Decidability Questions in Ostrowski Numeration Systems

Fabian Schmitthenner
fabian@schmitthenner.eu

30. März 2023

Version 1.0.1 from May 2023
(Revised version with minor corrections)

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Philipp Hieronymi

Zweitgutachter: Dr. Christian d'Elbée

Mathematisches Institut

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

List of Figures

1. Introduction

In this thesis, we investigate the decidability of some first-order theories. Let S be a finite or recursively enumerable set of relation and function symbols, which we also call signature.[1] A first-order theory T in the signature S is decidable when there is a decision algorithm, that takes a formula ψ as an input and decides if $T \models \psi$.
Mojżesz Presburger showed around 1929, that the first-order theory of natural numbers together with the addition operation is decidable (see [10] and [13]).

Theorem 1.1. *Let $S = \{+\}$ be the signature of Presburger arithmetic and let*

$$\mathcal{M} = (\mathbb{N}, (x, y) \mapsto x + y).$$

Then $\mathcal{M}$ is decidable.

In the original proof, formulas are translated to equivalent formulas without quantifiers. These may include a few new relation symbols like the modulo operation. The quantifier-free formulas can then be checked for their truth value. Büchi later showed the same theorem using a different approach. In this approach, finite state automata are used to show the decidability of Presburger arithmetic[4]. This approach can also be extended to show that certain extensions of the theory of Presburger arithmetic are also decidable. Which relations and functions can be added to this structure, such that the theory of the resulting structure is still decidable? When adding multiplication to the structure, the theory of the resulting structure is not decidable anymore. But we can add other relations to this structure and keep the property of the decidability of its theory. One interesting class of mathematical objects are sequences:

Definition 1.2. A function $s\colon \mathbb{N} \to \{0, 1\}$ is called a sequence. Define the signature $S' = (+, c)$. For a sequence s we define the S'-structure

$$\mathcal{M}_s = (\mathbb{N}, (x, y) \mapsto x + y, \{x \in \mathbb{N} \mid s(x) = 1\}).$$

In this structure, many properties of s can be captured using first-order formulas. For example, the following formula can be used to express, that the sequence s is eventually periodic:

$$\varphi = \exists p \exists n \neg \text{zero}(p) \wedge \forall m (c(n + m) \longleftrightarrow c((n + m) + p))$$

where $\text{zero}(p) = \forall m\ m + p = m$. Then $\mathcal{M}_s \models \varphi$ if and only if s is eventually periodic. Many other important properties can also be expressed using first-order-formulas. A list of such properties and their expression as S'-formulas can be found in [12, Section 8 First-order formulas for fundamental sequence properties]. If $\mathcal{M}_s$ is decidable, these properties about s can all be decided computationally. However, the computational requirements for executing such an algorithm can be extraordinary in terms of both runtime and required memory. Thus it may not be possible to execute such an algorithm in practice.

[1] In model theory, such a set is usually called a language. However, this term conflicts with the use of the term language in this thesis. We follow the convention also used in [5, footnote 1] and call such a set a signature.

One class of sequences that lead to a decidable theory of $\mathcal{M}_s$ are the k-automatic sequences:

Definition 1.3. A k-automatic sequence is a sequence $s\colon \mathbb{N} \to \{0,1\}$, such that there is a finite state automaton on the alphabet $\{0,\dots,k-1\}$ that accepts exactly the numbers n in base-k notation for which $s(n)=1$.

Theorem 1.4. *For k-automatic sequences s, the theory of $\mathcal{M}_s$ is decidable.*

This topic of analyzing properties of sequences s for which $\mathcal{M}_s$ is decidable is analyzed in great detail in [12]. In that book, the software Walnut is presented. Walnut can be used to execute the decision algorithm on formulas for decidable theories of structures of certain forms in practice[7]. One class of such structures are the structures $\mathcal{M}_s$ for which s is a k-automatic sequence. These structures are supported by Walnut. A user of Walnut needs to provide the automaton for the sequence s as well as a sentence φ. Walnut can then determine if $\mathcal{M}_s \models \varphi$. It works by assigning finite state automata recursively to all sub-formulas of φ. In the end, an automaton is assigned to φ itself. Walnut then checks if that automaton accepts any words. If so, the statement holds in $\mathcal{M}_s$, if not it doesn't hold. Walnut may not always finish in any reasonable amount of time, but does so for a wide range of interesting formulas and automata[2].
We expand on this approach in two ways. Firstly, we use Ostrowski numeration systems instead of a standard base-k system. Sequences that are not automatic in a base k numeration system may still be automatic in another numeration system. In Ostrowski numeration systems, specific corresponding Sturmian words are automatic. A characteristic Sturmian word is defined as follows:

Definition 1.5. Let $\alpha \in (0,1) \setminus \mathbb{Q}$ be an irrational number between 0 and 1. Then the characteristic Sturmian word of slope α is defined as $c_\alpha\colon \mathbb{N} \to \{0,1\}$

$$c_\alpha(n) = \lfloor \alpha(n+1) \rfloor - \lfloor \alpha n \rfloor$$

We will define Ostrowski numeration systems and elaborate on their relationship to Sturmian words in section 4. Using this relationship, it has been shown in [3], that $\mathcal{M}_{c_\alpha}$ has a decidable theory for quadratic α. Here, α is called quadratic when it is the solution of a polynomial with integer coefficients of degree 2. One important contribution of that paper is the definition of finite state automata that can recognize addition of numbers in specific Ostrowski representations. The automata are more complicated than the automata needed to recognize addition in base-k numeration systems.
Secondly, we use automata on infinite words instead of on finite words. There are more infinite than finite words over a finite alphabet. The set of finite words are countable whereas the set of infinite words is uncountable and has the same cardinality as $\mathbb{R}$. The approach of using infinite words allows to prove statements about larger sets. It was used

[2] According to [12, 1.3 Two simple examples, page 9], "the worst-case running time of the decision procedure is quite bad. Nevertheless, [...] the procedure is still practical for a very wide range of problems. It seems that most examples that people care about don't result in the worst-case behavior. Why this is so is still unknown."

in [5] to show that the combined theory $\{T : \forall \alpha \in (0,1) \setminus \mathbb{Q} \quad M_\alpha \models T\}$ is also decidable. Intuitively, infinite words are needed here because we need to show statements for all α which come from the set $(0,1)\setminus\mathbb{Q}$. This set is uncountable, it also has the same cardinality as $\mathbb{R}$. The paper uses the combined adder automaton from [3], that was used there as an intermediate step towards the adder automata for specific Ostrowski numeration systems. [5] then continues to give a brief description how that automaton can be adapted to an automaton working on infinite words with a finite alphabet. The main contribution of the present thesis is to construct an adder automaton with the same goal. We give a detailed description how to construct that automaton. Furthermore, we show that it is correct. We do this in two ways. First, by using a conventional mathematical proof. Secondly, by using software assistance. This improves the confidence in the correctness of the construction. The automaton constructed here has just 24 states, less than the automaton that emerges when following the steps outlined in [5, section 4][3].

In the first section after this introduction, we introduce important notions and concepts. We introduce finite and infinite words, languages and Buchi automata – a type of automaton working on infinite words. This is followed by a definition of first-order formulas, structures and the $\models$ relation.

Afterwards, we assign Buchi automata to first-order formulas and show how the truth value of first-order formulas in certain structures can be decided using this approach. As a first application, we prove that the theory of Presburger arithmetic is decidable. This approach can also be used to prove theorem 1.4, as the finite state automata for the sequences can be translated into corresponding Buchi automata in this context.

In section 4, we introduce Ostrowski representations and their relationship to Sturmian words. Ostrowski numeration systems are a numeration system where the individual digits can be arbitrarily large. This is not good for our purpose because we want to represent them with a finite alphabet. This is not an issue when considering only individual quadratic Sturmian words – for them the digits of the corresponding Ostrowski representation are bounded by some b that only depends on α. But in the more generic case the digits are not bounded. Thus, in section 5 we will introduce a representation of Ostrowski numbers with a finite alphabet. This is done by representing the individual digits by a base-2 representation. This approach is taken from [5, section 3]. Finally, we construct the adder on this representation.

In the last section, we apply the constructed adder using Pecan. Pecan is a software program, that can be used to construct the automata automatically and check first-order formulas for their truth value. It is similar to Walnut, but works with automata on infinite words instead of automata on finite words. We give a short introduction to

[3] It is claimed there, that the automaton constructed there has 82 states. I followed that construction myself and fixed a minor flaw in the argument, as one needs to start with an automaton working on Ostrowski numbers in least significant digit first notation instead of most significant digit first notation. Following that approach, I did not reach an automaton with 82 states, but instead one with 53 states. I have no idea where this number 82 comes from. There is also an explicit automaton given in `https://github.com/ReedOei/SturmianWords/blob/master/ostrowski-automata/bco_adder.txt` which has 221 states. I also don't understand how that automaton has been constructed.

Pecan. We then show the correctness of our adder automaton, that was constructed in section 5, using Pecan. This will show that it has indeed been constructed correctly. Afterwards, we will show, that no characteristic Sturmian word is eventually periodic using Pecan[4]. In the end, we will analyze the complexity of the involved automata when proving statements with Pecan and discuss how small changes in the formulas can have a big impact in the runtime of Pecan.

2. Notation and basic concepts

We write $\omega = \mathbb{N} = \{0, 1, 2, \dots\}$ for the natural numbers including 0 and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ for the natural numbers without 0. We sometimes use the set-theoretic notions of natural numbers where $n = \{0, \dots, n-1\}$.

2.1. Words and languages

Definition 2.1. Let Σ be a set, which we call alphabet. We define the set of finite words over Σ by

$$\Sigma^* := \bigcup_{i \in \omega} (i \to \Sigma)$$

An element of this set is called a finite word. We write $|w| = \mathrm{dom}(w)$ for the length of a finite word (which is a natural number) and write $w_i = w(i) \in \Sigma$ for the (i+1)-st symbol of the finite word. That is, the first symbol of a word has position 0. We also write $w = w_0 w_1 \dots w_{|w|-1}$ to denote finite words.

Definition 2.2. A subset $L \subseteq \Sigma^*$ of finite words is called a *language* (of finite words).

The set of finite words over a finite alphabet is countable. The set of all languages over a finite alphabet, however, is uncountable. Let L be a language over a finite alphabet Σ. Then it might be interesting to find out for a given word $w \in \Sigma^*$ if $w \in L$ or $w \notin L$. For some languages, this question is easier to answer than for other languages. On the easier side are regular languages. These are languages, that can be recognized by a (non-)deterministic finite state automaton.
There are only countably many words over a finite alphabet. Thus, it's not possible to represent $\mathbb{R}$ – or some large subset of it like $(0, 1) \setminus \mathbb{Q}$ – by finite words over a finite alphabet. To allow an identification with such large sets, we introduce the notion of ω-words.

Definition 2.3. The set of ω-words over an alphabet Σ is defined as

$$\Sigma^\omega := \omega \to \Sigma$$

Similarly to Definition 2.1, we call an element of this set an ω-word. We sometimes omit ω and just say word, if it's clear from the context. We write $w_i = w(i) \in \Sigma$ for

[4] This was also done in [5, theorem 7.1]. We give more details what happens during the execution of the proof here.

the (i+1)-st symbol in the ω-word. For $w \in \Sigma^*, v \in \Sigma^\omega$, we write $wv \in \Sigma^\omega$ for the concatenation

$$i \mapsto \begin{cases} w_i & i < |w| \\ w_{i-|w|} & i \geq |w| \end{cases}$$

For $w \in \Sigma^*$, we write w^ω for the ω-word

$$i \mapsto w_{i \bmod |w|}$$

A subset $L \subseteq \Sigma^\omega$ is called an *ω-language*.

As for languages over finite words, it's also interesting to ask for ω-languages what kind of structure they have. The question of wether a specific ω-word is part of the language is not as easy to ask in this case though. It's not even possible to characterize all ω-words in a finite manner (at least when $|\Sigma| \geq 2$). Thus, there is also no algorithm that takes an ω-word as input, as not all ω-words can be encoded. Some questions can still be asked though. For example, given an ω-language, we can ask if it is non-empty; and if so, we can ask for a witness.

2.2. Finite state automata

We start by defining a nondeterministic finite state automaton (NFA). This motivates the subsequent definition of Buchi automata.

Definition 2.4. Let Σ be a finite alphabet. A NFA over Σ is a tuple (S, I, T, F) where

S is the finite set of states.

$I \subseteq S$ is the set of initial states.

$T \subseteq S \times \Sigma \times S$ is the transition relation.

$F \subseteq S$ is the set of final states.

Let's define the words this automaton accepts.

Definition 2.5. Let $\mathcal{A} = (S, I, T, F)$ be an NFA over Σ and let $w \in \Sigma^*$. Then $\sigma\colon (|w| + 1) \to S$ is a run of w over $\mathcal{A}$ when

$\sigma(0) \in I$ and

$\forall j \in |w| \;\; (\sigma(j), w_j, \sigma(j+1)) \in T$

When $\sigma(|w|) \in F$, the run is an accepting run. For a given word w, $\mathcal{A}$ accepts w when there is an accepting run of w over $\mathcal{A}$.

This definition can be generalized to ω-words. The definition works similarly, the only problem is that the definition of an accepting run can't be based on the last state that is reached, as there is no last state. There are various different ways these acceptance criteria can be transformed such that they work for infinite words. Throughout this thesis, we will use Buchi automata, which use a certain acceptance condition. In section 6 we will also briefly touch automata for infinite words with different acceptance conditions.

Definition 2.6. Let Σ be a finite alphabet. A Buchi automaton over Σ is a tuple (S, I, T, F) where

S is the finite set of states.

$I \subseteq S$ is the set of initial states.

$T \subseteq S \times \Sigma \times S$ is the transition relation.

$F \subseteq S$ is the set of final states.

This definition is exactly the same definition as the definition of an NFA. However, we define accepted runs differently:

Definition 2.7. Let $\mathcal{A} = (S, I, T, F)$ be a Buchi automaton over Σ and let $w \in \Sigma^\omega$. Then $\sigma \colon \omega \to S$ is a run of w over $\mathcal{A}$ when

$\sigma(0) \in I$ and

$\forall j \in \omega \ (\sigma(j), w_j, \sigma(j+1)) \in T$

Define the infinity set $l(\sigma)$ of a run as

$$l(\sigma) = \{s \in S : \{i \in \omega : \sigma(i) = s\} \text{ is infinite}\}$$

Then σ is accepted by $\mathcal{A}$ when $l(\sigma) \cap F \neq \emptyset$. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ when there is an accepting run of w over $\mathcal{A}$. The language of $\mathcal{A}$ is the set of all words $w \in \Sigma^\omega$, that are accepted by some run of w over $\mathcal{A}$. We denote this language by $\mathcal{L}(\mathcal{A})$. A language $L \subseteq \Sigma^\omega$ is called ω-regular when there is a Buchi automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = L$. $w \in \Sigma^\omega$ is ω-regular when $\{w\}$ is ω-regular.

We draw automata by using circles for each of the states in S and draw an arrow from s to u annotated with b for each $(s, b, u) \in T$. Furthermore, the states in F are denoted by double circles and the states in S are denoted by inbound arrows.
Buchi automata have nice closure properties, as the following proposition shows.

Proposition 2.8. *Let Σ be a finite set and let $L_1, L_2 \subseteq \Sigma^\omega$ be ω-regular. Then*

$L_1 \cup L_2$ is ω-regular.

$L_1 \cap L_2$ is ω-regular.

$\Sigma^\omega \setminus L_1$ is ω-regular.

Furthermore, the resulting automata can be computed algorithmically.

Proof. For the first two statements, we give explicit automata recognizing these languages.
Let $\mathcal{A} = (S_A, I_A, T_A, F_A)$, $\mathcal{B} = (S_B, I_B, T_B, F_B)$ be Buchi automata with $\mathcal{L}(\mathcal{A}) = L_1$ and $\mathcal{L}(\mathcal{B}) = L_2$. Without loss of generality, let $S_A \cap S_B = \emptyset$. Then define $\mathcal{C} = (S_A \cup S_B, I_A \cup I_B, T_A \cup T_B, F_A \cup F_B)$. Then $\mathcal{L}(\mathcal{C}) = L_1 \cup L_2$.

For the second statement, set $S = S_A \times S_B \times \{\emptyset, \{A\}, \{B\}, \{A, B\}\}$. We take the product of the states of the original automata to remember these states. Furthermore, we add some more information to remember if we have recently seen a final states of the $\mathcal{A}$ or $\mathcal{B}$ parts. Once we've seen a final state in both parts, we also produce a final state and set back this part of the state to $\emptyset$.
Formalizing this, set

$$I = I_A \times I_B \times \{\emptyset\}$$
$$F = S_A \times S_B \times \{\{A, B\}\}$$

$$T = \left\{ ((s_A, s_B, g), c, (d_A, d_B, h)) : \begin{array}{r} (s_A, c, d_A) \in T_A \\ (s_B, c, d_B) \in T_B \\ h = t(g) \cup l(d_A, d_B) \end{array} \right\}$$

where

$$t(x) = \begin{cases} \emptyset & x = \{A, B\} \\ x & x \neq \{A, B\} \end{cases}$$

$$l(a, b) = \begin{cases} \emptyset & a \notin F_A, b \notin F_B \\ \{A\} & a \in F_A, b \notin F_B \\ \{B\} & a \notin F_A, b \in F_B \\ \{A, B\} & a \in F_A, b \in F_B \end{cases}$$

Then $\mathcal{L}(S, I, T, F) = L_1 \cap L_2$.
The third statement is more complicated and we will not show it here. A construction can be found for example in [6, Section 3.4.4 Complementation of Buchi Automata]. □

Remark 2.9. We can compute the number of states the resulting automata will have. The first construction uses at most $|S_A| + |S_B|$ states and the second construction at most $4|S_A||S_B|$ states. The number of states needed for the complementation automaton is much bigger and may need $2^{\mathcal{O}(n \log n)}$ states, where n is the number of states of the original automaton (see [6]).

2.3. First-order formulas and structures

Definition 2.10. A signature is a pair (S, ar) where S is the set of relation symbols and $ar \colon S \to \mathbb{N}$ is a function mapping each relation symbol to its arity. We may also denote signatures by a tuple of their relation symbols. An (S, ar)-structure $\mathcal{M}$ is a tuple (M, I) where I has domain S and for $R \in S$

$$I(R) \subseteq M^{ar(s)}.$$

We also write $R^{\mathcal{M}}$ for $I(R)$ and may denote a structure $\mathcal{M} = (M, I)$ over the signature $(R_1, \ldots, R_n)$ by the tuple $(M, R_1^{\mathcal{M}}, \ldots, R_n^{\mathcal{M}})$.

Usually, signatures are defined such that there can also be function and constant symbols in the signature. We only consider signatures without function and constant symbols here. Each signature S' with function and constant symbols can be converted to a signature S without those by converting each function symbol f in S' into a relation symbol R with $ar(R) = ar(f) + 1$. Theories and formulas can also be transformed back and forth[11]. Thus the decidability of the corresponding theories can also be transformed. For example, take the structure $\mathcal{M}_s$ from definition 1.2 in the introduction. Instead of defining $+$ as a 2-ary function symbol, we would define it as a 3-ary relation symbol with $+^{\mathcal{M}_s} = \{(x, y, z) \mid x + y = z\}$.

Definition 2.11. Let $L = (S, ar)$ be a signature. Then we define the set of formulas $\mathrm{F}(L)$ as the smallest set, such that

- $v_i \equiv v_j \in \mathrm{F}(L)$ for $i, j \in \omega$
- $R\left(v_{i_1}, \ldots, v_{i_{ar(R)}}\right) \in \mathrm{F}(L)$ for $R \in S, i_1, \ldots, i_{ar(R)} \in \omega$.
- $\varphi \wedge \psi \in \mathrm{F}(L)$ for $\varphi, \psi \in \mathrm{F}(L)$
- $\varphi \vee \psi \in \mathrm{F}(L)$ for $\varphi, \psi \in \mathrm{F}(L)$
- $\neg\varphi \in \mathrm{F}(L)$ for $\varphi \in \mathrm{F}(L)$
- $\exists v_j \varphi \in \mathrm{F}(L)$ for $j \in \omega, \varphi \in \mathrm{F}(L)$

We define the set of free variables of a formula φ, denoted by $\mathrm{free}(\varphi)$ recursively in the usual way:

$$\begin{aligned}
\mathrm{free}(v_i \equiv v_j) &= \{v_i, v_j\} \\
\mathrm{free}\left(R\left(v_{i_1}, \ldots, v_{i_{ar(R)}}\right)\right) &= \{v_{i_1}, \ldots, v_{i_{ar(R)}}\} \\
\mathrm{free}(\varphi \wedge \psi) = \mathrm{free}(\varphi \vee \psi) &= \mathrm{free}(\varphi) \cup \mathrm{free}(\psi) \\
\mathrm{free}(\neg\varphi) &= \mathrm{free}(\varphi) \\
\mathrm{free}(\exists v_j \varphi) &= \mathrm{free}(\varphi) \setminus \{v_j\}
\end{aligned}$$

For a finite signature L, $\mathrm{F}(L)$ is countable. We write $\forall v_j \varphi$ for $\neg\exists v_j \neg\varphi$, $\top$ for $\forall v_0 v_0 \equiv v_0$ and $\bot$ for $\neg\top$. An L-sentence is a formula without free variables. An L-theory is a set of L-sentences.

