From Context Transformation Algorithm to N^* model of Set Theory

Yauhen Yakimovich*
yauhen.yakimovich.mail@gmail.com

Monday 21st April, 2025
v1.0.0

Abstract

This paper proposes Set Theory models N of $ZF + DC$ that explicitly decides the Continuum Hypothesis (CH) as well as N^* of $ZF + DC_\lambda + I$ that implies the existence of *supercompact cardinal*. The approach employs advanced and somewhat novel techniques from mathematical logic, including Boolean-valued models and String Enumeration Formulas (SEF) to build *replicata* structures as used for the models. It follows that CH becomes resolved for all instances of large enough models of $ZF + DC$, providing a concrete solution to a long-standing open problem, so that Goedel's conjecture $2^{\aleph_0} = \aleph_2$ holds. Other results offer new insights into the structure of N^* and its implications for related mathematical theories, laying a foundation for further research on infinite sets.

*Please see the end of the paper for Acknowledgements

Contents

For Mary and Emilia

1 Preface

Embarking on the exploration of a simple algorithm years ago marked the beginning of an extraordinary journey for me. This quest soon transformed not only into a hobby, but rather what I affectionately refer to as my favorite long-term relationship. Back in 2007, as a software engineer with practical experience in the industry, I found myself increasingly drawn to the theoretical aspects of mathematics, eventually developing a keen interest in Set Theory.

The concept of applying the CTA algorithm in an infinite context did not emerge suddenly. Instead, it was a gradual realization, spurred by my encounter with Cantor's diagonal argument (which to my shame I completely misunderstood at first and was able to fully comprehend only years after graduating in technical field). This personal discovery was a significant turning point, opening up the new and unknown realm of infinite for me.

My initial intention was purely educational, aimed at expanding my understanding as much as possible. Over the years, this scholarly pursuit evolved. I gathered my scattered notes, gradually mustering the courage to compile them into a coherent manuscript. The objective was to produce a concise paper, offering a computer science perspective on the notion of infinities, particularly in relation to the continuum hypothesis.

However, as time passed, it became painfully clear through numerous trials and errors that a computer science lens alone was inadequate. It lacked the necessary precision not just for answering questions, but even for formulating them correctly. This realization necessitated a deeper, more fundamental approach to learning.

A pivotal moment in my journey was the acknowledgement that disproving the continuum hypothesis (CH) or proposing a naive theoretical example of its possibility was not a novel achievement. Paul Cohen's groundbreaking 1963 paper, which elegantly and rigorously demonstrated the independence of CH using the sophisticated method of forcing, was a humbling reminder of the complexity of these problems.

Despite these challenges, I remained motivated by the belief that my unique approach might still offer a fresh perspective or become a useful tool in the field. This paper is the result of years of learning, exploring, and refining these ideas.

Now, as I conclude this draft, I am ready to present my findings. This paper represents not just an academic endeavor, but also a personal journey from a software engineering background into the depths of mathematical theory.

2 Introduction

Traversing a binary string is a process that is trivial enough to imagine. This is equivalent to enumerating some sequence of values. Each value can be either zero or one. The process of enumerating such values allows labeling each position with a number or an *index*, so that we can always find out the value at certain position. A sequence of $\{0, 1\}$ equipped with such indexing will then be a binary string, but the same can be stated more formally.

Definition 1.

If the position of each value of some binary sequence can be enumerated by an index set $J \subseteq \mathbb{N}$, so that each index can be mapped to the binary value at the respective position, then we call such mapping $b : J \to \{0, 1\}$ a binary string.

Existence of the above mapping also means that the binary string b has the same "cardinality" as its index set J. We call such "cardinality" *the length of the binary string* and denote this as $|b| = |J|$. For example, if some string b has the length $n \in \mathbb{N}$, then this fact is noted as $|b| = n$ or with the lower subindex b_n.

Sometimes we also use round bracket or double quote notation if we need to explicitly indicate which exact finite sequence of $\{0, 1\}$ does represent some concrete binary string. For example, the following notations of a binary string b_n are equivalent and ordered by increasing brevity:

- $\backsim (1, 0, 0, 1, 1, 1, 0, 1, .., 0, 1, 1)_n$
- $\backsim$ "$1, 0, 0, 1, 1, 1, 0, 1, .., 0, 1, 1$"$_n$
- $\backsim$ "$10011101..011$"$_n$
- $\backsim$ $10011101..011_n$

A set of binary strings that have the same length is denoted as $B_n = \{b_n \in |n \in \mathbb{N}\}$.

Definition 2.

Consider a set of binary strings B_n of a finite length n from 2^n of possible strings of that length, *s.t.* $|B_n| = 2^n$. We say that two strings $x_n \in B_n$ and $y_n \in B_n$ are *equal* iff values in both strings are equal at each position j as in $x_n(j) = y_n(j), \forall j \in J$, where J is some index set shared by x_n and y_n.

By analogy of the opposite, the *inequality* of $x_n, y_n \in B_n$ follows from $\exists j, j \in J : x_n(j) \neq y_n(j)$.

Given the above definitions we are going to look at binary strings of some finite length in the first part of the paper and consider context dependent transformation of their values called *context transformation algorithm* (CTA). In the second part of the paper we will discuss CTA applications to more fundamental concepts such as cardinality of infinite binary strings. Some other generalizations of the CTA as well as listings of code are provided in appendix of this paper.

3 Finite case of CTA

3.1 Context depended operations

Following earlier definitions one can proceed with some basic transformations of binary strings. Consider a boolean logic operation like *xor* computed over values of two strings $\{x_n, y_n\} \subset B_n, n \in \mathbb{N}$ individually. The result of such string operation z_n can be obtained as value-wise or element-wise addition modulo 2, s.t. $z_n(j) = x_n(j) \oplus y_n(j), \forall j \in J$, where J is an enumeration or index set.

Computing a sequence $z_n(j), \forall j \in J$ is done index by index, defined by the shared enumeration set J with each position being treated independently. Meaning that the end result does not depend on the order of operations as long as the bits of the outcome binary string z_n can be put together in the right order $\forall j \in J$. We will call such operations *context-free operations*.

A different situation is a special case when some basic binary operation is applied to strings of bits in strictly dependent order. Indeed, the trivial case of performing a *xor* operation over two binary strings can be computed in parallel and the outcome will be always the same over and over again as long as we can match the indexes.

However, if one can control how the resulting z_n string is produced by adding some memory property to store previously computed or observed states, one will most likely come up with a stateful process of a binary string transformation. We will refer to such string operations augmented with the memory state as *context depended operations*.

Definition 3.

Consider an input binary string $b_n, n \in \mathbb{N}$ and a cursor $b_n(k)$ pointing at some index $0 \leq k < n$. A context of a fixed length $m < n$ observed to the left of k is a *prefix* enumerating values before index k as "$b_n(k-m), .., b_n(k-1)$".

3.2 Context Transformation Algorithm

Let us illustrate how one of such context depended algorithms can work. Let b_n be a finite input string. We will use a state lookup table (see fig. 1) for state tracking purposes. One needs to choose the length of the context as $m < n, m \in \mathbb{N}$, so that the left-hand of the lookup table will consists of all possible context strings of such length $C_m = \{$"000..00"$_m$, "000..01"$_m$, .. "111..11"$_m\}$, $|C_m| = 2^m$ that can be potentially found while traversing b_n with $k \in J =$ "0, .., $n-1$".

For convenience, prefix values which have not yet been observed before some cursor $k < m$ are padded with zeros. For example, for $k \geq 0$ and $m = 4$ first few observed context strings will be padded until no more padding is

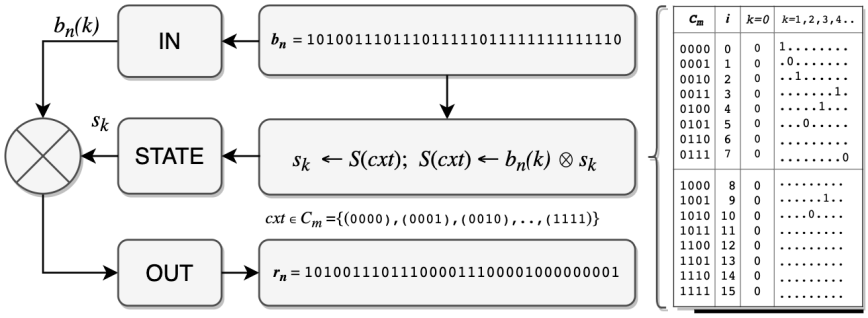


Figure 1: Context transformation algorithm. Finite case.

required as in $(0000, 000x, 00xx, 0xxx, xxxx)$, where $x \in \{0, 1\}$ is a value placeholder and $k \geq m$. Alternatively one can imaging a sliding window of fixed length $m < n$ moving from left to right over the indexes of b_n to observe all found context strings of that length.

We define a finite case of *CTA* as following:

1. **take** a finite binary string b_n as input - for example, let $b_n =$ "10100111011101111101111111111110"$_{n=33}$
2. **choose** the size of the context as $m < n, m \in \mathbb{N}$ - for example, $m = 4$
3. **point** cursor at position $k = 0, k \in J$, so that we can access first value $1 \leftarrow b_n(k)$ at the beginning of b_n
4. **initialize** all unobserved states in the state table with zeros $S(cxt) \leftarrow 0, \forall cxt \in C_m$
5. **set** the observed context $cxt \leftarrow$ "000..00"$_m$
6. **shift** bits of the observed context to the left by dropping a value at the first index $cxt(0)$
7. **update** the observed context string by appending the input value at cursor $cxt \leftarrow$ "$cxt(1), cxt(2), .., cxt(m-1), b_n(k)$"$_m$
8. **take** $s_k \leftarrow S(cxt)$
9. **output** next bit $r_n(k) \leftarrow b_n(k) \oplus s_k$
10. **update** memory state for observed context $S(cxt) \leftarrow b_n(k) \oplus s_k$, which will be 1 if the state has changed and 0 otherwise
11. **if** $k = n$ and cursor points at the end of the input string, **then** transformation is complete and we can terminate the algorithm **else** move cursor to the next index $k \leftarrow k + 1$ at the input string b_n (from left to right) and continue by jumping back to step (6)

3.3 Bijectivity of CTA

Above we have defined the CTA as a context-depended algorithm that takes a finite binary string b_n as an input and uses a simple *xor* operation applied to the input bits and the memory states to produce an output string r_n. For example and as shown on fig. 1 - $r_n \leftarrow CTA(b_n)$, so for the sample input the CTA result is:

$$b_n = \text{“}10100111011101111101111111111110\text{”}$$
$$r_n = \text{“}10100111011100001110000100000001\text{”}$$

Interestingly enough, *xor* operation on binary strings can be easily reversed. Which means, that one can also define an inverse version of the algorithm such that it will take r_n as input and produce the original values of b_n as output - $b_n \leftarrow CTA^{-1}(r_n)$.

Theorem 1.

If a context transformation algorithm on finite binary strings B_n can be represented by a function $CTA_m : B_n \to B_n$, where context length is $m < n : m, n \in \mathbb{N}$, then CTA_m is both a total computable and bijective function.

Proof. There exists a direct and inverse implementation of CTA_m in Turing-complete programming language[1]. □

[1] See ”CTA - code listing” in appendix - for direct and inverse algorithm implementation in python

4 Countable case of CTA

4.1 Infinite binary strings

So far we have looked at binary strings of finite length. Sets of such strings have been arranged as binary tables over which we run a context-depend transformation to produce an output binary string. Next we will see that sequences of $\{0, 1\}$ can be considered to be rather "large" or unrestricted. For example, not only input and output binary strings, but also individual rows in such tables can be repeatedly padded by zeros or any other algorithmically printable patterns consisting of finite substrings.

Instead of trying to print all of the elements at once of any such cyclic binary string in a single infinite loop, it would be much more pragmatic to approximate the process by printing out only finite subparts of an infinite string that are accessed at particular indexes $\forall j \in J$. For example, when accessing the subsets or snapshots of the values to be printed out or when implementing a tape to store a value using Turing Machine (TM). In other words, those values can be reached and printed out within a certain computational time[2]. We will agree that such access is happening by means of a *cursor* $b(j), \forall j \in J$ - a totally computable function that takes index as an input to return the desired value. We will also refer to such selective access of cyclic binary strings as *algorithmic access*.

Definition 4.

If there exists an algorithm such as a cursor function $b(j), \forall j \in J$ that can *access* some values of an infinite binary string by printing them out, we say about those values that they are *algorithmically reachable*.

Indeed, it is easy to construct a function that will map a sequence of natural numbers to a totally ordered set of binary strings. Such function will be merely a translation from each natural number into a binary format which are padded with zeros (or some repeatable pattern). This fact can be noted as one of the possible enumerations of any infinitely countable set of numbers[3] by a list of binary strings with *one-to-one* and *onto* correspondence.

Lemma 2.

---

[2] or within other reasonable constraints depending on the needs of the respective computational model if it is different from the deterministic TM

[3] such countable set can be defined as a *proper set* depending on the consistency needs of the logical foundation or computational framework, since we don't require infinitely countable set of binary strings to contain all possible variations of binary strings - see section *4.4 Uncountable cardinalities*

If $\mathcal{B}$ is an infinitely countable set of binary strings, then any enumeration of those strings produces an index set $I \subseteq \mathbb{N}$, where $\kappa : I \to \mathcal{B}$ is some particular enumeration of $\mathcal{B}$.

To simplify the notation, the length of the individual string $b \in \mathcal{B}$ is not restricted with any finite $n \in \mathbb{N}$ and the infinite padding is assumed if needed for the algorithmic access of values such as $b(j), j \in J$. It is easy to show that any finite non-cyclic part of such string can be algorithmically accessed as well[4]. More formally this can be stated as following:

Definition 5.

A string b is called semi-open[5] and infinitely countable on the left (right) or *left-infinite* (respectively, *right-infinite*), for short, iff there exists a bijective mapping $\alpha : \mathbb{N} \to J$, such that any index $\forall j = \alpha(n), n \in \mathbb{N}$ can access all values in b as in $b(j) = b(\alpha(n))$ starting from the right bits towards countably many on the left side of the string (respectively, $\exists\, \sigma(n) : \mathbb{N} \to J$, which is bijective and can access values $b(\sigma(n)) : n \in \mathbb{N}$ starting from the left bits towards countably many on the right side of the string).

Definition 6.

A string b is called open and *bi-infinite* iff it does not have a start and an end and is both left- and right-infinite.

By choosing yet a different way to index or map values, one can consider equally infinite strings which would be closed, instead of being open. Obviously closed bi-infinite strings are similar up to isomorphism of a particular mapping to the open bi-infinite strings - a form, which we adopt through the rest of our discussion and without loss of any generality.

Lemma 3.

Let o be open bi-infinite, c - closed bi-infinite and l, r be respectively left- and right-infinite binary strings. It is easy to show that c, l, o, r share the same cardinality equal to that of $|\mathbb{N}|$ or countable cardinality.

In order to show the last statement, it is enough to observe that o can be indexed by some $J \subseteq \mathbb{Z} : |J| = |\mathbb{Z}| = |\mathbb{N}|$.

[4] as long as there is enough memory to store the unique finite substring or it can be produced algorithmically as with translation of any natural number into a binary format

[5] as without end at least once

4.2 Generalized Context Transformation Algorithm

We will generalize CTA and apply it for a bi-infinite string $b : K \to \{0,1\}, K \subseteq \mathbb{Z}$ as an input. Let us start by considering a trivial case when b has a form of "..01010101.." sequence built by an infinite repetition of the (01) pattern in both left and right directions. In order to produce the output string, we will apply the generalized context transformation algorithm (GCTA) by starting somewhere in the middle of b and moving from left to right. Then we construct a GCTA table $T : (i,j) \to \{0,1\}$ with rows indexed by $i \in I, I \subseteq \mathbb{N}$ and columns by $j \in J, J \subseteq \mathbb{Z}$.

Definition 7.

Let b be a bi-infinite binary string with the cursor pointing at some index $k \in K, K \subseteq \mathbb{Z}$ somewhere in the middle of the string. A context observed to the left of k or a *prefix* is a left-infinite string equal to b and obtained by ignoring[6] all the values after the cursor starting with k as in "$k, k+1, k+2, k+3, ..$".

The left-hand of the table $T(i,j), j < 0$ consists of left-infinite strings $\{cxt_i\}, i \in I$ in order to track the already observed prefixes in b on the left. Let the cursor point at $k \in K, K \subseteq \mathbb{Z}$, so that we can access the values "$.., 0 \leftarrow b(k-2), 1 \leftarrow b(k-1), 0 \leftarrow b(k), 1 \leftarrow b(k+1), 0 \leftarrow b(k+2), ..$" and so on. We can add a first entry into the left-hand part of the table which would be "..01010101". As we move the cursor to the next position towards right $1 \leftarrow b(k+1)$, the next prefix to add into the left-hand part of $T(i,j), j < 0$ would be "..10101010".

The right-hand of the GCTA table would consists of right-infinite strings. Values of those strings would be state sequences tracked as

$$s_{i,j+1} \leftarrow S(j|cxt_i) \oplus b(k) \tag{1}$$

where $(i, j \geq 0), i \in I, j \in J$ are the indexes over the occurred transformations given the respective left-infinite context $cxt_i = (T(i,j) : i \in I, j < 0)$.

Lemma 4.

If GCTA table can be built by traversing the input string b, then $\exists \mu : k \to (i,j)$, s.t. input string index $k \in K, K \subseteq \mathbb{Z}$ can be always mapped to GCTA table indexes (i,j) by looking up the context cxt_i on the left-hand of T as well as tracking j on the right-hand of T.

If no transformation has been tracked yet, then conveniently[7] values of the right-hand part of the context-table are assumed to be countably many

[6] cutting off

[7] and without loss of generality

times padded with zeros towards right. In particular, $T(i,0) \to 0, i \in I$ are initial states. As we have now added two prefixes to the left-hand of the GCTA table for the first time, the right-hand of the context table will reflect this with two states being updated in two rows:

1. for $0 \leftarrow b(k), \mu(k) \to (0,1)$:
   update $s_{(i,j)} = s_{(0,1)} = s_{(0,0)} \oplus b(k) = 0$
   given $cxt_i = cxt_0 =$ "..01010101"

2. for $1 \leftarrow b(k+1), \mu(k+1) \to (1,1)$:
   update $s_{(i,j)} = s_{(1,1)} = cxt(1,0) \oplus b(k+1) = 1$
   given $cxt_i = cxt_1 =$ "..10101010"

The transformation can take place in the sense that one can produce an output string by knowing the position of the cursor k and traversing the GCTA table, so that one looks up the prefix cxt_i on the left-hand of the table and computes the *xor* of $b(k)$ with the respective state $s_{i,j}, j \geq 0$ on the right-hand of the table. Every time the cursor continues to move over b from left to right, the newly observed context gets switched and next bit will be produced as a *xor* result to be appended to the bi-infinite output.

It turns out that if we continue to iterate over the input string $b =$ "..01010101..", then no new left-infinite prefixes will be added to the left-hand of T. All the prefixes[8] have been already observed. Also on the right-hand of the table there would be only a single non-zero entry $T(i,j) = s(i,j) = s(1,1) = 1$. The rest of states will continue to be zero. The reason for this is the memory property of the GCTA, which indicates a certain triviality of $b =$ "..01010101.." as being repetitive or *cyclic* by construction.

Definition 8.

Let $T : (i,j) \to \{0,1\}$ be a GCTA table for a bi-infinite input string b with rows indexed by $i \in I, I \subseteq \mathbb{N}$ and columns by $j \in J, J \subseteq \mathbb{Z}$. If rows on left-hand of the GCTA table, namely when $j < 0$, consist only of observed left-infinite prefixes of the input string b obtained by scanning or traversing b from left to right as well as if rows in the right-hand of the GCTA table, namely for $j >= 0$, consist only either of updated state values or unset placeholders, then such GCTA table $T(i,j)$ is called *proper*.

Earlier, we have also mentioned the idea that cyclic strings like $b =$ "..01010101.." can be generated by a repetitive pattern or a program loop. Let us state this more formally by using the above notion of GCTA table.

Definition 9.

---

[8] the two of them

Let b be a bi-infinite string. If one can show that b consists only of a single repetitive pattern - a finite substring $s \subset b$ that can be printed by an infinite program loop, then we say that b is a *cyclic* bi-infinite string.

Theorem 5.

Let b be a bi-infinite input string and $T(i, j)$ be a proper GCTA table for b. A bi-infinite string b is *cyclic* iff $T(i, j)$ has a finite number of rows, i.e. $\exists e \in \mathbb{N} : i <= e$

Proof. Take any cyclic bi-infinite input string b. Since b is cyclic it means that b consists of some repetitive patterns, which are formed by some finite substring s. Now start scanning b from left to right in order to produce desired GCTA table $T(i, j)$. While scanning, one would move the cursor k over b in a loop which would be equivalent to continuously popping values from s and padding them on the right side to some $cxt_i =$ "$..b(k-2), b(k-1), b(k)$" shifting the cursor $k \leftarrow k + 1$. Each such padding operation will produce entries for the left hand of $T(i, j)$ table. Note that because s is finite, $\exists e \in \mathbb{N} : e = |s|$. It means that once we will reach the end of s, no new unique context row will be produced to be added to $T(i, j)$ left hand. Instead we will continue to advance the right hand column $j \leftarrow j + 1$ in order to append the state part of GCTA table with the observed states in form of right-infinite strings. Given that s is finite, so would be the number of rows in $T(i, j) : i <= e$. The opposite is trivial. We can run the GCTA algorithm and produce the output in form of the cyclic string b. □

Corollary 6.

Let b be a cyclic bi-infinite input string and $T(i, j)$ be a proper GCTA table for b. The number of rows in $T(i, j)$ will be not greater than the length of the repetitive substring $s \subset b$, i.e. $i <= e, |s| = e$.

Proof. By definition of b as cyclic string, there exists a finite substring $s \subset b, |s| = e, e \in \mathbb{N}$ which is also a single repetitive pattern underlying s. Run GCTA scanning and observe the number of unique prefixes added to the left-hand of the $T(i, j)$. Notice that they start repeating themselves only after enumerating values in s at least once. □

Let us consider another case of scanning a bi-infinite string b, that does not have a repetitive pattern.

Let b be a left-infinite string, printed out by an endless program loop. Each bit is passed from the binary conversion of the counting program, that can enumerate natural numbers and print out their values in binary format. Printing is done by padding individual bits of converted values from left to the right output part of b. By construction, string b will never repeat itself

meaning that left-handle of the GCTA table will contain infinitely many rows. This also means that after scanning of b the right-hand of the GCTA table will have values updated only in the first column $j = 0$. The rest of $j > 0$ will contain empty unset placeholders.

Definition 10.

Let b be a bi-infinite binary input string and $T(i,j), \forall i \in I, I \subseteq \mathbb{N}, \forall j \in J, I \subset \mathbb{Z}$ be a proper GCTA table for b. If there exists no substring $s \subset b$ which can be printed out repeatedly to produce b', s.t. $b = b'$, then b is called *non-cyclic* infinite binary string.

Note, that if one replaces counting program to print out values of b in the above definition to enumerate $\mathbb{Z}$ instead of $\mathbb{N}$, one will print out non-cyclic bi-infinite string instead of left-infinite one.

Theorem 7.

Let b be a bi-infinite input string and $T(i,j)$ be a proper GCTA table for b. A bi-infinite string b is *non-cyclic* iff there are countably-many rows in $T(i,j) : i \in I, |I| = |\mathbb{N}|$ and the right hand of $T(i,j)$ has only one column of state values filled in after scanning b.

In fact, GCTA table can be always built by traversing the input string b. Let's state this as a theorem.

Theorem 8.

If b is a bi-infinite input string, then $\exists \gamma : b \to T(i,j)$, where $T(i,j)$ is a proper GCTA table and γ is a bijective mapping.

Proof. Assume the opposite. One possibility is that $\exists b' \neq b : \gamma(b) = \gamma(b') = T(i,j)$. Let $T(i,j)$ consists of all left-infinite prefixes of b. Then there must be a left-infinite prefix $cxt_p \in T(i, j < 0)$ which is also a prefix of b' and is at most a single bit different from some prefix in b. However, this is not possible if $T(i,j)$ is a proper GCTA table. Another possibility is that $\exists T'(i,j) = \gamma(b) : T'(i,j) \neq T(i,j)$, where $T(i,j)$ is a proper GCTA table for the bi-infinite input string b. Put cursor at some k index in the middle of b and cut the left-infinite part before that cursor k as a first prefix to be added into $T(0, j < 0)$ and into $T'(0, j < 0)$ at the same time. As we have seen left-hand sides of both tables will be equivalent. Then let us consider the right-hand parts and start updating the states in both tables. Observe that there is no difference between values $T(0, j >= 0)$ and $T'(0, j >= 0)$ if GCTA is applied over the same b and hence γ is bijective. □

Corollary 9.

If b is a bi-infinite input string and $T(i,j) = \gamma(b)$ is a proper GCTA table for b, then $\exists o$, which is a bi-infinite output string for $GCTA(b)$, such that there exists a reverse transformation $b = GCTA^{-1}(o)$ for every bi-infinite input and output combinations.

Proof. Follows from bijectivity of CTA and $\gamma : b \hookrightarrow\!\!\!\!\rightarrow T(i,j)$. Indeed, GCTA scheme is extended around CTA by defining similar direct and inverse transformation algorithms over bi-infinite input string facilitated by the proper GCTA table to produce a bi-infinite output string and to reverse the output back into the same input. Let's assume the later is false. That means that for any unique output $o = GCTA(b)$ there exists $\exists T'(i,j)$, s.t. if the infinite extension of the inverse CTA is applied to it as $GCTA^{-1}$, then $GCTA^{-1}(o) = b', b' \neq b$. However, since $T(i,j)$ is proper and $\gamma : b \hookrightarrow\!\!\!\!\rightarrow T(i,j)$ is a bijective mapping, this leads to a contradiction. Hence, $GCTA^{-1}(o) = b$ since $T'(i,j) = T(i,j)$ and both inputs $b' = b$ are equal. □

4.3 Beyond countable

The above observations introduce bi-infinite binary strings and tables for context-dependent transformations. They also provide insights into relations between these objects which are always restricted within countable cardinality of $|\mathbb{N}|$.

