*V_5 + x*V_4*V_15 + x*V_11*S + x + x*V_15,
-V_37 + x*V_37 + x*V_22 + x*V_34*V_22 + x*V_8*V_31,
-V_38 + x*V_24*V_21 + x*V_19*V_38 + x*V_16*V_21 + x*V_6*V_38,
-V_39 + V_24*V_9 + V_19*V_27]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
  V_11, V_12, V_13, V_14, V_15, V_16, V_17, V_18, V_19, V_20, V_21, V_22, V_23,
  V_24, V_25, V_26, V_27, V_28, V_29, V_30, V_31, V_32, V_33, V_34, V_35, V_36,
  V_37, V_38, V_39], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[1]:
libname := ".",libname:
combine(equivalent(ps, x, n, 1));
combine(equivalent(ps, x, n, 2));
combine(equivalent(ps, x, n, 3));
```

B.4 Base 5

B.4.1 2-flimsy base-5

```
eqs := [-S + x*V_11 + x*V_7*V_12 + x*V_16*V_11 + x*S + x*V_15 + x + x*V_8*V_12
          + x*V_7*V_1*V_11,
-V_1 + x*V_1 + x*V_8*V_1 + x*V_7*V_1 + x,
-V_2 + x*V_10 + x*V_5*V_2 + x*V_14 + x + x + x*V_2 + x*V_5*V_20 + x*V_18*V_2
     + x*V_6 + x,
-V_3 + V_7*V_12 + V_16*V_11,
-V_4 + x*V_5*V_4 + x*V_14 + x + x*S + x*V_4 + x*V_18*V_4 + x*V_6 + x + x*V_3,
-V_5 + x*V_5 + x + x*V_19 + x*V_5*V_17 + x*V_5*V_9 + x*V_18*V_17,
-V_6 + V_10 + V_5*V_2 + V_14,
-V_7 + x + x*V_7*V_13 + x*V_16 + x*V_7 + x*V_8*V_13 + x*V_7*V_1,
```

```
-V_8 + V_7*V_13 + V_16,
-V_9 + x*V_5*V_9 + x*V_9 + x*V_18*V_9 + x,
-V_10 + x*V_10 + x + x*V_5*V_20 + x*V_18*V_20,
-V_11 + x*V_11 + x*V_3 + x*S + x*V_19*S + x*V_5*V_4 + x*V_14 + x + x*V_5*V_9*S
      + x*V_18*V_4 + x*V_6 + x,
-V_12 + x*V_12 + x*V_8*V_12 + x*V_7*V_1*V_11 + x*V_7*V_12 + x*V_16*V_11 + x*V_11
      + x*V_15 + x,
-V_13 + x*V_13 + x*V_8*V_13 + x*V_7*V_1 + x*V_7*V_13 + x*V_16 + x,
-V_14 + x*V_14 + x + x*V_10 + x*V_5*V_2 + x*V_14 + x + x*V_5*V_20 + x*V_18*V_2
      + x*V_6 + x,
-V_15 + V_19*S + V_5*V_4 + V_14,
-V_16 + x*V_7*V_1 + x*V_16 + x + x*V_8*V_1,
-V_17 + x*V_19 + x*V_5*V_17 + x + x*V_17 + x*V_5*V_9 + x*V_18*V_17,
-V_18 + V_19 + V_5*V_17,
-V_19 + x*V_19 + x + x*V_5*V_9 + x*V_18*V_9,
-V_20 + x*V_5*V_20 + x*V_20 + x*V_18*V_20 + x]:
Groebner[Basis](eqs, lexdeg([V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_10,
  V_11, V_12, V_13, V_14, V_15, V_16, V_17, V_18, V_19, V_20], [S])):
algeq := %[1]:
assume(x, positive):
f := solve(algeq, S):
ps := f[1]:
libname := ".",libname:
combine(equivalent(ps, x, n, 1));
combine(equivalent(ps, x, n, 2));
combine(equivalent(ps, x, n, 3));
```