Definition 2.12. Let $\mathcal{M}$ be an L-structure and let φ be an L-formula with $\mathrm{free}(\varphi) = \{x_1, \ldots x_n\}$. Let $m_1, \ldots m_n \in M$. We define $\mathcal{M}\frac{m_1,\ldots,m_n}{x_1,\ldots,m_n} \models \varphi(x_1, \ldots, x_n)$ recursively by

$$\begin{aligned}
\mathcal{M}\frac{m_1, m_2}{x_1, x_2} &\models x_1 \equiv x_2 \iff m_1 = m_2 \\
\mathcal{M}\frac{m_1, \ldots, m_k}{x_1, \ldots, x_k} &\models R(x_{i_1}, \ldots, x_{i_{ar(R)}}) \iff R^{\mathcal{M}}(m_{i_1}, \ldots, m_{i_{ar(R)}}) \\
\mathcal{M}\frac{m_1, \ldots, m_k}{x_1, \ldots, x_k} &\models \varphi \wedge \psi \iff \mathcal{M}\frac{m_1, \ldots, m_k}{x_1, \ldots, x_k} \models \varphi \text{ and } \mathcal{M}\frac{m_1, \ldots, m_k}{x_1, \ldots, x_k} \models \psi
\end{aligned}$$

$$\mathcal{M}\frac{m_1,\ldots,m_k}{x_1,\ldots,x_k} \models \varphi \vee \psi \iff \mathcal{M}\frac{m_1,\ldots,m_k}{x_1,\ldots,x_k} \models \varphi \text{ or } \mathcal{M}\frac{m_1,\ldots,m_k}{x_1,\ldots,x_k} \models \psi$$
$$\mathcal{M}\frac{m_1,\ldots,m_k}{x_1,\ldots,x_k} \models \neg\varphi \iff \mathcal{M}\frac{m_1,\ldots,m_k}{x_1,\ldots,x_k} \not\models \varphi$$
$$\mathcal{M}\frac{m_2,\ldots,m_k}{x_2,\ldots,x_k} \models \exists x_1\varphi \iff \text{ there is } m \in M \text{ s.t. } \mathcal{M}\frac{m,m_2,\ldots,m_k}{x_1,x_2,\ldots,x_k} \models \varphi$$

For sentences φ we write $\mathcal{M} \models \varphi$. Let T be an L-theory. Then $\mathcal{M} \models T$ when $\mathcal{M} \models \varphi$ for all $\varphi \in T$. For a sentence φ, we write $T \models \varphi$ when for all L-structures $\mathcal{M}$, if $\mathcal{M} \models T$ then $\mathcal{M} \models \varphi$. The theory of an L-structure is the set of L-sentences $\{\varphi : \mathcal{M} \models \varphi\}$. An L-theory T is complete, if for each L-sentence φ either $T \models \varphi$ or $T \models \neg\varphi$. It is consistent, when $T \not\models \bot$. An L-theory T is decidable, if there is an algorithm that gets φ as input, finishes on all inputs and decides whether $T \models \varphi$.

3. Decidability questions

In this section, we introduce a framework that can be used to show that theories of certain structures are decidable. We will later use this to show that the combined theory of characteristic Sturmian words is decidable. To give a first example of an application of this framework, we will use it to show that Presburger arithmetic is decidable. Afterwards, we will show that the main theorem needed to apply this framework is true. This theorem states, that certain structures are decidable. We give the precise statement below in theorem 3.3.

3.1. Decidability of Presburger arithmetic

As mentioned already in the introduction, it is well-known that Presburger arithmetic is decidable. We reformulate theorem 1.1 using the notations introduced in the previous section, that is using only relation symbols. We also add relation symbols for 0 and 1 here, although that would not not be necessary as 0 and 1 are also definable by $+$.

Proposition 3.1. *Let $S = (0, 1, +)$ be the signature of Presburger arithmetic, where $0, 1$ are one-ary relation symbols and $+$ is a 3-ary relation symbol. Let $\mathcal{M} = (\mathbb{N}, 0^{\mathcal{M}}, 1^{\mathcal{M}}, +^{\mathcal{M}})$ be the standard structure, that is $0^{\mathcal{M}} = \{0\}$, $1^{\mathcal{M}} = \{1\}$ and $+^{\mathcal{M}} = \{(x, y, z) \in \mathbb{N}^3 \mid x+y = z\}$. Then the theory of $\mathcal{M}$ is consistent, complete and decidable.*

Let's now introduce the main theorem we will use to show this proposition. To do so, we introduce the following definition beforehand.

Definition 3.2. Let Σ be finite, $M \subseteq \Sigma^\omega$, $n \in \mathbb{N}$ and $R \subseteq M^n$. For $w \in (\Sigma^n)^\omega$, let $w^{(i)} = j \mapsto w(j)_i$ be the projection of the word to its i-th component (for $1 \leq i \leq n$). Then we call a language $L \subseteq (\Sigma^n)^\omega$ *M-compatible with R* when for all $w \in (\Sigma^n)^\omega$ such that $w^{(1)}, \ldots, w^{(n)} \in M$ the following holds:

$$R\left(w^{(1)}, \ldots, R^{(n)}\right) \iff w \in L$$

The idea is to say that the language L is M-compatible with the relation R when it behaves the same way after projection for words that are projected to valid words in M. We don't care if other words are part of the language or not.

Theorem 3.3. *Let $L = (S, ar)$ be a signature. Let Σ be finite, and let $M \subseteq \Sigma^\omega$. Let $\mathcal{M} = (M, I)$ be a structure. Let M be $\omega - regular$. Furthermore, for each relation symbol $R \in I$, let there be an ω-regular language, that is M-compatible with $R^{\mathcal{M}}$. Then the theory of $\mathcal{M}$ is decidable.*

We use Buchi automata instead of deterministic or nondeterministic finite automata here. Later, when we will show that a more complicated structure is also decidable, we will need Buchi automata. It is a good opportunity now to show the concept of using Buchi automata to show the decidability of a structure in this easier example already. I have not seen a proof of the decidability of Presburger arithmetic using Buchi automata in the literature. That being said, the transformation from the proof using finite state automata, as done for example in [12, Section 6.3 Decidability of Presburger arithmetic] is straight-forward. Interestingly, one aspect of the proof is also easier: We don't need to make sure that words that only differ by a prefix or suffix of 0 are handled the same way by our automata. As our words have infinite length, there are no such alternative words. We also don't need to align words to the same length.
Let's now start with the actual proof. We assume for a moment already that theorem 3.3 is true. We will give a proof, that the theory of $\mathcal{M}$ is decidable:

Proof of proposition 3.1. The idea is to encode natural numbers as binary ω-words. Therefore, let $\Sigma = \{0, 1\}$ and let N be the language of all words that are eventually always 0:

$$N = \{w \in \Sigma^\omega \mid \{j \mid w_j \neq 0\} \text{ is finite}\}$$

We define $f\colon N \to \mathbb{N}$ by

$$f(w) = \sum_{i \in \omega} 2^i w_i.$$

f is a bijection. Furthermore, let's define the following sets:

$$\begin{aligned} 0^{\mathcal{N}} &\subseteq N & x \in 0^{\mathcal{N}} &\iff f(x) = 0 \\ 1^{\mathcal{N}} &\subseteq N & x \in 1^{\mathcal{N}} &\iff f(x) = 1 \\ +^{\mathcal{N}} &\subseteq N^3 & (x, y, z) \in +^{\mathcal{N}} &\iff f(x) + f(y) = f(z) \end{aligned}$$

Define the structure $\mathcal{N} = (N, 0^{\mathcal{N}}, 1^{\mathcal{N}}, +^{\mathcal{N}})$. $\mathcal{N}$ is isomorphic to $\mathcal{M}$ and thus its theories are the same. Thus, by theorem 3.3, it's enough to show that N is ω-regular and there are ω-regular languages that are N-compatible with $0^{\mathcal{N}}$, $1^{\mathcal{N}}$ and $+^{\mathcal{N}}$ resp. As 0 and 1 are one-ary and $(\Sigma^1)^\omega = (\Sigma^\omega)^1 = \Sigma^\omega$, $0^{\mathcal{N}}$ is N-compatible with itself and the same holds for $1^{\mathcal{N}}$. N, $0^{\mathcal{N}}$ and $1^{\mathcal{N}}$ are ω-regular by the automata given in figure 1.
It is only left to show that there is an ω-regular N-compatible language with $+^{\mathcal{N}}$. The remainder of the proof is about showing this fact. The idea is the following:

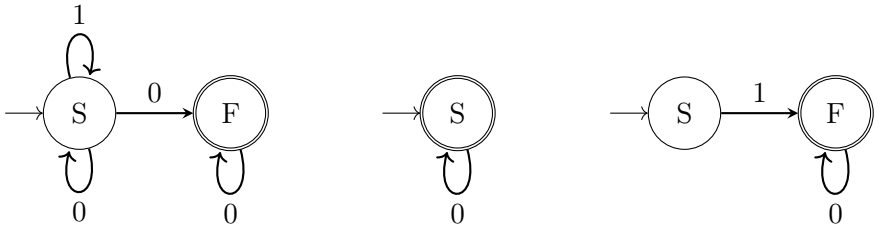

Figure 1: Buchi automata recognizing N(left), $0^{\mathcal{N}}$(middle) and $1^{\mathcal{N}}$(right)

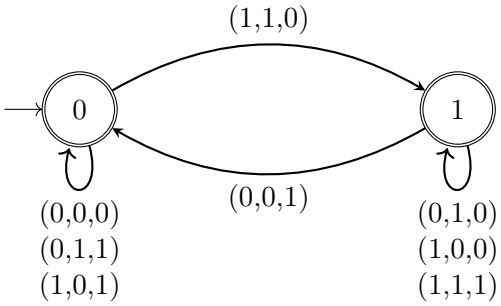

Figure 2: A Buchi automaton recognizing a language that is N-compatible with $+^{\mathcal{N}}$

1. We recursively define a sequence c_k for fixed $x, y, z \in N$. Furthermore $c_0 = 0 \iff f(x) + f(y) = f(z)$.
2. We show, that when $f(x) + f(y) = f(z)$ all c_k are in a small finite set; in our case here $c_k \in \{0, 1\}$.
3. We construct an automaton where we have one state for each element of this finite set, such that for any accepting run σ, $\sigma_i = c_i$ and conversely, if $f(x)+f(y) = f(z)$ then c is an accepting run.
4. We conclude that the language accepted by the automaton is N-compatible with $+^{\mathcal{N}}$.

Now let $x, y, z \in N$. We start by defining $(c_k)_{k\in\omega}$ by

$$c_k = \sum_{i=k}^{\infty} (z_i - x_i - y_i)2^{i-k}$$

Then $c_k \in \mathbb{Z}$, and $f(x) + f(y) = f(z) \iff c_0 = 0$. Also,

$$c_k = (z_k - x_k - y_k) + 2 \sum_{i=k+1}^{\infty} (z_i - x_i - y_i)2^{i-(k+1)} = (z_k - x_k - y_k) + 2c_{k+1}$$

$$\implies c_{k+1} = \frac{c_k + x_k + y_k - z_k}{2} \tag{3.1}$$

Now let $f(x) + f(y) = f(z)$. We show, that in this case $c_k \in \{0, 1\}$. We do this by induction over k. For $k = 0$ we have $c_k = 0$. Now let $c_k \in \{0, 1\}$. Then

$$c_{k+1} = \frac{c_k + x_k + y_k - z_k}{2} \begin{cases} \leq \frac{1+1+1-0}{2} = \frac{3}{2} \\ \geq \frac{0+0+0-1}{2} = -\frac{1}{2} \end{cases}$$

As $c_{k+1} \in \mathbb{Z}$, the only possible values inside that range are $c_{k+1} = 0$ and $c_{k+1} = 1$.
Now, we can construct the Buchi automaton with states 0 and 1 such that for any accepted run σ, $\sigma = c$. We set the transitions according to (3.1), that is for $c, c' \in \{0, 1\}$: $(c, (x, y, z), c') \in T \iff 2c' = c + x + y - z$. Additionally, 0 is the only initial state and both states 0 and 1 are accepting states. This resulting automaton is shown in figure 2. The language of this automaton is ω-regular by its construction. We still need to show, that it is N-compatible with $+^{\mathcal{N}}$. Therefore, let $w \in (\Sigma^3)^\omega$ be accepted by the automaton and let $w^{(1)}, w^{(2)}, w^{(3)} \in N$. Set $x = w^{(1)}$, $y = w^{(2)}$ and $z = w^{(3)}$. Let σ be an accepting run. As $w^{(1)}, w^{(2)}, w^{(3)} \in N$, there is $k \in \mathbb{N}$ such that $w_j = (0, 0, 0)$ for all $j \geq k$. There is only one transition labeled $(0, 0, 0)$ (the unique solution to $2c' = c$ with $c, c' \in \{0, 1\}$) which starts and ends in state 0. Thus, $\sigma_j = 0$ for all $j \geq k$. By definition, we also have $c_j = 0$ for $j \geq k$ (where c is the sequence on x, y, z). Because c_{j-1} is uniquely defined by (3.1) and the transitions of the automaton are defined according to the same formula, we have $c_j = \sigma_j \Rightarrow c_{j-1} = \sigma_{j-1}$. By induction, it follows that $c = \sigma$. As the only initial state of the automaton is 0, we have $c_0 = \sigma_0 = 0$ and thus $f\big(w^{(1)}\big) + f\big(w^{(2)}\big) = f\big(w^{(3)}\big)$. Conversely, let $w \in (\Sigma^3)^\omega$ be such that $w^{(1)}, w^{(2)}, w^{(3)} \in N$ and $f\big(w^{(1)}\big) + f\big(w^{(2)}\big) = f\big(w^{(3)}\big)$. Set $x = w^{(1)}, y = w^{(2)}, z = w^{(3)}$. We need to show, that c is an accepting run. We start by showing that it is a run by noting that $c_0 = 0$, 0 is an initial state and the transitions of the automaton are defined according to (3.1). It is also an accepting run as all states are accepting. □

3.2. Decidability of ω-regular structures

We want to turn back to theorem 3.3 now. Therefore, let (S, ar) be a signature, Σ be a finite set, $M \subseteq \Sigma^\omega$ be ω-regular and let $\mathcal{M}$ be an (S, ar)-structure. Furthermore, for any $R \in S$ let $L_R \subseteq \left(\Sigma^{ar(R)}\right)^\omega$ be an M-compatible ω-regular language with $R^{\mathcal{M}}$. In this setup, we recursively define Buchi automata for formulas.
Let $\varphi \in F(L)$ and let $(i_j)_{j=1,\dots,n}$ be a strictly ascending sequence (that is $i_1 < i_2 < \dots < i_n$) such that $\text{free}(\varphi) = \{v_{i_1}, \dots, v_{i_n}\}$. We define a Buchi automaton $B(\varphi)$ over Σ^n, s.t. for all $w \in (\Sigma^n)^\omega$ with $w^{(1)}, \dots, w^{(n)} \in M$ the following invariant holds:

$$w \text{ is accepted by } B(\varphi) \text{ if and only if } \mathcal{M}\frac{w^{(1)}, \dots, w^{(n)}}{v_{i_1}, \dots, v_{i_n}} \models \varphi \tag{3.2}$$

We construct these recursively, starting with atomic formulas. For each of the constructions below, it can be verified that (3.2) holds.

- For formulas of the form $v_j \equiv v_j$ we take the automaton, that consists of just one state and accepts any word:

$$B(v_j \equiv v_j) := (\{s_1\}, \{s_1\}, \{(s_1, c, s_1) : c \in \Sigma\}, \{s_1\})$$

- For formulas of the form $v_j \equiv v_k$ for $j \neq k$, we take the automaton, that consists of just one state and accepts all words consisting of letters of the form (c, c) for $c \in \Sigma$:
$$B(v_j \equiv v_k) := (\{s_1\}, \{s_1\}, \{(s_1, (c, c), s_1) : c \in \Sigma\}, \{s_1\})$$

- Now let $R \in S$ be a relation symbol with arity m. Let $\varphi = R(v_{i_{k_1}}, \ldots, v_{i_{k_m}})$. Then $\{k_1, \ldots, k_m\} = \{1, \ldots, n\}$. Let $D = (S, I, T, F)$ be the Buchi automaton recognizing L_R. Let
$$T' = \{(s_1, (c_1, \ldots, c_n), s_2) : (s_1, (c_{k_1}, \ldots, c_{k_m}), s_2) \in T\}$$
We set $B(\varphi) := (S, I, T', F)$.

- Let $\varphi, \psi \in F(L)$. Let $v_{i_1}, \ldots, v_{i_n}$ be the free variables in any of these formulas with $i_1 < \cdots < i_n$. Let $v_{i_{j_1}}, \ldots, v_{i_{j_k}}$ be the free variables of φ with $1 \leq j_1 < \cdots < j_k \leq n$. Let $B(\varphi) = (S_1, I_1, T_1, F_1)$. We construct a Buchi automaton $B' = (S_1, I_1, T_1', F_1)$ for φ over Σ^n by
$$T_1' = \{(s_1, (c_1, \ldots, c_n), s_2) : (s_1, (c_{j_1}, \ldots, c_{j_k}), s_2) \in T_1\}$$
Intuitively, B' is the Buchi automaton recognizing φ with all free variables in φ *and* ψ as input. By the same construction, let B'' be the Buchi automaton recognizing ψ with all these free variables as input. B' and B'' are both Buchi automata over Σ^n. Thus by proposition 2.8, we can construct Buchi automata $B^\cup$ and $B^\cap$ with $\mathcal{L}(B^\cup) = \mathcal{L}(B') \cup \mathcal{L}(B'')$ and $\mathcal{L}(B^\cap) = \mathcal{L}(B') \cap \mathcal{L}(B'')$. We can now define the automata for $\varphi \vee \psi$ and $\varphi \wedge \psi$:
$$B(\varphi \vee \psi) := B^\cup$$
$$B(\varphi \wedge \psi) := B^\cap$$

- By proposition 2.8 we can also construct the automaton $B^\neg$, that recognizes exactly the words not recognized by $B(\varphi)$. We set
$$B(\neg\varphi) := B^\neg.$$

- Now let φ be a formula with free variables $v_{i_1}, \ldots, v_{i_n}$ with $i_1 < \cdots < i_n$. We want to construct the automaton for $\exists v_k \varphi$. First, let's consider the case where k is not among $i_1, \ldots, i_n$. In this case, v_k does not appear freely in φ. Thus, we can just set
$$B(\exists v_k \varphi) := B(\varphi).$$
Next, let $k \in \{i_1, \ldots, i_n\}$. Let $k = i_j$. Our idea is to just "forget" the j-th component of our words, as we don't care about the concrete word in that component, there just needs to be some word. But there is one complication: The word also needs to be $\in M$. So it is slightly more complicated. Therefore, let $U = (S_U, I_U, T_U, F_U)$ be an automaton recognizing M and let $U' =$

$(S_U, I_U, \{(s_1, (c_1, \ldots, c_n), s_2) : (s_1, c_j, s_2) \in T_U\}, F_U)$. Let $D = (S, I, T, F)$ be an automaton that recognizes words that are recognized by both $B(\varphi)$ and U'. Finally, we define $B(\exists v_k \varphi) := (S, I, T', F)$ with

$$T' = \{(s_1, (c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n), s_2) : \exists c_j \in \Sigma(s1, (c1, \ldots, c_n), s_2) \in T\}.$$

All these constructions are constructive and can be done in an algorithm. So we can now give an algorithm to decide if $\mathcal{M} \models \varphi$.

Definition 3.4. We define the following algorithm, that decides if $\mathcal{M} \models \varphi$.
Input: L-sentence φ
Output: true when $\mathcal{M} \models \varphi$, false otherwise
Algorithm:

Calculate $B(\varphi) = (S, I, T, F)$. This is a Buchi automaton over the alphabet $\Sigma^0 = \{()\}$. Thus all edges are annotated with the same letter.

Let $G = (V, E)$ be a directed graph with vertices $V = S$ and edges $E = \{(s, t) : (s, (), t) \in T\}$.

Calculate the vertices R, that are reachable from I. This can be done for example by doing a breadth-first search. Set $P = R \cap F$.

For each vertex $v \in P$, check if there's a cycle from v to itself. This can also be done by a breadth-first search. If that's the case for any such vertex, return true.

If there is no such cycle, return false.

4. Ostrowski numeration systems

In the base-n system, the factor of a digit in position k is n^k which is a multiple of the factor of the digit at position $k-1$. The Ostrowski numeration system is an alternative numeration system, where the factor is instead a sum of the factors at the digits at position $k-1$ and $k-2$.

4.1. Definition of Ostrowski representations

Definition 4.1. Let $B = \mathbb{N}^+ \to \mathbb{N}^+$ be the set of sequences of positive integers. For $b \in B$ also write b_i instead of $b(i)$. For such a sequence b, we define sequences p_i and q_i recursively as follows[5]

$$\begin{aligned} p_{-1}(b) &= 1 & q_{-1}(b) &= 0 \\ p_0(b) &= 0 & q_0(b) &= 1 \\ p_k(b) &= b_k p_{k-1}(b) + p_{k-2}(b) & q_k(b) &= b_k q_{k-1}(b) + q_{k-2}(b) \end{aligned}$$

[5] I've taken these equations from [2, theorem 2.4.1] and [12, lemma 3.5.6]. These books define these sequences as the numerator and denominator of continued fractions and this definition is a deduction of that definition. Instead, we define these sequences directly in this way here.

We define the language $L'_b \subseteq \mathbb{N}^\omega$ as the set of words $w \in \mathbb{N}^\omega$, that have the following properties:

1. $\{j \in \omega \mid w_j \neq 0\}$ is finite.
2. for all $i \in \omega$, $w_i \leq b_{i+1}$.

Let $L_b \subseteq L'_b$ be the language where each word additionally has the following properties:

3. $w_0 < b_1$
4. For all $i \geq 1$ if $w_i = b_{i+1}$ then $w_{i-1} = 0$.

Define $f_b \colon L'_b \to \mathbb{N}$ as

$$f_b(w) = \sum_j w_j q_j(b) \tag{4.1}$$

f_b is well-defined because of property 1. We will now show, that $f_b \colon L_b \to \mathbb{N}$ is a bijection. An alternative proof of this theorem can be found in [2, theorem 3.9.1].