Each $T(i,j) = \gamma(b)$ has a generating property that by traversing the right-hand of $T(i, j < 0)$ the GCTA can produce a corresponding output. In order to produce a bijective output o a clear determinism needs to be followed. Every time the cursor k is pointing at the input string b is increased to $k+1$, then:

1. either *the switch of the context* takes place: an existing i row containing $cxt_i =$ "$..b(k-2), b(k-1), b(k)$" is selected or the new $i+1$ row is added to the table if such context has not been observed yet and is a different left-infinite string;

2. j is advanced to $j+1$ and updated.

An interesting aspect of that generating property is that sometimes one can produce a different output if the left-hand of $T(i,j)$ is substituted with the left-hand for a different input string $b' \neq b$. One can also introduce a modification into the generating algorithm to produce more than one output string if all of the right-hand $T(i, j > 0)$ is traversed. Later is possible if the left-hand remains algorithmically accessible in the same computation time.

Indeed, instead of continuously following the logic of switching between contexts when scanning an input string, one can choose to have the GCTA

table as an algorithmically constructible starting point[9] and simply traverse the right-hand to output multiple strings for each corresponding i row and cxt_i in that row, so that each row is potentially a left-infinite part of a different bi-infinite output string. In other words, to run GCTA in parallel one need to keep track of multiple j indexes in the above scheme. Each of them must be advanced and updated independently.

The above also means that, there exists an access algorithm defining a map between every countable element of some infinite sequence on the left-hand of the GCTA table and strictly many more elements obtained by traversing and generating new corresponding elements from the right-hand of the GCTA table. The same can be stated more formally by defining a scheme of accessing values using *general context generation algorithm* (GCGA). In order to do so, we would also require some more convenient tools to explain how the process of infinite binary strings enumeration can be organized.

[9] not necessarily proper

5 String Enumeration Formulas

5.1 Enumeration of infinite binary strings

Next, we will define *string enumeration formulas* (SEFs) by overloading the round brackets (parentheses) notation to indicate the occurrence of a cyclic string.

Let us revise the notations that we have already reserved to describe binary strings. We want to see in which cases the same can be expressed even shorter. To achieve this we would require such notation to be more compact and yet to denote the same value of the corresponding infinite binary string.

Let us start with the bi-infinite string which has a value of b, which can be computed as concatenation of repeating "01" substrings countably many times, as $b =$ "..010101.."[10]. We will denote such repetitive motif with round brackets. If no finite length index was specified, then such brackets indicate an infinite cycle, s.t. "..0101010101.." $\backsim$ "(01)". In other words "(01)" is a program that can be computed to print out the infinite value it is representing.

Definition 11.

Replicator or replication operator is a simple program denoted by round brackets that repeats the finite substring value wrapped in those brackets countably many times.

Definition 12.

String enumeration formula is a program implementing a second order replicator. Formulas can be nested in each other, so that one formula can contain either other formulas or concatenation of multiple replicators. Each replicator is always a balanced pair of brackets. Any closing bracket must be always preceded by the opening one, so that the whole expression can be parsed as a tree from the infix notation of the formula.

For example, a basic formula contains a single replicator and can be implemented by an infinite printing loop. We also say that two formulas "(01)" $\neq$ "(10)" represent different bi-infinite binary strings or are equal modular up to the shift from their initial index position at zero ("in the middle" of each string). Namely, $b = (01) : b(-1) = 1, b(0) = 0, b(1) = 1, ..., \forall i \in I, I \subset \mathbb{Z}$ and $b' = (10) : b'(-1) = 0, b'(0) = 1, b'(1) = 0, ..., \forall i \in I, I \subset \mathbb{Z}$.

[10] We reserve *two sequential dots* or ".." notation to denote omitted range in any assumed infinite binary string, but for other infinite sequences we keep *ellipses* with "..." to mean same

5.2 Categorization of SEFs

Now let's reiterate the idea of building an infinite binary string without any repetition. For that we will need another shorthand in our notations that we can use as a part of SEFs. We will use square brackets to note conversion of the natural number into binary format[11]. Here are a few examples of employing this notation:

- "([0])" $\backsim$ "(0)"
- "([1])" $\backsim$ "(1)"
- "([2])" $\backsim$ "(10)"
- "([3])" $\backsim$ "(11)"
- "([4])" $\backsim$ "(100)"
- ...

Intuitively, counting can be one of the obvious ways of generating examples of non-cyclic infinite binary strings, i.e. by enumerating natural number sequence in a single line. Following the above notation and adding a plus sign that would mean

"(0)[1+]" $\backsim$ "(0)[1][2][3][4][5][6][7][8].." $\backsim$
"..0000011011100101110111000.."

Although the square brackets notation does not necessarily enrich the expressiveness of SEFs, it is still a useful shortcut for the discussion purposes.

So far we have defined cyclic and non-cyclic infinite binary strings. These are a rather *trivial* but not the only category of infinite binary strings that one can encounter. If we attempt to break up these and other obvious kinds of strings generated by SEFs into categories either by looking at the characteristics of the GCTA tables produced after scanning those strings as input, we can get a better idea of the countability properties with regard to the width and the length of the resulting GCTA table (see fig. 2).

Following SEF notation one can define an infinite binary string by combining multiple formulas. A new category, presented in the above figure, is called *complex* and has not yet been discussed. Indeed, according to the proposed categorization a mixture of two trivial SEFs, the cyclic or non-cyclic strings, will produce a GCTA table inherently combining their characteristics. Let us provide an example for it.

Let b = "(01)(0)" be a bi-infinite binary input string. After scanning it with GCTA one obtains a proper countable table (as illustrated in table

[11] low-endian and without zero-padding

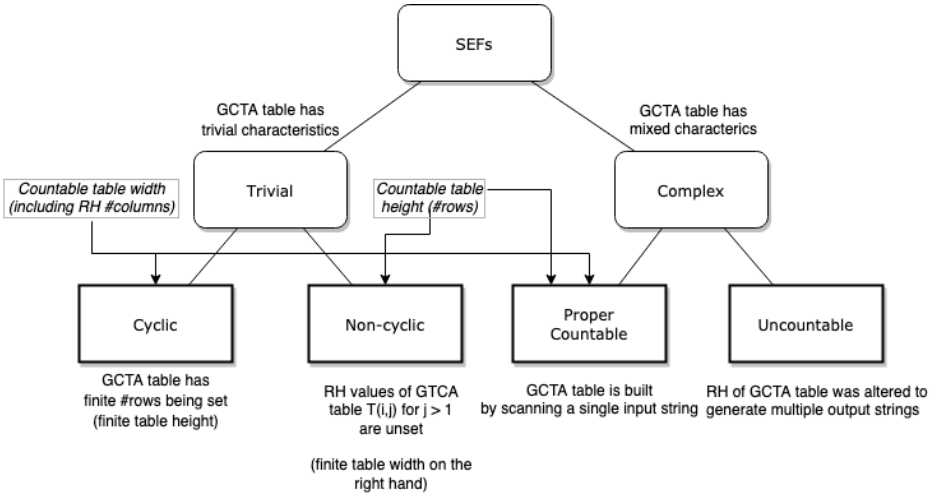


Figure 2: Categories of SEFs

Table 1: Example of a proper countable GCTA table for complex "(01)(0)"

#	LH	RH
1	"(01)"	"(0)"
2	"(01)0"	"1(0)1"
3	"(01)00"	"0"
4	"(01)000"	"0"
5	"(01)0000"	"0"
..	..	..

1). The mixed properties here are that the width and height of the table placeholders is both countable and set. In particular, first lines (*#1 and #2*) contain cyclic formulas which expand infinitely to the right[12]. Now next lines do not have values of the RH placeholders set starting with the column index $j > 0, j \in J, J \subset \mathbb{Z}$. Although $b =$ "(01)(0)" consists of two trivial cyclic strings, it is not trivial by itself. In the sense of proposed categorization, GCTA table characteristics observed are similar to both cyclic string "(01)" in the first lines and non-cyclic string "[1+]" for the rest.

5.3 Context argument

We start by scanning a bi-infinite non-trivial binary string $b =$ "(0)[1+]" and producing a proper GCTA table $T(i, j)$.

The left-hand of $T(i, j), \forall j < 0$ will contain all left-infinite strings observed during scanning of b by construction. Based on the characteristics of non-trivial infinite binary strings the right-hand of $T(i, j), i \in I, I \subset \mathbb{N}, j \in$

[12] validate RH value 1(0)1 at line *#2* for yourself

$J, J \subset \mathbb{Z}$ will exhibit both trivial non-cyclic and cyclic characteristics. This means we will have at least one cyclic-like bi-infinite entry "(0)1" obtained by scanning "(0)" resulting from zero-padding of b on the left (see line #1 in table 2)).

The other part of $b =$ "..[1+]" generates a unique countable sequence, which is continuously appended on the right. Scanning bits of this infinite sequence results in a list of countably many prefixes on the left-hand of $T(i,j), \forall j < 0$. The corresponding right-hand entries will be unset for all $\forall j > 0, i > 1$ in $T(i,j)$.

Table 2: Initial configuration with proper GCTA table for complex "(0)[1+]"

#	LH	RH
1	"(0)"	"(0)1"
2	"(0)1"	"0"
3	"(0)10"	"1"
4	"(0)101"	"1"
5	"(0)1011"	"0"
6	"(0)10110"	"1"
7	"(0)101101"	"1"
8	"(0)1011011"	"1"
9	"(0)10110111"	"0"
10	"(0)101101110"	"0"
11	"(0)1011011100"	"1"
..	..	..

Note that at this point string lists on both left-hand $LH = T(i,j) : j < 0$ and right-hand $RH = T(i,j) : j >= 0$ share the same cardinality due to line-by-line[13] correspondence. We will denote them as $|LH|$ and $|RH|$ respectively. Both are countable $|LH| = |RH| = |\mathbb{N}|$.

We will refer to the above setup as an *initial configuration* and use it as a starting point for some proofs below[14]. Will also refer to Cantor's slash argument[15] as well as employ some well accepted notations from the theory of smaller and Large Cardinals[16].

For convenience, we will refer to the output result of GCGA as a table $T'(i',j') = GCGA(b, T(i,j))$ containing those accessible strings or equivalently as to the left-hand $LH' = T'(i',j'), j' < 0$ and the right hand $RH' = T'(i',j'), j' \geq 0$. Like that we will treat $T'(i',j')$ as a cursor function and sometimes abuse or omit the index notation by meaning enumeration

[13] one-to-one and onto

[14] many similar configurations are equally acceptable for our purposes

[15] for example, as offered in comprehensive introductory overviews into the concept such as [1]

[16] namely, $\aleph_0 = |\mathbb{N}|$ - also see next section for more detailed recap of existing results and notations

in a wider sense as $\forall i' \in I', I' \supseteq \mathbb{N}$ and $\forall j' \in J', J' \supseteq \mathbb{N}$ when discussing $T'(i', j')$.

Now let us describe how GCGA can act on the initial configuration by assigning values to the previously unset placeholders on the right-hand $RH = T(i, j) : j >= 0$ in order to produce unobserved state and generate the multitude of output strings (*GCGA steps*):

1. create a reference of $T(i, j)$ from the initial configuration as $T'(i', j')$ which is both relative to original $T(i, j)$[17] and otherwise contains the same values except that it can also be modified or expanded with new placeholders at need so that all indexes are respectively relabeled;

2. start moving the cursor in the top-down direction from the point closer the top-middle of the $T'(i', j')$ by examining values at $i' = 0, j' = 1$ on the right hand of the proper table;

3. continue to increment access index i' until one can find the placeholder for the state value that has not yet been set at some $i' \geq 0, j' \geq 1$[18];

4. if the values for $j' > 0$ are unset, this means that the prefix on the same counter-part line on the left has been observed only once. Then one can label this location as *suitable for modification*;

5. next, GCGA can be run similar to GCTA steps to output the result with some difference in conditional logic as explained below;

6. continue with generation on every line by switching between context prefixes

7. if the the line at index i' can be marked as *suitable for modification*, then:

   (a) mark corresponding prefix in the LH of $T'(i', j'), j' < 0$ as copied, so that only the finite reference is stored[19]

   (b) keep expanding $T'(i', j')$ with copies of the marked string including the unset placeholder on the right, preparing them for the modification at the next steps

   (c) continue to run the necessary number of iterations for that i' so that one can cover the entire space of yet unset placeholders on the right-hand of $T'(i', j'), j' > 0$ by some countable index mapping $k \to (i', j'), k \in K, K \subset \mathbb{N}$

[17] acts as copy of $T(i, j)$

[18] optionally and in case of other input strings, one can also increment j up to some finite size of the SEF formula in the state (if desired) and look further on the right for the unset placeholders

[19] we interpret output with such finite separator reference as "multi-lined"

(d) once you run as many finite iterations of SEFs enumeration for each index pair (i, j) as required by the above one-to-one and onto mapping, append newly obtained binary values to the unset placeholders on the right of $T'(i', j'), j' > 0$

(e) optionally, generate an output string each time the new placeholder is appended

(f) update (i', j') for the next iteration of the cover on the same line either by remembering values or computing them from the cover mapping

(g) continue to loop over the covering map $k \to (i', j')$ to repeat the above sub-process from step 3 as many times as necessary[20] or until you have reached the output values desired by the access or a cursor function

Keep in mind, that every appending (generation) happening in the steps 7(c) and 7(d) of *GCGA steps* over previously unset values is interpreted as access (generation) of a new output infinite binary string which continues to expand with the new unobserved part of the prefix [21]. Unless designed, the described algorithm will never terminate for the specific input string $b =$ "(0)[1+]". In fact, this is not necessary for our purposes as it is enough to show that:

- the above description of GCGA algorithm generates new unique strings from the point of view of the particular "fork" or branch starting at some (i', j')
- as well as provides the ability to access the necessary values at the exact "fork" or branch observed after the given context in LH

It is easy to see that GCGA can be further modified to print out only desired or finite parts of the output. Recall, that we refer to such output strings as algorithmically reachable by a cursor.

We can observe that after applying GCGA to the $T(i, j)$, the table will not be proper anymore in a sense that the left-hand LH' contains duplicates, which are still unique strings only if considered jointly with their right-hand counter-parts in RH' list.

It is important to stress the following fact again - the property that makes every new string listed in the table $T'(i', j')$ unique is the peculiar design of GCGA, which goes beyond counting[22] and is simply a special way of accessing every traversable object that can be output by the algorithm with or without given context in LH.

[20] up to countably many times

[21] due to construction of the algorithm

[22] in a sense of recursive construction of natural numbers $\mathbb{N} = \{x_{i+1} = x_i + 1\}, \forall i \in I$

We have reached the point when we know how $T'(i', j')$ can work by applying GCGA to $T(i, j)$ values. It is still not clear if left and right "table halves" when accessed via the respective cursor functions $T(i, j)$ and $T'(i', j')$ can result in producing the output of the same left and right cardinality $|LH| \stackrel{?}{=} |RH'|$ or alternatively can access parts of one or more unique bi-infinite binary strings which we can show to be unequal.

Theorem 10.

Let $T'(i', j') = GCGA(b, T(i, j))$, where both $T'(i', j')$ and $T(i, j)$ are cursor functions as well as $T(i, j)$ is a proper table over input bi-infinite binary string b and b is proper countable. Cardinality of $|LH| < |RH'|$ when accessed via $T(i, j)$ and $T'(i', j')$ from input b respectively.

Proof. The above statement $|LH| < |RH'|$ can have a meaningful answer only when we recall that $T'(i', j')$ tries not just to access but also to generate new elements in the sense of the algorithm application explained above in *GCGA steps*. Let $b =$ "(0)[1+]" or any other proper countable input string. Proceed with initial configuration and apply GCGA steps to $T(i, j)$ proper table constructed with GCTA. We can produce any arbitrary or specifically requested output range through means of the cursor $T'(i', j')$.

Now let us show that the resulting output of applying GCGA will fall into the category of the proper uncountable strings. We can point out the exact difference in the end result of the traversing process up to specific index. In order to do so, choose any nearest (i', j') index pair suitable for modification (*GCGA step 7.c*). Observe that $j' > 0$ meaning the cursor is pointing at the yet unset placeholder in both $T(i', j')$ and $T'(i', j')$. Select the row i' from $T(i, j)$ table without any unset values as a left-infinite string $a := (T(i', j), \forall j < j')$. Iterate further through GCGA steps to pick next suitable value $v \in \{0, 1\}$ from the generation scheme (or pattern, such as "[1+]"). Assign that value at the cursor $T'(i', j') := v$. Select the row from $T'(i', j'), \forall j'$ without any unset values as a left-infinite string a'. Observe that $a' \neq a$ and is different up to a single placeholder that is still unset in $T(i', j')$. Then notice that string a is also a prefix of a' as in $a =$ "$.., a'[j' - 2], a'[j' - 1]$". If $a' \notin T$ table, then $|LH| < |RH'|$. Indeed, assume the opposite that $a' \in T$. By design of GCTA the $T(i, j)$ table is constructed as proper, meaning that $T(i', j')$ can have multiple states assigned for the prefix of a in RH iff the prefix of the left-infinite string a is observed in b multiple times. The latter is not the case. Neither prefix of a nor a is observed in b more than once, hence $a' \notin T$.

This new string a' has been just appended to T' table by GCGA, so that $i' := i + 1$. We can continue to generate similar new and unique strings and access them by means of the $T'(i', j')$ cursor an exact and finite number of

times[23], until we switch to the next index suitable for modification. However, that means that RH' is strictly uncountable and $|LH| < |RH'|$. □

One should avoid confusions in interpreting the results stated in the *Theorem 10*. We have shown that one can use GCGA [24] to achieve algorithmic accessibility when it comes to handling larger objects arranged from infinite binary strings. Such schemes can be used to construct and access objects like RH' demonstrating to have the property of strictly greater cardinality over $|\mathbb{N}|$. However, if we run GCGA implementation on a deterministic TM any produced output will be naturally of the same countable cardinality, just as the one of GCTA [25]. We are again in need of tools that can help us operate with such objects of greater uncountable cardinality more intuitively.

5.4 Uncountable cardinalities

In the previous section (see - *Theorem 10*) we have indicated yet another way[26] for how to count at infinity or expand upon large objects such as lists of infinite binary strings. Next, we will reference and introduce some more of notation to be able to discuss previously known theory and its results. Our intention is to better understand potential consequences of such presumably novel "counting" technique.

We begin by following into the original line of thinking as offered by George Cantor. Cantor introduces a theory for transfinite numerals called ordinals and defines their cardinalities. He also famously introduces the Cantor's diagonal or slashing argument[27]. The idea behind slashing is to compare cardinalities of rather large objects such as infinite lists of items by employing the bijective mapping[28].

Bijective mapping of objects is such a central tool to the discussion of cardinality (in the proposed set theory by Cantor and beyond) that we want to state it even more formally as a principle.

Principle 1 - Principle of one-to-one correspondence.

Objects have the same cardinality if and only if their elements are in one-to-one correspondence.

[23] by design and implementation of GCGA

[24] or similar schemes

[25] with the only difference in the printed-out values

[26] next to natural numbers as well as transfinite cardinal and ordinal numerals - read further for more detailed explanation

[27] Further we will refer to Cantor's diagonal argument simply as *slashing*

[28] An instructive overview of these and other ideas by Cantor[2] for matching cardinalities (starting with $|\mathbb{Q}| = |\mathbb{N}|$) has been cleverly outlined in "Hilbert's Hotel" by Gamow[3], p.17.

One-to-one correspondence between objects is usually proved by showing the existence of the respective bijective mappings or, equivalently, functions which are both one-to-one and onto[29].

Again, this principle turned out to be very insightful and helped to make sense out of many less-intuitive situations when objects were looked upon much more restrictively - for example, only from the perspective of trivial counting. Let us recap on that.

Recall that counting procedure provides us with the set of natural numbers $\mathbb{N} = \{1, 2, 3..\}$. Any taken subset of natural numbers $X \subset \mathbb{N}$ will be always limited by some largest number at the very end of such subsequence X, meaning that the cardinality of such subset is finite $|X| < |\mathbb{N}|$. However since the sequence of the natural numbers $\mathbb{N}$ by itself is always constantly appended with a new number which will be an increment of the previously seen largest number, we say that the cardinality of natural numbers is actually infinite and corresponds to the countable form of infinity. This can be captured by assuming the existence of some transfinite limit ω_0[30]. This limit ω_0 is strictly greater than any natural number[31] and will be a first transfinite number such that if $X' \subseteq \mathbb{N} \cup \{\omega_0\}$, then we say such subset has a first infinite or countable cardinality similar to that of natural numbers. Cardinality of sets that include ordinal limits is notated with aleph letters. For example, for the inclusion of the first ordinal limit ω_0 we use $\aleph_0$ and say $\aleph_0 = |X'| = |\mathbb{N}|$. The idea of transfinite numbers gets further developed by introducing the theory of *ordinal numbers*[2].

A set of natural numbers consist of elements which can be counted and hence *well-ordered*. Counting provides order-preserving bijection for well-ordered sets by labeling[32]. Each cardinal number can be related to some limit ordinal of the underlying set. For example, for natural numbers[33] that would be a first order type ω_0[34]. Thus we say that any of the ordinals starting with ω_0 in the well-ordered sequence $\omega_0,\ \omega_0 + 1,\ \omega_0 + 2,\ ..$ will be strictly greater than any natural number. All ω_0 ordinals such as $\{\omega_0,\ \omega_0 + 1,\ \omega_0 + 2,\ ..\}$ correspond to the first transfinite cardinal $\aleph_0$. Similar association is true for the next biggest transfinite cardinal $\aleph_1$, to which all ω_1 ordinals such as $\{\omega_1,\ \omega_1 + 1,\ \omega_1 + 2,\ ..\}$ can be mapped to as many to one[35]. We say that $\aleph_1$ is a first transfinite cardinal for the infinite set with

[29] Also see the set theoretical definitions for bijective functions as discussed, for example, in[4–6]

[30] ∞ sign was suggested to be replaced by Cantor with ω which is a less confusing notation[2]

[31] which is also a definition of any transfinite numeral - see [6] for much better introduction into ordinals

[32] and simply agreeing that the order of some elements is greater than the others

[33] by re-ordering a set we can assign it a new order type

[34] mind the index at omega for convenient mapping with alephs further on

[35] in most frameworks like Zermelo–Fraenkel (ZF) set theory this usually requires an *Axiom of replacement*

uncountable cardinality.

Although ω_0 and $\omega_0 + 1$ have different respective order types, they do not add more to the original cardinality $\aleph_0$ of the corresponding set $\mathbb{N}$[36]. To reiterate over the idea of cardinalities of ordinal transfinite numerals again - indeed, if ordered set $\mathbb{N} \cup \{\omega_0\} = \{0, 1, 2, 3, .., \omega_0\}$, then the cardinality $|\mathbb{N} \cup \{\omega_0\}|$ is still the same as $|\mathbb{N}| = \aleph_0$.

Recall that ordinals are defined as order preserving sets, but transitive addition in ordinal arithmetic is not commutative[37]. One can still have the well-ordering in form of right-side addition as in $\omega_0 < \omega_0 + 1$, even though it is not the same for left-side addition when $1 + \omega_0 = \omega_0$. This means that applying ordinal arithmetic will not result in order preserving bijections between $\mathbb{N} \cup \{\omega_0\}$ and $\{\omega_0\} \cup \mathbb{N}$. Namely, the later will have order type $1 + \omega_0 = \omega_0$, based on the bijection $\{\omega_0 \to 0, i \to i + 1 : \forall i \in \mathbb{N}\}$, and hence $1 + \omega_0 < \omega_0 + 1$.

Other interesting insights that we want to recap are based on the concept of powerset. This concept is a helpful example of yet another way to count things at infinity (next to transfinite ordinals). The concept itself is as following, if X is a set of items, then a set $\mathcal{P}(X)$ consists of all subsets of set X and is called a *powerset* of X.

Numbers can be constructed from sets. For example, applying only empty set one can show how to construct natural numbers [1]. The process of recursive application of *powerset* to sets with cardinalities of $\aleph_0$ and larger is called *transfinite exponentiation*. It can be also used to construct *ordinals* from sets[38]. Such sets are referred to as *omega sets*[39] of well-ordered numerals and can be used to show how greater orders of infinity (beyond countable) can be set up[40].

However, one has to be careful when working with such concepts as *set of sets*. It has been proven to be unexpectedly complex and may easily lead to paradoxes[41] if no special considerations are put in place[42]. Whenever possible we will replace the notion of *set of sets* either by considering *lists of items*, where items are more concrete objects than sets such as binary strings or by simply looking at classes of numerals[43].

[36] keep in mind that re-ordering the set does not change its cardinality, even in case of transfinite sets

[37] contrary to the definitions of ordinal arithmetic and in case if natural ordinal arithmetic is applied, one can achieve $1 + \omega_0 = \omega_0 + 1$

[38] for other than Cantor ordinals, please check von Neumann ordinals with an alternative proper-class construct: "each ordinal is the well-ordered set of all smaller ordinals"

[39] mind the ω letter we have used earlier

[40] ultimately by stating the existence of transfinite numerals as we have done earlier, see - *"The Transfinite Ordinals and Cantor's Mature Theory. In: Labyrinth of Thought"*[7], pp. 257-296 as well as [8]

[41] For example - *Russell's paradox*

[42] For instance - the requirement to operate within certain logical framework which relies on axiomatic systems such as ZF set theory

[43] Meaning that we try to implicitly employ the proper-class construct as proposed by

The powerset concept is used in another important result called Cantor's Theorem [44]:

Theorem 11.

If $|X| = \alpha$, then $|\mathcal{P}(X)| = 2^\alpha$ and $\alpha < 2^\alpha$, where 2^α is the total number of subsets of any set such as X.

As a consequence one can state[45] the existence of strictly greater cardinality $\aleph_0 < 2^{\aleph_0}$. An assumption that $\aleph_1 \stackrel{?}{=} 2^{\aleph_0}$ is equivalent to the *continuum hypothesis*[46], which needs more explanation.