Lemma 4.2. *Let $u \in L_b$ and $k \in \mathbb{N}$ such that $u_m = 0$ for all $m \geq k$. Then $f_b(u) < q_k$.*

Proof. Prove by induction over k. For $k = 0$ $u = 0^\omega$, so $f_b(u) = 0 < 1 = q_0$. For $k = 1$ we can write $u = u_0 0^\omega$, so $f_b(u) = u_0 < b_1 = q_1$. Now let $k \geq 1$ and let the statement be true for k and $k-1$. We show the statement for $k+1$. Therefore, let u be as defined in the statement. We separate the following cases:

Case 1: $u_k = b_{k+1}$. Then $u_{k-1} = 0$ by property 4 and thus

$$f_b(u) < q_{k-1} + u_k q_k = q_{k+1}$$

Case 2: $u_k < b_{k+1}$. Then $u_k \leq b_{k+1} - 1$ and thus

$$f_b(u) < q_k + (b_{k+1} - 1)q_k = b_{k+1}q_k = q_{k+1} - q_{k-1} \leq q_{k+1}$$

□

Theorem 4.3. *$f_b : L_b \to \mathbb{N}$ is a bijection.*

Proof. We start by showing surjectivity. We show inductively over $n \in \mathbb{N}$, that there is $w \in L_b$, such that $f_b(w) = n$. $n = 0$ is true by choosing $w = 0^\omega \in L_b$.
Now let the statement be true for $n-1$. Let $u \in L_b$ be an element, such that $f_b(u) = n-1$. We now define the sequence $u^{(0)}, u^{(1)}, \cdots \in L'_b$ by

$$u_j^{(i)} = \begin{cases} u_j & j > i + \delta_{u_i b_{i+1}} \\ u_j + 1 & j = i + \delta_{u_i b_{i+1}} \\ 0 & j < i + \delta_{u_i b_{i+1}} \end{cases}$$

where $\delta_{xy} = 0$ if $x \neq y$ and $\delta_{xy} = 1$ if $x = y$.

First, let's verify that $u^{(i)} \in L'_b$. Property 1 clearly holds for $u^{(i)}$, as property 1 holds for u and the sequences $u^{(i)}$ and u only differ in one position. If $u_i < b_{i+1}$, property 2 also clearly holds. If $u_i = b_{i+1}$ then $u_{i+1} < b_{i+2}$ by property 4 and thus property 2 also holds in this case. Together, $u^{(i)} \in L'_b$.
Now let k be minimal, such that $u^{(k)} \in L_b$. Such a k exists as $u^{(\max_j\{u_j \neq 0\}+2)} \in L_b$. We will now show inductively for all $l \leq k$, that

$$f_b\Big(u^{(l)}\Big) = n \tag{4.2}$$

To see this for $l = 0$, note that $u_0 < b_1$ and thus $\delta_{u_0 b_1} = 0$.
Now let $l = 1$ and $k \geq 1$. Then $u^{(0)} \notin L_b$, so $u_0^{(0)} = b_1$ or $u_1^{(0)} = b_2$ (or both). Let's look at these two cases separately (considering the both case as part of case 1).

Case 1 $u_1^{(0)} = b_2$. By property 4 of $u \in L_b$ and $u_1 = u_1^{(0)} = b_1$ we can follow, that $u_0 = 0$. Thus, $u_0^{(0)} = u_0 + 1 = 1$ and

$$\begin{aligned} f_b(u^{(1)}) &= (u_2+1)q_2 + \sum_{j\geq 3} q_j u_j \\ &= \overbrace{\underbrace{1}_{=u_0^{(0)}} q_0 + \underbrace{b_1}_{=u_1^{(0)}} q_1}^{q_2=} + u_2 q_2 + \sum_{j\geq 3} q_j u_j \\ &= f_b(u^{(0)}) = n \end{aligned}$$

Case 2 $u_0^{(0)} = b_1$ and $u_1^{(0)} < b_2$. Then we have $q_1 = q_{-1} + b_1 q_0 = b_1 = u_0^{(0)} q_0$ and

$$\begin{aligned} f_b(u^{(1)}) &= (u_1+1)q_1 + \sum_{j\geq 2} q_j u_j \\ &= u_0^{(0)} q_0 + u_1 q_1 + \sum_{j\geq 2} q_j u_j \\ &= f_b(u^{(0)}) = n \end{aligned}$$

So (4.2) holds for $l = 1$.
Now let $k \geq l \geq 2$ and let the statement hold for $l-1$. Then again $u^{(l-1)} \notin L_b$ (as otherwise $k \leq l-1$). We distinguish the following two cases:

Case 1 $u_{l-1} = b_l$. In this case $u_l < b_{l+1}$ by property 4 of u and thus $u^{(l)} = u^{(l-1)} = n$.

Case 2 $u_{l-1} < b_l$. Then $u_{l-1} = 0$ and $u_l = b_{l+1}$ as otherwise $u^{(l-1)} \in L_b$. So $u_{l-1}^{(l-1)} = 1, u_l^{(l-1)} = u_l = b_{l+1}$ and

$$\begin{aligned} f_b\Big(u^{(l)}\Big) &= (u_{l+1}+1)q_{l+1} + \sum_{j\geq l+2} u_j q_j = q_{l+1} + \sum_{j\geq l+1} u_j q_j \\ &= 1 \cdot q_{l-1} + u_l q_l + \sum_{j\geq l+1} u_j q_j = f_b\Big(u^{(l-1)}\Big) = n \end{aligned}$$

	1	2	i	$((12)^\omega)_i$	2^i
1	01	1	01	1	1
2	001	01	02	01	01
3	0001	11	001	001	11
4	0101	02	011	101	02
5	00001	001	021	011	12
6	01001	101	002	002	03
7	00101	011	012	102	13
8	000001	111	022	0001	04
9	010001	021	003	1001	001
10	001001	002	0001	0101	101
11	000101	102	0101	00001	011
12	010101	0001	0201	10001	111
13	0000001	1001	0011	01001	021

Figure 3: Examples of small numbers in different Ostrowski numeration systems $b_i = 1, b_i = 2, b_i = i$, $b_i = ((12)^\omega)_i$ and $b_i = 2^i$

So (4.2) holds for l as well. This completes the inner induction and thus (4.2) holds for all $l \leq k$. Thus $u^{(k)} \in L_b$ and $f_b(u^{(k)}) = n$, which completes the proof of the outer induction. In total, we have shown the surjectivity of f_b.
It's left to show that f_b is injective. Therefore, let's assume that f_b is not injective. Then there are $u, w \in L_b$ with $f_b(u) = f_b(w)$ and $u \neq w$. Let k be maximal, such that $u_k \neq w_k$. Without loss of generality, let $u_k < w_k$. Define u' and w' by

$$u'_j = \begin{cases} u_j & j < k \\ 0 & j = k \\ 0 & j \geq k \end{cases} \qquad w'_j = \begin{cases} w_j & j < k \\ w_k - u_k & j = k \\ 0 & j > k \end{cases}$$

Then u', $w' \in L_b$ and $f_b(u') = f_b(w')$ by construction. But by lemma 4.2, $f_b(u') < q_k$; and by equation (4.1) $f_b(w') \geq (w'_k - u'_k)q_k \geq q_k$. This contradicts the assumption, so f_b is injective. Thus, f_b is both injective and surjective. □

Let's now look at some examples. In figure 3 we can see how small numbers are represented in a few Ostrwoski representations. Each cell contains the first letters of the corresponding word without any 0s in the end. The Ostrowski representation $b_i = 1$ is also called Fibonacci representation and the representation $b_i = 2$ is called Pell representation[12, Section 3]. We represent them in least-significant-digit first notation here.[6]

4.2. Sturmian words

Sturmian words are sequences, that can be defined in the following way:

[6] In the literature I reviewed, the use of most-significant-first notation was more common. We use least-significant-digit first notation, as this is the only way to represent them as infinite words.

Definition 4.4. Let $\alpha \in (0,1) \setminus \mathbb{Q}$. The characteristic Sturmian word with slope α is the ω-word $c_\alpha \in \{0,1\}^\omega$, defined as follows:

$$c_\alpha(n) = \lfloor \alpha(n+1) \rfloor - \lfloor \alpha n \rfloor$$

The goal of this subsection is to show a relationship between characteristic Sturmian words and Ostrowski representations.

Proposition 4.5. *For each $\alpha \in (0,1) \setminus \mathbb{Q}$ there is a unique $b \in B$, such that*

$$\lim_{i \to \infty} \frac{p_i(b)}{q_i(b)} = \alpha$$

Furthermore, for such α and corresponding b there is the following relationship between c_α and f_b: for all $n \in \mathbb{N}^+$

$$c_\alpha(n) = 1 \iff \min_i \{ f_b^{-1}(n)_i > 0 \} \text{ is odd}$$

This theorem is also shown in [2, theorem 9.1.15]. Towards the goal of showing proposition 4.5, let's start with some facts. In the following, we sometimes write p_i and q_i instead of $p_i(b)$ and $q_i(b)$ when b is clear from the context.

Lemma 4.6. *Let $b \in B$. Then for every i the following equation holds:*

$$p_i q_{i-1} - p_{i-1} q_i = (-1)^{i+1} \tag{4.3}$$

Proof. We show this by induction over i. For $i = 0$, we have $0 \cdot 0 - 1 \cdot 1 = -1$. Now, let the formula be true for m. Then the formula holds for $m+1$ as well, as the following calculation shows:

$$\begin{aligned} p_{m+1} q_m - p_m q_{m+1} &= b_{m+1} p_m q_m + p_{m-1} q_m - b_{m+1} p_m q_m - p_m q_{m-1} \\ &= -\left(p_m q_{m-1} - p_{m-1} q_m \right) \\ &= -(-1)^{m+1} \qquad \text{by induction hypothesis} \\ &= (-1)^{m+2} \end{aligned}$$

□

Lemma 4.7. *Let k be even. Then, for all $l > k$, the following equation holds*

$$\frac{p_k}{q_k} < \frac{p_l}{q_l} \tag{4.4}$$

Similarly, let k be odd. Then for all $l > k$ the following holds:

$$\frac{p_k}{q_k} > \frac{p_l}{q_l} \tag{4.5}$$

Proof. We use lemma 4.6 and divide both sides of (4.3) by $q_k q_{k-1}$. This leads to the following equation

$$\frac{p_i}{q_i} - \frac{p_{i-1}}{q_{i-1}} = \frac{(-1)^{i+1}}{q_i q_{i-1}} \tag{4.6}$$

As a first step, let $l-k$ be even. Then the following sum has an even number of summands, and we can thus make groups of two terms that are always positive or negative:

$$\begin{aligned}\frac{p_l}{q_l} - \frac{p_k}{q_k} &= \sum_{i=1}^{l-k} \frac{(-1)^{k+i+1}}{q_{k+i}q_{k+i-1}} \\ &= \sum_{\substack{i=1 \\ 2|i}}^{l-k} \underbrace{(-1)^{k+i+1}}_{=(-1)^{k+1}} \underbrace{\left(-\frac{1}{q_{k+i-1}q_{k+i-2}} + \frac{1}{q_{k+i}q_{k+i-1}}\right)}_{<0} \\ &\begin{cases} > 0 & 2 \mid k \\ < 0 & 2 \nmid k \end{cases}\end{aligned} \tag{4.7}$$

Now let $l-k$ be odd. In this case $l+1$ has the same parity as k and thus

$$\begin{aligned}\frac{p_l}{q_l} - \frac{p_k}{q_k} = \underbrace{\frac{p_{l-1}}{q_{l-1}} - \frac{p_k}{q_k}}_{\begin{cases} > 0 & 2 \mid k \\ < 0 & 2 \nmid k \end{cases}} + \underbrace{\frac{(-1)^{l+1}}{q_l q_{l-1}}}_{\begin{cases} > 0 & 2 \mid k \\ < 0 & 2 \nmid k \end{cases}} \\ \begin{cases} > 0 & 2 \mid k \\ < 0 & 2 \nmid k \end{cases}\end{aligned}$$

This is the same inequality as (4.7), which thus holds in all cases. It is also an equivalent formulation of (4.4) and (4.5). □

Definition 4.8. Let $b \in B$. Then define

$$\boldsymbol{\alpha}(b) = \lim_{k\to\infty} \frac{p_k(b)}{q_k(b)}$$

Lemma 4.9. $\boldsymbol{\alpha}$ *is well-defined*

Proof. Using lemma 4.6 and dividing both sides by $q_k q_{k-1}$ we have

$$\frac{p_k}{q_k} - \frac{p_{k-1}}{q_{k-1}} = \frac{(-1)^n}{q_k q_{k-1}}$$

By definition, q_k is strictly monotonically increasing, so $\frac{1}{q_k q_{k-1}}$ is strictly monotonically decreasing, and because $q_k \in \mathbb{N}$ its limit is 0. Furthermore, by (4.4) and (4.5), $\{p_i/q_i \mid i > k\} \subseteq (p_k/q_k, p_{k-1}/q_{k-1})$. Thus, p_k/q_k is a Cauchy sequence and thus converges. □

Definition 4.10. Let $\alpha \in (0,1) \setminus \mathbb{Q}$. Define the sequence $(\alpha_i)_{i\in\omega}$ recursively by

$$\alpha_0 = \alpha \tag{4.8}$$

$$\alpha_{i+1} = \frac{1}{\alpha_i} - \left\lfloor \frac{1}{\alpha_i} \right\rfloor \tag{4.9}$$

Lemma 4.11. *$(\alpha_i)_{i\in\omega}$ is well-defined and $\alpha_i \in (0,1)\setminus\mathbb{Q}$ for all i.*

Proof. We show this by induction over i. For $i = 0$, the statement is true by definition. Let the statement now be true for i. To show, that α_{i+1} is well-defined, we need to show, that we don't divide by zero in (4.9). That is, $\alpha_i \neq 0$. But this holds already by the induction hypothesis. Furthermore, as α_i is irrational, so is $\frac{1}{\alpha_i}$ and after subtracting an integer, that property is still true. Thus α_{i+1} is also irrational. Also, by definition $\alpha_{i+1} \in [0,1)$; and because of irrationality it is also in the slightly smaller interval $(0,1)$. □

Theorem 4.12. *$\boldsymbol{\alpha} : B \to (0,1)\setminus\mathbb{Q}$ is a bijection.*

Proof. We start by showing that $\boldsymbol{\alpha}$ is surjective. Therefore, let $\alpha \in (0,1)\setminus\mathbb{Q}$. Let α_i be defined as in definition 4.10. Define $(b_i)_{i\in\omega}$ by

$$b_{i+1} = \left\lfloor \frac{1}{\alpha_i} \right\rfloor \tag{4.10}$$

We now show the following statement by induction over i:

$$\alpha = \frac{p_i + \alpha_i p_{i-1}}{q_i + \alpha_i q_{i-1}}$$

For $i = 0$ we have

$$\frac{p_0 + \alpha_0 p_{-1}}{q_0 + \alpha_0 q_{-1}} = \frac{0 + \alpha \cdot 1}{1 + \alpha \cdot 0} = \alpha$$

Now let the statement be true for i. We show, that the statement is also true for $i+1$. By (4.9) and (4.10), $\alpha_i^{-1} = \alpha_{i+1} + b_{i+1}$ and thus

$$\begin{aligned} \frac{p_{i+1} + \alpha_{i+1} p_i}{q_{i+1} + \alpha_{i+1} q_i} &= \frac{\overbrace{(b_{i+1} + \alpha_{i+1})}^{=\alpha_i^{-1}} p_i + p_{i-1}}{\underbrace{(b_{i+1} + \alpha_{i+1})}_{=\alpha_i^{-1}} q_i + q_{i-1}} && \text{expand by } \alpha_i \\ &= \frac{p_i + \alpha_i p_{i-1}}{q_i + \alpha_i q_{i-1}} = \alpha \end{aligned}$$

So $\alpha \in \left[\frac{p_i}{q_i}, \frac{p_{i-1}}{q_{i-1}}\right]$. As the sequence $\frac{p_i}{q_i}$ converges by lemma 4.9, α is its limit. In other words $\boldsymbol{\alpha}(b) = \alpha$ and thus $\boldsymbol{\alpha}$ is surjective.
We still have to show, that $\boldsymbol{\alpha}$ is injective. Therefore, let $b, c \in B$ and let $b \neq c$. Let k be minimal, such that $b_k \neq c_k$. Without loss of generality, let $b_k > c_k$. Clearly $p_j(b) = p_j(c)$ and $q_j(b) = q_j(c)$ for $j < k$. We also write p_j and q_j without parameter for such j in what follows.

Then

$$\begin{aligned} p_k(b)q_{k+1}(c) - p_{k+1}(c)q_k(b) &= p_k(b)(c_{k+1}q_k(c) + q_{k-1}) - (c_{k+1}p_k(c) + p_{k-1})q_k(b) \\ &= c_{k+1}(p_k(b)q_k(c) - p_k(c)q_k(b)) + \underbrace{p_k(b)q_{k-1} - p_{k-1}q_k(b)}_{=(-1)^{k+1}} \end{aligned}$$

Calculating the subterm further, we get

$$\begin{aligned} &p_k(c)q_k(b) - q_k(c)p_k(b) \\ =&(c_kp_{k-1} + p_{k-2})(b_kq_{k-1} + q_{k-2}) - (c_kq_{k-1} + q_{k-2})(b_kp_{k-1} + p_{k-2}) \\ =&(c_k - b_k)p_{k-1}q_{k-2} + (b_k - c_k)p_{k-2}q_{k-1} \\ =&(c_k - b_k)(p_{k-1}q_{k-2} - p_{k-2}q_{k-1}) \\ =&(c_k - b_k)(-1)^k \end{aligned}$$

Putting that into the original equation, we get

$$\begin{aligned} p_k(b)q_{k+1}(c) - p_{k+1}(c)q_k(b) &= c_{k+1}(b_k - c_k)(-1)^k + (-1)^{k+1} \\ &= (-1)^k(c_{k+1}(b_k - c_k) - 1) \end{aligned}$$

And dividing by $q_{k+1}(c)q_k(b)$ leads to

$$\frac{p_k(b)}{q_k(b)} - \frac{p_{k+1}(c)}{q_{k+1}(c)} = (-1)^k \underbrace{(\overbrace{c_{k+1}}^{\geq 1}\overbrace{(b_k - c_k)}^{\geq 1} - 1)}_{\geq 0} \underbrace{\frac{1}{q_{k+1}(c)q_k(b)}}_{>0} \tag{4.11}$$

Case 1 k even. Then by lemma 4.7 $\frac{p_k(b)}{q_k(b)} < \boldsymbol{\alpha}(b)$ and $\frac{p_{k+1}(c)}{q_{k+1}(c)} > \boldsymbol{\alpha}(c)$ and by (4.11), $\frac{p_k(b)}{q_k(b)} \geq \frac{p_{k+1}(c)}{q_{k+1}(c)}$, so $\boldsymbol{\alpha}(b) > \boldsymbol{\alpha}(c)$.

Case 2 k odd. Then by lemma 4.7 $\frac{p_k(b)}{q_k(b)} > \boldsymbol{\alpha}(b)$ and $\frac{p_{k+1}(c)}{q_{k+1}(c)} < \boldsymbol{\alpha}(c)$ and by (4.11), $\frac{p_k(b)}{q_k(b)} \leq \frac{p_{k+1}(c)}{q_{k+1}(c)}$, so $\boldsymbol{\alpha}(b) < \boldsymbol{\alpha}(c)$.

Thus in all cases $\boldsymbol{\alpha}(b) \neq \boldsymbol{\alpha}(c)$, so $\boldsymbol{\alpha}$ is injective. □

Definition 4.13. Let $b \in B$. Define the k-th difference $\beta_k(b) = q_k(b)\boldsymbol{\alpha}(b) - p_k(b)$ for $k \in \omega$ and define $g_b \colon L_b \to \mathbb{R}$ by

$$g_b(w) = \sum_i w_i\beta_i(b)$$

Lemma 4.14. *Let $b \in B$. Let α_i be defined as in definition 4.10 with $\alpha = \boldsymbol{\alpha}(b)$. Then for all $k \geq -1$*

$$\beta_k(b) = (-1)^k \prod_{j=0}^{k} \alpha_j$$

Proof. We prove this by induction over k. For $k = -1$, $\beta_{-1} = 0 \cdot \alpha - 1 = -1 = (-1)^{-1}$ and for $k = 0$, $\beta_0 = q_0\alpha - p_0 = \alpha = (-1)^0\alpha_0$. Now let $k \geq 0$ and let the statement be true for k and $k-1$. By (4.10) and (4.9), $b_{k+1} = \frac{1}{\alpha_k} - \alpha_{k+1}$. Then

$$\begin{aligned}
\beta_{k+1} &= q_{k+1}\alpha - p_{k+1} \\
&= q_k b_{k+1}\alpha + q_{k-1}\alpha - p_k b_{k+1} - p_{k-1} \\
&= b_{k+1}(q_k\alpha - p_k) + (q_{k-1}\alpha - p_{k-1}) \\
&= b_{k+1}\beta_k + \beta_{k-1} \\
&= \frac{1}{\alpha_k}\beta_k - \alpha_{k+1}\beta_k + \beta_{k-1}
\end{aligned}$$

Applying the induction hypothesis on k and $k-1$ we have $\beta_k = -\beta_{k-1}\alpha_k$. Putting this into the equation we can transform it further in the following way:

$$\begin{aligned}
&= \beta_{k-1}(-1 + \alpha_k\alpha_{k+1} + 1) \\
&= \beta_{k-1}\alpha_k\alpha_{k+1}
\end{aligned}$$

Applying the induction hypothesis on $k-1$ on this shows the equation for $k+1$ which concludes the induction step. □

Proposition 4.15. *Let $b \in B$. Then $g_b(L_b) \subseteq [-\boldsymbol{\alpha}(b), 1 - \boldsymbol{\alpha}(b))$*

Proof. Let $w \in L_b$. Let $\alpha = \boldsymbol{\alpha}(b)$ and we also write β_i for $\beta_i(b)$. As $\beta_i \geq 0$ for even i and $\beta_i \leq 0$ for odd i, we can estimate $g_b(w)$ in the following way

$$\sum_{i=0}^{\infty} w_{2i+1}\beta_{2i+1} \leq g_b(w) \leq \sum_{i=0}^{\infty} w_{2i}\beta_{2i}$$

Furthermore

$$b_{i+1}\beta_i = \left(-\alpha_{i+1} + \frac{1}{\alpha_i}\right)\beta_i = \beta_{i+1} - \beta_{i-1} \tag{4.12}$$

Thus

$$\begin{aligned}
\sum_{i=0}^{\infty} w_{2i+1}\beta_{2i+1} &> \sum_{i=0}^{\infty} b_{2i+2}\beta_{2i+1} \\
&= \lim_{k\to\infty} \sum_{i=0}^{k} b_{2i+2}\beta_{2i+1} \\
&= \lim_{k\to\infty} (\beta_{2k+2} - \beta_0) \\
&= (\lim_{k\to\infty} \underbrace{\beta_{2k+2}}_{\geq 0}) - \alpha \\
&\geq -\alpha
\end{aligned}$$

and

$$\begin{aligned}\sum_{i=0}^{\infty} w_{2i}\beta_{2i} &< (b_1-1)\beta_0 + \lim_{k\to\infty}\sum_{i=1}^{k} b_{2i+1}\beta_{2i} \\ &= (b_1-1)\beta_0 + \lim_{k\to\infty}\underbrace{\beta_{2k+1}}_{\leq 0} - \beta_1 \\ &\leq \underbrace{b_1\beta_0}_{\beta_1-\beta_{-1}} - \beta_0 - \beta_1 \\ &= -\beta_{-1} - \beta_0 = 1-\alpha\end{aligned}$$ □

Remark 4.16. It is possible to extend L_b to some $M_b \supseteq L_b$ and extend the domain of g_b to that set, such that $g_b\colon M_b \to [-\boldsymbol{\alpha}(b), 1-\boldsymbol{\alpha}(b))$ is a bijection, see [5, fact 2.7]. This fact can be used to generalize the statements here about characteristic Sturmian words to all Sturmian words as has been done in [5]. We don't do this in this thesis and restrict ourselves to characteristic Sturmian words.

Lemma 4.17. $\lim_{i\to\infty}\beta_i = 0$

Proof. We show that for all $i \in \omega$ the product $\alpha_i\alpha_{i+1} \leq \frac{1}{2}$. Together with lemma 4.14 this is enough to show the statement of this lemma. We distinguishing two cases:

Case 1 $\alpha_i \leq \frac{1}{2}$. As $\alpha_{i+1} \leq 1$ by lemma 4.9, the statement follows immediately.

Case 2 $\alpha_i > \frac{1}{2}$. Then $\frac{1}{\alpha_i} < 2$, so $\left\lfloor \frac{1}{\alpha_i} \right\rfloor = 1$. Then

$$\alpha_i\alpha_{i+1} = \alpha_i\left(\frac{1}{\alpha_i} - \underbrace{\left\lfloor \frac{1}{\alpha_i} \right\rfloor}_{=1}\right) = 1-\alpha_i < \frac{1}{2}$$ □

Lemma 4.18. *Let $b \in B$, $w \in L_b$ and $f_b(w) > 0$. Let $k = \min_i\{w_i \neq 0\}$. Then*

$$g_b(w) < 0 \iff k \text{ is odd}$$

and

$$g_b(w) > 0 \iff k \text{ is even}$$

Proof. We start by noting that by applying (4.12) in the following sum we get

$$\begin{aligned}\sum_{i=0}^{\infty} b_{k+1+2i+1}\beta_{k+1+2i} &= \sum_{i=0}^{\infty}(-\beta_{k+2i} + \beta_{k+2i+2}) \\ &= -\beta_k + \lim_{i\to\infty}\beta_{k+2i+2} = -\beta_k\end{aligned}$$

The last equation holds, because $\lim_{i\to\infty} \beta_i = 0$.
First, let k be odd. In this case, the two equivalences simplify to $g_b(w) < 0$ and $g_b(w) \leq 0$ respectively. The second inequality follows easily from the first, so it is enough to show $g_b(w) < 0$ in this case. Indeed, this is the case:

$$\begin{aligned} g_b(w) &= \underbrace{w_k}_{\geq 1}\underbrace{\beta_k}_{<0} + \underbrace{\sum_{i=0}^{\infty} \overbrace{w_{k+1+2i}}^{\leq b_{k+1+2i+1}} \beta_{k+1+2i}}_{<-\beta_k} + \underbrace{\sum_{i=0}^{\infty} w_{k+2+2i}\beta_{k+2+2i}}_{\leq 0} \\ &< \beta_k - \beta_k + 0 = 0 \end{aligned}$$

The inequality is strict because $w_{k+1+2i} = 0 < b_{k+1+2i+1}$ for large enough i (because of property 1 of definition 4.1). The case where k is even can be shown similarly. In that case, we need to show $g_b(w) > 0$ and this is the case by an analogous calculation:

$$\begin{aligned} g_b(w) &= \underbrace{w_k}_{\geq 1}\underbrace{\beta_k}_{>0} + \underbrace{\sum_{i=0}^{\infty} \overbrace{w_{k+1+2i}}^{\leq b_{k+1+2i+1}} \beta_{k+1+2i}}_{>-\beta_k} + \underbrace{\sum_{i=0}^{\infty} w_{k+2+2i}\beta_{k+2+2i}}_{\geq 0} \\ &> \beta_k - \beta_k + 0 = 0 \end{aligned}$$

□

Lemma 4.19. *Let $b \in B, w \in L_b$. Let $\alpha = \boldsymbol{\alpha}(b)$. Then*

$$g_b(w) - \alpha f_b(w) \in \mathbb{Z}$$

Proof. We note, that $\beta_i - \alpha q_i = q_i\alpha - p_i - \alpha q_i = -p_i \in \mathbb{Z}$. Thus

$$g_b(w) - \alpha f_b(w) = \sum_{j=0}^{\infty} w_i(\beta_i(b) - \alpha q_i(b)) \in \mathbb{Z}$$

□

Proposition 4.20. *For all $b \in B$ and $n \in \mathbb{N}^+$ the following equivalence holds:*

$$c_{\boldsymbol{\alpha}(b)}(n) = 1 \iff g_b(n) < 0$$

.

Proof. Let $\alpha = \boldsymbol{\alpha}(b)$. Note, that $g_b(n) < 0 \iff \lfloor g_b(n) \rfloor = -1$, so we need to show, that

$$c_\alpha(n) = -\lfloor g_b(n) \rfloor.$$

By lemma 4.19, there are $w, v \in \mathbb{Z}$ such that

$$\begin{aligned} g_b(n) &= \alpha n + w \\ g_b(n+1) &= \alpha(n+1) + v \end{aligned}$$

Thus

$$\begin{aligned} v - w &= g_b(n+1) - \alpha(n+1) - g_b(n) + \alpha n \\ &= g_b(n+1) - g_b(n) - \alpha. \end{aligned}$$

By proposition 4.15, we have

$$g_b(n+1) - g_b(n) \begin{cases} < (1-\alpha) - (-\alpha) = 1 \\ > (-\alpha) - (1-\alpha) = -1 \end{cases}.$$

so $v - w \in (-1-\alpha, 1-\alpha)$. As $v - w \in \mathbb{Z}$, it follows that $v - w = 0$ or $v - w = -1$. Now

$$\begin{aligned} c_\alpha(n) &= \lfloor (n+1)\alpha \rfloor - \lfloor n\alpha \rfloor \\ &= \lfloor g_b(n+1) - v \rfloor - \lfloor g_b(n) - w \rfloor \\ &= w - v + \lfloor g_b(n+1) \rfloor - \lfloor g_b(n) \rfloor. \end{aligned}$$

Case 1 $v - w = 0$. Then $g_b(n+1) = g_b(n) + \alpha \geq -\alpha + \alpha = 0$, so $\lfloor g_b(n+1) \rfloor = 0$ and thus

$$c_\alpha(n) = w - v + \lfloor g_b(n+1) \rfloor - \lfloor g_b(n) \rfloor = -\lfloor g_b(n) \rfloor$$

Case 2 $v - w = -1$. Then $g_b(n+1) = g_b(n) + \alpha - 1 < 1 - \alpha + \alpha - 1 = 0$, so $\lfloor g_b(n+1) \rfloor = -1$, and thus in this case $c_\alpha(n) = 1 - 1 - \lfloor g_b(n) \rfloor = -\lfloor g_b(n) \rfloor$ as well. □

Together, we now have the prerequisites to show proposition 4.5.

Proof of proposition 4.5. The first part of the proposition was shown by theorem 4.12. We recall the second part of the proposition. Let $b \in B$ and $n \in \mathbb{N}^+$. Then proposition 4.5 states that the following equation holds:

$$c_{\boldsymbol{\alpha}(b)}(n) = 1 \iff \min_i \{ f_b(n)_i > 0 \} \text{ is odd}$$

By proposition 4.20 we already know that $c_{\alpha(b)}(n) = 1 \iff g_b(n) < 0$ and by lemma 4.18 $g_b(n) < 0 \iff \min_i \{ f_b(n)_i > 0 \}$ is odd. Thus, the whole equivalence holds. □

5. Addition of numbers in Ostrowski representations

The goal of this section is to construct a Buchi automaton that takes $b \in B$ as well as 3 natural numbers x, y and z in the b-Ostrowski representation as input and accepts these only when $x + y = z$. To do this, we first need to clarify how exactly to represent b, x, y and z.

5.1. Representation with a finite alphabet

To be able to apply the theorems about ω-regular structures, we need to represent the Ostrowski representation as an ω-word with a finite alphabet. So far, we represented a number in Ostrowski representation as an element of $L_b \subseteq \mathbb{N}^\omega$, where the alphabet is not finite. When $\{b_i : i \in \mathbb{N}^+\}$ is bounded, L_b only contains a finite number of symbols, but when it's not bounded – for example take $b_i = i$ – then L_b really contains all $\mathbb{N}$ (that is $\{l \mid \exists w \in L_b\ \exists j\ w_j = l\} = \mathbb{N}$). Additionally, we also want to make some propositions about *all* Sturmian words, and for those we also need a way to represent them all with a finite alphabet. It would not be enough, that the number of digits needed for an individual Ostrowski numeration system is finite.
To work around this, we will represent the individual digits of the Ostrowski representation by binary numbers. As we represent an individual Ostrowski digit by multiple binary digits, we use an additional marker to recognize where the next Ostrowski digit starts. This idea is taken from [5, Section 3 #-binary encoding].

Definition 5.1. Fix $\Sigma = \{0, 1, \#\}$. Let

$$K = \{w \in \Sigma^\omega \mid w_0 = \#, \{j \mid w_j = \#\} \text{ is infinite}\}.$$

For $w \in K$, let $p_w \colon \mathbb{N} \to \omega$ s.t. $p_w(n)$ is the position of the $n+1$-st occurance of # in w. Note, that $p_w(0) = 0$ for any $w \in K$. Define $g \colon K \to \mathbb{N}^\omega$ by

$$g(w)_i = \sum_{j=0}^{p_w(i+1)-p_w(i)-2} 2^j w_{p_w(i)+1+j}$$

That is, $g(w)_i$ is the number written between the $(i+1)$-st and $(i+2)$-nd #, interpreted as binary number in least-significant-digit-first notation. We call two words $w, v \in K$ aligned, when $p_w = p_v$.

We note that g is surjective, but not injective.

Definition 5.2. Let $D = \{w \in K : \forall i \in \omega(w_{i+1} = \# \implies w_i = 1)\}$, that is D is the subset of K where each # except for the first one is preceeded by 1.[7] Define $\mu \colon D \to B$ by

$$\mu(w)_i = g(w)_{i-1}$$

Theorem 5.3. *μ is well-defined and bijective.*

Proof. Let $w \in D$. Then

$$\mu(w)_i = \sum_{j=p_w(i)+1}^{p_w(i+1)-1} 2^{j-p_w(i)-1} w_j \geq 2^{\overbrace{p_w(i+1)-p_w(i)-2}^{\geq 0}} \underbrace{w_{p_w(i+1)-1}}_{=1} \geq 1$$

[7] The definition of D is slightly different than the definition of R in [5, section 3.1] which this definition is based on. We restrict D a bit more such that μ becomes injective. This makes some arguments easier, as we don't need to consider equivalence classes of aligned words. Also, the automaton to recognize D is slightly easier this way.

Figure 4: automata recognizing K(left) and D(right)

So $\mu(w) \in B$. The mapping is also bijective, because the mapping of the individual allowed words between # signs to $\mathbb{N}^+$

$$\{u \in \{0,1\}^* : |u| \geq 1, u_{|u|-1} = 1\} \to \mathbb{N}^+$$
$$u \mapsto \sum_{i=0}^{|u|-1} 2^i u_i$$

is a bijection. □

Theorem 5.4. *K and D are ω-regular.*

Proof. K and D are recognized by the automata in figure 4. □

We have now transformed B to this finite alphabet, the next step is to also transform L_b accordingly

Definition 5.5. Let $d \in D$. Define $U_d = \{w \in K \mid g(w) \in L_{\mu(d)} \wedge p_w = p_d\}$.

Proposition 5.6. *Let $d \in D$. Then $g\colon U_d \to L_{\mu(d)}$ is a bijection.*

Proof. $g(U_d) \subseteq L_{\mu(d)}$ by definition. It is injective by the restriction to words that are aligned with d. For surjectivity, let $w \in L_{\mu(d)}$. Then $w_i \leq \mu(d)_{i+1}$. Then the 2-digit representation of w_i fits into $p_d(i+1) - p_d(i) - 1$ digits, as $\mu(d)_{i+1} = g_i(d)$ also fits into that. Thus, there is $u \in K$ with $g(u) = w$ and $p_u = p_d$. □

Proposition 5.7. *Let $U = \{(d, w) \in D \times K \mid w \in U_d\}$. Let $\bar{U} = (i \mapsto (d_i, w_i))$. Then $\bar{U}$ is K-compatible with U and $\bar{U}$ is ω-regular.*

Proof. $\bar{U}$ is K-compatible with U by definition. $\bar{U}$ is recognized by the automaton in figure 5 and is thus ω-regular. The different states of this automaton relate to the different properties of L_b defined in definition 4.1. F and F' ensure the first property, that the number is eventually 0. Properties 2-4 are ensured by the rest of the automaton. The $<$ and $=$ states ensure, that $w_i < b_{i+1}$ whenever a path through $<$ is taken. The 0 and $0'$ states can only be taken, when they are followed by a number where $w_i = b_{i+1}$ in the $=$ state. This ensures, that $w_i = b_{i+1}$ is also possible, but only when $w_{i-1} = 0$. The automaton also ensures, that the first component is in D. □

Definition 5.8. Let $d \in D$. Define the bijection $h_d\colon U_d \to \mathbb{N}$ by $h_d = f_{\mu(d)} \circ g$.

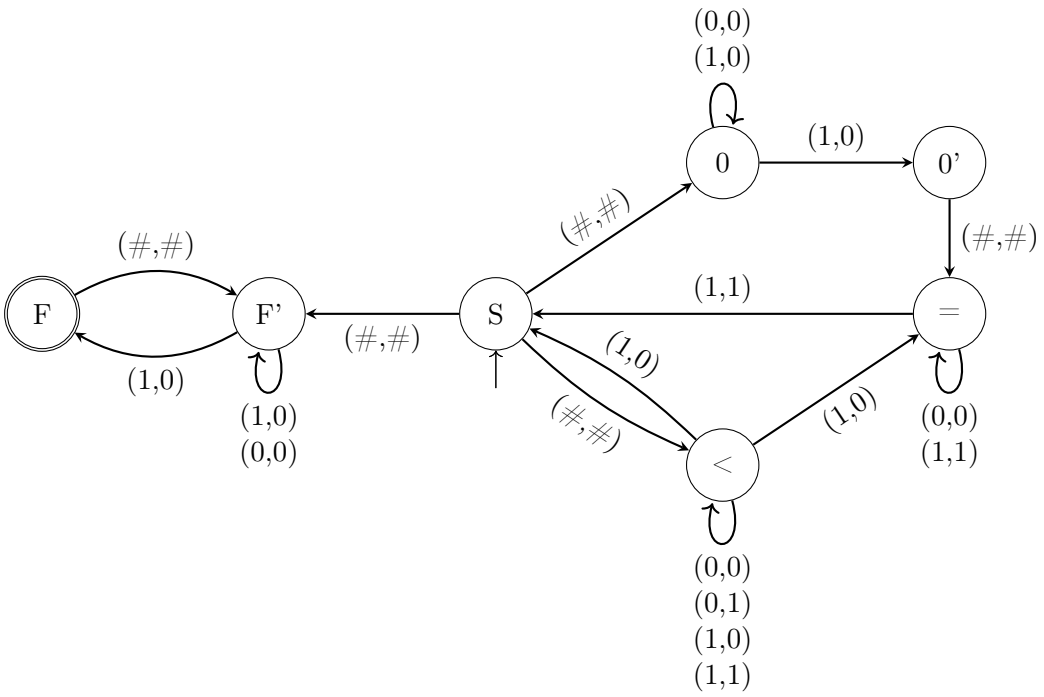

Figure 5: An automaton recognizing $\bar{U}$

5.2. Addition on L_b

Let $b \in B$ and let $x, y, z \in L_b$. We want to work towards an automaton, that can recognize if $f_b(x) + f_b(y) = f_b(z)$. Therefore, set $u \in \mathbb{Z}^\omega$ by $u_j = x_j + y_j - z_j$. Then $\{i \mid u_i \neq 0\}$ is finite and

$$f_b(x) + f_b(y) = f_b(z) \iff \sum_j u_j q_j(b) = 0 \tag{5.1}$$

Now let $n = \max_i\{(x_i, y_i, z_i) \neq (0,0,0)\}$. We define two sequences $r_k, s_k \in \mathbb{Z}$ for $k \in \omega$ recursively, s.t. the following invariant holds for all $k \in \omega$:

$$\sum_{j \geq k} u_j q_j = r_k q_k + s_k q_{k-1} \tag{5.2}$$

The idea of considering this equation is taken from [3, Section 2.2], but I have added a few ideas of my own here, too. Notably, we take a slightly different sum here, s.t. the definition of (r_i, s_i) is also useful when $f_b(x) + f_b(y) \neq f_b(z)$.[8]
For all $k > n$ we can fulfill this equation by setting $r_k = s_k = 0$. For $k < n$ we can then define r_k, s_k recursively, s.t. the equation is fulfilled. Let r_{k+1}, s_{k+1} already be defined,

[8] We also switched r and s here compared to [3], so r here is s there and the other way around

s.t. (5.2) holds for $k+1$. To motivate our choice for r_k, s_k, let's see what choices there are to make the invariant hold:

$$\begin{aligned} & u_k q_k + \sum_{j \geq k+1} u_j q_j = \sum_{j \geq k} u_j q_j \\ \iff \quad & u_k q_k + r_{k+1} q_{k+1} + s_{k+1} q_k = r_k q_k + s_k q_{k-1} \\ \iff \quad & u_k q_k + r_{k+1} b_{k+1} q_k + r_{k+1} q_{k-1} + s_{k+1} q_k = r_k q_k + s_k q_{k-1} \\ \iff \quad & (u_k + r_{k+1} b_{k+1} + s_{k+1} - r_k) q_k = (s_k - r_{k+1}) q_{k-1} \end{aligned}$$

This equation has multiple solutions for r_k, s_k, but setting both sides of this equation to 0 is a natural and easy choice.[9] We will use this and define r_k, s_k in the following way:

Definition 5.9. Define sequences $(r_k, s_k)_{k \in \omega}$ (depending on the choice of b, x, y, z) recursively by $r_k = s_k = 0$ for $k > n = \max_i \{(x_i, y_i, z_i) \neq (0,0,0)\}$ and

$$s_k = r_{k+1} \tag{5.3}$$
$$r_k = u_k + r_{k+1} b_{k+1} + s_{k+1} \tag{5.4}$$

for $0 \leq k \leq n$.

Then (5.2) holds and applying it to $k = 0$ leads to

$$\sum_{j \geq 0} q_j u_j = r_0 q_0 + s_0 q_{-1} = r_0.$$

Thus, we can rephrase (5.1) to

$$f_b(x) + f_b(y) = f_b(z) \iff r_0 = 0 \tag{5.5}$$

Lemma 5.10. *Let $f_b(x) + f_b(y) = f_b(z)$. Then for all $i \in \omega$ $r_i, s_i \in \{-1, 0, +1\}$.*

Proof. The proof here is written in my own words and adapted to the notation used here, but highly influenced by [3, theorem 4] which contains a proof of the same statement. We show it by induction over i. For $i > n$, $r_i = s_i = 0 \in \{-1, 0, +1\}$. Now let the statement be true for $i+1$, that is $r_{i+1}, s_{i+1} \in \{-1, 0, 1\}$. We show that $r_i, s_i \in \{-1, 0, +1\}$. By (5.3) $s_i = r_{i+1} \in \{-1, 0, +1\}$. It's left to show that $r_i \in \{-1, 0, +1\}$. By lemma 4.2, $\sum_{j \leq k-1} x_j < q_k$ and the same is true when replacing x in that inequality with y and z. Thus,

$$-q_k + 1 \leq \sum_{j \leq k-1} u_j q_j \leq 2q_k - 2 \tag{5.6}$$

[9] In [3, Proof of Proposition 3], it is argued that because q_k and q_{k-1} are coprime, both sides have to be zero; but I couldn't follow that argument, as $t q_k q_{k-1}$ with $t \in \mathbb{Z}$ would be possible as well. It is also not necessary as we can pick $t = 0$.

By the prerequisites of this lemma, $\sum_j q_j u_j = 0$, so

$$\begin{aligned}
\sum_{j\leq k-1} u_j q_j &= -\sum_{j\geq k} u_j q_j \\
&= -u_k q_k - \sum_{j\geq k+1} u_j q_j \\
&= -u_k q_k - r_{k+1} q_{k+1} - s_{k+1} q_k \\
&= -u_k q_k - r_{k+1} b_{k+1} q_k - r_{k+1} q_{k-1} - s_{k+1} q_k \\
&= -(u_k + r_{k+1} b_{k+1} + s_{k+1}) q_k - r_{k+1} q_{k-1} \\
&= -r_k q_k - r_{k+1} q_{k-1} \qquad (5.7)
\end{aligned}$$

Putting (5.7) into (5.6) we get

$$\begin{aligned}
& -q_k + 1 \leq -r_k q_k - r_{k+1} q_{k-1} \\
\Longrightarrow \quad & r_k q_k \leq q_k - 1 - \underbrace{r_{k+1}}_{\geq -1} q_{k-1} \\
& \leq q_k - 1 + \underbrace{q_{k-1}}_{\leq q_k} \\
& \leq 2q_k - 1 < 2q_k \\
\Longrightarrow \quad & r_k < 2
\end{aligned}$$

In the other direction, we have

$$\begin{aligned}
& -r_k q_k - r_{k+1} q_{k-1} \leq 2q_k - 2 \\
\Longrightarrow \quad & r_k q_k \geq -r_{k+1} q_{k-1} - 2q_k + 2 \qquad (5.8)
\end{aligned}$$

Case 1 $r_{k+1} \leq 0$. In this case $r_k q_k \geq -2q_k + 2 > -2q_k \Longrightarrow r_k > -2$.