We define the cardinality of real numbers as $\mathfrak{c} = |\mathbb{R}|$. Cantor gave two proofs that the cardinality of the set of integers is strictly smaller than that of the set of real numbers $\aleph_0 < \mathfrak{c}$[47]. However, those proofs provided no indication to which extent or how far does the difference of cardinality in $\aleph_0 < \mathfrak{c}$ goes, so Cantor has proposed the continuum hypothesis (CH)[48] as a possible solution to this question. The CH states that there exists no set S for which $\aleph_0 < |S| < \mathfrak{c}$[49].

5.5 Formal SEF languages

So far we have provided a number of definitions to describe infinite binary strings. Good news is that our discussion and results are not based on dealing with highly abstract sets[50] yet, but rather with concrete objects like lists of binary strings[51]. This means we did not rely much on any foundational framework[52], so the results in *Theorem 10* are mostly independent. Unfortunately at this point such results are still not formal enough to explain what exactly do we mean by *uncountability* or how to interpret our claims about cardinality of inspected objects. Instead of moving forward with developing a complete theory of infinite binary strings from scratch[53] by doing further build-up on presented GCGA concept, we intend to harden and refine our definitions referencing something well-known. We will do so by obtaining

---

ZF everywhere in our discussion

[44] see [6] and for conceptual proof outline refer to [1]

[45] Alternatively, one will rely on the proof using Cantor's diagonal argument[1] applied to the list of all binary strings

[46] Assuming the axiom of choice, there is the smallest cardinal number $\aleph_1$ greater than $\aleph_0$, and the *continuum hypothesis* is in turn equivalent to the equality $\aleph_1 = 2^{\aleph_0}$ [4].

[47] Cantor's first uncountability proof and Cantor's slashing

[48] see [6, 9] as well as [10] for more informal but comprehensive covering

[49] which is a weaker version of the CH formulation

[50] and equally large, such as *von Neumann universe* in ZFC

[51] residing and computable within the realm of theoretical computer science

[52] such as ZFC

[53] in terms of mathematical logic

an exact explanation based on already existing results in classic set theory[54] and theoretical computer science[11].

With such motivation in mind, this subsection will focus on refining definitions of SEFs as formal languages[55]. Next subsection will describe an algorithm that we will refer to as *continuum access scheme* (CAS) and benefit from the definitions.

Recall that in computer science[11] a formal language is defined over some alphabet. An alphabet is usually a finite set or list of symbols or sometimes strings. For example in both cases, $\Sigma = \{0, 1\}$ or $\Sigma = \{ab, ba, c\}$ are alphabets.

Definition 13.

If Σ is a finite set of strings called alphabet, then Σ^* is an infinite list of all possible concatenations over those strings, s.t. $|\Sigma^*| = \aleph_0$.

The star symbol notation used in Σ^* is called Kleene-closure of Σ. Alternatively, Σ^* is said to be closed under string concatenations including the empty string ϵ[56].

A formal language consists of *words* whose letters are taken from an alphabet and are *well-formed* according to some specific set of rules.

Definition 14.

If Σ is a finite alphabet, then a *formal language* L over an alphabet Σ is defined as a set $L := \{w \; is \; valid : \forall w \in \Sigma^*\}$, where each word w is a *well-formed* or *valid* expression of L.

Words of formal languages can be also referred to as expressions or formulas. Now we want to take a closer look at what does the well-formed expression means or the notion of formula validity for SEFs by summarizing what has been discussed about SEFs so far as a set of rules.

Definition 15.

A formula ϕ is a valid String Enumeration Formula iff all of the following is true:

1. ϕ is an input-free program generating an infinite binary string as an output in countable time[57]

[54] which we have most briefly recapped and referenced in the previous section - also see [4]

[55] following Chomsky hierarchy

[56] a zero length string, similar to the empty set element

[57] meaning that number of output characters or the cardinality of the output is never greater than $\aleph_0$ - also see *Definition 11*

2. formula has always a finite length $|\phi| = m : m \in \mathbb{N}$ unless specified otherwise[58]

3. replicator syntax is a balanced pair of round brackets such that number of balanced bracket pairs is finite $|\phi|_(= |\phi|_) = n : n \geq 0, n \in \mathbb{N} \cup \{0\}$

4. round brackets can be nested which is interpreted as recursive replicator calls

5. a replicator can not be defined over an empty string

Given the notion of SEF validity we can define a *didactic* procedure as an algorithm accepting only valid input expressions. This also allows stating the definition of a formal language for SEFs.

Definition 16.

A Formal String Enumeration Formula language or *SEF language* is a formal language constructed from syntactically valid formulas over alphabet Σ as $L_{SEF} := \Delta(\Sigma^*)$, where Δ[59] is a validation procedure or *didactic* that can recognize only well-formed expressions of L_{SEF}. $\Sigma := V \cup \gamma$ is a union between set V of vocabulary symbols and set γ of syntactical symbols (syntax).

One can state an equivalent definition.

Definition 17.

L_{SEF} is a *SEF language* constructable from a tuple (Σ, Δ) iff $\Delta \subset \Sigma^*$ s.t. all formulas in Δ must be both syntactically valid and computable as SEFs.

Note that for most purposes formal languages are defined over finite alphabet. The same is with SEF languages unless specifically considered.

Definition 18.

If $L_{SEF} := (\Sigma, \Delta)$ is a SEF language and $\exists k \in \mathbb{N} : |\Sigma| \leq k$, then we say that L_{SEF} has an alphabet with finite cardinality k.

This is different from the cardinality of the language which is a set of all valid formulas.

[58] one can explicitly specify countable length for $\phi, |\phi| = \omega_0 : \omega_0 > n, \forall n \in \mathbb{N}$

[59] do not confuse with Δ_0 formulas[6] of first order logic

Definition 19.

If $L_{SEF} := (\Sigma, \Delta)$ is a SEF language then the cardinality of the language is the same as $|\Delta|$.

Finally, we say that the language is finite if it can recognize strings of finite length.

Definition 20.

If $L_{SEF} := (\Sigma, \Delta)$ is a SEF language and each formula of the language is finite $\forall \phi \in \Delta : |\phi| \leq n, n \in \mathbb{N}$ then we say that L_{SEF} is a finite language.

Theorem 12.

If $L_{SEF} := (\Sigma, \Delta)$ is a finite SEF language then L_{SEF} is also a regular language.

Proof. Recall the result from [11] that every formal language is a regular language iff it can be recognized using a regular expression. Observe that regular expressions themselves form a regular language or can be parsed by a regular expression. Since every SEF expression can be reduced to a regular expression by mapping replicator syntax to a Kleene-closure, it means that there exists a regular expression to parse SEFs. □

Obviously, even if both the cardinality of the alphabet[60] and the cardinality of the SEF language itself[61] is infinite, it can be still restricted by some greater cardinality unless shown or defined otherwise.

Please note that so far we have been working only with finite SEF languages and will continue to do so. For example, $L_{SEF} := (\Sigma, \Delta) = (\{0, 1\} \cup \{(,), :\}, \Delta)$ is a finite SEF language equipped with round brackets as replicator symbols and colon for access symbol and $\forall \phi \in \Delta : |\phi| \leq n, n \in \mathbb{N}$. If needs be, the extended version of the syntaxes may include square brackets for binary conversion of in-line decimals as well as plus and star generating short-hands for enumeration scanning.

Next we would need one more definition to count how many times one sub-string can occur in another string.

Definition 21.

Given L_{SEF} is a SEF language and $\exists z, s : z, s \in \Delta$ are valid strings in that language, such that one string can occur in another one at least once, i.e. $z \in s$. This fact is noted as $|s|_z$ and is called cardinality of z in s.

[60] i.e. $|V \cup \gamma| \geq \aleph_0$

[61] language contains formulas of countable length

In case when we can count how many times one sub-string can occur in another string, then this is noted as $|s|_z = \lambda : \lambda \in \mathbb{N}$.

5.6 Continuum access scheme

We base on the setting similar to the GCGA initial configuration. We start by scanning a string $b =$ "(0)[1+]" with a small difference in approach but without loss of generality. Instead of scanning bit by bit, we jump over the full binary representation from one integer "(0)[1][2]" to the next one "(0)[1][2][3]". Such scanning is marked as $b =$ "(0)[1∗]". Like that all bits of each integer are concatenated at full for every string we add to LH of the table (see fig. 3).

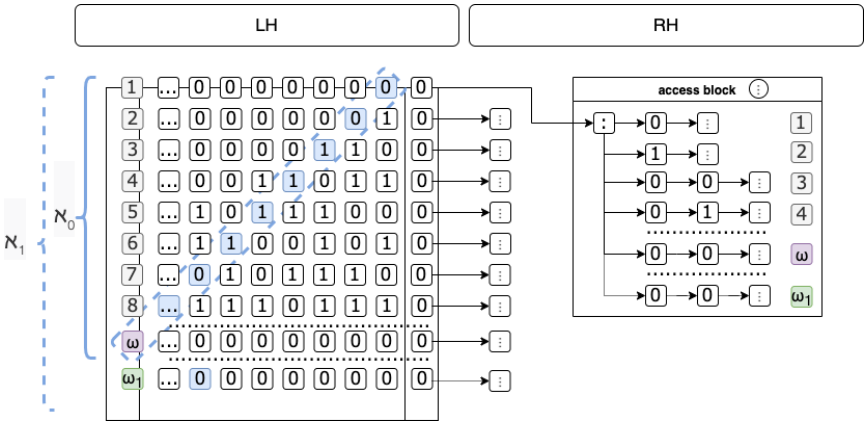


Figure 3: Continuum access scheme

We also extend the SEF formula language of infinite binary string enumeration using a colon symbol. On the RH of the table we have something called *access block* labeled with three vertical dots. The expanded version of this block contains the access step noted with colon symbol. Keep in mind that the access block is recursive. For example, a string formula that indicates some instance of algorithmic access from the LH context with the span into the RH is designated as "(0)[1]:01:010(:0)", where the string replicator on the very right end "..(:0)" represents a continuous access action into the RH. When one needs to compute such an infinite binary string formula, one can imagine a computation process by expanding each replicator without stop. The replicator with the access colon symbol means continuous recursive access with ever expanding context on the left hand of each colon. This schematic representation describes nothing else but the recursive application of the GCGA and can be seen as yet another generalization step through the employment of the recursive definition.

On one hand, we don't necessarily need to restrict number of times when the colon symbol appears in such extended SEF notation. Indeed,

such cases can be called an infinite algorithmic access scheme or simply *continuum access scheme*. On the other hand, if we would prefer to restrict number of occurrences for the colon symbol by some finite limit, that should be possible as well.

Now let us consider another simplification with the finer subcase of applying continuum access scheme (see fig. 3) but only a limited number of times.

Definition 22.

If $L_{SEF} := (V \cup \gamma, \Delta_\delta)$ is a finite SEF language, which syntax is equipped with a colon as an access symbol and any formula in its didactic set Δ_δ contains colon symbol but only limited number of times, then we say that language L_{SEF} has some finite depth of δ. Specifically, $\exists \delta \in \mathbb{N}, \forall s \in \Delta : |s|_: \leq \delta$.

Access schema for languages with finite depth can be imagined as similar to fig.3 but without having an infinite recursion call inside the access block[62]. Number of calls becomes limited by counting. We are particularly interested in this case as we can show not only that there is no bijection between LH and RH parts of the corresponding computationally expanded GCGA table[63], but specifically to extend and pin-point the relationship between LH and RH parts.

Let $L_{SEF} := (V \cup \gamma, \Delta_1)$ be a finite SEF language with a binary vocabulary $V := \{0, 1\}$, typical syntax $\gamma := \{(,), :\}$ and didactic with a single access depth, s.t. $\forall s \in \Delta : |s|_: = 1$. Consider a proper table with cursor $T(i, j)$ obtained by scanning a proper countable bi-infinite input binary string b and a cursor function $T'(i', j')$ such that $T'(i', j') = GCGA(b, T(i, j))$ which will allow algorithmic access as exposed in *Theorem 10*. For instance, if $\exists i', j' : i' \in I'$ and $j' \in J'$ are some indexes and there is a formula $f :=$ "(0)1:01" corresponding to $i' \in I'$, then the sub-string before the colon (acting as an access symbol) will correspond to the context of the expanded string accessed via cursor function $T'(i', j') : \forall j' < 0, j' \in J'$ and will be equal to "(0)1". While the part after the colon corresponds to the value-state of the expanded string from $T'(i', j') : \forall j' > 0, j' \in J'$ and will be equal to "01".

The above approach is similar for finite languages of greater depth. For example, in case of $L_{SEF} := (V \cup \gamma, \Delta_2)$ if there is a formula $f :=$ "(0)1:01:1", then the whole sub-string before the second occurrence of the colon symbol will correspond to the context when the second access is evaluated. However, for greater clarity we replace multiple-prime notation with depth index. This applies to cursor functions $T_\delta(i, j)$, so that

[62] a block with the three vertical dots symbol

[63] by applying result from *Theorem 10*

$$
\begin{aligned}
if\ \delta = 1{:}\quad & T_1(i,j) := T'(i',j') \\
if\ \delta = 2{:}\quad & T_2(i,j) := T''(i'',j'') \\
& ...
\end{aligned}
$$

and so on. Also further we assume that depending on the context the cursor indexes i, j are remapped from their index supersets accordingly. For example, $j' \in J'$ is always re-mapped from $j'' \in J''$[64] via $\alpha : J'' \to J'$ based on depth and occurrence of the access symbol and given that $J' \subseteq J''$. Such mapping is equivalently rewritten as $\alpha : J_2 \to J_1$, where $j_1 \in J_1, j_2 \in J_2, J_1 \subseteq J_2$. The same is true for LH' and RH'' being rewritten as LH_1 and RH_2.

Theorem 13.

Let $L_{SEF} := (\{0,1\} \cup \{(,),:\}, \Delta_\delta)$ be a finite SEF language with a binary vocabulary, typical syntax and didactic with a finite depth $\delta \in \mathbb{N}$. If a cursor function $T_\delta(i,j)$ implements continuum access schema for L_{SEF} limited by some finite $\delta \in \mathbb{N}$ for every formula in didactic Δ_δ, then for every algorithmically accessed value-state sub-string $\forall \nu_\delta \in RH_{\delta+1} : \nu_\delta := T_{\delta+1}(i,j) : \forall j > 0, j \in J_{\delta+1}, i \in I_{\delta+1}$ given context $\phi_\delta \in LH_\delta : \phi_\delta := T_\delta(i,j) : \forall j < 0, j \in J_\delta, i \in I_\delta$ there exists surjection

$$
\begin{aligned}
\rho_\delta{:}\ RH_{\delta+1} &\to LH_\delta \\
\nu_\delta &\mapsto \phi_\delta
\end{aligned}
$$

such that at least for $\delta = 0 : |LH_\delta| < |RH_{\delta+1}|$ and $|RH_{\delta+1}| \leq |\mathcal{P}(LH_\delta)|$, where $\mathcal{P}(X)$ is a powerset of X.

Proof. The theorem is proven by following the construction of the continuum access schema.

A finite case for the existence of the surjection $\rho{:}\ RH_1 \to LH$ is trivial. It is enough to show that $\rho : \nu \mapsto \phi$, where $\forall n \in \mathbb{N}, m \in \{1, .., 2^n\} : \nu =$ "$[m]$", $\phi =$ "$[n]$", which is the case for $T_1(i,j)$ table with any finite number of $i \leq 2^n$ lines. Given that the $|\mathcal{P}(X)| = 2^{|X|}$ if X is a finite set, then the existence of the surjection $\rho{:}\ RH_1 \to LH$ is satisfied for LH of $T(i,j)$ and RH_1 of $T_1(i,j)$.

For the infinite case we need to consider $|LH| \geq \aleph_0$. This can be achieved by slashing LH as shown on fig.3, then adding an inverted diagonal as new string labeled ω to $LH := LH \cup \{\omega\}$ and extending RH with the link

[64] projected or pulled from the superset - for example, this can be implemented via Levy collapse of cardinals[6]; we also discuss implementation of CAS in later subsections

to access block accordingly. So far we have not commented on how the RH part of $T(i,j)$ is expanded. Continuum access is implemented based on Theorem 10 result by recursively applying GCGA as in $T_{\delta+1}(i,j) := GCGA(b, T_\delta(i,j))$, where input b can be ignored after initial application. The role of $RH_{\delta+1}$ extension is performed by the access block, so that each entry in the access block is interpreted as value-state sub-string before the next iteration down the recursive call. Like that value-state of the previous iteration of the access call becomes appended (concatenated) to the new context of the next access call.

Note that given L_{SEF} and for all countably and recursively extendable cases of $LH_\delta : \delta \in \mathbb{N}$ the access block always represents an infinite access scheme to all possible infinite binary strings, the cardinality of which by Cantor's theorem equals to $2^{\aleph_0}$ due to existing bijection between the set of all infinite binary strings and continuum of real numbers $\mathbb{R}$. The same theorem states $2^{\aleph_0} = |\mathcal{P}(\aleph_0)|$[6].

Both for finite and infinitely recursive access blocks we still rely on the algorithmic access as a possible selective reference to higher value-states of $\delta \geq 1$. Specifically, one can think of $RH_{\delta+1}$ values getting pulled down into LH_δ at need [65].

Now in our case, L_{SEF} language is finite and $\delta \leq 1$. Since the access block recursion is called only finite number of times we can only state that the right-hand is not greater than cardinality of the continuum or $|RH_{\delta+1}| \leq |\mathcal{P}(\aleph_0)|$

Given that $|LH| = \aleph_0$ and using generalization of Theorem 10 by induction, one has $|LH_\delta| < |RH_{\delta+1}|$. This means there exists a surjection $\rho_\delta \colon RH_{\delta+1} \to LH_\delta$, specifically such that $|LH_\delta| < |RH_{\delta+1}|$ and $|RH_{\delta+1}| \leq |\mathcal{P}(LH_\delta)|$ at least for $\delta = 0$. □

A valid concern would be to ask if such continuum access schema is computable on a general computer or an equivalent TM. The answer is *no* and it follows from the following explanation. Indeed, from computer theoretical perspective this still means that not every infinite binary string can be accessed[66] with existing computers. Another way to say the same is that *every algebraic number is computable* or *every transcendental number is incomputable*[67]. Even if one would take a step back to assume the existence of some special-purpose formalism[68] that would permit efficient implementation of the algorithm[69] for the described continuum access schema for all left-infinite countably extendable expansions of contexts in $LH_\delta : \delta \in \mathbb{N}$

[65] only when provided with the specific context

[66] see *halting problem* for details

[67] every Chaitin's constant is the example of incomputable number - also see subsection *Computability of SEF languages*

[68] uncommon enough, beyond general computer programs

[69] or rather a schema

then most likely such algorithm is neither in RE nor in co-RE complexity class[70].

One can notice that once we have applied slashing to LH (see fig.3) as according to the explanation in the above proof, we can repeat the process to obtain ω_1 string and produce an uncountable extension of $|LH| = \aleph_1$. Technically there is nothing stopping us from doing so at this point. Another peculiarity is that the proof of Theorem 12 was obtained only for the case of at least $\delta = 0$, without full generalization by induction. The reason behind this is a valid concern that we don't necessarily know yet how big is exactly the cardinality of the continuum $\mathfrak{c} = 2^{\aleph_0}$. In fact, besides highlighting such concern, the Theorem 12 does not provide any additional insights except already known results due to Cantor's work[2, 4, 8]. However, we did manage to provide an illustration of relation between cardinalities of $\aleph_0$, $\aleph_1$ and $2^{\aleph_0}$ in form of continuum access schema using infinite binary strings as well as the connection to the set theoretical axioms[71].

5.7 Uncountability of SEF languages

On one hand we have looked at the definitions of formal SEF languages which have a finite alphabet and consist of countably many formulas of finite length. This aligns nicely with the well-known formalism in computer science theory such as regular languages. On the other hand, the motivation behind SEFs was to have an expression which produces an infinite binary string. But if we want to expand on our understanding of uncountability, we would need to look into extending the definition of formal SEF languages to be able to handle expressions of infinite length.

Let us revise some properties of the formulas in a finite SEF language $L_{SEF} := (\Sigma, \Delta)$. Each valid $\phi \in \Delta$ can be computed into some infinite binary string $b \in \mathbb{B}, |b| = \aleph_0$ to produce an output. In fact, the opposite is also possible if one needs to take some infinite binary string $b \in \mathbb{B}, |b| = \aleph_0$ as an input and recognize it by an ω-regular expression[12].

Lemma 14.

If $L_{SEF} := (\Sigma, \Delta)$ is a finite SEF language of countably infinite cardinality $|L_{SEF}| = |\Delta| = \aleph_0$, then each finite and valid $\phi \in \Delta, |\phi| = k, \forall k \in \mathbb{N}$ is itself an ω-regular language that contains exactly one binary infinite string it can recognize.

[70] this however should be more carefully reviewed for the ω-regular languages and respective "ω-RE" class if only such can be meaningfully defined for some extended version of Chomsky hierarchy

[71] ultimately, by building up on the result of Theorem 10 - so that uncountability concept is fully aligned with the original set theoretical notion by Cantor

Proof. By definition in [11], Γ^* is a formal language iff it is an infinite list or a sublist of all possible finite strings built over some alphabet Γ. If one considers infinite words or ω-words they will constitute an ω-language. Similar to regular languages, ω-language is ω-regular iff it can be recognized using an ω-regular expression. This can be shown by replacing replicator in each $\phi \in \Delta$ with the ω-Kleene closure as defined for the ω-regular expressions. Since ϕ can produce only one infinite (countable) binary string when computed as SEF, the corresponding ω-regular language will also contain only one such string. □

However, the above still means that a single infinite binary string can be recognized by a multitude of ω-regular expressions or many $\phi \in \Delta$ formulas. But each SEF ϕ produces only one infinite binary string. Again, given that $\mathbb{B}$ is a list of all infinite binary strings, then such mapping as $f : \Delta \to \mathbb{B}$ is not necessary a surjection unless we select some particular subset $\Delta' \subset \Delta$, s.t. $f' : \Delta' \to \mathbb{B}$ will be a surjection[72].

Corollary 15.

If $L_{SEF} := \Delta(\Sigma^*)$ is a finite SEF language of countably infinite cardinality $|L_{SEF}| = \aleph_0$ such that for each finite and valid $\phi \in \Delta, |\phi| = k, \forall k \in \mathbb{N}$ there exists a corresponding ω-regular expressions which can accept ω-regular language, then L_{SEF} itself is ω-regular.

Proof. L_{SEF} can be parsed by ω-regular expression. □

A SEF language that can recognize a formula of infinite (countable) length is called infinite.

Definition 23.

If $\mathcal{L}_{SEF}$ is a SEF language and $\mathcal{L}_{SEF} := \{\phi \; is \; valid \; : |\phi| = \aleph_0, \forall \phi \in \Delta(\Sigma^\omega)\}$ (where Σ^ω is closed under ω-Kleene closure over the Σ alphabet), then we say that $\mathcal{L}_{SEF}$ is infinite.

Lemma 16.

In general if $\mathcal{L}_{SEF} := (\Sigma, \Delta_\delta), \delta \leq \omega$ is an infinite SEF language of uncountable cardinality and $\Delta_\delta : \Sigma^\omega \to \mathcal{L}_{SEF}$, then $|\Sigma^\omega| = |\Delta_\delta| = 2^{\aleph_0}$[73] and Δ_δ is undecidable[74].

[72] It is an important idea that is used later in the paper to look at the equivalence classes of SEFs

[73] see the generalization of the cantor set as an argument to illustrate this

[74] in general turning-undecidable sense due to uncountability of pre-image and image of the Δ_δ mapping, although even countability will be a sufficient condition for being undecidable

We continue by defining a production function that computes finite and countable SEFs into infinite binary strings. For the rest of the section assume $\mathcal{L}_{SEF} := \Delta_0(\Sigma^\omega)$ be an infinite SEF language of uncountable cardinality over a finite alphabet $\Sigma := \{0,1\} \cup \{(,)\}$ with binary vocabulary and typical syntax, so that each expression ϕ of either finite or countable length is valid in zero-depth didactic $\forall \phi \in \Delta_0(\Sigma) : |\phi| \geq n, \forall n \in \mathbb{N}$.

Definition 24.

Let $\mathcal{L}_{SEF} := \Delta_0(\Sigma^\omega)$ be an infinite SEF language of uncountable cardinality over a finite alphabet $\Sigma := \{0,1\} \cup \{(,)\}$. If $\mathbb{B}$ is a list of all infinite binary strings, then $\exists \pi : \Delta_0 \to \mathbb{B}$ and π is called a *production* function.

Previously we have applied GCGA scanning to classify infinite binary strings by putting them, into four categories *cyclic, non-cyclic, proper countable* and finally *uncountable*. We have also used finite SEFs as a notational shortcut for convenience. Note that formulas that we have mentioned before were mostly finite. However, according to the requirement in the most recent extended *Definition 17*: for formulas in didactic Δ_0 to be considered as valid they must be either of finite or countable length. The purpose for this is that we want to consider a broader number of cases for SEFs. Specifically another important kind of SEFs should be taken into account when discussing the structure of Δ_0 that outputs infinite binary strings and is in fact equivalent to the output.

Definition 25.

If $\phi \in \Delta_0$ is a valid formula of countable length $|\phi| = \omega_0 : \omega_0 > n, \forall n \in \mathbb{N}$, s.t. once computed ϕ will output an infinite binary string that contains no infinite repeatable cycle (pattern), then ϕ is called *countable fairly non-cyclic formula* or simply *fair*[75].

One can state the same using a production function.

Lemma 17.

A formula $\phi \in \Delta_0$ of countable length is fair iff there exists no other formula containing a replicator $\nexists \tau \in \Delta_0 : |\tau|_(= |\tau|_) = n : n > 0, n \in \mathbb{N}$ that can produce an equivalent infinite binary string as an output $\pi(\tau) = \pi(\phi)$, where $\pi : \Delta_0 \to \mathbb{B}$ is a production function.

[75] by analogy with a notion of a fair coin in probability theory, that can produce somewhat complex infinite strings, which exhibit fairly random properties like being difficult to compress

Lemma 18.