Case 2 $r_{k+1} = 1$. Then $r_k q_k \geq -q_{k-1} - 2q_k + 2 \geq -3q_k + 2 > -3q_k \Longrightarrow r_k > -3$. Now assume $r_k = -2$. Then by (5.4)

$$\begin{aligned}
& -2 = u_k + b_{k+1} + s_{k+1} \\
\Longrightarrow\ & u_k = -2 - b_{k+1} - s_{k+1} \leq -b_{k+1} - 1
\end{aligned}$$

But $u_k = x_k + y_k - z_k \geq -z_k \geq -b_{k+1}$ which contradicts the assumption. Thus, also in this case, $r_k > -2$.

In total, we have $-2 < r_k < 2$, so $r_k \in \{-1, 0, 1\}$ as well. □

Proposition 5.11. *Let $f_b(x) + f_b(y) = f_b(z)$. Then for all $i \in \omega$*

$$(r_i, s_i) \in \{(0,0), (0,1), (0,-1), (-1,0), (-1,1), (-1,-1), (1,-1)\}$$

Proof. This proof here is influenced by [3, Section 2.3], which also contains a proof of this statement. By lemma 5.10, $(r_i, s_i) \in \{-1, 0, 1\} \times \{-1, 0, 1\}$. It is left to show that $(r_i, s_i) \neq (1, 0)$ and $(r_i, s_i) \neq (1, 1)$.
We show this by contradiction, therefore let's assume that there is some $b \in B, i \in \omega$ with $f_b(x) + f_b(y) = f_b(z)$, $r_i = 1$ and $s_i \in \{0, 1\}$. Pick the minimal i such that this is the case. Then $i > 0$ as $r_0 = 0$ by (5.5). Thus, by (5.4)

$$r_{i-1} = u_{i-1} + r_i b_i + s_i = \underbrace{u_{i-1}}_{\geq -b_i} + b_i + s_i \geq s_i \geq 0$$

As $r_{i-1} \leq 1$ by lemma 5.10, it follows that $r_{i-1} \in \{0, 1\}$.

Case 1 $r_{i-1} = 1$. By (5.3) $s_{i-1} = r_i = 1$, so $(r_{i-1}, s_{i-1}) = (1, 1)$ in this case. But that means, that i is not minimal with that property, so this can't be the case.

Case 2 $r_{i-1} = 0$. Then $s_i = 0$ and $u_{i-1} = -b_i$. This is only possible when $x_{i-1} = 0, y_{i-1} = 0, z_{i-1} = b_i$. Then $i \geq 2$, $z_{i-2} = 0$ and by (5.2)

$$\sum_{j=0}^{i-2} u_j q_j = -r_{i-1} q_{i-1} - s_{i-1} q_{i-2} = -q_{i-2}$$

On the other hand,

$$\sum_{j=0}^{i-2} u_j q_j \geq -\sum_{j=0}^{i-2} z_j q_j = -\underbrace{\sum_{j=0}^{i-3} z_j q_j}_{< q_{i-2}} - \underbrace{z_{i-2}}_{=0} q_{i-2} > -q_{i-2}$$

So this case is also not possible.

This completes the proof by contradiction. □

5.3. Addition on U_d

So far, we assigned tuples (r, s) to positions of $w \in L_b$. Our automaton will work on $U_{\mu^{-1}(b)}$ instead of L_b. This inspires the following definition 5.14. The idea of doing the definition in the way it is done here and all the further statements in this section are my own ideas.

Definition 5.12. Let $d \in D$. We define the function

$$t_d(n) = |\{i < n \mid d_i = \#\}|$$

to be the number of occurrences of # before position n in d.

Lemma 5.13. *Let $d \in D$. Then for any $m \in \mathbb{N}$*

$$t_d(p_d(m)) = m$$

and for any $n \in \mathbb{N}$

$$p_d(t_d(n)) \geq n.$$

d_i	#	1	#	0	1	#	1	1	#	0	0	1	#	1	#	1	#
$t_d(i)$	0	1	1	2	2	2	3	3	3	4	4	4	4	5	5	6	6
$p_d(t_d(i))$	0	2	2	5	5	5	8	8	8	12	12	12	12	14	14	16	16
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 6: $t_d(i)$ and $p_d(t_d(i))$ for an exemplary $d \in D$

Proof. By definition, $p_d(m)$ is the position of the $m+1$-st occurrence of #. So $d_{p_d(m)} = \#$ and $\{i < p_d(m) \mid d_i = \#\} = m$. For the second statement, note that $p_d(m)$ is the largest number n', s.t. $|\{i < n' \mid d_i = \#\}| = m$. So with $m = t_d(n)$ we get $n' \geq n$. Intuitively, this can also be seen in the example in figure 6. □

Definition 5.14. Let $d \in D$ and let $x, y, z \in U_d$. Let r_i, s_i be sequences as defined in definition 5.9 on $\mu(d)$, $g(x)$, $g(y)$ and $g(z)$. We define sequences $(r^{(i)}, s^{(i)}, c^{(i)})_{i\in\omega}$ as follows

$$r^{(i)} = r_{t_d(i)} \tag{5.9}$$

$$s^{(i)} = s_{t_d(i)} \tag{5.10}$$

$$c^{(i)} = -\sum_{j=i}^{p_d(t_d(i))-1} 2^{j-i}(r_{t_d(j)}d_j + x_j + y_j - z_j) \tag{5.11}$$

Lemma 5.15. *For any $i \in \omega$*

$$c^{(p_d(i))} = 0$$
$$c^{(p_d(i)+1)} = s_{i+1} - r_i$$

Proof. For the first statement, $p_d(t_d(p_d(i))) = p_d(i)$ and thus the sum in (5.11) has no summands and $c^{(p_d(i))} = 0$. For the second statement, $t_d(p_d(i) + 1) = i + 1$ and thus

$$\begin{aligned} c^{(p_d(i)+1)} &= -\sum_{j=p_d(i)+1}^{p_d(i+1)-1} 2^{j-p_d(i)-1}(r_{i+1}d_j + x_j + y_j - z_j) \\ &= -(r_{i+1}\mu(d)_{i+1} + g(x)_i + g(y)_i - g(z)_i) \\ &= -\underbrace{(g(x)_i + g(y)_i - g(z)_i + r_{i+1}\mu(d)_{i+1} + s_{i+1})}_{=r_i \text{ by (5.4)}} + s_{i+1} \\ &= s_{i+1} - r_i \end{aligned}$$

□

Proposition 5.16. *Let $d \in D$, $x, y, z \in U_d$. Then for $i \in \omega$ with $d_i \neq \#$*

$$r^{(i+1)} = r^{(i)} \qquad s^{(i+1)} = s^{(i)} \qquad c^{(i+1)} = \frac{1}{2}\left(c^{(i)} + r^{(i)}d_i + x_i + y_i - z_i\right) \tag{5.12}$$

and for $i \in \omega$ with $d_i = \#$, the following holds

$$r^{(i+1)} = s^{(i)} \qquad c^{(i+1)} = s^{(i+1)} - r^{(i)} \qquad c^{(i)} = 0 \tag{5.13}$$

Furthermore, when $h_d(x) + h_d(y) = h_d(z)$ also

$$r^{(0)} = 0 \qquad c^{(1)} = s^{(1)} \tag{5.14}$$

Proof. We start by showing (5.12). Let $i \in \omega$ with $d_i \neq \#$. Then $t_d(i+1) = t_d(i)$ and thus $r^{(i+1)} = r_{t_d(i+1)} = r_{t_d(i)} = r^{(i)}$ and with the same argument $s^{(i+1)} = s^{(i)}$. Furthermore

$$\begin{aligned} c^{(i)} &= -\sum_{j=i}^{p_d(t_d(i))-1} 2^{j-i}(r_{t_d(i)}d_i + x_i + y_i - z_i) \\ &= -2^{i-i}(r_{t_d(i)}d_i + x_i + y_i - z_i) + \underbrace{\left(-\sum_{j=i+1}^{p_d(t_d(i))-1} 2^{j-i}(r_{t_d(j)}d_d + x_j + y_j - z_j)\right)}_{=2c^{(i+1)}} \\ &= -(r^{(i)}d_i + x_i + y_i - z_i) + 2c^{(i+1)} \end{aligned}$$

Solving this equation for $c^{(i+1)}$ leads to the right-most equation of (5.12).
Now let $d_i = \#$. Then $t_d(i+1) = t_d(i)+1$ and thus by (5.3) $r^{(i+1)} = r_{t_d(i+1)} = r_{t_d(i)+1} = s_{t_d(i)} = s^{(i)}$. The other two equations of (5.13) follow directly from lemma 5.15 applied to $t_d(i)$. It can be applied like this because $d_i = \#$ and thus $p_d(t_d(i)) = i$.
For the last part of the proposition, $d_0 = \#$ and $r^{(0)} = r_{t_d(0)} = r_0 = 0$ by (5.5). Then by (5.13) applied to $i = 0$ we have $c^{(1)} = s^{(1)} - r^{(0)} = s^{(1)} - 0 = s^{(1)}$. □

Lemma 5.17. *Let $h_d(x) + h_d(y) = h_d(z)$. Then for all i the following holds*

1. $(r^{(i)}, s^{(i)}) \in \{(0,0),(0,1),(0,-1),(-1,0),(-1,1),(-1,-1),(1,-1)\}$
2. $c^{(i)} \in \{-2,-1,0,1,2\}$

Proof. The first statement follows directly from the definition and proposition 5.11. We prove the second statement by induction over i. So let the statement be true for all $j \leq i$. We prove the statement for i by a case-by-case analysis.

Case 1 $i = 0$. In this case $p_d(0) = 0$ and thus by lemma 5.15, $c^{(0)} = 0$.

Case 2 $i \geq 1$ and $d_{i-1} = \#$. Then by (5.13), $|c^{(i)}| = |s^{(i)} - r^{(i-1)}| \leq 2$

Case 3 $i \geq 1$ and $d_{i-1} \neq \#$. Then by (5.12)

$$\begin{aligned} c^{(i)} &= \frac{1}{2}\left(c^{(i-1)} + r^{(i-1)}d_{i-1} + x_{i-1} + y_{i-1} - z_{i-1}\right) \\ &\begin{cases} \leq \frac{1}{2}(2 + (+1)\cdot 1 + 1 + 1 - 0) = \frac{5}{2} \\ \geq \frac{1}{2}(-2 + (-1)\cdot 1 + 0 + 0 - 1) = -\frac{4}{2} \end{cases} \end{aligned}$$

So $-2 \leq c^{(i)} \leq 2\frac{1}{2}$ and because $c^{(i)} \in \mathbb{Z}$ it follows that $c^{(i)} \in \{-2,-1,0,1,2\}$. □

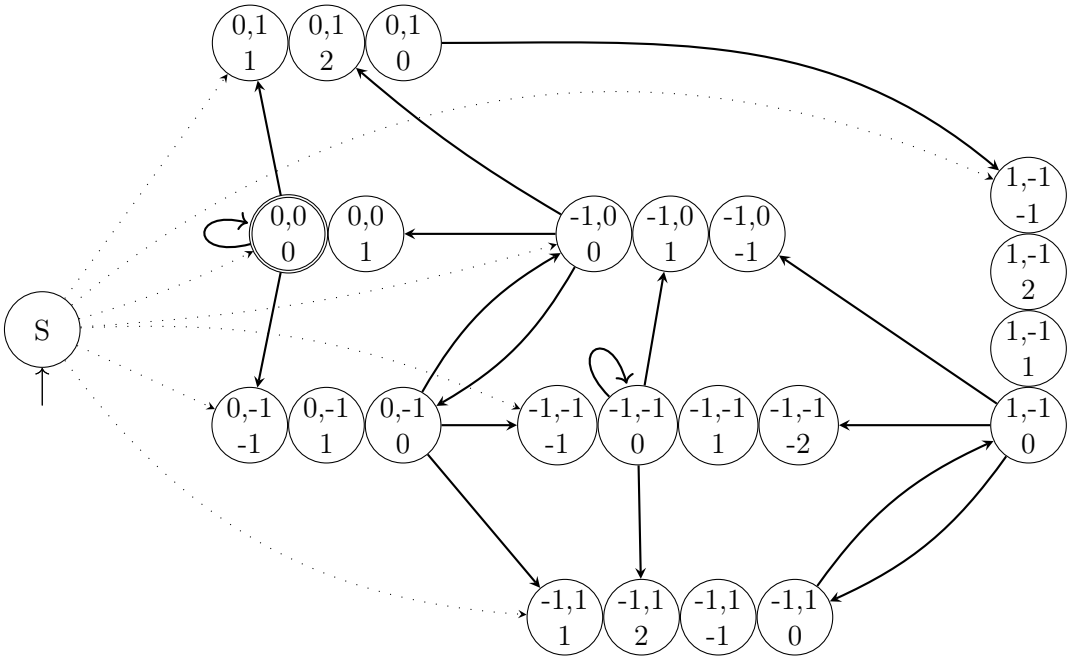

Figure 7: Addition automaton showing only edges labeled $(\#, \#, \#, \#)$. All other edges go from (r, s, c) to (r', s', c') with $(r, s) = (r', s')$. These are shown in figure 8.

Lemma 5.18. *Let $d \in D$, $x, y, z \in U_d$. Let $(\bar{r}^{(i)}, \bar{s}^{(i)}, \bar{c}^{(i)})_{i \in \omega}$ be a sequence that conforms to* (5.12) *and* (5.13)*. Furthermore, let there be a k such that $(\bar{r}^{(i)}, \bar{s}^{(i)}, \bar{c}^{(i)}) = (0, 0, 0)$ for all $i > k$. Then $(\bar{r}^{(i)}, \bar{s}^{(i)}, \bar{c}^{(i)}) = (r^{(i)}, s^{(i)}, c^{(i)})$ for all $i \in \mathbb{N}$.*

Proof. Let $n = \max_i((g(x)_i, g(y)_i, g(z)_i) \neq (0, 0, 0))$. Then for all $j > n$ we have $(r_j, s_j) = (0, 0)$ by definition 5.9. Let $m = \max(p_b(n+1), k)$. For any $j \geq m$ we have $(\bar{r}^{(j)}, \bar{s}^{(j)}, \bar{c}^{(j)}) = (0, 0, 0)$. Furthermore by definition 5.14 we have $r^{(j)} = r_{\underbrace{t_d(j)}_{\geq n+1}} = 0$ and $s^{(j)} = s_{t_d(j)} = 0$ as well. Also,

$$c^{(j)} = -\sum_{k=j}^{p_d(t_d(j))-1} 2^{k-j} \left(\underbrace{r_{t_d(k)}}_{=0} d_k + \underbrace{x_k}_{=0} + \underbrace{y_k}_{=0} - \underbrace{z_k}_{=0} \right) = 0$$

So for all $j \geq m$ we have $(\bar{r}^{(j)}, \bar{s}^{(j)}, \bar{c}^{(j)}) = (r^{(j)}, s^{(j)}, c^{(j)})$. We now show by induction, that this equation holds for all j. Let the statement already hold for $j+1$. We show the statement for j. We show this statement separately in these 2 cases.

Case 1 $d_j \neq \#$. Then by (5.12) $r^{(j)} = r^{(j+1)} = \bar{r}^{(j+1)} = \bar{r}^{(j)}$. With the same argument $s^{(j)} = \bar{s}^{(j)}$. Solving the right equation in (5.12) for $c^{(i)}$ and substituting i for j we get

$$c^{(j)} = 2(c^{(j+1)} - r^{(j)} d_j - x_j - y_j + z_j)$$

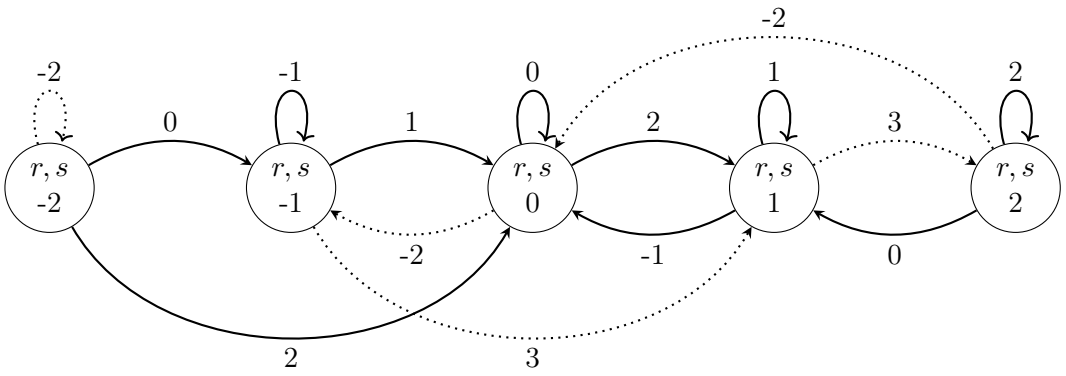

Figure 8: Edges of the form (d, x, y, z) with $d, x, y, z \in \{0, 1\}$. Edges labeled with v contain all symbols (d, x, y, z) such that $rd + x + y - z = v$. The dotted edges only exist for some values of r. $v = 3$ is only possible for $r = 1$ and $v = -2$ is only possible for $r = -1$.

and

$$\bar{c}^{(j)} = 2(\bar{c}^{(j+1)} - \bar{r}^{(j)} d_j - x_j - y_j + z_j)$$

As $c^{(j+1)} = \bar{c}^{(j+1)}$ by the induction hypothesis and $r^{(j)} = \bar{r}^{(j)}$ by the argument above, the right sides of these 2 equations are the same. Thus $c^{(j)} = \bar{c}^{(j)}$ as well.

Case 2 $d_j = \#$. Then by (5.13) $s^{(j)} = r^{(j+1)} = \bar{r}^{(j+1)} = \bar{s}^{(j)}$, $r^{(j)} = c^{(j+1)} + s^{(j+1)} = \bar{c}^{(j+1)} + \bar{s}^{(j+1)} = \bar{r}^{(j)}$ and $c^{(j)} = 0 = \bar{c}^{(j)}$. □

5.4. A Buchi automaton recognizing addition on U_d

We now construct a Buchi automaton $\mathcal{B}$ on Σ^4, such that for all $d \in D, x, y, z \in U_d$

$$h_d(x) + h_d(y) = h_d(z) \iff \mathcal{B} \text{ accepts } (i \mapsto (d_i, x_i, y_i, z_i))$$

We do this by constructing an automaton with states labeled by (r, s, c) where r, s, c are bounded as in lemma 5.17, which gives us our states $S \subseteq \{-1, 0, 1\} \times \{-1, 0, 1\} \times \{-2, -1, 0, 1, 2\}$. Let's define the transition relation $T \subseteq S \times \{0, 1, \#\}^4 \times S$. Proposition 5.16 shows which transitions we need to add.
(5.13) show us which transitions on $(\#, \#, \#, \#)$ we need to add:

$$((r, s, c), (\#, \#, \#, \#), (r', s', c')) \in T \iff r' = s \wedge c = 0 \wedge c' = s' - r$$

That is, we add transitions from $(r, s, 0)$ to $(s, t, t - r)$ labeled $(\#, \#, \#, \#)$.
In addition, we add transitions of the form

$$((r, s, c), (d, x, y, z), (r, s, c'))$$

for which $c + rd + x + y - z = 2c'$, as we can see from (5.12). From the resulting automaton we can remove some states that don't have any incoming transitions.

By definition 5.9, there is some $k > 0$, such that $r_j = s_j = 0$ for all $j > k$. Thus by definition 5.14 and lemma 5.15, $(r^{(p_d(j))}, s^{(p_d(j))}, c^{(p_d(j))}) = (0,0,0)$ for all $j > k$. This inspires the choice to set $(0,0,0)$ to be the only final state.
Now let $h_d(x) + h_d(y) = h_d(z)$. Then $c^{(1)} = s^{(1)}$ by (5.14), thus the possible values for $(r^{(1)}, s^{(1)}, c^{(1)})$ are of the form (r, s, s). We add an initial state to the automaton with transitions labeled $(\#, \#, \#, \#)$ to all the 7 states of this form.
The final automaton has 24 states. It is displayed in figure 7. Figure 7 shows only the transitions of the form $(\#, \#, \#, \#)$, the remaining transitions are shown in figure 8.

Theorem 5.19. *Let $d \in D, x, y, z \in U_d$. Then the adder automaton accepts $(i \mapsto (d_i, x_i, y_i, z_i))$ if and only if $h_d(x) + h_d(y) = h_d(z)$.*

Proof. First, let the adder automaton accept $(i \mapsto (d_i, x_i, y_i, z_i))$. Then there is an accepting run σ. Set $(\bar{r}^{(k)}, \bar{s}^{(k)}, \bar{c}^{(k)}) = \sigma(k)$ for $k \geq 1$ and $(\bar{r}^{(0)}, \bar{s}^{(0)}, \bar{c}^{(0)}) = (0, \bar{r}^{(1)}, 0)$. Then (5.12) and (5.13) hold for this sequence by the definition of the automaton for $k \geq 1$. For $k = 0$ this is slightly harder to see. As $d_0 = \#$ we need to show the 3 equations of (5.13) for $k = 0$. The first one, $\bar{r}^{(1)} = \bar{s}^{(0)}$, holds by the definition of $\bar{s}^{(0)}$. The outgoing transitions from the initial state go to states of the form (r, s, s), so we have $\bar{c}^{(1)} = \bar{s}^{(1)}$. Thus, $\bar{c}^{(1)} = \bar{s}^{(1)} = \bar{s}^{(1)} - \bar{r}^{(0)}$ as $\bar{r}^{(0)} = 0$ by definition. This is the second equation of (5.13). The third equation, $\bar{c}^{(0)} = 0$ holds by definition. Together, (5.12) and (5.13) hold for the whole sequence. Furthermore, as $x, y, z \in U_d$ there is a $k \in \mathbb{N}$ such that $(x_j, y_j, z_j) \in \{(0,0,0), (\#, \#, \#)\}$ for all $j > k$. As $(0,0,0)$ is the only final state of the adder automaton, there is a $l > k$, such that $\sigma(l) = (0,0,0)$. We show $\sigma(j) = (0,0,0)$ for all $j > l$ as well. We show this by contradiction. Consider that there is $j > l$ with $\sigma(j) \neq (0,0,0)$. Let j be minimal with that property. Then $\sigma(j-1) = (0,0,0)$ and $\sigma(j) \neq (0,0,0)$. We do a case-by-case analysis:

Case 1 $(x_{j-1}, y_{j-1}, z_{j-1}) = (0,0,0)$. Then $d_{j-1} \in \{0,1\}$ and by (5.12) we have

$$\bar{c}^{(j)} = \frac{1}{2}\left(\underbrace{\bar{c}^{(j-1)}}_{=0} + \underbrace{\bar{r}^{(j-1)}}_{=0} d_{j-1} + \underbrace{x_{j-1}}_{=0} + \underbrace{y_{j-1}}_{=0} - \underbrace{z_{j-1}}_{=0}\right) = 0.$$

Also $\bar{r}^{(j)} = \bar{r}^{(j-1)} = 0$ and $\bar{s}^{(j)} = \bar{s}^{(j-1)} = 0$, so $\sigma(j) = (0,0,0)$ which is a contradiction.