Every fair formula $\phi \in \Delta_0$ is idempotent in the sense of production $\pi(\phi) = \phi$ or equivalently $\exists \eta : \mathrm{H} \to \mathrm{H}, \mathrm{H} = \Delta_0 \cap \mathbb{B}$, s.t. $\eta(\eta(\phi)) = \eta(\phi)$ is idempotent and H is an ideal of Δ_0.

Next to fair formulas we can also differentiate countable formulas that contain at least one replicator.

Definition 26.

If $\phi \in \Delta_0$ is a valid formula of countable length $|\phi| = \omega_0 : \omega_0 > n, \forall n \in \mathbb{N}$, s.t. once computed ϕ will output an infinite binary string that contains at least one infinite repeatable cycle (pattern), then ϕ is called *countable unfair formula* or simply *unfair*.

We are interested in learning more about the cardinality of infinite languages. Before we dive further into exploring the relation between Δ_0 and $\mathbb{B}$, let us recall another very useful result due to Cantor. We want to extend the recursive definition of the space called the *Cantor set*[76].

The Cantor ternary set $\mathcal{C}$ is created by iteratively deleting an open middle third from a set of line segments. Note that we can generalize the iterative construction of the ternary Cantor set. Instead of cutting out a middle and leaving only 2 segments on each iteration, we can also consider taking multiple segment cuts on $[0, 1]$. Then we will repeat the process on each iteration for each of the remaining segments infinitely many times.

Definition 27.

Let $\mathcal{L}_{SEF} := (\Sigma, \Delta)$ be an infinite SEF language of uncountable cardinality over a finite alphabet $|\Sigma| = b, \forall b \in \mathbb{N}$, b being the length or *base* of the alphabet. A generalized Cantor set $\mathcal{C}^b$ is produced by recursively cutting each segment into b subsegments infinitely many times, s.t. each point of the infinite $\mathcal{C}^b$ can be uniquely identified by an infinite (countable) b-ary string $\tau \in \mathcal{L}_{SEF}$ used as a tree path over $\mathcal{C}^b$.

Note that for $b = 1$ the set $\mathcal{C}^b$ is trivial and always corresponds to an interval $[0, 1]$, so no obvious cutting is possible[77]. It is easy to see that the above definition extends on the construction of the ternary Cantor set $\mathcal{C}$, which corresponds to $b = 2$. As a result $\mathcal{C}^b$ inherits well-known properties of $\mathcal{C}$ such as:

[76] as provided in [6] and more comprehensively in [13]

[77] however, if we continue to cut into pairs of intervals intersecting only at limit points, each iteration may illustrate well what is called *countable chain condition* - see *Suslin Problem* in [6]

1. *zero length* - the Cantor set is nowhere dense, and has Lebesgue measure 0

2. *cardinality of continuum* - the Cantor set contains an uncountably infinite number of points so that $|\mathcal{C}| = \mathfrak{c}$

3. *fractional dimension* - the Cantor set is self-similar to copies of itself[78]

Lemma 19.

Cardinality of the generalized Cantor set with any finite base is the same as the cardinality of the ternary Cantor set, namely $\forall b \in \mathbb{N}$:

1. $\mathfrak{c} = |[0,1]|$

2. $|[0,1]| = |\mathcal{C}^b|$

3. $|\mathcal{C}^2| = 2^{\aleph_0}$

Proof. Point (1) follows from the fact that a closed interval $[0,1]$ [79] is isomorphic to the real line $\mathbb{R}$ and $|\mathbb{R}| = \mathfrak{c} = |[0,1]|$.

Point (2) follows from the fact that each set $|\mathcal{C}^b|$ lies inside the interval $[0,1]$ which it tries to subdivide in an infinite manner. For $b = 2$ each point on $[0,1]$ can be described by an infinite binary path $p \in \{0,1\}^\omega$ over the tree of subdivisions of $[0,1]$. The same is true for $b > 2$, namely each $p \in \{0,..,b\}^\omega$ corresponds to a uniquely mapped point of $[0,1]$ through the countable recursive process of subdivision of $[0,1]$ into b segments. Hence, there exists a bijection between paths to edges of $\mathcal{C}^b$ segments and unique points of $[0,1]$. This implies $|\mathcal{C}^2| = |\{0,1\}^\omega|$. Observation that $|\{0,1\}^\omega| = 2^\omega = 2^{\aleph_0}$ already implies (3).

However, we will also consider an alternative argument. Each path $p \in \{0,1\}^\omega$ is a countable binary string, i.e $|p| = \aleph_0$. According to *Definition 120*, there is, indeed, a one-to-one correspondence between $\{0,1\}^\omega$ and $\mathcal{P}(A)$ of some set A. This means that at a countable number of iterations there is a countable subset of $[0,1]$ or $\exists A$, s.t. $|A| = \aleph_0$ iff $A \subset \mathcal{C}^2_\omega \subseteq \mathcal{C}^2$. Now, (3) follows from $|A| = \aleph_0 \implies |\mathcal{P}(A)| = 2^{\aleph_0}$[80]. □

The above statement is a paraphrased line of thinking by Cantor on his approach to CH. This sequence of equalities $\mathfrak{c} = |\mathcal{C}^b| = |\mathcal{C}^2| = 2^{\aleph_0}$ involves observations that may be not immediately obvious. In turn, previous lemma has some important consequences for our discussion.

[78] The Hausdorff dimension of the Cantor set is equal to $ln(2)/ln(3)$[13]

[79] or, equivalently, an open interval $(0,1)$

[80] Please see Fraenkel morphism, which is discussed much later in this paper, as well as the proof in [**jech**] on p.28

Theorem 20.

Let $\mathcal{L}_{SEF} := \Delta_\delta(\Sigma^\omega), \forall \delta \in \mathbb{N} \cup \{0\}$ be an infinite SEF language of uncountable cardinality over a finite alphabet $\Sigma := \{0,1\} \cup \{(,),:\}$. If $\mathbb{B}$ is a list of all infinite binary strings, then $|\mathcal{L}_{SEF}| = |\Delta_\delta| = |\mathbb{B}| = \mathfrak{c}$.

Proof. Given Σ is a finite alphabet with base $\gamma = |\Gamma|, \gamma > n, \forall n \in \mathbb{N}$, for the smallest $\gamma = 2 : \Gamma := \{0,1\}$ we have an infinite ω-regular language that can recognize all infinite binary strings $\mathbb{B} \subseteq \Gamma^\omega$ and a corresponding ternary Cantor set $|\mathbb{B}| = |\mathcal{C}|$. In case of $\mathcal{L}_{SEF} := \Delta_\delta(\Sigma^\omega)$ we have an alphabet base $b = |\Sigma| = 5$ and a corresponding b-ary Cantor set of the same cardinality $|\mathcal{C}^b| = |\mathcal{C}^5| = |\Sigma^\omega|$. Since $\mathcal{L}_{SEF}$ consists only of valid expressions, $\Delta_\delta \subseteq \Sigma^\omega$, we know that $|\Delta_\delta| \leq |\mathcal{C}^5|$. In general for some two alphabets if $|\Gamma| = \gamma, |\Sigma| = b : \gamma \leq b$, then also $|\Gamma^*| \leq |\Sigma^*|$ for finite strings or $|\Gamma^\omega| \leq |\Sigma^\omega|$[81] for the ω-regular languages with strings of countable length as in this context. So $|\Delta_\delta|$ cardinality must be somewhere between $|\mathcal{C}^2| \leq |\mathcal{C}^3| \leq |\mathcal{C}^4| \leq |\Delta_\delta| \leq |\mathcal{C}^5|$. But according to *Lemma 19*[82] $|\mathcal{C}^b| = |\mathcal{C}| = \mathfrak{c} : \forall b \in \mathbb{N}$, hence $|\Gamma^\omega| = |\mathcal{C}| = \mathfrak{c} = |\mathbb{B}| = |\Delta_\delta| = \mathcal{L}_{SEF}$[83]. □

In order to understand the scale to which any infinite SEF language can be uncountable, we have matched the cardinalities of two lists: all infinite string enumeration formulas $|\mathcal{L}_{SEF}|$ with all infinite binary strings $|\mathbb{B}|$. Assuming the CH is false, defining[84] an infinite SEF language of uncountable cardinality κ does not necessary mean that $\kappa = \mathfrak{c}$, so showing this explicitly is an important result.

5.8 Equivalence classes of SEFs

We are approaching a part of our discussion when we are ready to describe what is an equivalence class of string enumeration formulas recognized by $\mathcal{L}_{SEF} := \Delta_\delta(\Sigma^\omega), \delta = 0$[85].

A finite case of all finite binary strings $\mathcal{B}$ can help us to understand the nature of another important countable subset of all valid but finite formulas L_{SEF}.

Lemma 21.

If $\mathcal{B}$ is a list of all finite binary strings, then $|\mathcal{B}| = \aleph_0$.

[81] Although the future corollary of the current proof would be $|\Gamma^\omega| = |\Sigma^\omega|$

[82] alternatively one can construct a different proof with an inverse argument to apply Schröder–Bernstein theorem

[83] $|\mathcal{L}_{SEF}| = |\Delta_\delta|$ by definition and $\mathfrak{c} = |\mathbb{B}|$ is true due to *Lemma 19*

[84] or constructing

[85] for now and without loss of generality, we will not consider infinite SEF languages with depth of continuum access schema greater than zero - recall that $\forall \phi \in \Delta_0 : |\phi|_: = 0$

Proof. $\exists\beta : \mathbb{N} \to \mathcal{B}$ by conversion of every natural number $n \in \mathbb{N}$ into a finite binary string representation and β is a bijection. □

Lemma 22.

If $L_{SEF} := \Delta_0(\Sigma^*)$ is a finite SEF language over alphabet $\Sigma := \{0,1\} \cup \{(,)\}$, s.t. for $\forall\phi \in L_{SEF}, |\phi| \in \mathbb{N}, \forall b \in \mathcal{B}, |b| \in \mathbb{N}$ there exists a surjection $\exists\rho : \phi \mapsto b$, then $|L_{SEF}| = |\Delta_0| = \aleph_0$.

Proof. Assume that for $\forall\phi \in L_{SEF}, |\phi| \in \mathbb{N}, \forall b \in \mathcal{B}, |b| \in \mathbb{N}$ there exists a projection $\exists\rho : \phi \mapsto b$ which is defined by replacing $\{(,)\}$ symbols with empty string, so that $|\mathcal{B}| \leq |L_{SEF}|$. From previous definitions[86] as we have $L_{SEF} = \Delta_0 \subset \Sigma^*$ and $|\Sigma^*| = \aleph_0$. After applying Schroeder–Bernstein theorem, it follows that $|\Delta_0| = |L_{SEF}| = |\mathcal{B}| = \aleph_0$. □

Assume one would like to enumerate all finite SEFs $\forall\phi \in L_{SEF} : |\phi| \in \mathbb{N}$ based on *Definitions 17-20*. The question would be if there exists some efficient algorithm that could eventually print out a list of all possible and valid formulas for L_{SEF}? According to *Theorem 12* we know that L_{SEF} is a regular language, unfortunately it remains open if there exists an even more efficient algorithm to enumerate L_{SEF}[87].

Another specificity is that if any two finite SEFs are to be fully computed, their computation result will not be unique. Meaning that one can easily pick two or more different formulas that will still print out a similar infinite binary string as an output. Indeed, this is easy to illustrate by example for some finite $\phi, \tau \in L_{SEF}$ and $|\phi|, |\tau| \in \mathbb{N}$. Let assume that $\phi :=$ "100111(01)". Note that one can pick $\tau \neq \phi : \tau :=$ "100111(01)(01)". In fact, for cases as in this example with some finite $\phi :=$ "100111(01)" there always exists an infinite number of formulas that would print out a similar result just as ϕ. However, ϕ will be the shortest formula or the formula with the minimum length for such equivalence class. We can state this more formally. Again we proceed by considering a finite case first.

Definition 28.

If $\exists\eta : \eta(\Phi) = \Phi$ acting as identity function for all subsets $\forall\Phi \subset L_{SEF}$, then η partitions a list of valid but finite SEFs in L_{SEF}, so that $\exists \mathrm{E}_{SEF} \subset L_{SEF}$ which is a list of unique labels of every equivalence class of such partitioning. A straightforward way to define such identity function η would be to use production function $\pi : \mathcal{L}_{SEF} \to \mathbb{B}$ [88], so that each equivalence class results

[86] Definitions 13, 17, 18, 19, 20

[87] see subsection - *Computability of SEF languages*

[88] here and when applicable, for the countable case the actual domain of π is just L_{SEF} as in $\pi : L_{SEF} \to \mathbb{B}$, where $L_{SEF} \subset \mathcal{L}_{SEF}$

in producing the same output value in form of the infinite binary string meaning that $\exists b \in \mathbb{B} : \Phi := \{\phi : \pi(\phi) = b, \forall \phi \in L_{SEF}\}$, where Φ is an equivalence class uniquely labeled either by b or by the shortest formula $\phi \in \Phi : \phi = \inf \Phi$.

This means that if we want to use the production function $\pi : \mathcal{L}_{SEF} \to \mathbb{B}$ to print out unique results of all finite valid formulas in L_{SEF}, it is enough to enumerate only respective labels of the equivalence classes $\mathrm{E}_{SEF} \subset L_{SEF}$. This would look like $\{\pi(\phi) : \phi \in \mathrm{E}_{SEF}\} \subset \mathbb{B}$. It is obvious that $|\mathrm{E}_{SEF}| = |L_{SEF}| = \aleph_0$[89]. But it also makes sense to get a better understanding of how exactly $|\mathrm{E}_{SEF}|$ is organized by providing few intuitive examples of the initial enumeration attempts.

Enumeration of E_{SEF} can be well-ordered or sorted alpha-numerically, since L_{SEF} is a finite language and we can assign a weight for each character in the alphabet $V \cup \gamma = \{0, 1, (,)\}$. We have devised an algorithm to help to enumerate and sample first groups of labels in E_{SEF} as a table $\{E(n, m) \subset \mathrm{E}_{SEF} : \forall n, m \in \mathbb{N}\}$, where n is the base binary string length or number of $\{0, 1\}$ and m is a number of balanced bracket pairs in each finite formula (see fig. 4).

The choice to arrange the grouping in the enumeration table by n, m has to do with some redundancies that the algorithm had to discard according to the definition of the labeling set E_{SEF}. First, every formula produces an infinite string by definition, meaning that there are no formulas that can output a finite binary string $\mathcal{B} \cap \pi(\mathrm{E}_{SEF}) = \emptyset$[90]. Second, ..

Theorem 23.

If $\mathrm{E}_{SEF} \subset L_{SEF}$ is an enumeration of labels of the equivalence classes $\forall \Phi \subset L_{SEF}$ of finite SEFs, s.t. each label is a shortest formula in equivalence class $\phi \in \Phi : \phi = \inf \Phi$, then $|\mathrm{E}_{SEF}| = |L_{SEF}| = \aleph_0$.

Proof. Given that $|L_{SEF}| = \aleph_0$ is of countable cardinality, any infinite partitioning of L_{SEF} will result in the same cardinality. We define the identity function using production function π as an equivalence criterion for any two formulas $\forall \phi, \tau \in \Phi : \pi(\phi) = \pi(\tau)$. Assume enumeration of labels in E_{SEF} is finite, then no new formulas can be added to such list so that $\exists \phi \in \mathrm{E}_{SEF} : |\phi| = n, n \in \mathbb{N}$ is the last and the longest of all the shortest labels picked from each of the respective equivalence classes. But this is not the case, since one can always find a new formula $\phi' \in L_{SEF}, |\phi| = k, k \in \mathbb{N}$ by enumerating formulas of greater length $k > n$ that are not in the list of labels yet $\phi' \notin \mathrm{E}_{SEF}$. If this new formula candidate will turn out to produce a new infinite binary string $\pi(\phi')$ that is different from any of the

[89] see the theorem below

[90] all labels $\forall \phi \in \mathrm{E}_{SEF}$ are finite unfair formulas that output countable strings

seen outputs of the $\forall\phi \in \mathrm{E}_{SEF} : \pi(\phi) \neq \pi(\phi')$, then it is added to the list $\mathrm{E}_{SEF} := \mathrm{E}_{SEF} \cup \{\phi'\}$. Otherwise, new formula candidate is selected. Like this E_{SEF} list will grow until countably many labels are enumerated. Hence, exhausting partitioning of L_{SEF} is infinite and $|\mathrm{E}_{SEF}| = |L_{SEF}| = \aleph_0$. □

Now we want to show that similar partitioning exists for uncountable case of $\mathcal{L}_{SEF}$.

Theorem 24.

If $\mathcal{E}_{SEF} \subset \mathcal{L}_{SEF}$ is an infinite enumeration of labels of the equivalence classes $\forall\Phi \subset \mathcal{L}_{SEF}$, then $|\mathcal{E}_{SEF}| = |\mathcal{L}_{SEF}| = \mathfrak{c}$.

Proof. Firstly let us show the existence of $\mathcal{E}_{SEF}$ and then try to match its cardinality with $\mathcal{L}_{SEF}$. We can show the existence of $\mathcal{E}_{SEF}$ by illustrating the labeling algorithm that helps to enumerate $\mathcal{E}_{SEF}$, so that each label is unique. Notice that the production function $\pi : \mathcal{L}_{SEF} \to \mathbb{B}$ is an injection since any formula $\forall\phi \in \mathcal{L}_{SEF}$ can be computed into some infinite binary string $\pi : \phi \mapsto b, b \in \mathbb{B}$. In the same time, when π is applied to a list of labels of the equivalence classes $\mathcal{E}_{SEF} \subset \mathcal{L}_{SEF}$, so that $\pi : \mathcal{E}_{SEF} \to \mathbb{B}$, it also acts as surjection in the sense that for each infinite binary string $\forall b \in \mathbb{B}$ there exists a unique valid formula $\phi \in \mathcal{E}_{SEF}$ such that either the formula is finite and unfair[91], or $|\phi| = \aleph_0$. In the countable case formula ϕ can be either *fair*, so that $\pi(\phi) = \phi = b$ is idempotent, or *unfair*. For simplicity, we agree that the labeling algorithm has a convention that if at least one formula in the equivalence class is finite, then the label matches with some equivalence class $\Phi \subset \mathrm{E}_{SEF}, \mathrm{E}_{SEF} \subset \mathcal{E}_{SEF}$. Otherwise, if all formulas in some equivalence class Φ are countable $\nexists\phi \in \Phi : |\phi| \in \mathbb{N}$, then we pick $b = \pi(\phi), \forall\phi \in \Phi, b \in \mathbb{B}$. Like this both countable fair and countable unfair formulas will receive infinite binary string $b = \pi(\phi)$ as their respective class labels.

Such labeling algorithm shows that infinite partitioning of $\mathcal{L}_{SEF}$ exists. Since $\pi : \mathcal{E}_{SEF} \to \mathbb{B}$ acts as bijection between $\mathcal{E}_{SEF}$ and $\mathbb{B}$, we have $|\mathcal{E}_{SEF}| = |\mathbb{B}| = \mathfrak{c} = |\mathcal{L}_{SEF}|$. □

The above labeling algorithm produces only disjoint equivalence classes[92]. Imagine that the countable formula in some equivalence class $\Phi \subset \mathcal{E}_{SEF} : \tau \in \Phi, |\tau| = \aleph_0$ produces an infinite output which was already seen in the enumeration of finite SEFs $\pi(\tau) \in \pi(\mathrm{E}_{SEF})$. It is only possible if the same equivalence class also contains the shortest possible finite formula $\phi \in \Phi : \phi = \inf \Phi, |\phi| \in \mathbb{N}$, which is also a label of this equivalence class. Let us define such a labeling algorithm more explicitly.

[91] contains at least one replicator since $\mathrm{E}_{SEF} \subset \mathcal{E}_{SEF}$

[92] as per definition of the notion of the equivalence class in set theory

Definition 29.

Given $\mathcal{E}_{SEF} \subset \mathcal{L}_{SEF}$ is an infinite enumeration of labels of the equivalence classes $\forall\Phi \subset \mathcal{L}_{SEF}$, we can define an algorithm $\lambda : \mathcal{L}_{SEF} \to \mathcal{E}_{SEF}$ that can be used to assign labels to those classes $\forall\Phi \subset \mathcal{L}_{SEF}$ as following:

1. If $\exists\phi \in \Phi$, s.t. $|\phi| \in \mathbb{N}$, then take the shortest possible finite formula $\tau \in \Phi : \tau = \inf \Phi$ as the label of the class $\lambda(\Phi) := \tau$
2. Otherwise if $\nexists\phi \in \Phi$, s.t. $|\phi| \in \mathbb{N}$ and all formulas are of countable length only $\forall\phi \in \Phi : |\phi| = \aleph_0$, then $\lambda(\Phi) := \pi(\Phi)$[93].

Lemma 25.

All equivalence classes of countable partitioning of L_{SEF} are countable $\forall\Phi \in \mathrm{E}_{SEF}, \mathrm{E}_{SEF} \in L_{SEF} : |\Phi| = \aleph_0$.

Lemma 26.

All equivalence classes of uncountable partitioning of $\mathcal{L}_{SEF}$ are at least countable $\forall\Phi \in \mathcal{E}_{SEF}, \mathcal{E}_{SEF} \in \mathcal{L}_{SEF} : |\Phi| \geq \aleph_0$, except if Φ is labeled by a countable formula which is also fair[94].

Proof. There exists an infinite amount of formulas that can produce the same infinite binary output string $\forall\phi \in \Phi, \Phi \subset \mathcal{E}_{SEF}$ unless ϕ is countable fair. We can show this by infinite concatenation of formulas from $\mathrm{E}_{SEF} \subset \mathcal{L}_{SEF}$ or by infinite concatenation of reducible parts of any equivalence class label formula ϕ. □

Now we can take a deeper insight into the enumeration of labels of the equivalence classes $\mathrm{E}_{SEF} \subset L_{SEF}$ by applying slashing to its production image.

Theorem 27.

Countable enumeration of labels of the equivalence classes of finite SEFs $\mathrm{E}_{SEF} \subset L_{SEF}$ is complete in a sense that $\nexists\phi' : \phi' \in \mathrm{E}_{SEF}$ if $\pi(\phi') = b', b' \in \mathbb{B}$ and b' was produced by Cantor slashing of $\pi(\mathrm{E}_{SEF})$.

Proof. □

[93] This is important for the next subsections as we would write $\lambda(\mathcal{U}_{SEF}) \subset \mathcal{L}_{SEF}$ meaning that we care to pick each $b = \pi(\phi), \phi \in \Phi, |b| = \aleph_0$ as a respective unique label for each equivalence class $\Phi \subset \mathcal{U}_{SEF} : |\Phi| = \aleph_0$

[94] In later case a fair countable formula $\phi \in \Phi$ labels equivalence class which contains only itself.

Corollary 28.

If $b' \in \mathbb{B} : b' = \pi(\phi'), \phi' \in \mathrm{E}_{SEF}$ was produced by Cantor slashing of $\pi(\mathrm{E}_{SEF})$, then ϕ' is countable unfair formula and $\phi' \in \mathcal{E}_{SEF}$ but $\phi' \notin \mathrm{E}_{SEF}$.

5.9 Continuo Cantor Argument

At this point we want to recap on the results in *Theorem 10* to indicate that *GCGA* algorithm is not just instrumental as an alternative way to show uncountability, but it can be also equipped as a part of an equivalent and even more general tool than slashing. Instead of applying Cantor slashing in recursive iterations to yield greater and greater indexes of $\aleph_k, k \in K$[95], we can approach this simultaneously and in one go. We will refer to such *in situ et statim* approach as *Continuo Cantor Argument (CCA)* or *c-slashing* for short.

Definition 30.

Let $(\Omega, \mathfrak{F}_c, P_c)$ be a discrete probability space[14] for c-slashing, where $\Omega = \{0,1\}^{\omega_0}$ is at most countable *sample space* of an endlessly tossed coin s.t. the tossing outcomes are arbitrary[96] and form a countable binary string $b \in \mathbb{B}$, $\mathfrak{F} \subseteq 2^{\Omega}$ is a σ-algebra[97] containing all information about possible outcomes and corresponds to a countable partition $\Omega = \bigcup_{o \in O} B_o, |O| = \aleph_0$[98], and, finally, $P : \Omega \to [0,1]$ is a probability measure function[99].

Now imagine a computational process as a countable sequence of deterministic events[100] that produces a countable list of infinite (countable) binary strings. Such result can be accessed in form of an infinite table[101] or a cursor function using respective indexes for algorithmic access $T(i,j), i \in I, j \in J, |I| = |J| = \aleph_0$.

We say that c-slashing is a process similar to Cantor diagonal argument, when applied to $T(i,j)$ values[102] so if a string $b' \notin T(i,j)$ is a result of slashing, then it can be "virtually" added to and accessed from $T'(i',j')$

[95] where K is some index up to a certain countable cardinality $|K| \leq \aleph_0$ or greater if reasonable

[96] in a sense, that each outcome does depend only on $(\Omega, \mathfrak{F}, P)$ space

[97] sometimes also referenced as a field of sets[14] or a σ-field

[98] in fact, this σ-algebra contains all countable partitions of 2^{Ω} by definition

[99] for most our needs equal probabilities are assigned $\forall b \in \mathbb{B}$, so the uniform probability measure does suffice

[100] In general, these events are fully independent of the c-slashing probability space $(\Omega, \mathfrak{F}_c, P_c)$, as such computations are almost always countably certain to happen or are deterministic in the sense that they require countable amount of time.

[101] following into the prerequisites similar as for the initial configuration in Theorem 10

[102] see fig. 3 for similar illustration of slashing to produce ω_0

cursor function extended over $T(i,j)$, s.t. $i' \in I', j' \in J', |I'| = |J'| = \kappa, \kappa \leq |T(i,j) \cup \{b'\} \cup \ldots| \leq \aleph_1$.

Definition 31.

If $B \subset \mathbb{B}$ is a countable list of countable binary strings and $r_c \leftarrow (\Omega, \mathfrak{F}_c, P_c)$ is an infinite binary string $r_c \in \mathbb{B}$ generated from some subset of events $C \subset \mathfrak{F}_c$ with probability P_c, then we say that *c-slashing* is an extension over a set $B' := [B]|r_c$ resulting in a strictly greater cardinality of the new set $|B'| > |B|$.

Definition 32.

$B' := [B]|r_c$ is c-slashing of $B \subset \mathbb{B}, |B| = \aleph_0$ iff B' is the result of the following algorithmic process:

1. Start by remapping indexes to access all input strings of B as left-infinite binary strings via a cursor function over infinite table, meaning $LH := T(i, j < 0)$;

2. To complete initial configuration for GCGA, extend and initialize $RH := T(i, j \geq 0)$ on the right-hand with one single additional column leaving other placeholders unset.[103];

3. Initialization is done by:

   (a) generating a random infinite (countable) binary string according to probability space $r_c \leftarrow (\Omega, \mathfrak{F}_c, P_c)$

   (b) assigning $\forall i \in I : T(i, j = 0) \leftarrow r_c(i)$ [104];

4. Result of applying $GCGA$ can be algorithmically accessed over space of greater cardinality $T'(i', j') := GCGA(B, r_c), |T'(i', j')| > |T(i, j)|$.

Lemma 29.

If c-slashing $B' := [B]|r_c$ is applied to $B \subset \mathbb{B}, |B| = \aleph_0$ so that $r_c \leftarrow (\Omega, \mathfrak{F}_c, P_c)$ is generated from the subset of events $C \subset \mathfrak{F}_c$ under condition that r_c contains only a single non-zero value $|r_c|_1 = 1, |r_c|_0 = \aleph_0$, then such special case of c-slashing is equivalent to obtaining B' by applying cantor argument to extend B.

[103] Usually first column of $T(i, j = 0)$ is set to the scanned successive values from some input string $b \in \mathbb{B}$ by running $GCTA$ over b as a left-infinite sliding window - see the setting of Theorem 10. But for c-slashing this step is skipped as it will not add any new strings to the LH.

[104] so that r_c is transposed and its values are used to initialize the column at $T(i, j = 0)$

5.10 A case for $\neg CH$

We continue to work with the structure of $\mathcal{E}_{SEF}$, which would lead us to consider a strong case for $\neg CH$. From the result in *Corollary 28* we have learned that slashing can be used to construct a countable binary string $b' = \pi(\phi), b' \in \mathbb{B}$, which is computed from a countable unfair formula $\phi \in \mathcal{L}_{SEF}$. Although it may seem that there are simpler ways to construct countable unfair formulas such as using countable concatenations of finite unfair strings $\forall b \in \mathrm{E}_{SEF}$[105], generated collections of such concatenations will exclude not only much of possible redundancies typical for countable unfair formulas[106], the resulting product cardinalities will remain limited by the original cardinality of the initial sets[107]. This changes when we seek to apply c-slashing to $B = \pi(\mathrm{E}_{SEF})$.

Theorem 30.

Let c-slashing $B' := [B]|r_c$ be applied to $B = \pi(\mathrm{E}_{SEF}), B \subset \mathbb{B}, |B| = \aleph_0$ so that $r_c \leftarrow (\Omega, \mathfrak{F}_c, P_c)$ is generated from the subset of events $C \subset \mathfrak{F}_c$ chosen under one of the two conditions:

1. *condition A*: r_c contains equal counts of 0-s and 1-s $|r_c|_1 = |r_c|_0 = \aleph_0$;
2. *condition B*: r_c contains up to finite difference between counts of 0-s and 1-s $||r_c|_1 - |r_c|_0| < n, n \in \mathbb{N}, |r_c|_1 = \aleph_0, |r_c|_0 = \aleph_0$.

If the above is true and $\exists B'$, then there exists some list of countable unfair formulas $\exists \mathcal{U}_{SEF} : B' = \lambda(\mathcal{U}_{SEF}), \aleph_0 < |B'| \leq |\mathcal{U}_{SEF}|$ which can be used to produce SEF equivalence class labels (countable binary strings) disjoint from E_{SEF}, specifically $\lambda(\mathcal{U}_{SEF}) \subseteq \mathcal{E}_{SEF} : \lambda(\mathcal{U}_{SEF}) \cap \mathrm{E}_{SEF} = \emptyset$.

Theorem 31.

List of countable fair formulas $\mathcal{F}_{SEF} \subset \mathcal{E}_{SEF}$ is closed under finite c-slashing.

Corollary 32.

There exists a list of countable fair formulas which can be used to produce SEF equivalence class labels (countable binary strings), specifically $\exists \mathcal{F}_{SEF} = \lambda(\mathcal{F}_{SEF}) : \mathcal{F}_{SEF} \subseteq \mathcal{E}_{SEF}$ and $|\mathcal{F}_{SEF}| > \aleph_0$.

Finally, let us look into the statement which is equivalent to a weaker condition to show $\neg CH$.

[105] By somewhat distant analogy as to using cylinder sets (or product sets) for generation

[106] such as reducing number of nested brackets or many tautological repetitions

[107] for instance, $\aleph_0 = \aleph_0 \times \aleph_0$, unless we apply slashing or knowledge of Cantor's theorem ending up either with $\aleph_1$ or with $2^{\aleph_0} = |\mathcal{P}(\mathbb{N})|$

Theorem 33 - Strong-case Theorem for $\neg CH$ (SNCH).

There exists a list of countable SEF formulas satisfying a weaker condition for $\neg CH$, namely $\exists B' : \aleph_0 < |B'| < 2^{\aleph_0}$.

Proof. We will break our proof into several steps:

1. Consider $B := \lambda(\mathrm{E}_{SEF}) = \pi(\mathrm{E}_{SEF}), B \subset \mathbb{B}, |B| = \aleph_0$
2. Obtain $B' := [B]|r_c$ by applying c-slashing to B as in *Theorem 30*, so that $B' = \lambda(\mathcal{U}_{SEF})$, where $\mathcal{U}_{SEF}$ is a list of countable unfair formulas s.t. $\lambda(\mathcal{U}_{SEF}) \subseteq \mathcal{E}_{SEF}$ and $|\lambda(\mathcal{U}_{SEF})| > \aleph_0$.
3. We know that $|\mathrm{E}_{SEF}| = \aleph_0$. Now assume that the cardinality of $\lambda(\mathcal{U}_{SEF})$ is rather big, so that disjoint union of equivalence class labels from both lists of all finite and of all countable unfair formulas can exhaust (cover) equivalence classes of countable SEFs $\mathcal{E}_{SEF} = \mathrm{E}_{SEF} \cup \lambda(\mathcal{U}_{SEF})$[108] and $|\mathcal{E}_{SEF}| = |\lambda(\mathcal{U}_{SEF})| = \mathfrak{c}$.
4. However, according to *Corollary 32* there exists another list of labels of the equivalence classes of countable SEFs $\mathcal{F}_{SEF} \subset \mathcal{E}_{SEF}$ of greater than countable cardinality $|\mathcal{F}_{SEF}| > \aleph_0$ and disjoint from other labels $\mathcal{F}_{SEF} \cap (\lambda(\mathcal{U}_{SEF}) \cup \mathrm{E}_{SEF}) = \emptyset$ by definition[109]. This makes our assumption in the previous step false. In fact, we see that $\mathcal{E}_{SEF} = \mathrm{E}_{SEF} \cup \mathcal{F}_{SEF} \cup \lambda(\mathcal{U}_{SEF})$[110] and $|\mathcal{E}_{SEF}| = \mathfrak{c}, |\mathcal{E}_{SEF}| > |\lambda(\mathcal{U}_{SEF})|$
5. Given that $B' = \lambda(\mathcal{U}_{SEF})$, we have $|B'| < 2^{\aleph_0}$. We also know that $B := \lambda(\mathrm{E}_{SEF}) = \pi(\mathrm{E}_{SEF})$ and $|B'| > |B|, |B| = \aleph_0$. Hence, $\aleph_0 < |B'| < 2^{\aleph_0}$.

□

[108] where $\cup$ stands for disjoint union

[109] $\mathcal{F}_{SEF} \cap \mathcal{U}_{SEF} = \lambda(\mathcal{F}_{SEF}) \cap \lambda(\mathcal{U}_{SEF}) = \emptyset$ as fair formulas do not contain any replicators by definition as well as $\lambda(\mathcal{U}_{SEF}) \cap \mathrm{E}_{SEF} = \emptyset$ by definition of the labeling algorithm λ

[110] note that such finite disjoint union can be also seen as three equivalence classes of a finite partitioning of $\mathcal{E}_{SEF}$ which helps to assign them as representatives to three distinct cardinalities using *Axiom of Regularity* and definition of cardinals as $|X| = |Y|$ relation[6]. Also see Scott's trick[15, 16]

6 Accessibility, Choice and Fair Determinacy

6.1 Computability of SEF languages

The results presented in *Theorems 30-33* will require a number of clarifications to make sure that we can offer a proper interpretation for statements like $\mathcal{E}_{SEF} = \mathrm{E}_{SEF} \uplus \mathcal{F}_{SEF} \uplus \lambda(\mathcal{U}_{SEF})$. Indeed, the definition of a list of countable unfair formulas $\mathcal{U}_{SEF} \subset \mathcal{L}_{SEF}$ as it has been introduced by applying c-slashing in *Theorem 30* still works well assuming CH is true[111]. It was enough to consider the case that $|\mathcal{U}_{SEF}| > \aleph_0$, and c-slashing illustrates how such list can be succinctly constructed. However, given the new result of $\neg CH$ in *Theorem 33*, it becomes evident that in order to have a strict partitioning of $\mathcal{E}_{SEF}$ into $\mathcal{F}_{SEF} \uplus \lambda(\mathcal{U}_{SEF})$ next to countable E_{SEF}, the actual cardinalities of both $\mathcal{F}_{SEF}$ and $\mathcal{U}_{SEF}$ have to be much bigger[112]. This means that we have to update our definitions, if they are to be strict enough and consistent with our newly learned understandings.

Definition 33.

$\mathcal{U}_{SEF} \subset \mathcal{L}_{SEF}$ is a list of all countable strictly unfair SEFs iff the following holds:

1. all formulas in $\mathcal{U}_{SEF}$ are of countable length $\forall \phi \in \mathcal{U}_{SEF} : |\phi| = \aleph_0$
2. all formulas in $\mathcal{U}_{SEF}$ have no replicator over empty string $\forall \phi \in \mathcal{U}_{SEF} : (k = \text{“}x{\cdot}y{\cdot}z\text{”}) \wedge (|\phi|_k > 0) \wedge (x = () \wedge (|y|_(= |y|_) = 0) \wedge (|y| > 0) \wedge (z =))$
3. all formulas in $\mathcal{U}_{SEF}$ are strictly unfair - $\forall \phi \in \mathcal{U}_{SEF} : (\phi = \text{“}x \cdot y\text{”}) \wedge \forall x, y : (1 \leq |x|_(\leq |y|_)) \wedge (|\phi|_(= |\phi|_))$.
4. if $g_\alpha \in G$ is some generating procedure[113] from a family of all such procedures G and $\alpha \in I(G), |I(G)| \leq \mathfrak{c}$ is an index set over it, s.t. $U_\alpha := g_\alpha(\mathrm{E}_{SEF}), g_\alpha \in G$, then $\mathcal{U}_{SEF}$ is closed under α-indexed unions $\mathcal{U}_{SEF} := \bigcup_{\alpha \in I} U_\alpha$

where “$x \cdot y$” is a concatenation of two strings $\forall x, y : x, y \in \Sigma^\omega$ and $\Sigma := \{0, 1, (,)\}$[114]

A similar definition can be provided for $\mathcal{F}_{SEF}$.

Definition 34.

$\mathcal{F}_{SEF} \subset \mathcal{L}_{SEF}$ is a list of all countable strictly fair SEFs iff the following holds:

[111] and is sufficient for the proof as used in *Theorem 33*
[112] than initially considered or constructed by c-slashing
[113] similar to c-slashing such as conditions A and B in *Theorem 30*
[114] Also see *Definition 23*

1. all formulas in $\mathcal{F}_{SEF}$ are of countable length $\forall\phi \in \mathcal{F}_{SEF} : |\phi| = \aleph_0$
2. all formulas in $\mathcal{F}_{SEF}$ are strictly fair $\forall\phi \in \mathcal{F}_{SEF} : |\phi|_(= |\phi|_) = 0$
3. if $p_\alpha \in P$ is some generating procedure from a family of all such procedures P and $\alpha \in I(P), |I(P)| < \mathfrak{c}$ is an index set over it, s.t. $F_\alpha := p_\alpha(\mathrm{E}_{SEF}), p_\alpha \in P$, then $\mathcal{F}_{SEF}$ is closed under α-indexed unions $\mathcal{F}_{SEF} := \bigcup_{\alpha\in I} F_\alpha$

Given both definitions the partition statement $\mathcal{E}_{SEF} = \mathrm{E}_{SEF} \cup \mathcal{F}_{SEF} \cup \lambda(\mathcal{U}_{SEF})$ is consistent again[115]. We also avoid $\mathcal{F}_{SEF}$ and $\mathcal{U}_{SEF}$ to be universal sets, thus escaping unnecessary set theoretical paradoxes. However, one can see that both definitions rely on a rather abstract requirement - to be *closed under α-indexed unions*, which is limited by the cardinality of the index set[116]. Although mathematically concise, these definitions might not be computationally very practical.

At this point we want to ask a very radical question - to which extent we can attempt to reasonably enumerate $\mathcal{F}_{SEF}$ and $\mathcal{U}_{SEF}$ lists of SEFs? And if exhaustive enumeration is not possible, can we attempt any practical procedure to approach this? Computational enumeration is the next possible proxy of using finite and semi-finite[117] tools in our attempt to comprehend the nature of infinite.

We proceed with clarification on computational strictness as a practical form of mathematical rigor. Going forward we accept *Defintion 25* and *Definition 26* only as definitions of *strictly fair* and *strictly unfair* countable binary strings.

Definition 35.

If $\mathcal{L}_{SEF}$ is an infinite SEF language and $\phi \in \mathcal{L}_{SEF}$ is a formula in this language, then ϕ is called *non-strictly fair* iff there exists a finite prefix $\exists\varphi : |\phi|_\varphi = 1, |\varphi| \in \mathbb{N}$, s.t. $|\varphi|_(= |\varphi|_) = 0$ at any countable moment of scanning time for the input ϕ.

In other words there exists no TM that can decide if ϕ is *strictly fair* without stopping, which makes strict fairness undecidable. In the same manner we cannot check for strict unfairness of the countable formula or if such formula is strictly valid[118]. But we can still design a recursive procedure[119] which is capable of yielding a finite prefix $\varphi : |\phi|_\varphi = 1$ at any moment of

[115] Otherwise we would fail to have exhaustive partition, only $\mathrm{E}_{SEF} \cup \mathcal{F}_{SEF} \cup \lambda(\mathcal{U}_{SEF}) \subseteq \mathcal{E}_{SEF}$

[116] Although the nature of G and P set generating families remains abstract, we assume that their cardinalities are not greater or even strictly less than that of the continuum

[117] countable

[118] e.g. has the right number of brackets

[119] in a dovetailing fashion[17]

countable time[120] and run as a part of another bigger enumerative recursion. Such procedure will simultaneously check for non-strict fairness and unfairness[121]. It turns out that non-strict fairness may still be good enough for some of our purposes in proofs. Alternatively, strict fairness can be induced by construction[122].

Finally, there is another specific case why non-strictness is useful. A generating procedure can be exotic enough, so that we have to deal with the formula of uncountable length $\phi \notin \mathcal{L}_{SEF} : |\phi| > \aleph_0$[123]. In this very special case relying on *Defintion 35* still permits treating of the input as countable.

Lemma 34.

If $\mathcal{L}_{SEF}$ is an infinite SEF language, then $\exists\phi \in \mathcal{L}_{SEF}$ such that there exists no TM that can decide if ϕ is *strictly fair*.

Assume we are in need of a strictly countable fair formula. Instead of trying to spend countable time to scan $\phi \in \mathcal{L}_{SEF}$ for bracket symbols, an alternative approach would be to show that we can generate $\tau \in \mathcal{F}_{SEF}$ such that absence of bracket symbols is guaranteed by the generating procedure itself. As long as SEF τ does not contain a single replicator (hence does not produce infinite output loops of finite patterns), it is strictly fair by definition.

It turns out that the process of generating some strictly countable fair SEF $\phi \in \mathcal{L}_{SEF}$ is closely related to the very concept of randomness. Given that we have a good definition of randomness at hand[124], we can consider using it as generation procedure of $\phi \in \mathcal{L}_{SEF}$. Chaitin Omega number Ω_U number is the halting probability of a universal Chaitin (self-delimiting Turing) machine. Every Omega number is both computably enumerable[125] (the limit of a computable, increasing, converging sequence of rationals) and random (its binary expansion is an algorithmic random sequence).

Theorem 35.

Let $p \in \mathbb{B}$ be an infinite binary expansion (string) of some real number $x \in (0, 1)$, s.t. for some universal Chaitin machine $x = \Omega_U$. If $\exists\phi \in \mathcal{L}_{SEF}$ is a countable SEF, s.t. $p = \pi(\phi)$, then ϕ is also a strictly countable fair SEF $\phi \in \mathcal{F}_{SEF}$.

[120] this pretty much reflects the ideas of computability of the ω-regular expression

[121] It does not mean that we intend to construct a proof in form of a program and let it run without stop. The result of such a program would still be an infinite enumeration. If we don't want to wait forever, our proof should be more resourceful

[122] basically if we know a-priori that SEFs are strictly countable fair

[123] not by our choice but due to some abstract nature of the exotic computational model

[124] such as Chaitin Omega number Ω_U mentioned in [18]

[125] abbreviated as c.e., meaning that it is in RE class

Proof. Chaitin Omega number Ω_U is defined using universal Chaitin machine U. s.t.

$$\Omega_U = \sum_{p \in \Pi:\ p \text{ halts}} 2^{-|p|},$$

where $|p|$ is the length of the binary string p and set $\Pi \subseteq \{0,1\}^\omega$ consists only of allowed[126] programs on U, so that each program is prefix-free (no string is a prefix of another string).

According to Kraft's inequality $\sum_{p \in \Pi} 2^{-|p|} \leq 1$, which shows that Ω_U is also the probability that program p halts.

We know that $|p|$ is countable[127]. Now assume that p is produced from a countable unfair SEF. In that case it will contain at least one replicator. But since each valid program $\forall p \in \Pi$ must be prefix-free and no string is a prefix of another string, p can not contain repetitive prefixes. Which means p must be strictly fair.

□

Alternatively, algorithmic randomness of Ω_U implies equal probability for zeros and ones to occur in p. Assume $p = \pi(\phi)$ and ϕ is countable unfair SEF, so that p contains infinite repetitions of some finite "(τ)" as a replicator (as by definition of strictly countable unfair SEF). Not only infinite repetitions of finite τ can change the distribution of probabilities to being non-random. It turns out that replicators can be well-compressed contradicting the algorithmic randomness of Ω_U.

In other words, every Chaitin Omega number is also a strictly countable fair SEF. However there are simpler approaches to encode countable binary string in prefix-free manner or without even finite substring repetitions. In that sense, the requirement for a countable string to be strictly fair is much weaker than algorithmic randomness.

Definition 36.

Consider a list $\mathcal{A}_\alpha$ of infinite sequences containing permutations of natural numbers (without repetitions), where $\alpha \in I$ is some index. Meaning that first entry of the list is $\mathcal{A}_1 = (a_0, a_1, a_2, a_3, a_4, ..) = (1, 2, 3, 4, 5, ..)$, second is obtained as permutation of indexes as in $\mathcal{A}_1 = (a_1, a_0, a_2, a_3, a_4, ..) = (2, 1, 3, 4, 5, ..)$ and so on. So that $\mathcal{A}$ contains infinitely many permutations of the infinite sequence with natural numbers[128].

Now if we can encode each entry of the list as SEFs $\Psi := \{\psi : \psi =$

[126] specially encoded

[127] by given and from the fact that each Chaitin Omega number is c.e.

[128] somewhat by analogy with Baire space in set theory, but no repetitions

$encode(\mathcal{A}_\alpha)\}$ in such a form that we write only a short binary expansion[129] of each natural number, then the result of such encoding can be rewritten as SEF $\psi : \psi \in \Psi, \Psi \subset \mathcal{L}_{SEF}$, s.t. : $\psi :=$ "..$[a_i][a_{i+1}][a_{i+2}]$..", where $\mathcal{A}_\alpha = (.., a_i, a_{i+1}, a_{i+2}, ..), i \in \mathbb{N}$. Then such list of SEFs Ψ is called SEF permutation list of natural numbers.

Lemma 36.

If Ψ is a SEF permutation list of natural numbers, then $\Psi \subset \mathcal{F}_{SEF}$ and each SEF $\forall \psi \in \Psi : \psi$ is strictly countable fair.

Proof. Proof is by construction. Each $\psi \in \Psi$ is encoded as prefix-free string and hence does not contain any replicators. □

Knowing that each SEF in Ψ is strictly countable fair by construction and $|\Psi| = \aleph_0$, we can as well treat them as non-strictly fair as they don't need any actual computational check. We would now try to use them and construct an alternative proof for *Theorem 33*, but this time we will use lists of much finer cardinalities.

In the alternative proof for $\neg CH$ case, we will not use c-slashing and its GCGA construction[130], but only re-use the configuration for continuous access scheme. It turns out that it can be sufficient to work with much finer sets.

Theorem 37 - Weaker-case Theorem for $\neg CH$ (WNCH).

If $\Psi \subset \mathcal{F}_{SEF}, |\Psi| = \aleph_0$, then $\exists U'_{SEF} \subset \mathcal{U}_{SEF}$, s.t. $|\Psi| < |U'_{SEF}|$ and $\aleph_0 < |U'_{SEF}| < \mathfrak{c}$.

Proof. Instead of computing binary expansions of SEFs, we would work with their values directly[131] as arguments or references to the subset of the domain of the production function $\pi(\mathcal{L}_{SEF})$. We start by arranging a table with Ψ as a list of countable fair formulas $\Psi \subset \mathcal{F}_{SEF}, \mathcal{F}_{SEF} \subset \mathcal{L}_{SEF}$ on the LH.

1. Let $U_{SEF} \subset \mathcal{U}_{SEF}$ be the list of countable unfair that we want to access via value-states on the RH. The setting is as in continuous access scheme[132].

2. We did not apply any slashing on the LH, so $|\Psi| = \aleph_0$. This means we can choose any prefix $\psi \in \Psi$ in countable time and try to access values matched on the RH.

[129] if $n \in \mathbb{N}$ and SEF s := "$[n]$", then its binary expansion $b := \pi(s) = \pi(\text{"}[n]\text{"})$ has length at most $log(n)$, i.e. $|b| \leq log(n)$

[130] as we did in *Theorem 33*

[131] to illustrate their computability in countable time

[132] see *Definition 22* and *Theorem 13* for finite L_{SEF} and $\delta = 1$

3. Next, for each $\forall\psi \in \Psi$ we will always iterate over the same list of all finite unfair formulas E_{SEF} which we have chosen to put on the RH as value-states[133].

4. As the result of employed algorithmic access we can approximate $U_{SEF} \subset \mathcal{U}_{SEF}$ for any RH as SEF concatenation[134].

5. Assume that $|U_{SEF}| = |\Psi \times \mathrm{E}_{SEF}|$. Given the countable time of our access or the cardinalities in the product we get $|U_{SEF}| = \aleph_0$.

6. Now let us show that there exists an surjection $\nu : U'_{SEF} \to U_{SEF}$, s.t. $U'_{SEF} \subset \mathcal{U}_{SEF}$ and $U_{SEF} < U'_{SEF}$[135].

7. The surjection $\nu : U'_{SEF} \to U_{SEF}$ can be constructed in countable time by taking every countable fair part $\psi \in \Psi$, and generating two formulas instead of one to construct the domain of ν U'_{SEF}[136]: take original $\phi \in U_{SEF}$ as is "ϕ" plus another one wrapped into a replicator "(ϕ)". Note that there always exists "(ϕ)" for every $\phi \in U_{SEF}$ since each ϕ has form of $A \cdot \overline{b}$.

8. To construct U'_{SEF} with $|U'_{SEF}| > \aleph_0$, we introduce a systematic method for generating new formulas. For each $\phi \in U_{SEF}$, apply a replicative process that generates exponentially many variants. Specifically, for any ϕ, construct formulas ϕ_n for $n \in \mathbb{N}$ by appending a sequence $1^n \cdot 0$ as a suffix, where 1^n is a repetition of 1 n-times followed by 0. This ensures a countably infinite family of new formulas is generated for each ϕ, effectively multiplying the cardinality.

9. Since the above step produces exponentially many new formulas for each ϕ, the cardinality of U'_{SEF} exceeds $\aleph_0$ while still being enumerable within the constraints of $\aleph_0$-time dovetailing. Importantly, this replication method retains a bijection with $\mathbb{N} \times \mathbb{N}$ on the RH, confirming that we do not break the requirement of computability in countable time[137].

10. Again, note that we cannot print out in countable time[138] all the new RH list U'_{SEF} as we arrive at $|U_{SEF}| < |U'_{SEF}|$ and $\aleph_0 < |U'_{SEF}|$.

[133] notice that like this we still keep enumeration under the countable time in dovetailing fashion

[134] see countable unfair from $A \cdot \overline{b}$ in concatenation table - fig. 6

[135] Alternative way to continue the proof would be to do binary expansion on the RH and apply GCGA as in *Theorem 10* to obtain a different $U'_{SEF} = GCGA(U_{SEF})$. Like this we arrive at $\aleph_0 < |U'_{SEF}|$. This binary expansion step we cannot do in countable time. But we can approximate it by partial access - eventually at a limit, somewhat similar how one would approximate computation of real or even Chaitin numbers

[136] alternative approach is to take original "ψ" from LH and wrap it into replicator ones "(ψ)" and twice "$((\psi))$"

[137] as we don't expand the table

[138] only approximate a countable fragment via partial algorithmic access

11. Since $|\mathcal{L}_{SEF}| = \mathfrak{c}$ (see *Theorem 24*) and $U'_{SEF} \subset \mathcal{U}_{SEF} \subset \mathcal{L}_{SEF}$, we observe that $|U'_{SEF}| \leq \mathfrak{c}$. We can show that $|U'_{SEF}| \neq \mathfrak{c}$ by construction from $U'_{SEF} \cap \mathcal{F}_{SEF} = \emptyset, \mathcal{F}_{SEF} \subset \mathcal{L}_{SEF}$. Hence, $\aleph_0 < |U'_{SEF}| < \mathfrak{c}$.