Case 2 $(x_{j-1}, y_{j-1}, z_{j-1}) = (\#, \#, \#)$. Then $d_{j-1} = \#$ as well. Then by (5.13) we have $\bar{c}^{(j)} = \bar{s}^{(j)} - \bar{r}^{(j-1)} = \bar{s}^{(j)}$ and $\bar{r}^{(j)} = \bar{s}^{(j-1)} = 0$. We know, that $d_j \neq \#$ and thus by (5.13) we have

$$2\bar{c}^{(j+1)} = \bar{c}^{(j)} + \bar{r}^{(j)} d_j + x_j + y_j - z_j = \bar{s}^{(j)} + 0 \cdot d_j + 0 + 0 - 0 = \bar{s}^{(j)}.$$

So $2 | \bar{s}^{(j)}$ and as $\bar{s}^{(j)} \in \{-1, 0, 1\}$ it follows that $\bar{s}^{(j)} = 0$. Also, $\bar{c}^{(j)} = \bar{s}^{(j)} = 0$ and $\bar{r}^{(j)} = \bar{s}^{(j-1)} = 0$. Thus, we have a contradiction in this case as well.

We conclude, that $(\bar{r}^{(j)}, \bar{s}^{(j)}, \bar{c}^{(j)}) = (0,0,0)$ for all $j > l$. We now have all the prerequisites to apply lemma 5.18 and we get $r_0 = r^{(0)} = \bar{r}^{(0)} = 0$. Then by (5.5) the addition equality holds.

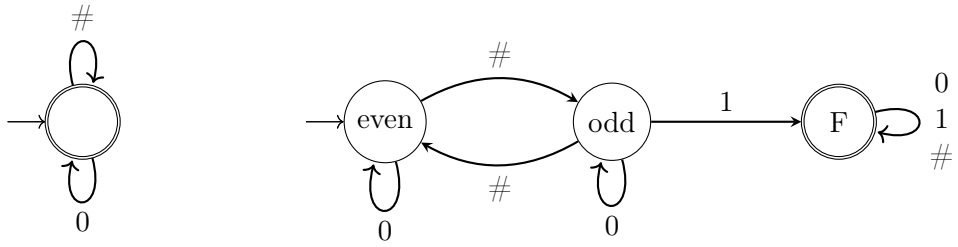

Figure 9: Automata recognizing 0(left) and recognizing if a word in Ostrowski representation is Sturmian(right)

Conversely, let $h_d(x) + h_d(y) = h_d(z)$. Then

$$\sigma(i) = \begin{cases} (r^{(i)}, s^{(i)}, c^{(i)}) & i \geq 1 \\ \mathrm{S} & i = 0 \end{cases}$$

is an accepting run by construction of the automaton and proposition 5.16. □

5.5. Application to Sturmian words

With the definitions done so far, we construct a structure with a decidable theory now. Let $L_0 \subseteq \Sigma^\omega$, $L_c \subseteq \Sigma^\omega$ be the languages of the Buchi automata given in figure 9 and let $L_+ \subseteq (\Sigma^4)^\omega$ be the language of the addition automaton constructed in the last section.

Definition 5.20. Define the signature $S = \{D, U, 0, c, +\}$ with $ar(D) = 1, ar(U) = 2, ar(0) = ar(c) = 1, ar(+) = 4$. We define the S-structure $\mathcal{K}$

$$\mathcal{K} = (\Sigma^\omega, D^\mathcal{K}, U^\mathcal{K}, 0^\mathcal{K}, c^\mathcal{K}, +^\mathcal{K}).$$

with the following definitions for the relation interpretations:

$$\begin{aligned} D^\mathcal{K} &= D \text{ as defined in definition 5.2} \\ U^\mathcal{K} &= U \text{ as defined in proposition 5.7} \\ 0^\mathcal{K} &= L_0 \\ c^\mathcal{K} &= L_c \\ +^\mathcal{K} &= \{(a, x, y, z) \mid (i \mapsto (a_i, x_i, y_i, z_i)) \in L_+\} \end{aligned}$$

Proposition 5.21. *$\mathcal{K}$ has a decidable theory.*

Proof. Σ^ω is clearly ω-regular. D is ω-regular by theorem 5.4. U is Σ^ω-compatible with $\bar{U}$ which is ω-regular by proposition 5.7. $0^\mathcal{K} = L_0$, $c^\mathcal{K} = L_c$ and $+^\mathcal{K}$ is Σ^ω-compatible with L_+. L_0, L_c and L_+ are all ω-regular by construction. Thus by theorem 3.3 $\mathcal{K}$ has a decidable theory. □

From the decidability of the theory of $\mathcal{K}$ we can follow the more natural statement, that the combined theory of Presburger arithmetic with characteristic Sturmian words is decidable.

Theorem 5.22. *Let $S' = \{+, 0, c\}$ be the signature of Presburger arithmetic together with a unary relation symbol. That is $ar(0) = ar(c) = 1, ar(+) = 3$. Let $M_\alpha = (\mathbb{N}, \{(x, y, z) \in \mathbb{N}^3 \mid x + y = z\}, \{0\}, \{n \mid c_\alpha(n) = 1\})$ be the S'-structure where c is mapped to the characteristic Sturmian word with slope α. Let $T = \{\phi \in F(S') \mid \text{free}(\phi) = \emptyset, \forall \alpha \in (0, 1) \setminus \mathbb{Q}\ M_\alpha \models \phi\}$ be the combined theory. Then T is decidable.*

Proof. The idea is to transform formulas $\phi \in \mathrm{F}(S')$ into formulas $\psi \in \mathrm{F}(S)$, such that $\mathcal{K} \models \psi \iff \phi \in T$. As $\mathcal{K}$ has a decidable theory, doing this transformation in a computable way is enough to show this theorem.
We start by defining helper mappings $u_j \colon F(S') \to F(S)$. For this helper mapping, the following invariant should hold:

> Let $\varphi \in F(S')$ and $j \in \omega$. Let v_j not appear anywhere in φ (not even inside some $\exists$). Let $\text{free}(\varphi) = \{v_{i_1}, \ldots, v_{i_n}\}$. Then for all $\alpha \in (0, 1) \setminus \mathbb{Q}$ and $m_1, \ldots, m_n \in \mathbb{N}$ the following equivalence holds:
>
> $$M_\alpha \frac{m_1, \ldots, m_n}{v_{i_1}, \ldots, v_{i_n}} \models \varphi \iff \mathcal{K} \frac{d}{v_j} \frac{h_d^{-1}(m_1) \ldots h_d^{-1}(m_n)}{v_{i_1} \ldots v_{i_n}} \models u_j(\varphi) \qquad (5.15)$$
>
> where $d = \mu^{-1}(\boldsymbol{\alpha}^{-1}(\alpha))$.

We define u_j by recursion over the complexity of the formula and start with the atomic formulas.

$$\begin{aligned} u_j(v_i \equiv v_k) &:= v_i \equiv v_k \\ u_j(+(v_i, v_k, v_l)) &:= +(v_j, v_i, v_k, v_l) \\ u_j(0(v_i)) &:= 0(v_i) \\ u_j(c(v_i)) &:= c(v_i) \end{aligned}$$

We show for these atomic formulas, that (5.15) holds, starting with $\varphi = v_i \equiv v_k$:

$$\begin{aligned} M_\alpha \frac{m_1, m_2}{v_i, \ldots, v_k} \models v_i \equiv v_k &\iff m_1 = m_2 \iff h_d^{-1}(m_1) = h_d^{-1}(m_2) \\ &\iff \mathcal{K} \frac{d}{v_j} \frac{h_d^{-1}(m_1), h_d^{-1}(m_2)}{v_i, v_k} \models v_i \equiv v_k \end{aligned}$$

Next, let's consider $\varphi = +(v_i, v_k, v_l)$. In this case

$$\begin{aligned} M_\alpha \frac{m_1, , m_2, m_3}{v_i, v_k, v_l} \models +(v_i, v_k, v_l) &\iff m_1 + m_2 = m_3 \\ &\iff h_d(h_d^{-1}(m_1)) + h_d(h_d^{-1}(m_2)) = h_d(h_d^{-1}(m_3)) \\ &\iff (d, h_d^{-1}(m_1), h_d^{-1}(m_2), h_d^{-1}(m_3)) \in +^{\mathcal{K}} \qquad \text{by theorem 5.19} \\ &\iff \mathcal{K} \frac{d}{v_j} \frac{h_d^{-1}(m_1), h_d^{-1}(m_2), h_d^{-1}(m_3)}{v_i, v_k, v_l} \models +(v_j, v_i, v_k, v_l) \end{aligned}$$

When $\varphi = 0(v_i)$, (5.15) holds as well:

$$M_\alpha \frac{m_1}{v_i} \models 0(v_i) \iff m_1 = 0 \iff h_d^{-1}(m_1) \in L_0 \iff \mathcal{K}\frac{d}{v_j}\frac{h_d^{-1}(m_1)}{v_i} \models 0(v_i)$$

Let's also consider the last case of an atomic formula, where $\varphi = c(v_i)$. Then

$$\begin{aligned} M_\alpha \frac{m_1}{v_i} \models c(v_i) &\iff c_\alpha(m_1) = 1 \\ &\iff c_\alpha(m_1) = 1 \\ &\iff \min_k\{f^{-1}_{\mu(d)}(m_1)_k > 0\} \text{ is odd} \qquad \text{by proposition 4.5} \\ &\iff t_d(\min_k\{h_d^{-1}(m_1)_k = 1\}) \text{ is odd} \\ &\iff h_d^{-1}(m_1) \in L_c \\ &\iff \mathcal{K}\frac{d}{v_j}\frac{h_d^{-1}(m_1)}{v_i} \models c(v_i) \end{aligned}$$

Let the mapping now be already defined for φ and ϕ. Then we recursively define u_j

$$\begin{aligned} u_j(\varphi \wedge \phi) &:= u_j(\varphi) \wedge u_j(\phi) \\ u_j(\varphi \vee \phi) &:= u_j(\varphi) \vee u_j(\phi) \\ u_j(\neg\varphi) &:= \neg u_j(\varphi) \\ u_j(\exists v_i \phi) &:= \exists v_i(U(v_j, v_i) \wedge u_j(\phi)) \end{aligned}$$

We now show, that (5.15) holds for all formulas. We do this by induction over the complexity of the formula. For atomic formulas, we already showed above, that (5.15) holds. For the first three of recursive definitions, it's easy to see that (5.15) holds. We show it for the last case, as that is a bit more complicated. Therefore, let (5.15) hold already for ϕ. We show it for $\varphi = \exists v_{i_1}\phi$:

$$\begin{aligned} M_\alpha \frac{m_1 \dots m_n}{v_{i_1} \dots v_{i_n}} &\models \exists v_k \phi \\ &\iff \text{there exists } m \in \mathbb{N} \text{ s.t. } M_\alpha \frac{m}{v_k}\frac{m_1 \dots m_n}{v_{i_1} \dots v_{i_n}} \models \phi \\ &\iff \text{there exists } m \in \mathbb{N} \text{ s.t. } \mathcal{K}\frac{d}{v_j}\frac{h_d^{-1}(m)}{v_k}\frac{h_d^{-1}(m_1)\dots h_d^{-1}(m_n)}{v_{i_1} \dots v_{i_n}} \models u_j(\phi) \\ &\iff \text{there exists } m \in U_d \text{ s.t. } \mathcal{K}\frac{d}{v_j}\frac{m}{v_k}\frac{h_d^{-1}(m_1)\dots h_d^{-1}(m_n)}{v_{i_1} \dots v_{i_n}} \models u_j(\phi) \qquad (5.16) \end{aligned}$$

Now $U_d \subseteq \Sigma^\omega$ and for $m \in \Sigma^\omega$

$$m \in U_d \iff (d, m) \in \bar{U} \iff \mathcal{K}\frac{d}{v_j}\frac{m}{v_k} \models U(v_j, v_k)$$

and we can further equivalently reformulate (5.16) to

$$\begin{aligned}
&\iff \text{there exists } m \in \Sigma^\omega \text{ s.t. } \mathcal{K}\frac{d}{v_j}\frac{m}{v_k}\frac{h_d^{-1}(m_1)\dots h_d^{-1}(m_n)}{v_{i_1}\dots v_{i_n}} \models U(v_j, v_k) \wedge u_j(\phi) \\
&\iff \mathcal{K}\frac{d}{v_j}\frac{h_d^{-1}(m_1)\dots h_d^{-1}(m_n)}{v_{i_1}\dots v_{i_n}} \models \exists v_k(U(v_j, v_k) \wedge u_j(\phi))
\end{aligned}$$

Now let $\phi \in \mathrm{F}(S')$ be a sentence. Let $j \in \omega$ be minimal such that v_j is not appearing anywhere in ϕ. Then we define $\psi \in \mathrm{F}(S)$ by

$$\psi = \neg\exists v_j(D(v_j) \wedge \neg u_j(\phi))$$

Then $\mathcal{K} \models \psi \iff \phi \in T$ as the following calculation shows:

$$\begin{aligned}
\mathcal{K} \models \psi \iff &\text{ there is no } b \in \Sigma^\omega \text{ s.t. } d \in D \text{ and } \mathcal{K}\frac{d}{v_j} \not\models u_j(\phi) \\
\iff &\text{ there is no } \alpha \in (0,1) \setminus \mathbb{Q} \quad \mathcal{K}\frac{\mu^{-1}(\boldsymbol{\alpha}^{-1}(\alpha))}{v_j} \not\models u_j(\phi) \\
\iff &\text{ there is no } \alpha \in (0,1) \setminus \mathbb{Q} \quad M_\alpha \not\models \phi \\
\iff &\text{ for all } \alpha \in (0,1) \setminus \mathbb{Q} \quad M_\alpha \models \phi \\
\iff &\phi \in T
\end{aligned}$$

□

6. Proving theorems in practice

We have now shown, that the theory of $\mathcal{K}$ is decidable. Thus we know that there is an algorithm, that can check for a formula φ if it holds in $\mathcal{K}$ or not. But can we also run such an algorithm in practice? Pecan is a software that can be used to show statements about ω-regular structures and implements the ideas outlined in section 3.2 in a software program. We will start by giving a short introduction into it and then use it to show some statements. We will use it to get more confidence that the adder automaton constructed in the last section is indeed correct and also use it to show that characterstic Sturmian words are never eventually periodic. We will also analyze the possibilities and limits of this approach in general and Pecan specifically.

6.1. Introduction to Pecan

Pecan is a software that can be used to check properties of formulas in practice. More information about Pecan can be found in [9]. The code of Pecan is available on Github at `https://github.com/ReedOei/Pecan`. A manual is also available there. Pecan implements the decision algorithm as outlined in section 3.2. It is influenced by Walnut, but works with automata on infinite words instead of on finite words. Pecan is written in Python. It uses Spot[1], a library written in C++, to represent and manipulate automata. All the complicated and resource-intensive computation that is needed to manipulate automata is done in this layer.

Spot uses a set of atomic propositions to represent the input alphabet. As a result, the cardinality of the alphabet is always a power of 2. Having n atomic propositions $a_1, \ldots, a_n$, the input alphabet consists of all the variations of $\bigwedge_{i=1}^{n} (\neg) a_i$. Edges are not labeled directly with letters from this alphabet, but instead with boolean formulas on the atomic propositions. This makes operations like projection easier. To project some part of the alphabet away, just leave out some of the atomic propositions. Combinations of multiple automata is also easier this way. It also helps in having fewer edges. But it also means, that the 3-letter alphabet needed to represent our Ostrowski numbers need 2 atomic proposition to have enough space (plus another two for the definition of the actual Ostrowski representation used).

Pecan allows to represent one variable by multiple such atomic propositions in the underlying representation in Spot. A variable can only be used at a place where a variable with the same number of atomic propositions is expected. This helps abstracting the underlying representation away, but it still sometimes "leaks" through the abstraction and may thus be good to understand.

Spot can represent automata with various acceptance conditions. In general, every transition is labeled with a finite amount of colors. The acceptance condition can then be formulated as a boolean formula on the expression `Inf(color)` and `Fin(color)`, which means that the particular color is only finitely often – resp infinitely often – visited in a run. A Buchi automaton has just one color – let's call it 1 – and the acceptance condition `Inf(1)`. Furthermore, all transitions leaving the same state are labeled with the same set of colors[10]. A Co-Buchi automaton is an automaton with one color where the acceptance condition is `Fin(1)`. The result of some operations on Buchi automata may result in automata, that are not Buchi. For example, complementing a Buchi automaton may result in a Co-Buchi automaton. At various times, Pecan will then call some *postprocess* function to convert such an automaton to a Buchi automaton again. Postprocessing can be a very expensive operation. Spot has some limitations what kind of automata it can represent. For example, the states are referenced in edges in a 32 bit integer, which means that an automaton cannot have more than about 4 billion nodes[1, Section 8 Shortcomings and one Future Direction].

Pecan can read automata in different formats using the `#load` command and will assign it to a relation symbol then. Using these relations, more complicated formulas can be built. Pecan supports the normal operations like $\vee$, $\wedge$, $\exists$ and $\neg$. Other operations are supported, too, but translated to equivalent formulations. For example $\forall v.\varphi$ is translated to $\neg\exists v.\neg\varphi$. $\rightarrow$ and $\leftrightarrow$ are also translated to equivalent formula consisting of only $\vee, \wedge$ and $\neg$. Pecan also supports restricting variables after $\exists$ and $\forall$ with a predicate. This is often quite useful. Some of these translations are shown in figure 10. Pecan also supports some "syntactic sugar" to write more complicated low-level formulas in short expressive high-level forms. It supports the idea of having types assigned to variables and can then interpret operators like `+` differently depending on the type. We will not use this high-level syntactic sugar here and ignore its existence for the remainder of this thesis.

[10] An automaton with that property is said to have "state-based acceptance".