□

The above proof is a simplified version of previous $\neg CH$ case[139]. We also argue that this proof is algorithmically accessible in countable time[140]. We have mentioned earlier that continuum access schema is undecidable and in fact it allows only partial access to continuum unless shown how it can be properly implemented[141]. So we benefit only from a similar table-like setting with LH and RH listings rather than using CAS fully.

Corollary 38.

If $\mathcal{F}_{SEF}$ is a list of all strictly countable fair SEFs and $\mathcal{U}_{SEF}$ is a list of all strictly countable unfair SEFs, then $|\mathcal{F}_{SEF}| < |\mathcal{U}_{SEF}|$

Proof. Show that there exists an injection $\nu : \mathcal{F}_{SEF} \to \mathcal{U}_{SEF}$, similar as in previous theorem. Inequality is strict since $\mathcal{F}_{SEF} \cap \mathcal{U}_{SEF} = \emptyset$. □

We wanted to dedicate some final remarks in this subsection to enumeration of finite SEFs in E_{SEF} (see fig. 4). As pointed out earlier in *Theorem 12* L_{SEF} is a regular language since it can be recognized by a regular expression.

Indeed, replication operators used in unfair SEFs can be rewritten as ω-regular expressions. As long SEFs are of finite length, they can be recognized by a regular expression themselves. From Chomsky hierarchy we know that regular languages are subset of almost all other languages in the hierarchy[142]. We also know that each ω-regular language can be accepted by a Linear Bounded Automata (LBA).

Still, as mentioned in subsection *Equivalence classes of SEFs*, we are not sure if there exists an efficient algorithm for E_{SEF} enumeration. To produce initial results (as shown in fig. 4) we had to apply a lot of heuristics and formula reductions. It becomes obvious rather quickly that part of the

[139] without relying on results in *Theorem 10*, c-slashing and (disjoint) equivalence classes in complete cover of $\mathcal{E}_{SEF}$ partition, but we may still require the choice function to be able to index over all reals which may imply ZF+AC[6]

[140] Unfortunately we don't provide a formal proof for showing that the *Theorem 37* can be achieved also computationally by running on ω-automaton or similar (non-DC) TM formalism. But one not very straight-forward way to show this would be to use Church-Turing thesis to identify pointwise equivalent program or formula as a point on $x \in \mathbb{R}$. Then show that x is c.e.

[141] for example, by assuming

[142] can be accepted by a Finite State Machine (FSM)

enumeration task to find minimum regular expressions can be as difficult as PSPACE-complete[143].

6.2 The Paradox of Finality

Up to this point, our exploration has predominantly centered on the computational approach to infinity. In the subsequent subsection, we will pivot our attention to the foundational principles of Set Theory when discussing infinity. Specifically, we aim to explore how the concept of infinite collections of strings, which we have treated computationally, translates into Set Theory. What does it mean to formalize these collections as sets? How will this shift change our understanding of infinity?

Before introducing the theorem, it is important to address a key distinction between how we think about infinite sets in a computational sense versus within Set Theory. In a computational or algorithmic framework, infinite sets are often treated as though they are always dynamically constructed, such as through iterative processes like string concatenation. However, this approach can lead to paradoxes when self-referencing or circular dependencies are involved. For example, attempting to define a self-referencing mapping like $\mu : \mathcal{L}_{SEF} \times \mathcal{U}_{SEF} \to \mathcal{U}_{SEF}$ — where SEF denotes String Enumeration Formula (see *5 - String Enumeration Formulas*) — results in contradictions if we assume these infinite sets are always dynamically constructed. Such constructions are never "final" but remain perpetually "under construction," leading to inconsistencies. In contrast, Set Theory finalizes these sets through a static, step-by-step cumulative hierarchy V, as defined in Zermelo-Fraenkel (ZF) Set Theory (see *8.2 - Expected Length of Strings*).

The *finality paradox* arises when sets reference each other circularly. Both sets attempt to construct subsets of each other simultaneously, leading to confusion about whether the sets are well-defined. For instance, if we restrict the mapping to a small subset of SEFs, it avoids contradiction by becoming an identity mapping. This simplification reveals the limitations of the dynamic approach, which remains perpetually "under construction" and cannot guarantee well-definedness in the presence of circular references.

In contrast, Set Theory adopts a fundamentally different perspective. Sets are not constructed dynamically, contrary to how the running programs might behave, but are instead defined to exist axiomatically. In this framework, sets are fully established and governed by the rules of Zermelo-Fraenkel (ZF) Set Theory from the very beginning. This avoids the paradoxes of self-reference and circular dependencies that arise in computational models, ensuring that infinite sets are rigorously and consistently defined.

This axiomatic approach allows us to rigorously define infinite sets and discuss them without falling into contradictions. Sets, like any mathematical

[143] Just as the problem of finding an equivalent or minimal FSM, equivalence problem for regular expressions is known to be PSPACE-complete[11]

objects, are constructed using axioms. This simple yet the central idea is a corner stone of Set Theory (and arguably the whole of mathematics).

Theorem 39 - Axiomatic Existence.

If a set X exists [144], then it is a member of the cumulative hierarchy V, which is a proper class encompassing all sets in ZF. Formally, if $\exists X : X \in V$, then X is a set in ZF.

The cumulative hierarchy V is constructed in stages[145], indexed by ordinal numbers. Each stage, V_α, consists of all sets that can be formed from sets in earlier stages. Starting with the empty set at V_0, the hierarchy builds successively larger collections of sets using operations such as forming power sets and unions. The entire hierarchy V is the union of all these stages and represents the "universe" of all sets that can exist within ZF. This structure ensures that every set is grounded in a well-defined and logically consistent framework.

Proof. This follows directly from the axioms of Zermelo-Fraenkel Set Theory [5, 6]. Specifically, the Axiom of Foundation ensures that every set is contained within a well-defined stage of the cumulative hierarchy V. Additionally, the Axiom of Replacement guarantees that sets constructed via definable functions remain within V. Consequently, any set X that exists according to ZF is necessarily an element of V. □

Another way to interpret the above statement is to emphasize that sets are not dynamically constructed or accessed through a computational process but instead already exist axiomatically within the universe of sets. This means that when we refer to a set, we are not invoking a process of step-by-step construction, as might be done in a computational framework. Instead, the set is fully established and exists independently of any procedural creation. The purpose of stating this is to underscore the absence of an "on-the-fly" construction process for sets, which distinguishes the axiomatic nature of set existence in ZF from any computational model[146].

Equivalent statements hold in other axiomatic set theories, such as Neumann-Bernays-Gödel (NBG) or Morse-Kelley (MK) Set Theory. Ultimately, it is a matter of relative consistency of the model and the encoding of formulas. Usually, everything, including formulas, can be encoded as sets. Depending on the need, some statements can be represented as Boolean-valued algebras

[144] For example, within the framework of Zermelo-Fraenkel (ZF) Set Theory—see the next subsection for an explanation of how the axiom of choice or alternative weaker conditions relate to this theorem

[145] Also see *8.2 - Expected Length of Strings*

[146] This should also not be confused with constructing sets that are used for the inner models of ZF

or mapped from binary strings into sets. One can imagine this as referencing a set from an infinite library, where each book represents a set already defined by the Set Theory axioms.

6.3 Importance of choice

Recall the definition of terms *choice function*, *well-ordering* and the related theorem as according to [6].

Axiom - Axiom of Choice (AC).

Every family of nonempty sets has a *choice function*.

If S is a family of sets and $\emptyset \notin S$, then a *choice function* for S is a function c on S such that $c(X) \in X$ for every $X \in S$[147].

Definition 37 - Well-Ordering.

A linear ordering $<$ of a set P is a *well-ordering* if every nonempty subset of P has a least element[148].

Theorem 40 - Zermelo's Well-Ordering Theorem (WOT).

Every set can be well-ordered.

The question of well-ordering of continuous line of reals is intimately interwoven with the CH[149]. Today we know that:

1. $AC \iff WOT$
2. CH is independent of ZFC
3. AC is independent of ZF

In fact, well-ordering of sets is something that cannot be shown in ZF alone. Both statements in (1) are equivalent since it can be shown [9] that AC and WOT imply each other[150]. Furthermore, even relying on weaker statements[6] such as CUT requires the use of at least some weaker version of choice axiom in addition to ZF.

[147] or equivalently, when using $S = (X_i)_{i \in I}$, where I is some index set, $c(X_i) \mapsto X_i, \forall i \in I$

[148] such "least" element also can be called "first"

[149] This is also one of the reasons why D. Hilbert has linked both questions in his statement of the famous first problem[9].

[150] Also see Hartogs's Theorem and [19]

Axiom - Axiom of Countable Choice (CC).

For each sequence $(X_n)_{n\in\mathbb{N}}$ of non–empty sets X_n, the product set $\prod_{n\in\mathbb{N}} X_n$ is non–empty[151].

Theorem 41 - Countable Union Theorem (CUT).

The union of a countable family of countable sets is countable.

Specifically, using CC implies CUT[152]. "It turns out that the Axiom of Choice is independent of the other axioms of set theory and that many mathematical theorems are unprovable in ZF without AC"[6].

Let us consider yet another candidate for a weaker choice condition, which we would also apply later. In order to obtain a notion of such a choice axiom, we will require a definition of the relaxed binary relation.

Definition 38.

if K is a binary *consensus*[153] relation on a nonempty class X, and if for every $x \in X$ there exists $y \in X$ s.t. xKy, then the following holds true for K:

1. *irreflexivity* - $\forall x \in X : \neg(xKx)$
2. *symmetry* - $\forall x, y \in X : xKy \to yKx$
   and inversely, $\forall x, y \in X : \neg(xKy) \to \neg(yKx)$
3. *transitivity* - $\forall x, y, z \in X : xKy \wedge yKz \to xKz$
   and inversely, $\forall x, y, z \in X : \neg(xKy) \wedge \neg(yKz) \to \neg(xKz)$

Going forward we will mostly use $\wr$ as the notation for consensus relation on classes and sets. The above definition is almost similar to the equivalence relation except for reflexivity. At first look, consensus relation seems intensional by being defined on the class. However, after a closer look, consensus is clearly extensional in terms of ϵ-relation. In fact, it is able to form quotient sets just like its dual under the compliment. Obviously there must be a simple explanation for this. Our newly defined relation is also known as "is not equal to" binary relation, so that $\sim=\wr^{\complement}$[154].

[151] Note that, here or in general - given some index set I, the elements of the product set $\prod_{i\in I} X_i$ are choice functions $(x_i)_{i\in I}$, namely, functions $x : I \to \bigcup_{i\in I} X_i$ pointing from index to each element of (X_n) - i.e. satisfying $x(i) = x_i \in X_i$ for each $i \in I$[9]

[152] Please see p. 23 as well as *Diagram 2.21.* on p. 18 in [9]

[153] as *to agree on something*

[154] where notation $X^{\complement}$ means that $X^{\complement}$ is a complement of set X

Axiom - Consensus By Choice (CB).

Every family of nonempty sets has a consensus relation.

In simple words, by assuming CB - we can index any nonempty set X, so that the only information we learn about X is that it contains some subsets without any additional knowledge of how to order them. At first, it seems tautological with the membership relation, but it is sufficient to index sets of greater cardinality such as $|X| > \aleph_0$. The best way to do this is to break X into quotient sets $[X] = X/\between$ which can behave almost similar or dual to equivalence classes. With the aim of proving consistency, such quotients can be described by formulas as consensus classes. Now let us still check if our newly obtained notion CB is indeed a strictly weaker condition than AC.

Lemma 42.

Equivalent are:

1. AC
2. WOT
3. CB

Proof. We will proceed as following:

1. we know that $AC \iff WOT$
2. show that $WOT \implies CB$
3. show that $CB \implies WOT$

$WOT \implies CB$: Assuming WOT, there is a well-ordering $<$ of X. One can define a relation $K \subset X \times X$ s.t. it is an equivalence relation $\sim$ on X (when $S_i, S_j \in X, \forall i, j \in I$ and $\between = \sim^{\complement}$ so that $S_i \neq S_j \implies S_i \between S_j$).

$CB \implies WOT$: Assuming CB, we have a consensus relation $\between$ on set X and can define quotient sets $[X] = X/\between$, then $\forall i, j \in I : \exists S_i, S_j \in [X]$, s.t. $S_i \between S_j \implies S_i \neq S_j$. Like this one can define cardinality $|X| = |I|$ by using the fact that we know of all unique subsets of X. Finally, we can define a well-order relation $K \subset X \times X$ by applying trichotomy on X. Like that $\forall i, j \in I : \exists S_i, S_j \in [X]$, s.t. $|S_i| < |S_j| \implies S_i K S_j$. In fact, we can do so for all sets (or on those, where consensus relation is provided). Hence, all such sets can be well-ordered. □

Note that in the above proof we eventually relied on the definition of cardinality in the broader sense by using equivalence classes. This permitted

existence of indexing sets and injections, which allowed to use the Law of Trichotomy. The law is usually defined for linear order, stating that exactly one of the conditions is true: $a < b$, $a = b$ or $a < b$. It was shown by Hartog in 1915 [19] that applying trichotomy to alephs implies AC. So, unsurprisingly, $CB \iff AC$. There exists enough of known alternative forms of AC[155].

Thanks to Goedel's work on Constructible Sets[156] we know that CH holds in the inner model of ZFC in ZF ($V = L$). Although, even assuming consistency of ZFC, Goedel's Second Incompleteness Theorem (GSIT) shows that $Con(ZFC)$ is unprovable in ZFC. And, thanks to P. Cohen's method of forcing, we know about (2) independence of CH from ZFC and (3) independence of AC from ZF. Which means that AC and CH cannot be proven by means of ZF alone.

Following intuition outlined in [9], another way of looking at CH is that it is a much stronger condition than AC. Let us also recall related definitions.

Definition 39 - Generalized Continuum Hypothesis (GCH).

GCH, states that for infinite cardinals $\mathfrak{a}$ and $\mathfrak{b}$ the inequalities $\mathfrak{a} \leq \mathfrak{b} < 2^{\mathfrak{a}}$ imply $\mathfrak{a} = \mathfrak{b}$.

Definition 40 - Aleph–Hypothesis (AH).

AH, states that $2^{\aleph_\alpha} = \aleph_{\alpha+1}$ for each ordinal α.

Note that $GCH \implies AC$, $AH \implies AC$ and $GCH \impliedby AH$. But as we discussed earlier through this paper $\neg CH \implies \neg(GCH \wedge AH)$[157].

Lastly, we only touched briefly on the implications of employing the axiom of choice[158]. Fully exploring this would indeed be a monumental task on its own. Instead, our focus will now shift to another, yet crucial principle for this paper called *dependent choice*.

6.4 Dependent choice

Let us consider two definitions of the axiom of the *dependent choice*. First one would be in the fashion of category theory [24]. Note that in the original definition due to [24] the domain of the sequence are natural numbers.

[155] According to [9, 20] there is a list of 383 such "forms". Unfortunately, the consequence of either keeping or leaving out AC can lead directly towards various forms of *disaster* (to quote [9]).

[156] *Theorem 13.18* in [6]

[157] although CH is not equivalent to AH

[158] We warmly encourage the reader to delve into the intricacies of the consequences of the axiom of choice in [21–23]

But we slightly adjust it by saying that, without loss of generality, we do understand DC in the way that the length of the resulting sequence is at most $\aleph_0$ and hence use ordinals as the index set of the sequence[159].

Definition 41 - Dependent Choice.

A relation $R : X \to Y$ is total iff every element of X is related to at least one element of Y. If X is a nonempty set and R is a total binary relation on X, then there exists a sequence $x : I \to X$, s.t. $\forall \alpha \in I : (x_\alpha, x_{\alpha+1}) \in R, I \subset Ord, |I| = \aleph_0$.

It is called dependent choice because the available choices for $x_{\alpha+1}$ depend on the choice of x_α made at the previous stage. It is strictly weaker than AC, but strictly stronger than CC.

The second one would be more generic and due to [25].

Definition 42 - Dependent Choice (DC_κ).

Let κ be an infinite cardinal. We denote by DC_κ the following statement: Every κ-closed tree without maximal elements has a chain of order type κ. We use $DC_{<\kappa}$ to abbreviate $\lambda < \kappa, DC_\lambda$, and for $\kappa = \aleph_0$ we simply write DC.

Both definitions are equivalent if $\kappa = \aleph_0$. This also follows from the next lemma [6, 25].

Lemma 43.

The following is equivalent:

1. DC.
2. Every pruned tree with ω levels has a branch.
3. The Löwenheim–Skolem theorem for countable languages: every structure in a countable language has a countable elementary submodel.
4. For every $\alpha \geq \omega$ and every countable $A \subseteq V_\alpha$ there is a countable elementary submodel M of V_α such that $A \subseteq M$.
5. For every $\alpha \geq \omega$ there is a countable elementary submodel $M \prec (V_\alpha, \in)$.

[159]Not to be lost in nomenclature, but we rely on the definition of alephs that $\aleph_0 = \omega$, thus the index set I can explicitly exceed the ordinality, but not the cardinality of the domain of natural numbers

Note that to better understand points (4) and (5) and the details about V_α please see in [25] *Definition 2.1* of Hereditary sets $H(\kappa)$ and their properties, including relation to Von Neuemann Cummulative hierarchy V_κ. Namely, $H(\kappa)$ is a transitive set of height κ and when κ is a strong limit cardinal, then $H(\kappa) = V_\kappa$.

Next to CC, among other basic consequences of DC we have: CUT, the regularity of ω_1, etc[160].

Now let us propose a new axiom but very similar in spirit with DC using subsequences.

Definition 43 - Subsequence.

Let I be an index set and $k : I \to I$ an increasing function. If $(x_\alpha)_{\alpha \in I}$ is a sequence with range over X, i.e. $x : I \to X$, then another sequence $(y_\gamma)_{\gamma \in I}$ defined by $y_\gamma := x_{k(\alpha)}$ is called a subsequence. Moreover, if the index set is clear from the context we just write $(x_\alpha)_\alpha$ for sequences.

Thus, one can also write $(x_{k(\alpha)})_\alpha$ to denote the subsequence of $(x_\alpha)_\alpha$. This is great, but we also want such subsequences to be more restricted and not to have any gaps, so that we can match them with our definition of a string. To achieve this, we want the function k to map to a *contiguous* subset of the index set I.

Let's clarify the notion of "contiguous"[161] for ordinals Ord[162].

Definition 44 - Ordinal contiguity.

If $I \subset Ord$ is an ordinal, then a subset $J \subseteq I$ is contiguous if for every $\alpha, \beta \in J$ with $\alpha < \beta$, all the ordinals between α and β are also in J.

With this in mind, we can define a function $k : I \to I$ to be contiguous if its range is a contiguous subset of I.

Definition 45 - Contiguous function (on ordinals).

Let $k : I \to I$ be an increasing function. The function k is called *contiguous* in I if the range of k, $ran(k)$, is a contiguous subset of I. That is, for every $\alpha, \beta \in ran(k)$ with $\alpha < \beta$, all the ordinals (or natural numbers, if $I = \mathbb{N}$) between α and β are also in $ran(k)$.

Observation 1.

---

[160] See *Theorems 3.2 and 3.3* in [25]. Also consult [21, 22] for further reading on the DC consequences.

[161] Not to be confused with contiguity of sequence of measures in probability theory.

[162] Obviously for natural numbers it will be the same.

Note that "contiguous" and "dense" are not the same, although they might seem related in some contexts. Let's clarify the distinction:

- **Contiguous**:
  - As defined above, a function $k : I \to I$ is contiguous if its range is a contiguous subset of I. This means that if α and β are in the range of k and $\alpha < \beta$, then every element between α and β is also in the range of k.
  - In the context of natural numbers, this means that there are no "gaps" in the sequence. For instance, the subsets $\{1, 2, 3\}$ and $\{4, 5, 6, 7\}$ are contiguous, but $\{1, 3, 4\}$ is not.
- **Dense**:
  - A set S is dense in a topological space T if every point in T is either in S or is a limit point of S. In simpler terms, between any two distinct points of S, there is another point of S.
  - In the context of the real numbers $\mathbb{R}$, the rational numbers $\mathbb{Q}$ are dense because between any two real numbers, there exists a rational number. This illustrates the concept of density by showing that no matter how close two real numbers are, a rational number can always be found between them.

For instance, the rational numbers $\mathbb{Q}$ are dense in the real numbers $\mathbb{R}$ because between any two rational numbers, there's another rational number. However, the rational numbers are not contiguous in $\mathbb{R}$ because there are "gaps" (i.e., irrational numbers) between them.

One hand there are special topologies when any interval (or more generally, any connected subset) is trivially dense in itself. For example, the interval $[1, 2]$ is dense in the topological space that consists of just the interval $[1, 2]$ because between any two points in this space, there are points of the interval itself. So, in that particular case, the provided notion of *contiguity* can be seen as strictly stronger than the property of being only *dense*.

But on the other hand and in general, this is not true. Meaning that *contiguous* and *dense* do not necessarily imply each other. For example, contiguous or every (topologically) connected interval in $\mathbb{R}$ is not necessarily dense in $\mathbb{R}$ itself.

Definition 46 - Strings and substrings from sequences.

Let $(x_\alpha)_{\alpha \in I}$ be a sequence with index set I and range over X. In this context we will also refer to this sequence simply as *string* x. If $k : I \to I$ is a contiguous function, then subsequence $(y_\gamma)_{\gamma \in I} := (x_{k(\alpha)})_\alpha$ is a called a *substring* y of the *string* x. We note this as $y \; \varepsilon \; x$.

Later we plan to define a much more detailed theory around notions of a string and substring[163]. But for now the following additional definitions will suffice.

Definition 47 - Concatenation of strings from sequences.

Let $x = (x_\alpha)_{\alpha \in I}$ and $y = (y_\beta)_{\beta \in J}$ be two strings with index sets I and J respectively. The concatenation of x and y, denoted by $x \cdot y$, is the string $z = (z_\gamma)_{\gamma \in K}$ where $K = I \cup \{\sup(I) + \beta \mid \beta \in J\}$.

Specifically, for each $\gamma \in K$:

$$z_\gamma = \begin{cases} x_\alpha & \text{if } \gamma = \alpha \text{ for some } \alpha \in I \\ y_\beta & \text{if } \gamma = \sup(I) + \beta \text{ for some } \beta \in J \end{cases}$$

In this definition, $\sup(I)$ represents the supremum of the index set I. The term $\sup(I) + \beta$ ensures that the indices of the concatenated string z are unique and ordered.

Definition 48 - Substring counting in sequences.

Let $x = (x_\alpha)_{\alpha \in I}$ be a string and $y = (y_\beta)_{\beta \in J}$ be a substring. The count of occurrences of y in x is denoted by $|x|_y$. Formally, $|x|_y$ is the number of distinct contiguous functions $k : J \to I$ such that for all $\beta \in J$, $y_\beta = x_{k(\beta)}$.

Note that, by the definition of contiguity, k is inherently order-preserving; that is, if $\beta_1 < \beta_2$, then $k(\beta_1) < k(\beta_2)$. In the above definition, y does not have to appear contiguously in x. However, if y is contiguously repeated in x at least twice and up to countably many times, we can define such behavior with a replication operator.

Definition 49 - Replication of strings.

We call $\alpha \in Ord$ the order of replication in the following sense. Let z be contiguously repeating subsequence (substring) in y, so that y is a periodic sequence with period z. Then, $y =$ "$(z)^\alpha$", where "$(z)^\alpha$" is called replicator, iff $|y|_z = \alpha$.

Note that if the order of replication is at least countable, and it is informative enough for the discussion, then we abuse the notation by simply writing $y =$ "(z)", assuming that omitted $\alpha \geq \omega$.

Finally, we come to our new axiom.

[163] Namely, see String Theory in *Definition 116*

Definition 50 - Countable Aperiodicity (CA).

Let X be a nonempty set and R be a total binary relation on X. Given a countable index set $I \subset Ord$ with $|I| = \aleph_0$, there exists a sequence $x : I \to X$ such that:

1. For all $\alpha \in I$ with $\alpha + 1 \in I$, $(x_\alpha, x_{\alpha+1}) \in R$.
2. For any contiguous subsequence $y : J \to X$ (with $J \subset I$) of the form $y = (x_{k(\alpha)})_{\alpha \in J}$, where $k : J \to I$ is a contiguous index mapping, there does not exist another contiguous subsequence $z : K \to X$ (with $K \subset J$) such that y can be expressed as a countable concatenation of z.

We say that x is *countably aperiodical* or *fair*.

In the context of the SEF theory, when sequences are interpreted as strings, the CA axiom posits the existence of a sequence (or equivalently, a string) $(x_\alpha)_\alpha$ that does not contain any replicator $y =$ "(z)". Basically, if $X = \{0, 1\}$, the CA axiom implies the statement: *"There exists a fair countable binary string of at least ω length"*.

Definition 51 - Fair String.

A countable string x is *fair* if no substring y of x can be expressed as a countable repetition of another substring z.

Definition 52 - Unfair String.

A countable string x is *unfair* if it isn't fair. A fair string contains no unfair substrings.

It should be emphasized that the presence of CA may initially appear superfluous if one assumes that fair strings can be inherently formulated solely within the ZF axioms. However, this assumption is not completely accurate. Indeed, CA being equivalent to DC clarifies the crucial role and significance of DC in the theoretical development concerning fair countable sequences.

Theorem 44 - $DC \iff CC + CA \iff GCTA(x)$.

The following statements are equivalent:

1. DC
2. $CC + CA$

3. $GCTA(x)$: GCTA table can be constructed by scanning a fair binary input string x.

Proof. (1) $\rightarrow$ (2): DC is known [9] to be strictly stronger than CC. Observe that CA is semantically equivalent to the statement that "there exists a fair string of at least ω length". To prove by contradiction we assume the opposite. Then, we play a game that decides between whether a countable binary string x is fair or not. It turns out, that DC implies that such a game is always determined and one of the players has always a winning strategy. Hence, CA follows from the *Corollary 48* and $DC \implies CA$.
(2) $\rightarrow$ (3): Recall that left hand (LH) of the GCTA table is a list or a matrix of infinite contexts. The table does not have a fixed start, so we pick some arbitrary index $j \in \mathbb{Z}$ and copy the left part of the countable string as a first element of the string. Rows are called transformation contexts and indexed with $i \in \mathbb{Z}$. For each row (or for each next context entry) the value of the string is obtained by a shifting the previous row by a single bit to the left and appending one bit (at $j + i$) to the right. This constructs a list of countably many shifted and copied strings based on the input string x. Such process is called scanning[164]. It is clear that the construction process invokes CC. Now, given that fair string exists as follows from (2), we can pick x to be a fair string and have that $CC + CA \implies GCTA(x)$.
(3) $\rightarrow$ (1): Recall that by *Definition 41* the DC axiom implies existence of total binary relation R, s.t. $\forall \alpha < \omega_1 : (x_\alpha, x_{\alpha+1}) \in R$. Observe that if x is a fair string, then $GCTA(x)$ will represent such R. Indeed, the construction of the main table (LH) is already described in the previous step called the scanning process. If one adds binary representation of the row index for each row on the right hand (RH) as finite binary string, then one can map each context on the LH to x_α and each state on the RH to $x_{\alpha+1}$. Like this $GCTA(x) \implies DC$. □

To clarify the last point of the above proof, here we have worked with $GCTA(x)$ with height $\kappa = \omega$, rather than $GCGA(x)$ when the height of the table in "expanded configuration" (post-generation, when new rows are added to the table) is at least $\kappa = \omega_1$. In that sense and even intuitively[165], $GCTA(x)$ resembles and is equivalent to DC_ω, when $expanded(GCGA(x))$ is equivalent to DC_{ω_1}.

As we wrap up this subsection, we present a pivotal lemma that sets a proper accord on the discussion of the existence of fair strings.

Lemma 45 - Uncountability of fairs.

---

[164] This is discussed in greater detail earlier in the paper, starting with *Generalized Context Transformation Algorithm - Subsection 4.2*
[165] Please illustrations in *Continuum access scheme - Subsection 5.6*

Let F be a large enough family of pairwise distinct fair strings, i.e. $F = \{x : x \in \{0,1\}^\omega \wedge x \text{ is fair }\}$. Then, there exists a subset of F with uncountable cardinality, i.e., $\exists F' \subseteq F : |F'| > \aleph_0$.

Proof. Assume $ZF + DC$. Construct $GCTA(x)$ table T (LH side) by scanning a fair input string x (as described in the proof of the *Theorem 44*). We complete the proof by applying cantor diagonal argument (slashing) to produce x' and observe that:

1. x' is also fair (by being almost adjoint[166] with countably many shifted copies of x)
2. x' is not present in the table $T \implies F' = T \cup \{x'\}$ and $|F'| > \aleph_0$

□

Alternatively, one can proceed with the following ending of the above proof. Choose countable binary string y as a single column for the RH, s.t. there is only finite number of arbitrary distributed ones $|y|_1 < \omega$. Use T as an input parameter for $GCGA$ algorithm. Basically, here we describe a setup for *c-slashing* (see *Continuo Cantor Argument - Subsection 5.9*), which can be noted as $T' := [T]||y$, where T' will be an *expanded*(T) and T is a $GCGA(x)$ table. If $F' = T'$, then the rest of the proof follows.

Curiously enough, a related but slightly different and more general construction than obtained T' is called *Ulam matrix* (for example see [6] for the definition of the concept and a proof of *Ulam Theorem* in ZFC). We will revisit this observation much later in the paper.

6.5 Partial Order by Reverse Contiguity

We will expand on the idea of contiguity by defining a partial order[167] over a set of countable binary strings. This approach aims to categorize strings based on their structural properties, especially focusing on the notion of fairness and repetitiveness in their composition.

Definition 53 - Perfectly Fair String.

A countable binary string x is called *perfectly fair* if it does not contain any countable substring y that can be expressed as the concatenation of at least two identical substrings, i.e., $\nexists y : y = z \cdot z$ where $|y| = \aleph_0$.

This definition captures the essence of a perfectly fair string as one that avoids any form of infinitely repetitive substructure.

One can immediately think of the category of all the other countable strings which are still fair but not perfectly fair.

[166] See *Definition 57*

[167] See *Definition 69*

Definition 54 - Imperfectly Fair String.

A countable fair string is called *imperfect* if it is not perfectly fair.

Provision of the above subcategories will come handy in getting a deeper understanding of how to construct an internal partial order within fair countable strings by using the same idea of contiguity.

Given these definitions, it is evident that a perfectly fair string is significantly less contiguous than any imperfect string. This is primarily because any imperfect string will inherently possess a more repetitive structure, consisting of substrings that repeat themselves, are adjacent, or are closely aligned. This leads us to further scrutinize the interplay between repetition and proximity of substrings.

We can assert that a string $y = x \cdot z \cdot x$ is less contiguous than a string $w = x \cdot x$, or even more so than $v = x \cdot x \cdot z$. This highlights our interest in repetitive or similar occurrences of substrings being more tightly or closely intertwined. Here by being more contiguous we literally mean that for any SEF variations build from combination of countable infinite string segments x, z, for example such as $y = x \cdot z \cdot x$ or $y = x \cdot x \cdot z$ or $y = z \cdot x \cdot x$, we have that any contained (potentially non-repetitive or fair) segment z tends to be smaller, i.e. $|z| \to min$. Such a notion can be effectively captured by defining it as a partial order, which necessitates additional definitions for SEF kind.

Definition 55 - Partition of SEFs by kind.

Let x be a countable binary string and s_x a corresponding String Enumeration Formula (SEF), such that $\pi(s_x) = x$. The *SEF kind* of string is defined as follows:

1. *Unfair Finite (UF)*: SEF has finite length $|s_x| < \aleph_0$ and x does not contain any (perfectly or imperfectly) fair substring;
2. *Perfectly Fair (PF)*: SEF has countable length $|s_x| = \aleph_0$ and x is perfectly fair[168];
3. *Imperfectly Fair (IF)*: SEF has countable length $|s_x| = \aleph_0$ and x is imperfectly fair[169];
4. *Unfair Countable (UC)*: SEF has countable length $|s_x| = \aleph_0$ and x contains at least one fair substring.

Briefly, we can also note this as $kind(s_x) := \{UF, PF, IF, UC\}$.

[168] see *Definition 53*
[169] see *Definition 54*

The introduction of the above four SEF kinds - Unfair Finite (UF), Perfectly Fair (PF), Imperfectly Fair (IF), and Unfair Countable (UC) - enables a refined classification of countable binary strings based on the length of the corresponding SEF and presence of infinite repetitions. This classification is key in establishing partial order within infinite countable strings, which we will define for each countable binary string x by looking at its preimage $\pi^{-1}(x)$ and applying the concept of contiguity.

Contiguity here will also quantify the adjacency of repetitive substrings within a binary string. This metric aids in capturing inner partial order that reflects the intricacy of infinite string patterns up to a perfectly fair string, going beyond mere length. The partial order is thus defined by the balance between repetition and uniqueness in a string's structure, offering a structured approach to categorizing infinite binary sequences.

To sum up, comparing countable binary strings by contiguity can be done by implying length, repetition and adjacency of similar (repetitive) segments. Let us state this more precisely.

Definition 56 - Partial Comparison by Contiguity.

Let us assume the following:

(i) ZF and CC;

(ii) x and y be two at most countable binary strings, s.t. $|x| \leq \aleph_0$ and $|y| \leq \aleph_0$;

(iii) $\pi(s_x) = x$ and $\pi(s_y) = y$, where s_x and s_y are corresponding SEFs preimages for x, y;

(iv) $P := \{0, 1, 2, 3\}$ is a set of partial order (p.o.) comparison results;

(v) if $|x|_a \geq 2$, then there exists a function $\mu : \{0,1\}^\omega \times \{0,1\}^\omega \to Ord$ that finds all adjacency segments $Adj_a(x)$ as any substrings between two closest occurrences of substring a, measures the length of those arbitrary connecting segments and is mapped to the lower bound $\mu(a, x) \mapsto q$ of any found length, i.e. $|q| \leq |k| : \forall k \in Adj_a(x)$ or to the shortest adjacency.

(vi) there exists a function that can compare ordinals defined as $cmp : Ord \times Ord \to P$, specifically meaning that for any $\alpha, \beta \in Ord$ we have:

  (a) $cmp(\alpha, \beta) = 3$ iff $\alpha > \beta$

  (b) $cmp(\alpha, \beta) = 2$ iff $\alpha = \beta$

  (c) $cmp(\alpha, \beta) = 1$ iff $\alpha < \beta$

(d) $cmp(\alpha, \beta)$ is never equal to 0 which means *incomparable*

(vii) there exists a function $\rho : \{0,1\}^\omega \times \{0,1\}^\omega \to P$, where $ran(\rho)$ can be respectively interpreted as:

(a) $\rho(x, y) = 3$ iff $x > y$

(b) $\rho(x, y) = 2$ iff $x = y$

(c) $\rho(x, y) = 1$ iff $x < y$

(d) $\rho(x, y) = 0$ iff both strings x and y are *incomparable*

Furthermore, the last function ρ can be used to define p.o. by comparing any two at most countable binary strings x, y according to the following scheme or algorithm[170]:

1. **by trivial equivalence**: if $x = y$, then map $\rho(x, y)$ to 2;

2. **by length**:

   (a) if $|x| \neq |y|$, then map $\rho(x, y)$ to $cmp(|x|, |y|)$;

   (b) if $|s_x| \neq |s_y|$, then map $\rho(x, y)$ to $cmp(|s_x|, |s_y|)$;

3. **by the kind of the corresponding SEF**: if $kind(s_x) \neq kind(s_y)$, then we map the kind of each SEF to the following linear order $UF < PF < IF < UC$, so that $\rho(x, y) \to \{1, 3\}$ accordingly; otherwise:

   (a) if $kind(s_x) = kind(s_y) = UF$, then we map both s_x, s_y SEFs to the reverse of the linear order (so that "(0)" is the formula for the most contiguous and "(1)" for the second most contiguous countable binary string) as enumerated by SEF equivalence classes[171], so that $\rho(x, y) \to \{1, 3\}$ accordingly;

   (b) if $kind(s_x) = kind(s_y) = PF$, then map $\rho(x, y)$ to 0[172];

   (c) if $kind(s_x) = kind(s_y) = IF$, find two substrings a and b, s.t. $|x|_a = \alpha$ and $|y|_b = \beta$ with the highest counts of occurrence[173] α, β respectively, then:

      i. if $\alpha \neq \beta$, then map $\rho(x, y)$ to $cmp(\alpha, \beta)$;

      ii. if $\alpha = \beta$ and $\alpha + \beta \geq 4$, then check for shortest adjacency and map $\rho(x, y)$ to $cmp(\mu(a, x), \mu(b, y))$ iff $\mu(a, x) \neq \mu(b, y)$;

[170] if the step concludes the mapping or the result of the function, then other steps are ignored

[171] as discussed in subsection *Equivalence classes of SEFs* and end of subsection *Computability of SEF languages*, each SEF is a part of the equivalence class, all of which can be enumerated and hence linearly ordered given that $|\mathrm{E}_{SEF}| = \aleph_0$ - also see fig. 4

[172] we are interested only in infinite contiguity and thus ignore finite repetitions of finite substrings

[173] here and further, higher bound is meant

iii. otherwise, map $\rho(x, y)$ to 0[174];

(d) if $kind(s_x) = kind(s_y) = UC$, firstly - reduce s_x and s_y SEFs to s'_x and s'_y by replacing any fair string with the next unique unseen finite binary string that becomes a reference to the replaced infinite fair string[175], secondly - find two first fair substrings[176] a and b, s.t. $|x|_a = \alpha$ and $|y|_b = \beta$ with the highest counts of occurrence α, β respectively, then:

i. if $r := \rho(\pi(s'_x)$ and $\pi(s'_y)), r \in \{1, 3\}$, then map $\rho(x, y)$ to r;

ii. if $kind(\pi^{-1}(a)) = kind(\pi^{-1}(b))$ and $\alpha \neq \beta$, then map $\rho(x, y)$ to $cmp(\alpha, \beta)$;

iii. if $r := \rho(a, b)$ and $r \in \{1, 3\}$, then map $\rho(x, y)$ to r;

iv. remove all occurrences of a from x as x' and b from y as y' and find $r := \rho(x', y')$, then proceed by rule - if $r = 2$ return 0, otherwise return r;

v. finally[177], map $\rho(x, y)$ to 0.

It is immediately obvious that the above definition can be helpful in defining partial order. Nonetheless, we will come back to exploring the notion of contiguity in greater detail only in section *9 In search of String Theory for SEF*.

6.6 Perfect and Fair Games

Theorem 46 - Perfect Determinacy.

A game that decides between whether a countable binary string x is perfectly fair or not is determined.

Proof. We assume $ZF + DC$. The proof goes like this. Both players need to provide a finite number $n \in \mathbb{N}$ each turn. They can do so by benefitting from applying DC every time a new element of the sequence needs to be generated. A number from a previous step is remembered and used to make a dependent choice. This guarantees that one can always make a choice of a new number. It's a game of perfect information. Both players

[174] again, this can be further refined, but we ignore finite repetitions

[175] Basically, if all finite strings in the SEF formula can be mapped to ordinals, so that some $\sigma \in Ord$ is the maximum ordinal, then the first occurrence of a fair string will be replaced by the binary representation of $\sigma + 1$ reference and so on. However, it also means that second and following occurrences of the same fair will get the same reference, so fairs do not get "anonymized"

[176] e.g. first occurrence in the left to right search with maximum replication order

[177] we reached the countable depth of recursion, although this step should never get reached after exhausting all fairs in previous step, so it is more of a dead code rather than a "safety net"

keep track of every finite number translated into binary and appended to a countable binary string x that has been observed so far[178]. Player I wins if the countable binary string is not perfectly fair, otherwise Player II claims the win. i.e. if x is perfectly fair. There is a round for every natural number $n \in \mathbb{N}$, and after all rounds are played (meaning that both players have the opportunity to move). We want to show that at least one of the players has a wining strategy.

Player II strategy is straightforward. He needs to avoid a situation when the observed sequence has a subsequence $y \ \varepsilon \ x$ that can be represented as a concatenation of at most two similar subsequences $y = z \cdot z$. However, it seems that Player I does not stand a chance. No matter, what adjustments are introduced by Player I in the original number provided by DC, unfortunately for him, Player II can always choose his moves carefully to avoid countable duplication by concatenation. For example, Player I has played some $m \in \mathbb{N}$. Then, player II can check if the number that he has picked to play for the same round e.g. $m' \in \mathbb{N}$ will make him lose. To avoid this, player II can imagine appending the number to the end of x, right after finite $\{0,1\}^{\lceil log2(m) \rceil}$ sequence of player I, as another finite $\{0,1\}^{\lceil log2(m') \rceil}$ sequence to see if this will form a period z and lead to $y \ \varepsilon \ x$ containing a contiguous duplication, i.e. $z \cdot z = y$. This means, Player II can always counterplay moves by player I by providing a different number (either by applying binary inverse or by adding another number provided by DC, etc.). In conclusion, after all countably many rounds are played, Player II has always a winning strategy and game is determined. □

Theorem 47 - k-degree Fair Determinacy.

A game to decide if a fair string x with degree of replication $k < \omega$ is determined.

Proof. The proof extends on the previous result which is a special case of $k = 2$. In the previous proof of *Theorem 46* we have showed that a contiguous repetition of at least two countable binary substrings is enough to decide that the string in question is not perfectly fair, i.e. $z \cdot z \ \varepsilon \ x$. Let's rewrite this as $z \cdot z = (z)^2$, so that the degree of replication is exactly $k = 2$. Now we are interested in showing determinacy for more general case when $(z)^k \ \varepsilon \ x, k < \omega$.

The game and winning condition as similar as for the proof for $k = 2$. For clarity, we also assume that the game is rational, i.e. both player will try their best to win and not give up until both have participated in at least countably many rounds. We know from the previous proof, that player II has a wining strategy for $k = 2$. This time in order to win, player II

[178] Since this is a transfinite game the string has countable length

needs a slight adjustment to his previous approach. Initially both players follow a joined strategy in an effort to form a $(z)^k$ period as a substring of x, where $k \in \mathbb{N}$ is a finite number. However, as soon as $\exists y : y \ \varepsilon \ x$ with k replication degree, player II follows the similar strategy as for $k = 2$, essentially preventing player I from winning by forming x with any higher replication degree that can exceeds k (on every round player II plays). Player II has always a winning strategy and game is determined. □

Corollary 48 - Fair Determinacy.

A game that decides between whether a countable binary string x is fair or not is determined.

Proof. By definition, countable fair string should never have a substring $\exists y : y \ \varepsilon \ x$ with a countable replication degree. The proof follows directly from *Theorem 47* when $k < \omega$. □

6.7 Filters, Ideals and Partitions of the Continuum

In this paper we have talked about two prominent ways of how to think about the continuum[179]. The first one would be to think about the continuum as a powerset of natural numbers with cardinality $|\mathcal{P}(\mathbb{N})| = 2^{\aleph_0} = \mathfrak{c}$. The other one, would be a collection of all countable binary strings $\mathbb{B}$, which has the same cardinality as the Cantor Set[180] and also equals to $|\mathbb{B}| = 2^{\aleph_0} = \mathfrak{c}$. Let us have a closer look on how one can define a poset on such collection of strings $\mathbb{B}$, which will give us enough structure such as filters and ideals.

In order to recall the necessary definitions for filters, we will break down the comparison between filters on sets and filters on posets[181]. The definitions are in fact quite similar, as they both reflect a kind of "closure" property. The difference comes directly from the nature of the underlying structures we want to consider: *sets* versus *posets*.

Definition 57 - Filter on sets.

F is a filter on set X iff the following holds:

1. If $A, B \subset X$ and $A, B \in F$, then $A \cap B \in F$ (Closed under intersection)
2. If $A, B \subset X$, $A \in F$ and $A \subset B$, then $B \in F$ (Closed under supersets)
3. The empty set is not in F: $\emptyset \notin F$

[179] There are of course infinitely more ways of how to think about continuum including the real line, $[0, 1]$ interval, set of all functions $\{f : \mathbb{N} \to \{0, 1\}\}$, etc.

[180] See *Lemma 19*

[181] specifically, as used in general forcing models and principles such as Martin's Axiom [6]

Definition 58 - Filter on posets (Order Filter).

F is a filter on poset $(P, \leq)$ iff the following holds:

1. If $p, q \in F$, there is some $r \in F$ such that $r \leq p$ and $r \leq q$ (Closed under "meets")
2. If $p \in F$, $q \in P$ and $p \leq q$, then $q \in F$. (Closed upwards)
3. F is not empty.

Let us provide similar definitions, but now for ideals. We will again use unions and subsets for sets, and joins and downward closure for posets.

Definition 59 - Ideal on Sets.

An ideal I on a set X is defined as follows:

1. If $A, B \in I$, then $A \cup B \in I$ (Closed under union).
2. If $A \in I$ and $B \subset A$, then $B \in I$ (Closed under subsets).
3. The whole set X is not in I: $X \notin I$.

Definition 60 - Ideal on Posets (Order Ideal).

An ideal I on a poset $(P, \leq)$ is defined as follows:

1. If $p, q \in I$, there is some $r \in I$ such that $p \leq r$ and $q \leq r$ (Closed under "joins").
2. If $p \in I$, $q \in P$ and $q \leq p$, then $q \in I$ (Closed downwards).
3. I is not empty.

Ultrafilter is a dual of prime ideal. A filter (ideal) that contains a finite set is called *principal*. Conversely, a filter (ideal) that contains only infinite sets is called *non-principal*.

Given that both the set of all binary strings $\mathbb{B}$ and the powerset of natural numbers have the same cardinality $|\mathbb{B}| = |\mathcal{P}(\mathbb{N})| = \mathfrak{c}$, there exists a bijection between them (by definition). This is tactically interesting for us, since, if one can define a partial order or poset on $\mathbb{B}$, then whatever structures and properties (like filters and ideals above) one can show to exists for the poset on $\mathbb{B}$, it can be derived to exists on other instances of sets with continuum cardinality via such bijection.

Let $(\mathbb{B}, \leq)$ be such poset. Our idea would be that $(\mathbb{B}, \leq)$ can be defined with either contiguity or substring relation. Moreover, one relation can be

seen as an extension for the other, as they both are complementary. But in order to show this we are still missing an important component, which would be a set of labels of equivalence classes of string enumeration formulas[182]. Given that all of such formula are constructed using string replication we will define such structure more explicitly.

Definition 61 - Replicata.

Let κ, λ be ordinals, $\kappa < \lambda$ and $|\kappa| = |\lambda|$. Then, R^{λ}_{κ} be a label set of equivalence classes of SEFs ($|\lambda|$-length strings over alphabet $\Sigma^{\star} \supset \{0, 1, (,)\}$). Also let $\{0,1\}^{|\lambda|}$ be a set of infinite (λ-length) binary strings and $\pi : R^{\lambda}_{\kappa} \to \{0,1\}^{|\lambda|}$ a production function evaluating each formula to its replicator-free form $\{0,1\}^{|\lambda|}$. Finally, let $P \subseteq R^{\lambda}_{\kappa}$ be some partial order[183] on the label set. We call $(R^{\lambda}_{\kappa}, P, \pi)$ structure a *replicata* (or a replicated space) with replication degree κ.

The length of SEFs in $dom(\pi) = R^{\lambda}_{\kappa}$ is $|\lambda|$. We want π to be defined as bijection, hence the length of $\{0,1\}$ strings in $ran(\pi)$ also equals to $|\lambda|$. In this section of the paper we will be interested in a countable case when $\omega \leq \lambda < \omega_1$. So, for convenience, when λ is omitted in the above notation, we assume $|\lambda| = \aleph_0$ to actually mean length[184], or, in other words, we implicitly invoke countable union theorem, so that the union of countable strings is again a string of length λ within $\omega \leq \lambda < \omega_1$ and $R^{<\omega_1}_{\kappa} = R_{\kappa}$. Also, for the rest of the section the other most frequent case of interested is countable $|\kappa|$. Hence, we imagine fixing some arbitrary large countable ordinal $\kappa < \lambda < \omega_1$ and write $dom(\pi) = R_{<\omega_1}$ and respectively $ran(\pi) = \{0,1\}^{<\omega_1} = \mathbb{B}$ (meaning the same fixed ordinal κ for $dom(\pi)$ and $ran(\pi)$), i.e. $\pi : R_{<\omega_1} \to \mathbb{B}$.

We will abuse notation for SEFs according to our convenience. We assume that alphabet $\Sigma^{\star}$ is extended with polynomial notation, very much in line how we worked with SEFs in the course of last subsections. For example, we will continue to use small Latin letters to note variables that can reference other formulas and exponents in the replication operator as shorthand for replication degree.

Now, we need to clarify about bijectivity of π. Note that unfair countable (UC) SEFs still can be mapped one-to-one to the production image of $\pi(s_x) = x$ as long as we can represent and match the unfair finite substrings of the SEF s_x in preimage. Hence, the proof in *Theorem 24* can be extended,

[182] also abbreviated as SEFs as understood and discussed much earlier in the paper in *Theorem 24*, but also in *Theorem 23*

[183] It is assumed that P comes with the binary relation, which we sometimes omit to specify - see comments below the *Definition 69*

[184] Equally, every time when we write $\{0,1\}^{\omega}$ we actually mean $\{0,1\}^{|\omega|}$ or $\{0,1\}^{<\omega_1}$. i.e. we always assume *CUT*

so that bijectivity of π holds also when we use the shortest SEFs as labels to mark equivalence classes $[s_x]$ for $s_x \in S_{UC}$[185].

Lemma 49 - Bijectivity of π.

If $(R_{<\omega_1}, P, \pi)$ is a replicata, then production π is well-defined and always bijective, i.e. $|R_{<\omega_1}| = |\mathbb{B}|$.

Proof. Assume $ZF+CC$. If $x = \pi(s_x)$, then $kind(s_x) := \{UF, PF, IF, UC\}$ (by *Definition 55*) determines the kind of s_x essentially helping to partition $R_{<\omega_1}$ into disjoint classes $R_{<\omega_1} = S_{UF} \cup S_{PF} \cup S_{IF} \cup S_{UC}$.

If $(R_{<\omega_1}, P, \pi)$ is a replicata, defined from labels of SEF equivalence classes, then by definition $[s_x] = \{\varphi \in R_{<\omega_1} : \pi(\varphi) = \pi(s_x)\}$ is an equivalence class of formulas that point to the same image (π is constant). Recall that the label s_x is either chosen to be the shortest SEF $s_x : |s_x| = min(\{|\varphi| : \varphi \in [s_x]\})$ for unfair finite strings (since there are countably many unfair finite SEFs $|S_{UF}| = \aleph_0$ - *Theorem 23*), or is equal to the image $x = \pi(s_x)$ iff x is a fair string ($s_x \in S_{PF} \vee s_x \in S_{IF}$). Again, labeling is based on the idea that one can always pick the shortest label for the equivalence class of unfair finite (assuming CC). Fair SEFs are labeled "as is" by their production image (identity). Finally, unfair countable (UC) strings are essentially a combination of countably many strings as concatenation of both unfair finite and fair (countable). So also for $s_x \in S_{UC}$ one can pick a much "shorter" SEF by abbreviating unfair finite subparts. Although such SEF will remain countable $|s_x| = \aleph_0$ due to fair substrings, i.e. $\exists y : y \; \varepsilon \; x \wedge (s_y \in S_{PF} \vee s_y \in S_{IF}) \in \Longrightarrow \; |s_y| = \aleph_0 = |s_x|$.