Original	Translation
$\forall v.\varphi$	$\neg\exists v.\neg\varphi$
$\forall v$ is $R(x,y).\varphi$	$\neg\exists v.(R(x,y,z) \vee \neg\varphi)$
$\exists v$ is $R(x,y).\varphi$	$\exists v.(R(x,y,z) \wedge \varphi)$
if φ then ϕ	$\neg\varphi \vee \phi$
$\varphi \to \phi$	$\neg\varphi \vee \phi$
φ iff ϕ	$(\neg\varphi \vee \phi) \wedge (\neg\phi \vee \varphi)$

Figure 10: Various translations of formula in Pecan

Pecan can do various operations on formulas:

- Check if a formula is true
- If a formula with free variables is true for at least one assignment to the variables, give an example of an eventually periodic such assignment to the variables. It is shown by providing the prefix and the periodic part.
- Write the automata assigned to a formula to a file

6.2. Improving confidence in the adder automaton

We will now use Pecan to check that our adder automaton is correct, using the same approach as was used in [5, Section 4], which itself lend it from [8, Remark 2.1].
We want to show, that for all $d \in D$ and $x, y, z \in U_d$

$$\mathcal{K}\frac{d,x,y,z}{v_1,v_2,v_3,v_4} \models +(v_1,v_2,v_3,v_4) \iff h_d(x) + h_d(y) = h_d(z) \tag{6.1}$$

We can do this by induction over $h_d(y)$. For $h_d(y) = 0$, we use the bijectivity of h_d to get the following equivalence:

$$\mathcal{K}\frac{d,x,y,z}{v_1,v_2,v_3,v_4} \models +(v_1,v_2,v_3,v_4) \iff x = z$$

In other words the base case of the induction is the following

$$\mathcal{K} \models \forall a \text{ is R}.\forall x,y,z \text{ are U}(a).\text{zzero}(y) \to (+(a,x,y,z) \longleftrightarrow \text{eq}(x,z))$$

We can formulate a successor predicate using just eq, leq and U predicates as defined in figure 11 and figure 5. Here, eq is the automaton recognizing the = relation and leq is an additional relation symbol that we add to $\mathcal{K}$ defined by the automaton in figure 11. It is defined, such that for any $d \in D$ and $x, y \in U_d$ we have

$$\mathcal{K}\frac{x,y}{v_1,v_2} \models \text{leq}(v_1,v_2) \iff h_d(x) \leq h_d(y).$$

With these definitions, we can define a successor predicate in the following way:

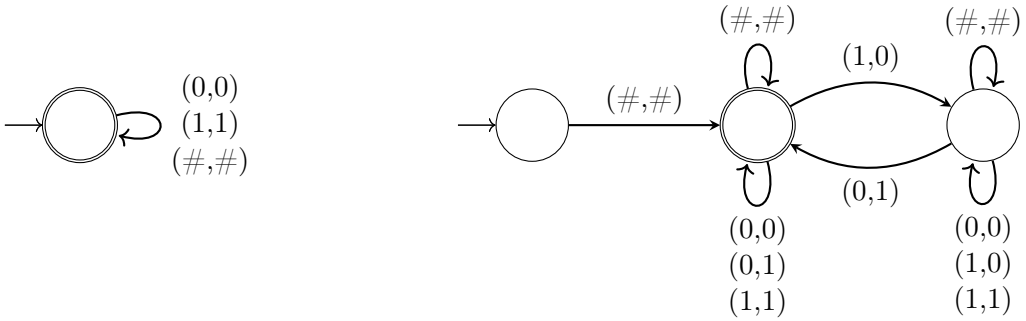

Figure 11: Automata for *eq*(left) and *leq*(right)

$$\begin{aligned}\text{succ(a, n, m)} := \text{leq}(n,m) \wedge \text{U}(a,n) \wedge \text{U}(a,m) \wedge \neg\text{eq}(n,m) \\ \wedge (\forall p.(\text{U}(a,p) \rightarrow (\text{leq}(p,n) \vee \text{leq}(m,p))\end{aligned}$$

For the induction step, let (6.1) be true for a fixed $d \in D$ and $y \in U_d$ (and all $x, z \in U_d$). We want to show the equivalence for $d' = d, y' = h_d^{-1}\big(h_d(y)+1\big)$ and arbitrary $x', z' \in U_d$. Let's assume, that $h_d(z') > 0$. Set $x = x'$ and $z = h_d^{-1}(h_d(z') - 1)$. Then by the induction hypothesis, we have

$$\mathcal{K}\frac{d, x', y', z'}{v_1, v_2, v_3, v_4} \models +(v_1, v_2, v_3, v_4) \Longleftrightarrow h_d(x) + h_d(y) = h_d(z)$$

This can be reformulated to the following

$$\begin{aligned}\mathcal{K} \models &\forall a \text{ is R.} \forall x, y, z, y', z' \text{ are U}(a). \\ &(\text{succ}(a, y, y') \wedge \text{succ}(a, z, z')) \rightarrow (+(a, x, y, z) \longleftrightarrow +(a, x, y', z'))\end{aligned}$$

In case $h_d(z') = 0$, clearly $\mathcal{K}\frac{d,x',y',z'}{v_1,v_2,v_3,v_4} \not\models +(v_1, v_2, v_3, v_4)$. We can also reformulate this as follows[11]:

$$\mathcal{K} \models \forall a \text{ is R.} \forall x', y, y', z' \text{ are U}(a).\text{succ(a, y, y')} \wedge \text{zzero}(z') \rightarrow \neg + (a, x', y', z')$$

We can check these 3 sentences in Pecan. This checks that the adder automaton is defined correctly under the assumption that the automata R, U, leq, eq and zero are correct. We execute the following Pecan code:

```
succ(a, x, y) := leq(x, y) & _U(a, x) & _U(a, y) & !eq(x, y)
    & forall z. if _U(a, z) then (leq(z, x) | leq(y, z))
```

[11] This last case is missing in [5, Section 4] as well as [8, Remark 2.1], but it is necessary. I double-checked and found a variant of +, call it fakeadd, that adheres to the first two conditions, but not this third one. It can be defined as follows: $\text{fakeadd}(a, x, y, z) = \text{succ}(a, z, y) \vee +(a, x, y, z)$. I verified this with Pecan, the script can be found in script 4 in appendix E.

```
Theorem ("Addition base case (0 + y = y)", {
    forall a. forall x,y,z are _U(a).
        if zzero(y) then (add(a,x,y,z) iff eq(x,z))
}).

Theorem ("Addition inductive case (x + s(y) = s(x + y))", {
    forall a is _D. forall x,y,z,u,v are _U(a).
        if (_U(a, x) & succ(a,u,y) & succ(a,v,z)) then
            (add(a,x,y,z) iff add(a,x,u,v))
}).

Theorem("Addition inductive case (x + s(y) != 0)", {
    forall a is _D. forall x,y,z,u are _U(a).
        if (succ(a, y, u) & zzero(z)) then !add(a, x, u, z)
}).
```

To be able to execute this code, we need to supply the automata to Pecan. This can be done using `#load` statements for the relations `_D(a)`, `_U(a, x)`, `leq(x, y)`, `eq(x, y)`, `zzero(z)`, `add(a, x, y, z)` and `sturmian(n)`. They look like this:

```
#load("automata/U.txt", "pecan", _U(a, x))
```

I have written files for the automata for D, U, leq, eq, zzero and sturmian manually, they can be found in appendix D. The adder automaton is much more complicated. Instead of writing it manually, I have written a small NodeJS program that generates the adder automaton. The source code for that file can also be found in appendix C. Additionally, the code can also be found at `https://gitlab.com/fkz/ostrowski`.
Executing Pecan leads to the following result:

```
Addition base case (0 + y = y) is true.
Addition inductive case (x + s(y) = s(x + y)) is true.
Addition inductive case (x + s(y) != 0) is true.
```

Thus, we can now have more confidence that the adder automaton is indeed correct.

6.3. Analyzing the runtime of Pecan programs

Let's dive a bit deeper into the things that happen under the hood of the execution of the above program. I have gathered the results for this section by executing Pecan with the `-ddd` option. This option tells Pecan to output more information about what's going on under the hood. Let's check what happens when Pecan encounters the `Addition base case (0 + y = y)` theorem. After parsing the formula, it transforms it into a formula that only contains the primitives $\exists$, $\neg$, $\wedge$ and $\vee$. This step transforms $\forall$`a. ∀x,y,z are _U(a). if zzero(y) then (add(a,x,y,z) iff eq(x,z))` to the following formula

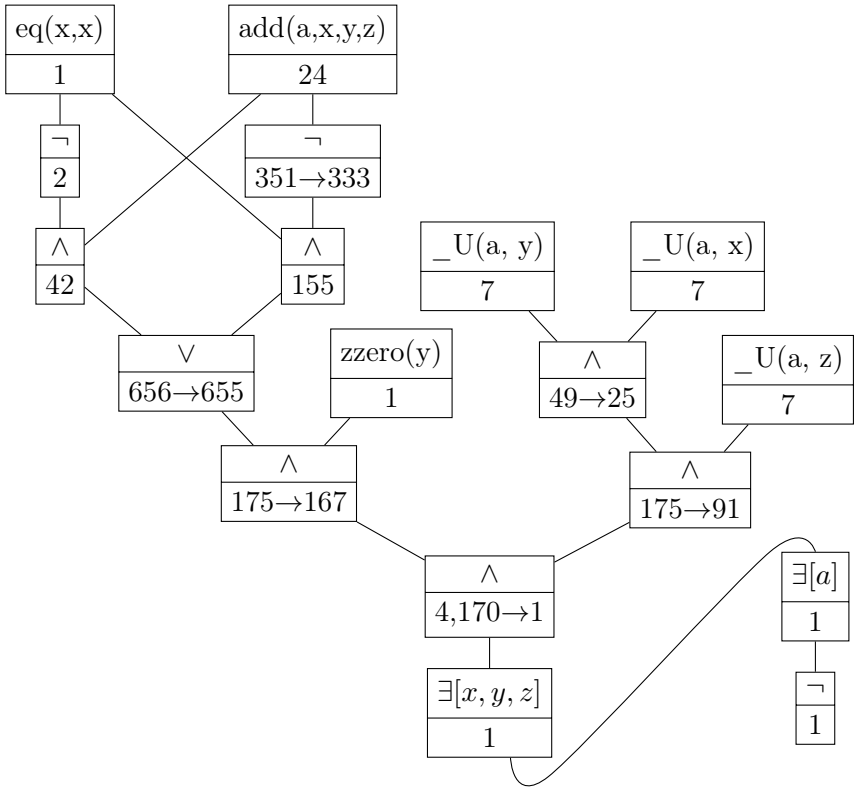

Figure 12: Number of states of automata calculated for subformula of the addition base case

```
(¬(∃[a]. (¬(¬(∃[x, y, z]. (
  ((_U(a, y) ∧ _U(a, x)) ∧ _U(a, z)) ∧
  (¬((¬zzero(y)) ∨
    (((¬add(a, x, y, z)) ∨ eq(x, z)) ∧ ((¬eq(x, z)) ∨ add(a, x, y, z)))
)))))))))
```

Then it optimizes this formula. Here, double negations are removed and De Morgan's laws are applied. After this step, the formula is simplified to the following:

```
(¬(∃[a]. (∃[x, y, z]. (
  ((_U(a, y) ∧ _U(a, x)) ∧ _U(a, z)) ∧
  (zzero(y) ∧
    ((add(a, x, y, z) ∧ (¬eq(x, z))) ∨ (eq(x, z) ∧ (¬add(a, x, y, z))))
)))))
```

Afterwards, it recursively computes automata for all the subformulas. Automata are simplified at each step. We show the number of states of the different automata in figure 12. When the number of states changed after simplification, both numbers are

shown. We can see there, that the negation of the adder automaton already has more than 300 states. The largest involved automaton has 4170 states, that occurred directly before the ∃ operations. But it got simplified to just one state, probably because Pecan does an emptiness check on the automaton as one of the simplification attempts. If that succeeds, the automaton can be simplified to an automaton with just 1 state.
We can do the same analysis for `Addition inductive case (x + s(y) = s(x + y))`. After simplification, Pecan generates the following formula for it.

```
(¬(∃[a]. (_D(a) ∧ (∃[x, y, z, u, v]. (
  ((((_U(a, y) ∧ _U(a, u)) ∧ _U(a, v)) ∧ _U(a, z))
   ∧ _U(a, x)
  ) ∧
  ((_U(a, x) ∧ (succ(a, u, y) ∧ succ(a, v, z))) ∧
   ((add(a, x, y, z) ∧ (¬add(a, x, u, v))) ∨
    (add(a, x, u, v) ∧ (¬add(a, x, y, z)))
)))))))
```

And the `succ` operation is defined as follows

```
succ(a,x,y)=
 (leq(x, y) ∧
  (_U(a, x) ∧
   (_U(a, y) ∧
    ((¬eq(x, y)) ∧
     (¬(∃[z]. (_U(a, z) ∧ ((¬leq(z, x)) ∧ (¬leq(y, z))))))
 ))))
```

We can analyze the size of the involved automata with the same approach. We don't show the number of states of the automata of all the involved subformulas here as that would take a lot of space. But it's interesting to show how the *succ* function is translated. Doing so also shows why we defined the function the way we did define it. When looking at the definition of `Addition inductive case` one might ask why we used the U relations inside the formula. As we already use them at the beginning of the formula in the quantifier, we could have also used the following way to formulate it:

```
succ2(a, x, y) := leq(x, y) & !eq(x, y)
    & forall z. if _U(a, z) then (leq(z, x) | leq(y, z))

Theorem ("Addition inductive case' (x + s(y) = s(x + y))", {
    forall a is _D. forall x,y,z,u,v are _U(a).
        if (succ2(a,u,y) & succ2(a,v,z)) then
            (add(a,x,y,z) iff add(a,x,u,v))
}).
```

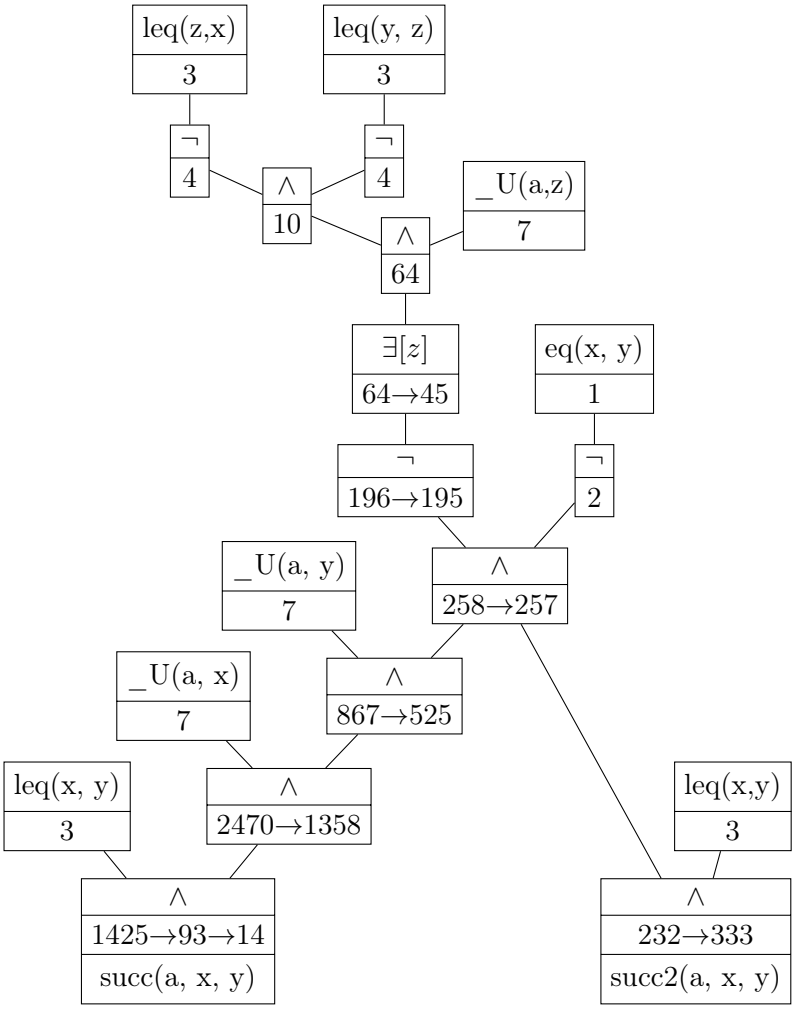

Figure 13: Automata involved in calculating succ(a,x,y) *(left)* and succ2(a,x,y) *(right)*

I have tried executing Pecan on this version of the theorem unsuccessfully. I even tried running it on a machine with more than 128GB of RAM memory. But after running for more than one hour, it had consumed 128GB of memory and then died with a memory-related error. We can compare this to the original program. The original program only differs from this one by having 3 more occurrences of U relations, two in the *succ* function and one in the theorem. That original execution finished in only a few seconds and consumed only a few megabytes of memory.

Why is there such a big difference? We show in figure 13 how many states the automaton for the `succ` and `succ2` functions have. The `succ` function has just 14 states, but the `succ2` function has 333 states. Interestingly, the number of states increased from an earlier 232 states. This is probably the case, because Pecan attempted to convert the automaton to a Buchi automaton. This can increase the number of states of an automaton

with a different acceptance condition.
It is notable that the automaton for the `succ2` function has much more states than the automaton for the `succ` function. One thing that appears important to note is that all the subformulas have to be fully expressed by the automata in Pecan. So even if they're later simplified, for example if some formula φ is only used in the expression $\varphi \wedge \bot$, Pecan can't skip the calculation of the automaton for φ. So it sometimes appears to be useful to combine some restrictions to some subformula already deeper in a big formula. Instead of doing that only at the end. Even doing so in the end and inside the formula – and thus producing a more complicated formula on first sight – can help to make the structure of some subformula easier and thus help in producing smaller automata for them. It is especially useful when some subformula becomes non-satisfiable. Then the corresponding automaton can be simplified to an automaton with just one state.

6.4. Sturmian words are not eventually periodic

As a last application, we want to check now if any Sturmian words are eventually periodic. We do this by using the following statement:

```
forall a is _D.
forall n, p are _U(a).
if !zzero(p) then
exists m, u are _U(a).
add(a, m, p, u) & leq(n, m) & sturmian(m) & !sturmian(u)
```

Pecan transforms this formula into its low-level form and then does its simplification. The result is the following formula:

```
(¬(∃[a]. (_D(a) ∧ (∃[n, p]. ((_U(a, p) ∧ _U(a, n)) ∧
  ((¬zzero(p)) ∧
   (¬(∃[m, u]. ((_U(a, m) ∧ _U(a, u)) ∧
     (add(a, m, p, u) ∧ (leq(n, m) ∧ (sturmian(m) ∧ (¬sturmian(u))))))
)))))))))
```

It is notable, that there are a limited number of negation operations there. We only have 3 negations. This was intentional, we have written the formula in such a way, that it would translate into a low number of negation operations, as negation is a computationally very expensive operation. The first negation is not particularly problematic, as the negation is done on a pretty small automaton. The last negation is also happening after the formula has already been simplified. The most challenging is then the negation in the middle. Indeed, we can see this in figure 14. We have shown the number of states of all the involved automata there. The automaton, that is negated in the middle consists of 483 states (once simplified) and after negation this number explodes to 597,468. This automaton then grows to some extent when another small automaton with 25 states is added with a $\wedge$ operation to 2,307,195 states. Fortunately, there are no words that this automaton accepts at all. This can be checked at a low computational cost by

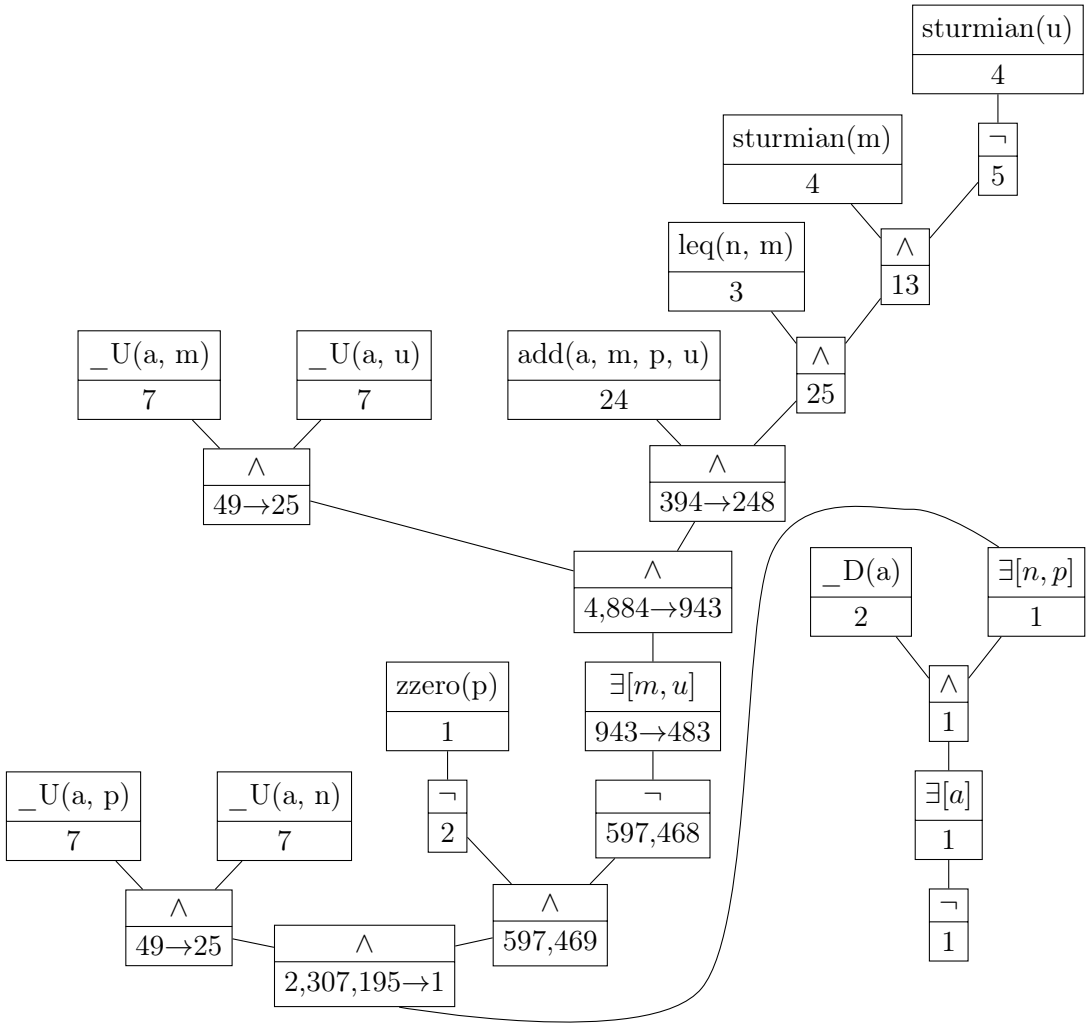

Figure 14: Number of states of automata showing Sturmian words non-periodic

an emptiness check, which Pecan does here before attempting any more complicated simplifications, that may take a long time on such a big automaton. Then Pecan replaces the automaton with an empty automaton consisting of just 1 state. Assigning automata to the rest of the formula is straightforward from there.

An interesting discovery for me was the fact that not all involved automata are Buchi automata. For example, the big automaton with 2,307,195 states has the following acceptance condition: `Inf(0) & (Inf(1) | Fin(2))`. It appears that Pecan did an emptiness check directly on that automaton. Luckily, it didn't try to convert it to a Buchi automaton, as that would have probably taken a very long time.

6.5. Outlook

In the previous section, we have seen that the runtime and complexity of the automata used in Pecan and Spot depend on the exact formulation of the formulas and base automata used. It can change a lot when subtle details are changed and also depends on implementation details of Pecan and Spot. While this problem of exploding runtime cannot be solved in general (negation of automata is expensive generally, see remark 2.9), there may be some improvements that can be made for practically relevant problems. Some improvement ideas:

- Is there a heuristic that tells us how confident we can be of obtaining a response within a reasonable amount of time when formulating a formula in a certain way?
- Which heuristics in automata transformations should be applied in Pecan, and at which points?
- Which simplification attempts should be avoided, as they take a lot of time without achieving their goal of automata simplification?
- Should automata with other acceptance conditions be used? When should they be transformed to Buchi automata again and should that happen at all?
- Another interesting approach might be to introduce new operations on automata, that produce automata that are only in the equivalence class of automata that are M-compatible for some set M. This might allow to produce smaller automata without having to attach further predicates inside formula, as the implementation of such an operation has more space for optimization.

Besides the fact that no Sturmian words are never eventually periodic [5, Automatically Proving Theorems about Sturmian Words] contains some more statements about Sturmian words, that have been checked with Pecan. I have not checked these further statements, but I have tried to check yet another theorem with Pecan:

> For all $\alpha \in (0,1) \setminus \mathbb{Q}$ there exist integers $0 < i < j$ such that for all n, we have either $c_\alpha(n) = c_\alpha(n+i)$ or $c_\alpha(n) = c_\alpha(n+j)$.

I tried to formulate this theorem in a few different ways as first-order formulas, but Pecan never finished successfully (except when I made some errors in the formulation of the first-order formula). I wonder if some of the ideas mentioned above might help to improve computational and memory needs for such Pecan programs and might help making more statements feasible. It might also help Pecan to succeed in more variations of formulations of statements. After all, the goal is to have an *automatic* theorem prover. When we have to know a lot of details about the inner workings of Pecan to be able to formulate statements in specific ways, one could arguably say that such a prove is not automatic.

A. Deutsche Zusammenfassung / German summary

In dieser Arbeit wird die Entscheidbarkeit von bestimmten Theorien gezeigt. Bekanntermaßen ist die Theorie von Presburger Arithmetik entscheidbar:

Theorem A.1. *Sei $\mathcal{L} = \{+\}$ die Signatur von Presburger Arithemtik und sei $\mathcal{M} = (\mathbb{N}, (x, y) \mapsto x + y)$ die Standardstruktur. Dann ist die Theorie von $\mathcal{M}$ entscheidbar.*

Das heißt, dass es einen Algorithmus gibt, der für jede $\mathcal{L}$-Formel φ entscheidet, ob $\mathcal{M} \models \varphi$. Das Hinzufügen von Folgen kann dazu führen, dass die Theorie nicht mehr entscheidbar ist, oder aber sie kann entscheidbar bleiben.

Definition A.2. Sei $s\colon \mathbb{N} \to \{0, 1\}$ eine Folge und sei $\mathcal{L}' = (+, S)$. Definiere die $\mathcal{L}'$-Struktur $\mathcal{M}_s$ folgendermaßen:

$$\mathcal{M}_s = (\mathbb{N}, (x, y) \mapsto x + y, \{n \in \mathbb{N} \mid s(n) = 1\})$$

Viele Eigenschaften von s lassen sich als $\mathcal{L}'$-Formel φ formulieren, sodass $\mathcal{M}_s \models \varphi$ genau dann gilt, wenn s die entsprechende Eigenschaft besitzt. Zum Beispiel lässt sich die Eigenschaft "s wird irgendwann periodisch" durch eine solche Formel formulieren. Falls die Theorie von $\mathcal{M}_s$ nun entscheidbar ist, gibt es einen Entscheidungsalgorithmus für solcherlei Fragestellungen.
Für eine bestimmte Art von Folgen s, die durch einen endlichen Automaton darstellbar sind, ist die Theorie tatsächlich entscheidbar. In dieser Arbeit schauen wir uns den Spezialfall der Folgen der stürmscher Wörter an. Stürmsche Wörter sind folgendermaßen definiert:

Definition A.3. Für $\alpha \in (0, 1) \setminus \mathbb{Q}$ ist das charakteristische stürmsche Wort $c_\alpha\colon \mathbb{N} \to \{0, 1\}$ definiert durch

$$c_\alpha(n) = \lfloor \alpha(n + 1) \rfloor - \lfloor \alpha n \rfloor.$$

$\mathcal{M}_s$ ist für bestimmte stürmsche Wörter entscheidbar. Um das zu sehen, kann man das Ostrowski Numerationssystem benutzen. In diesem sind stürmsche Wörter leicht durch endliche Automaten charakterisierbar.
Unser Ziel ist aber auch, zu zeigen, dass die gemeinsame Theorie von Sätzen, die für alle stürmschen Wörter gelten, entscheidbar ist:

Theorem A.4. *Die Theorie*

$$T = \{\varphi \mid \mathcal{M}_{c_\alpha} \models \varphi \text{ für alle } \alpha \in (0, 1) \setminus \mathbb{Q}\}$$

ist entscheidbar.

Um dies zu sehen müssen unendliche Wörter statt nur endlicher Wörter benutzt werden. In dieser Arbeit werden in Kapitel 2 nötige Grundlagen dazu eingeführt. Es geht zunächst um unendliche Wörter und Sprachen und dann um Büchi-Automaten. Büchi-Automaten sind eine Verallgemeinerung des endlichen Automaten auf unendliche Wörter. Im darauffolgenden Kapitel wird eine Methode erläutert, die benutzt werden kann, um zu zeigen,

dass Theorien von Strukturen entscheidbar sind. Dieses wird beispielhaft angewandt, um zu zeigen, dass Pressburger Arithmetik entscheidbar ist. In Kapitel 4 werden dann Ostrowski-Numerationssysteme und deren Verbindung zu stürmschen Wörtern vorgestellt. Darauf schließt das Kapitel mit dem Hauptresultat an. Dort wird zunächst eine Variante vorgestellt, Ostrowski-Zahlen mit einem endlichen Alphabet zu repräsentieren. Dazu werden die einzelnen Ziffern der Ostrowski-Representation wiederum mittels Binärdarstellung dargestellt. Dies ist notwendig, da ansonsten unendlich viele Ziffern notwendig wären; Büchi-Automaten aber nur mit einem endlichen Alphabet arbeiten können. Schließlich widmet sich dieses Kapitel der Konstruktion eines Automaten zur Addierung von Zahlen in Ostrwoski-Numerations-Systemen. Dieser Automat wird explizit hergeleitet, in Form von Graphen dargestellt sowie seine Richtigkeit bewiesen. Der Automat hat nur 24 Zustände und ist damit vergleichsweise klein.
Im letzten Kapitel wird dann noch auf die praktische Umsetzung des maschinengestützten Beweisens von Eigenschaften über T eingegangen. Zunächst wird die Software Pecan eingeführt, die benutzt werden kann, um Sätze in ω-regulären Strukturen auf ihre Richtigkeit zu überprüfen. Pecan benutzt die Softwarebibliothek Spot, um Automaten auf unendlich langen Wörtern darzustellen und zu verwenden. Spot unterstützt nicht nur Büchi-Automaten, sondern auch noch allgemeinere Akzeptanzbedingungen. Dies wird etwas erläutert. Dann wird gezeigt, wie Pecan benutzt werden kann, um das Vertrauen zu vergrößern, dass der in Kapitel 4 konstruierte Addierer-Automat für Ostrowski-Numerationssysteme tatsächlich korrekt ist. Abschließend wird mit dieser Methode gezeigt, dass Stürmische Wörter nie periodisch werden. Anhand dieser Beispiele wird außerdem erläutert, wie sehr die benötigte Zeit zur Berechnung des Wahrheitsgehalts einer Aussage und die Komplexität der enthaltenen Automaten von Details der Formulierung abhängen und es werden Verbesserungsideen vorgeschlagen.

B. Supplementary digital materials

This section provides an overview of the different folders that accompany this thesis as supplementary digital materials. The content was originally included on a USB flash drive and is now available at `https://gitlab.com/fkz/ostrowski`.

Folder pecan-docker This folder contains files that can be used to create a Docker image that can be used to execute Pecan. Using a Docker image is a good way to use Pecan, as all needed software is encapsulated. On the host operating system only Docker needs to be installed. The following command can be used to build the Docker image: `docker build --pull -t pecan pecan-docker`

Folder pecan-scripts This folder contains the scripts, that are referenced in the Bachelor thesis. These scripts were executed and some of the results are part of the Bachelor thesis. They depend on the automata in the automata subfolder and consists of the following 4 Pecan files:

- `script-1-improving-confidence-in-the-adder-automaton.pn`
- `script-2-improving-confidence-in-the-adder-automaton-alternative-succ-function.pn`
- `script-3-sturmain-words-are-not-eventually-periodic.pn`
- `script-4-fake-improving-confidence-in-the-adder-automaton.pn`

These scripts can also be found in appendix E.

Folder pecan-scripts/automata Here are the automata, that are needed by the scripts. They are also contained in appendix D and appendix C. All automata are hand-written, except for the adder automaton. The adder automaton `add.txt` has been generated by executing `gen-adder.js`. This script is also contained in this folder.

Folder pecan-script-results The results of executing the 4 scripts in the pecan-scripts folder are stored here. These scripts were executed with the `-ddd` option to output detailed information about the involved subautomata. This information has been used in the thesis in figures 12-14.

Folder pecan-script-runner This folder contains some scripts, that I used to execute the Pecan scripts. They download all the necessary files to a remote machine and also download the Docker image there. The results of the execution of the Pecan program are then uplaoded to a Git repository. I used these scripts to execute long running Pecan scripts on Hetzner cloud machines, but it should also be possible to execute them on any Linux machines. They just need Docker and Systemd preinstalled, all other software is automatically installed, if not already installed. The script has hardcoded locations and access tokens for the git repository, where the results should be pushed, as well as the docker registry where the pecan docker image can be downloaded (that can be created with the `pecan-docker` folder). To reproduce execution of these scripts, these locations and credentials need to be changed.

C. Source code to produce the adder automaton

To be able to use the adder automaton in Pecan, it needs to be supplied to it. As the automaton is already pretty big, I created a small program that generates the automaton in the pecan syntax. It is written in Javascript and can be executed with node. It will generate a file *add.txt* in the current directory when called. Instead of # we use 2 for the seperator symbol here, so our alphabet is $\{0, 1, 2\}$ instead of $\{0, 1, \#\}$. This is expected for the Pecan format. In the Spot representation, it will then be represented using 2 atomic propositions.

Listing 1: gen-adder.js

```
const fs = require("fs").promises;

const stateCount = 7;

const rIndex = [
    0, 0, 0,
    -1, -1, -1,
    1
];

const sIndex = [
    1, 0, -1,
    1, 0, -1,
    -1
];

function getState(r, s) {
    for (let i = 0; i < stateCount; ++i) {
        if (rIndex[i] == r && sIndex[i] == s) {
            return i;
        }
    }
    return undefined;
}

let binaryEdges = [];
for (let i = 0; i < 16; ++i) {
    binaryEdges.push([
        i % 2,
        Math.floor(i / 2) % 2,
        Math.floor(i / 4) % 2,
        Math.floor(i / 8) % 2
    ]);
```

```
}

function binarySum(binaryEdge, r) {
    return r*binaryEdge[0] + binaryEdge[1] + binaryEdge[2] -
        binaryEdge[3];
}

let visitedPositions = Array.from(
    {length: stateCount},
    () => { return {}; }
);
let toVisit = [];

function nextVisit(state, carry) {
    toVisit.push([state, carry]);
}

function addVisited(state, carry) {
    visitedPositions[state][carry] = true;
}

function visit() {
    let next;
    while (next = toVisit.pop()) {
        if (!visitedPositions[next[0]][next[1]]) {
            return next;
        }
    }
    return undefined;
}

async function visitPosition(file, state, carry) {
    addVisited(state, carry);
    const r = rIndex[state];
    const s = sIndex[state];

    const isFinalBool = r == 0 && s == 0 && carry == 0;
    const isFinal = isFinalBool ? "1" : "0";

    await file.write(`\n${state}${carry}: ${isFinal}\n`);

    for (const e of binaryEdges) {
        const sum = binarySum(e, r) + carry;
        if (sum % 2 == 0) {
```

```
            const newCarry = sum / 2;
            const line = `${e[0]} ${e[1]} ${e[2]} ${e[3]}` +
                ` -> ${state}${newCarry}\n`
            await file.write(line);
            nextVisit(state, newCarry);
        }
    }

    if (carry === 0) {
        for (let t = -1; t <= 1; ++t) {
            const newState = getState(s, t);
            const newCarry = t - r;
            if (newState !== undefined) {
                await file.write(
                    `2 2 2 2 -> ${newState}${newCarry}\n`
                );
                nextVisit(newState, newCarry);
            }
        }
    }
}

async function visitStart(file) {
    await file.write("start: 0\n");
    for (let i = 0; i < stateCount; ++i) {
        const s = sIndex[i];
        nextVisit(i, s);
        await file.write(`2 2 2 2 -> ${i}${s}\n`)
    }
}

async function main() {
    const file = await fs.open("add.txt", "w");
    await file.write("{0,1,2} {0,1,2} {0,1,2} {0,1,2}\n");
    await visitStart(file);
    let next;
    while (next = visit()) {
        await visitPosition(file, next[0], next[1]);
    }
    await file.close;
}

main();
```

D. Source code of other needed automata

We give the definition of the other needed automata here. They are a textual representation of the automata given in figure 4, figure 11, figure 5 and figure 9.

Listing 2: D.txt

```
{0,1,2}

start: 1
2 -> other

other: 0
0 -> other
1 -> other
1 -> start
```

Listing 3: eq.txt

```
{0,1,2} {0,1,2}

start: 1
0 0 -> start
1 1 -> start
2 2 -> start
```

Listing 4: leq.txt

```
{0,1,2} {0,1,2}
start: 0
2 2 -> leq

leq: 1
0 0 -> leq
0 1 -> leq
1 0 -> gt
1 1 -> leq
2 2 -> leq

gt: 0
0 0 -> gt
0 1 -> leq
1 0 -> gt
1 1 -> gt
2 2 -> gt
```

Listing 5: U.txt

```
{0,1,2} {0,1,2}

start: 0
2 2 -> smaller
2 2 -> zero
2 2 -> final

smaller: 0
0 0 -> smaller
0 1 -> smaller
1 0 -> smaller
1 1 -> smaller
1 0 -> equal
1 0 -> start


zero: 0
0 0 -> zero
1 0 -> zero
1 0 -> zero_f

zero_f: 0
2 2 -> equal

equal: 0
0 0 -> equal
1 1 -> equal
1 1 -> start

final: 0
0 0 -> final
1 0 -> final
1 0 -> final_f

final_f: 1
2 2 -> final
```

Listing 6: zero.txt

```
{0,1,2}

start: 1
0 -> start
2 -> start
```

Listing 7: sturmian.txt

```
{0,1,2}

start: 0
2 -> odd

odd: 0
0 -> odd
2 -> even

even: 0
0 -> even
1 -> final
2 -> odd

final: 1
0 -> final
1 -> final
2 -> final
```

E. Pecan scripts

These Pecan scripts can also be found on the flash drive in the folder `pecan-scripts`. All files start with the following lines to load the needed automata:

```
#load("automata/D.txt", "pecan", _D(a))
#load("automata/U.txt", "pecan", _U(a, x))
#load("automata/leq.txt", "pecan", leq(x, y))
#load("automata/eq.txt", "pecan", eq(x, y))
#load("automata/zero.txt", "pecan", zzero(z))
#load("automata/add.txt", "pecan", add(a, x, y, z))
#load("automata/sturmian.txt", "pecan", sturmian(n))
```

We only show the remaining lines for the 4 scripts, that are used in this thesis.

Listing 8: script-1-improving-confidence-in-the-adder-automaton.pn

```
succ(a, x, y) := leq(x, y) & _U(a, x) & _U(a, y) & !eq(x, y)
            & forall z. if _U(a, z) then (leq(z, x) | leq(y, z))

Theorem ("Addition base case (0 + y = y)", {
    forall a. forall x,y,z are _U(a).
        if zzero(y) then (add(a,x,y,z) iff eq(x,z))
}).

Theorem ("Addition inductive case (x + s(y) = s(x + y))", {
    forall a is _D. forall x,y,z,u,v are _U(a).
        if (_U(a, x) & succ(a,u,y) & succ(a,v,z)) then
            (add(a,x,y,z) iff add(a,x,u,v))
}).

Theorem("Addition inductive case (x + s(y) != 0)", {
    forall a is _D. forall x,y,z,u are _U(a).
        if (succ(a, y, u) & zzero(z)) then !add(a, x, u, z)
}).
```

Execution of this program returns the following

```
Addition base case (0 + y = y) is true.
Addition inductive case (x + s(y) = s(x + y)) is true.
Addition inductive case (x + s(y) != 0) is true.
```

Listing 9: script-2-improving-confidence-in-the-adder-automaton-alternative-succ-function.pn

```
succ2(a, x, y) := leq(x, y) & !eq(x, y)
    & forall z. if _U(a, z) then (leq(z, x) | leq(y, z))

Theorem ("Addition base case (0 + y = y)", {
    forall a. forall x,y,z are _U(a).
        if zzero(y) then (add(a,x,y,z) iff eq(x,z))
}).

Theorem ("Addition inductive case (x + s(y) = s(x + y))", {
    forall a is _D. forall x,y,z,u,v are _U(a).
        if (succ2(a,u,y) & succ2(a,v,z)) then
            (add(a,x,y,z) iff add(a,x,u,v))
}).

Theorem("Addition inductive case (x + s(y) != 0)", {
    forall a is _D. forall x,y,z,u are _U(a).
        if (succ2(a, y, u) & zzero(z)) then !add(a, x, u, z)
}).
```

This program doesn't finish succesfully. Instead, after running for about 90 minutes and using more than 130GB of RAM, it fails with the following error:

```
terminate called after throwing an instance of 'std::bad_alloc'
```

Listing 10: script-3-sturmain-words-are-not-eventually-periodic.pn

```
Theorem ("Sturmian words are not eventually periodic", {
    forall a is _D.
    forall n, p are _U(a).
    if !zzero(p) then
    exists m, u are _U(a).
    add(a, m, p, u) & leq(n, m) & sturmian(m) & !sturmian(u)
}).
```

Execution of this program returns the following

```
Sturmian words are not eventually periodic is true.
```

Here we also show the last two of the common `#load` statements, as the add relation is loaded under a different name.

Listing 11: script-4-fake-improving-confidence-in-the-adder-automaton.pn

```
#load("automata/add.txt", "pecan", addn(a, x, y, z))
#load("automata/sturmian.txt", "pecan", sturmian(n))

#import("base3formats.pn")

succ(a, x, y) := leq(x, y) & _U(a, x) & _U(a, y) & !eq(x, y)
            & forall z. if _U(a, z) then (leq(z, x) | leq(y, z))

fake_add(a,x,y,z) := succ(a, z, y) | addn(a, x, y, z)
add(a,x,y,z) := fake_add(a,x,y,z)

Theorem ("Addition base case (0 + y = y)", {
    forall a. forall x,y,z are _U(a).
        if zzero(y) then (add(a,x,y,z) iff eq(x,z))
}).

Theorem ("Addition inductive case (x + s(y) = s(x + y))", {
    forall a is _D. forall x,y,z,u,v are _U(a).
        if (_U(a, x) & succ(a,u,y) & succ(a,v,z)) then
            (add(a,x,y,z) iff add(a,x,u,v))
}).

Display example base3FormatI {
    _U(a,x) & _U(a,y) & _U(a,z) &
    add(a,x,y,z) & !addn(a,x,y,z)
}.

Theorem("Addition inductive case (x + s(y) != 0) not true", {
    !(forall a is _D. forall x,y,z,u are _U(a).
        if (succ(a, y, u) & zzero(z)) then !add(a, x, u, z))
}).
```

Execution of this program returns the following

```
Addition base case (0 + y = y) is true.
Addition inductive case (x + s(y) = s(x + y)) is true.
[(z,2100(20)^ω),(y,2010(20)^ω),(x,2010(20)^ω),(a,2011(21)^ω)]
Addition inductive case (x + s(y) != 0) not true is true.
```

The third line of the output is an example of values for x, y, z, d such that fake_add(d, x, y, z) holds, but $+(d, x, y, z)$ does not hold. We have $d = \#011(\#1)^\omega$, $x = h_d^{-1}(2)$, $y = h_d^{-1}(2)$ and $z = h_d^{-1}(1)$. Clearly $2 + 2 \neq 1$.

References

[1] Alexandre Duret-Lutz et al. "From Spot 2.0 to Spot 2.10: What's New?" In: *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*. Vol. 13372. Lecture Notes in Computer Science. Springer, Aug. 2022, pp. 174–187. DOI: 10.1007/978-3-031-13188-2_9.

[2] Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences. Theory, applications, generalizations*. English. Cambridge: Cambridge University Press, 2003. ISBN: 0-521-82332-3. DOI: 10.1017/CBO9780511546563.

[3] Aseem Baranwal, Luke Schaeffer, and Jeffrey Shallit. "Ostrowski-automatic sequences: theory and applications". English. In: *Theor. Comput. Sci.* 858 (2021), pp. 122–142. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2021.01.018.

[4] J. R. Büchi. "Weak second-order arithmetic and finite automata". English. In: *Z. Math. Logik Grundlagen Math.* 6 (1960), pp. 66–92. ISSN: 0044-3050. DOI: 10.1002/malq.19600060105.

[5] Philipp Hieronymi et al. "Decidability for Sturmian words". In: *CoRR* abs/2102.08207 (2021). arXiv: 2102.08207. URL: https://arxiv.org/abs/2102.08207.

[6] Christof Löding. "Methods for the transformation of ω-automata: Complexity and connection to second order logic". In: *Diplomata thesis, Christian-Albrechts-University of Kiel* (1998). URL: https://old.automata.rwth-aachen.de/users/loeding/diploma_loeding.pdf (visited on 02/02/2023).

[7] Hamoon Mousavi. "Automatic Theorem Proving in Walnut". In: *CoRR* abs/1603.06017 (2016). arXiv: 1603.06017. URL: http://arxiv.org/abs/1603.06017.

[8] Hamoon Mousavi, Luke Schaeffer, and Jeffrey Shallit. "Decision algorithms for Fibonacci-automatic Words, I: Basic results". en. In: *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 50.1 (2016). DOI: 10.1051/ita/2016010. URL: http://www.numdam.org/articles/10.1051/ita/2016010/.

[9] Reed Oei et al. "Pecan: An Automated Theorem Prover for Automatic Sequences using Büchi Automata". In: *CoRR* abs/2102.01727 (2021). arXiv: 2102.01727. URL: https://arxiv.org/abs/2102.01727.

[10] M. Presburger. *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt*. German. C. R. Congrès Math. Pays slaves 92-101, Zusatz ebenda, 395 (1930). 1930.

[11] Noah Schweber. *Are function symbols unnecessary in first-order logic?* Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/3353180.

[12] Jeffrey Shallit. *The logical approach to automatic sequences. Exploring combinatorics on words with Walnut*. English. Vol. 482. Lond. Math. Soc. Lect. Note Ser. Cambridge: Cambridge University Press, 2023. ISBN: 978-1-108-74524-6; 978-1-108-77526-7. DOI: 10.1017/9781108775267.

[13] Ryan Stansifer. *Presburger's Article on Integer Airthmetic: Remarks and Translation*. Tech. rep. TR84-639. Cornell University, Computer Science Department, Sept. 1984.