To complete the proof, we want to argue that π can be well-defined. This can follow from showing first that $kind(s_x) := \{UF, PF, IF, UC\}$ can partition $R_{<\omega_1}$. All four kinds are disjoint by definition. Let us determine $kind(s_x)$ according to exclusive logical steps, so that one and only one of the following is true:

1. it is determined that $s_x \in S_{UF}$ iff one can check that $s_x \in S_{UF}$ by applying CC to go through the list of all finite SEFs;
2. it is determined whether $s_x \in S_{PF}$ - follows from *Theorem 46*;
3. it is determined that $s_x \in S_{IF}$ iff $s_x \notin S_{PF}$ and s_x is fair (follows from *Corollary 48*);
4. otherwise, $s_x \in S_{UC}$.

Production π is naturally defined for S_{UF} (by evaluating replicators as countable concatenation of finite substrings between parenthesis - see *Definition 49*). It is also defined for all fairs as identity. Finally, S_{UC} is discussed

[185] rather than replicator-free identity

above. Now that we can differentiate between kinds of SEFs, well-definition of π follows. This also means that for each label s_x of the equivalence class $< s_x >$ we have one-to-one correspondence with the binary image $x \in \mathbb{B}$ (by construction of $R_{<\omega_1}$). Hence, π is bijective. □

Lemma 50 - Partition of $R_{<\omega_1}$.

If $(R_{<\omega_1}, P, \pi)$ is a replicata, then $R_{<\omega_1}$ can be partitioned into $m \geq 2, m \in \mathbb{N}$ pairwise disjoint subsets. Meaning that there exists a partition function $f : R_{<\omega_1} \to \{1, .., m\}$.

Proof. Follows directly from *Lemma 49*, where we show that $kind(s_x)$ is determined for $m \leq 4$ (assuming one can join classes to show partition of $R_{<\omega_1}$ into $2, 3$ disjoint subsets). For other cases when $m \geq 5$, one can pick $k = m - 3$ and extend $kind(s_x)$ definition, with up to countably many k-degree fair classes of SEFs - see *Theorem 47*. □

It will be appropriate to review our recent observations regarding *fair determinacy* and the *partition of* $R_{<\omega_1}$ in greater detail. These findings seem noteworthy and will play a significant role in the subsequent sections of this paper. One of the way of doing so is in the context of some known combinatorial results[186] about *colouring infinite graphs and the Prime Ideal Theorem*. For some $n \in \mathbb{N}$ consider the following statement:

P_n : If G is a graph such that every finite subgraph of G is n-colourable, then G itself is n-colourable.

The following implications are provable in ZF without AC[187]:

$$PIT \implies P_{n+1} \implies P_n \implies C(\infty, n), C(\infty, 2) \implies P_2$$

Surprisingly, Läuchli showed in [27] that P_3 implies PIT, and consequently, for all $n \geq 3$, the equivalence $P_n \implies PIT$ is provable in ZF without AC.

Recall that PIT stands for *Prime Ideal Theorem*. It is a well-known and powerful choice principle (weaker than AC), which we did not cover yet:

Theorem 51 - Prime Ideal Theorem (PIT) for Boolean algebras.

If I is an ideal in a Boolean algebra[188], then I can be extended to a prime ideal.

[186] See p.133 in [26]
[187] Set Theory without the Axiom of Choice
[188] See *Definition 68*

Sometimes seemingly related but essentially equivalent statement can be called *Boolean Prime Ideal* (BPI)[189] may be employed to establish numerous fundamental theorems in set theory without requiring the full axiom of choice [28].

Lemma 52.

The following are equivalent[190]:

1. every non-empty Boolean algebra has a prime ideal (BPI).
2. If $\{X_i \mid i \in I\}$ is a family of compact Hausdorff spaces, then $\prod_{i \in I} X_i$ is a compact Hausdorff space.
3. For every I, $\{0,1\}^I$ is compact, where $\{0,1\}$ is the discrete space with two elements.
4. The compactness theorem for first-order logic.
5. The completeness theorem for first-order logic.
6. If R is a commutative ring with a unit, then every ideal is contained in a prime ideal (PIT for commutative rings).

Lemma 53.

The following are consequences of BPI. None of these statements is provable in ZF alone, and none of them imply BPI:

1. Hahn-Banach theorem.
2. Every set can be linearly ordered[191].
3. There exists a non-measurable set.
4. The Banach-Tarski paradox.
5. There is a finitely additive probability measure on $\mathcal{P}(\mathbb{N})$ which vanishes on singletons.
6. If a vector space has a basis, then every two bases have the same cardinality.
7. Every two algebraic closures of a field F are isomorphic.

[189] Meaning that $PIT \iff BPI$ - see *lemma 52*

[190] This and next lemma are almost literally taken from[28], which is a great brief introduction on the matter of "choice" next to [9] and [21]

[191] Weaker form of the full axiom of choice, when sets that are not necessarily well-orderable or even countable

PIT can be considered as an attractive path to extend the framework of $ZF + DC$, since PIT and DC are in fact independent of each other. To further quote [28]:

> It is worth to mention that BPI is consistent with the statement "There is an infinite set of real numbers which does not have a countably infinite subset". The latter is a contradiction to DC, and therefore we cannot prove DC in $ZF + BPI$. In the other direction it is consistent with $ZF + DC$ that every set is Lebesgue measurable, in which case BPI fails, and therefore we cannot prove BPI from $ZF + DC$. So these two principles are indeed independent. Moreover, while $ZF + DC + BPI$ seems to be sufficient to prove a lot of results of interest, this theory still cannot prove the axiom of choice in full.

Let us reiterate the result by Läuchli showed in [27] based on combinatorics of graph coloring in greater detail. The result offers a technical combinatorial connection between Boolean algebras, graph theory, and set theory and implies PIT within the framework of ZF without AC. It shows that certain properties of graphs constructed from Boolean algebras can provide insights into algebraic structures within these algebras, such as existence of prime ideals.

Theorem 54 - $P_3 \implies PIT$**.**

In set theory without AC, there exists a function G that maps each Boolean algebra B to a graph $G(B)$, satisfying the following properties:

1. If $G(B)$ is 3-colorable, then there exists a prime ideal in B.
2. Every finite subgraph of $G(B)$ is 3-colorable.

The proof of this theorem relies on a combinatorial lemma on finite graphs. This lemma involves techniques from graph theory to establish properties related to colorability and structure within the graphs $G(B)$. Rather than presenting the entire proof[192], we will discuss the key notions behind the theorem to see how this result can be applied to the replicata $R_{<\omega_1}$.

1. **Graph Construction (Function G):**
   - For each Boolean algebra B, the function G assigns a graph $G(B)$.
   - This function G establishes a correspondence between Boolean algebras and graphs.

[192]For the actual proof please see the original paper [27]

2. **Colorability of $G(B)$:**
    - The graph $G(B)$ constructed from a Boolean algebra B satisfies a particular property: it is 3-colorable.
    - This means that the vertices of $G(B)$ can be colored with three colors in such a way that no two adjacent vertices have the same color.
3. **Connection to Prime Ideals:**
    - The theorem establishes a connection between the colorability of $G(B)$ and the existence of prime ideals in B.
    - Specifically, if $G(B)$ is 3-colorable, then there exists a prime ideal in B.
    - This connection implies that certain properties of the graph $G(B)$ correspond to algebraic properties within the Boolean algebra B.
4. **Finite Subgraphs:**
    - The theorem also guarantees that every finite subgraph of $G(B)$ is 3-colorable.
    - This property ensures that the colorability of $G(B)$ extends to all possible finite configurations of vertices and edges within the graph.

We don't necessarily need to show full PIT in most general sense, that it holds for every set. In fact, we just have mentioned above that - it would be impossible to achieve in $ZF + DC$ due to independence of BPI and DC. But, as it will turn out later, for our purposes a much weaker statement will suffice. The statement is based on *Theorem 5.15* in [26] but also true for the above *lemma 52*, again with restriction to a single set X.

Theorem 55 - Restricted Prime Ideal Theorem $BPI(X)$.

If B be a Boolean algebra on infinite set X, s.t. $B = (X, \wedge, \vee, \neg, \mathbf{0}, \mathbf{1})$, then the following statements are equivalent:

1. $PIT(X)$: if I is an ideal in a Boolean algebra B, then I can be extended to a *prime ideal*.
2. $UltrafilterTheorem(X)$: if F is a filter over set X, then F can be extended to an *ultrafilter*.
3. Consistency principle on X: for every binary mess M (set of binary functions defined on finite subsets of X) on X, there exists a binary function f on X which is consistent with M.

4. Compactness theorem for propositional logic on X: Let Σ be a set of formulae for Propositional logic, $X \subseteq \Sigma$ be a set of Propositional variables and function f a *realization* of Propositional logic (a map from all formulas to two-valued Boolean algebra $f : \Sigma \to (\{0,1\}, +, \cdot, -, \mathbf{0}, \mathbf{1})$). If every finite subset of Σ is satisfiable, then also Σ is satisfiable[193].

5. $BPI(X)$: Boolean algebra B has a prime ideal.

Given the restriction to a single set X, it is enough to show at least one of the statements in *Theorem 55* and the rest of the argument chain, including $BPI(X) \implies PIT(X)$, will follow from the more general proof *Theorem 5.15* in [26]. The above restricted formulation allows us to show the special case we aimed at:

Lemma 56 - Boolean Prime Ideal on $B(P(R_{<\omega_1}))$.

$BPI(R_{<\omega_1})$

Proof. Assume $ZF+DC$. We want to show that if $(R_{<\omega_1}, P, \pi)$ is a replicata and B is a corresponding Boolean algebra on set R_w, then we can show $PIT(R_{<\omega_1})$. We will do so in two steps:

(i) There exists a Boolean algebra B formed from SEFs in $R_{<\omega_1}$.

(ii) $G(B)$ can be 3-colored.

(i) We want to show that B is a Boolean algebra on $R_{<\omega_1}$. We start by letting $B^* = (X, \vee, \wedge, \neg, \mathbf{0}, \mathbf{1})$ be a Boolean algebra on set X and $X = \pi(R_{<\omega_1})$, consisting of $\{0,1\}^\omega$ strings (π is bijective - *lemma 49*). Boolean logical operators can be defined directly[194] as so-called boolean bitwise operations (see *Definition 67*) - on the index by index bases for a pair of sequences representing strings $\forall x, y \in X$:

- $x \vee y := z$, s.t. $\forall i \in \mathbb{N} : z(i) = x(i) \vee y(i)$
- $x \wedge y := z$, s.t. $\forall i \in \mathbb{N} : z(i) = x(i) \wedge y(i)$
- $\neg x := z$, s.t. $\forall i \in \mathbb{N} : z(i) = \neg x(i)$
- $\mathbf{0} := \pi(\text{“}(\mathbf{0})\text{”})$
- $\mathbf{1} := \pi(\text{“}(\mathbf{1})\text{”})$

The above definitions allows extending B^* from two-valued Boolean algebra. Technically, we assume that B is isomorphic to the Boolean algebra B^* and can be defined directly on SEFs, s.t. $B = (\pi^{-1}(0, 1^\omega), \vee, \wedge, \neg, \mathbf{0}, \mathbf{1})$. For

[193] See [26] for exact definitions of mapping formulas to $\mathbf{1}$ to be satisfiable

[194] Or from universal logical gate NAND - see *lemma 68*

the detailed proof that B^* is a Boolean algebra on binary $0, 1^\omega$ strings see *Theorem 70*.

(ii) Let us show that $G(B)$ can be 3-colored. By *Lemma 50*, there exists a partition of $R_{<\omega_1}$, s.t. $R_{<\omega_1} = S_{UF} \cup S_{FC} \cup S_{UC}$, where $S_{FC} = S_{PF} \cup S_{IF}$ are all fair countable SEFs (see *Definition 55*). Here we employ *fair determinacy* (*Corollary 48*) abbreviating as FD to decide whether $\forall s_x \in B : FD(s_x) \implies s_x \in S_{UF} \vee s_x \in S_{FC} \vee s_x \in S_{UC}$. Let $G(B) = (V, E)$, specifically vertices are equal to the elements of Boolean algebra $V = B$. So, essentially, we configure countable binary strings to be vertices of $G(B)$. Since pi and $G(B)$ are bijective, one can color any vertex of $v \in V$ that correspond to pairwise disjoint $\{S_{UF}, S_{FC}, S_{UC}\}$ into $\{red, green, blue\}$ respectively. Now it remains to show that G is connected and there are no edges which are connecting vertices of the same color, i.e. $\forall i, j \in I(V) : (v_i, v_j) \in E \wedge c(v_i) \neq c(v_j)$, where $c : V \to \{red, green, blue\}$ is a coloring function. To do this, we connect edges by the following rules:

(a) each $u \in S_{UC}$ can be mapped into the template formula $\phi \in S_{UF}$, so that one or countably many finite binary substrings are replaced by $x_1, .., x_n$ variables

(b) if $v \in S_{UF}$ is such a template $v \leftarrow \phi(x_1, .., x_n)$ (equivalence modulo substitution of concrete $x_1, .., x_n, n \in \mathbb{N}$ with exact finite binary strings) that can be used to generate $u \in S_{UC}$ by replacing (substituted) $x_1, .., x_n, n \in \mathbb{N}$ variables of ϕ with some $y_1, .., y_n \in Y, Y \subseteq S_{FC}$, then we connect those vertices with the edge $(v, u) \in E$

(c) furthermore, if $w \in S_{FC}$ can be substituted as one of the above variables $x_1, .., x_n, n \in \mathbb{N}$ for some $v \leftarrow \phi(x_1, .., x_n)$ so that $w = y_i$ and $\phi(y_1, .., y_n) \in S_{UC}$, then we connect those vertices with the edge $(w, v) \in E$

To show that there are, indeed, no edges connecting the same colors (by the above construction) it is enough to recall that $R_{<\omega_1}$ consists of the equivalence class labels and every SEF $s_x \in R_{<\omega_1}$ is a unique label of the whole class $[s_x]$, s.t. $\pi(\varphi) = x : \varphi \in [s_x]$. For instance, this also means that there is a bijection between every v and ϕ if $v \in S_{UF}$ is substitution template $v \leftarrow \phi(x_1, .., x_n)$. Also $x_1, .., x_n$ variables can be matched one-to-one to the $n \in \mathbb{N}$ finite substrings of the unfair finite SEF v. There are cycles in $G(B)$, but there are no edges between the vertices of the same color and between any $(w, u) : w \in S_{FC} \wedge u \in S_{UC}$. Finally, observe that vertices belonging to S_{UF} and S_{UC} as well as the other subset of vertices belonging S_{UF} and S_{FC} individually form two bipartite subgraphs, which can be joined over vertices belonging to S_{UF} together into a 3-colored graph G. Hence, if $G(B)$ is constructed in the above way, then it can be 3-colored.

Finally, the proof follows by applying result from *Theorem 54* (that can be restricted to any X): $G(B)$ can be 3-colored $\implies$ $PIT(R_{<\omega_1})$. And according to *Theorem 55*, $PIT(R_{<\omega_1}) \iff BPI(R_{<\omega_1})$. □

6.8 Smaller and bigger cardinals

According to the preceding *Theorem 55* we have that

$$PIT(R_{<\omega_1}) \implies Ultrafilter(R_{<\omega_1})$$

This consequence warrants further investigation. But before exploring its potential applications, let us carefully examine the underlying properties and structure of prime ideals on $R_{<\omega_1}$, given that we already know quite a lot about what SEFs constituting $R_{<\omega_1}$ are and how they are constructed. We will do so in this subsection. We will also touch on the topic of *large cardinal axioms*.

In particular, we would be interested in how can we define a *non-principal* ideal I on $R_{<\omega_1}$ (that can be extended to a non-principal prime ideal). We start with a subset of perfectly fair SEFs $S_{PF} \subset S_{FC} \subset R_{<\omega_1}$. Let us resume the narrative from few subsections earlier by coming back to *lemma 45* about the uncountability of fairs[195]. Our next goal is to reiterate the previous proof to see if we can say more about the structure of fairs S_{FC} and the cardinality of its partition $S_{FC} = S_{PF} \cup S_{IF}$.

We begin by offering a definition for almost disjoint countable binary strings derived from the definition of almost disjoint sets in [6].

Definition 62 - Almost disjoint strings.

Let x and y be countable binary strings that can be seen as characteristic functions corresponding[196] to respective sets X and Y. We say that the pair (x, y) consists of *almost disjoint* strings iff intersection $X \cap Y$ is finite.

Lemma 57 - Finite symmetrical difference.

Let x and y be countable binary strings. If the following is true:

- $z = x \oplus y$ is a XOR operation on countable binary strings (symmetrical difference)
- z can be written in form "$a_0(0)a_1..(0)a_n(0)a_{n+1}$" : $n \in \mathbb{N}$ called *finite symmetrical difference* (where each a_n is some non-empty finite binary substring of z except a_0 and a_{n+1}, which can be optionally empty)

[195] For brevity, we often shorten and say just *fairs* instead of *fair strings*. Again, recall that a fair string is simply a string with a corresponding fair formula.

[196] For instance, by morphism provided in *Definition 120*

Then, both pairs $(\neg x, y)$ and $(x, \neg y)$ are almost disjoint.

Proof. Our goal is to show $(\neg x, y)$ and $(x, \neg y)$ form almost disjoint pairs.

1. **XOR Operation**: The operation $x \oplus y$ yields a binary string z marking positions where x and y differ, i.e., $z_i = 1$ iff $x_i \neq y_i$. A finite symmetrical difference implies z contains a finite number of 1s, demarcating a finite set of positions where x and y diverge.

2. **Almost Disjoint Strings**: By definition, strings x and y are almost disjoint if their corresponding sets, interpreted via their characteristic functions, have a finite intersection.

   (a) For $(\neg x, y)$: The intersection of interest consists of positions where $\neg x$ and y both have 1s, which occur where x has 0s and y has 1s. Given z is finitely bounded, the subset of positions contributing to this intersection is finite, satisfying the almost disjoint criterion.

   (b) For $(x, \neg y)$: Analogously, we consider positions where x has 1s and $\neg y$ has 1s, equating to positions where y has 0s. The finiteness of z ensures this intersection is also finite.

Given the finite nature of $z = x \oplus y$, it follows directly that both $(\neg x, y)$ and $(x, \neg y)$ exhibit finite intersections in their corresponding sets, thus qualifying as almost disjoint pairs. This satisfies the lemma's statement by leveraging the definition of almost disjoint strings and the characteristics of the XOR operation. □

As it is evident from the previous lemma, it might be handy to define a concept that is congruent to almost disjoint strings.

Definition 63 - Almost adjoint strings.

Let x and y be countable binary strings. If both pairs $(\neg x, y)$ and $(x, \neg y)$ — formed by negating one element in each pair — are almost disjoint, then we term (x, y) as *almost adjoint* strings.

There is a natural corollary from *lemma 57*.

Corollary 58.

If $z = x \oplus y$ can be written in the form of finite symmetrical difference, then the pair (x, y) consists of two almost adjoint strings.

Proof. Follows directly from *Lemma 57* and *Definition 63*. □

Lemma 59 - Uncountability of perfect fairs.

There exists a subset of perfectly fair strings $I_{PF} \subset S_{PF}$, s.t. $|I_{PF}| > \aleph_0$.

Proof. Assume $ZF + DC$. Let $x \in S_{PF}$ be a perfectly fair countable binary string. Construct a $GCTA(x)$ table T similar to the proof of *Lemma 45*. Observe that the rest of argument in that proof also holds for perfectly fair x instead of just fair string. □

Lemma 60 - XOR closure of S_{UF}.

Let $(R_{<\omega_1}, P, \pi)$ be a replicata with $S_{UF} \subset R_{<\omega_1}$ being a set of all unfair finite formulas, which are labels of the respective equivalence classes under π. Then, S_{UF} is closed under XOR:

$$\forall s_x, s_y \in S_{UF} : s_z = s_x \oplus s_y \implies s_z \in S_{UF}$$

Proof. We show this in several steps:

(i) Let us recall construction of unfair finite SEF equivalence classes S_{UF} as explained not only in *Definition 55* but also much earlier in the paper in subsection *Equivalence classes of SEFs* as well as enumeration of the initial segment in fig. 4[197]. Specifically, there are countably many unfair finite SEF equivalence classes $|S_{UF}| = \aleph_0$ as according to *Theorem 23*. This follows from the fact that by construction $\forall s \in S_{UF} : |s| < m, m \in \mathbb{N}$, i.e. each label of the SEF equivalence class is the shortest formula (from that class) of finite length.

(ii) One of the issues we have is that XOR is not well-defined for SEFs, which are strings built up using $\{0, 1, (,)\}$ symbols. So we need to explain what we mean by XOR for SEF. By *Lemma 49* production π is bijective when the domain is restricted to labels of the equivalence classes $[s_x]$. In general, without such restriction π is surjective and several different forumals can be evaluated into the same countable binary string in $ran(\pi)$. But since we can meaningfully define XOR rather for two countable binary strings (than for two SEFs), when we say $s_z = s_x \oplus s_y$ we actually mean that we first compute $z = \pi(s_x) \oplus \pi(s_y)$ and then obtain $s_z = \pi^{-1}(z)$, s.t. s_z is actually the shortest formula or a label of $[s_z]$ equivalence class.

(iii) By definition s_z is unfair finite iff $z = \pi(s_z)$ contains a countable repetition of a finite binary substring. Pick $u \; \varepsilon \; x$ and $v \; \varepsilon \; y$ as such finite strings from both XOR operands in $z = x \oplus y$, so that indexes of their countable repetitions overlap. Note that no matter how nested are parentheses in formulas s_x and s_y, it is always possible to pick such finite substrings by looking at the formula from left to right even if $|s_x| \neq |s_y|$. For example, consider partial order defined on S_{UF} by substring relation on the formulas. It becomes obvious that $\forall x \in S_{UF}, \exists u = \{0, 1\}^n : |u| = n, n \in \mathbb{N} \wedge$ "(u)" $\varepsilon \; x$. So, there are always some finite binary substrings strings u, v, which (also and as well if concatenated countably many times) will be part of respective $\pi(x)$ and $\pi(y)$ (again, see fig. 4). This means that we can reduce our

[197] see subsection *Labels of the finite SEF equivalence classes* in Appendix

question to simply showing that "(u)" $\oplus$ "(v)" is unfair finite. The latter is as trivial as "(w)" = "(u)" $\oplus$ "(v)" iff $w = u' \oplus v'$, where operands are selected to match their finite length by adding missing symbols through repetition[198]: $u' \ \varepsilon$ "(u)", $v' \ \varepsilon$ "(v)" : $|u'| = |v'|$. Such alignment of finite length is possible by perfectly matching the periods of both strings (see *lemma 61*).

□

Lemma 61 - Alignment of Repeated Binary Strings.

Consider two non-empty binary strings x and y with lengths n and m respectively, where $n, m \in \mathbb{N}$. There exists a length, common to both, at which the periodic repetition of x and y results in their perfect alignment.

Proof. Given two non-empty binary strings x and y with lengths n and m, respectively, we aim to show that there exists a common length where periodic repetitions of both strings align perfectly.

The least common multiple (LCM) of two numbers n and m, denoted $\text{lcm}(n, m)$, is the smallest positive integer that is divisible by both n and m. According to the definition of LCM:

$$\text{lcm}(n, m) = n \cdot i = m \cdot j \quad \text{for some } i, j \in \mathbb{N}.$$

This relationship means that $\text{lcm}(n, m)$ is a multiple of both n and m. Therefore, if both x and y are repeated indefinitely to create strings of length $\text{lcm}(n, m)$, x will have completed exactly i cycles and y will have completed j cycles.

Thus, by construction, every character in the infinitely repeated string for x at positions $k \cdot n$ (for any k) aligns with the corresponding characters in the infinitely repeated string for y at positions $k \cdot m$, ensuring perfect alignment at all these positions. Hence, $\text{lcm}(n, m)$ serves as the required common length for perfect alignment of the periodic repetitions of x and y. □

Corollary 62 - Unfair-Finite Boolean Algebra.

Let $(R_{<\omega_1}, P, \pi)$ be a replicata with $S_{UF} \subset R_{<\omega_1}$ being a set of all unfair-finite formulas and $B_{UF} = \pi(S_{UF})$. Then,

$$(B_{UF}, \vee, \wedge, \neg, \mathbf{0}, \mathbf{1})$$

is a Boolean algebra.

[198] Alternatively, one can remove extra symbols to match the length.

Proof. Follows from similar arguments as above regarding XOR closure of S_{UF}. Knowing XOR properties (Closure, Associativity, Commutativity, Identity Element, Inverse Element, Distributivity) one can define OR, AND, NOT operations on countable binary strings $\forall x, y \in B_{UF}$ similar to how $\vee, \wedge, \neg$ are defined[199] for B^* in *lemma 56*.

Next, reiterate the argument (as in the previous proof) that it is enough to show closure on a limited part of any unfair finite formula. Replace $\oplus$ with any of the $\vee, \wedge, \neg$ operations in the statement: "(w)" = "(u)"$\oplus$"(v)" iff $w = u' \oplus v'$, where u' and v' are perfectly aligned (see *lemma 61*). Observe that such statement will hold for all three operations: $\vee, \wedge, \neg$.

Closure under all boolean operations for B_{UF} implies that B_{UF} is a Boolean algebra. Furthermore, $S_{UF} \subset R_{<\omega_1} \implies B_{UF} \subset B^*$. □

Corollary 63.

$(B_{UF}, \oplus, \cdot, \mathbf{0}, \mathbf{1})$ is a ring, where multiplication is $x \cdot y = x \wedge y$.

Lemma 64 - Cardinality of perfect and imperfect fairs.

Perfect fairs have the same cardinality as the imperfect fairs:

$$|S_{PF}| = |S_{IF}|$$

Proof. Can be shown by invoking Schröder-Bernstein theorem that there exist a bijective $h : S_{IF} \to S_{PF}$ iff there exists f and g, s.t.:

- $f : S_{IF} \to S_{PF}$ is injective - with index $\forall \alpha \in I(S_{IF})$, we have an automorphism $\mu : S_{IF} \to S_{IF}$ defined as $z_\alpha \mapsto x_\alpha$, where $\forall z_\alpha \in S_{IF}, \exists x_\alpha \in S_{IF} : x_\alpha \, \varepsilon \, z_\alpha$ (Mind the trivial case $x_\alpha = z_\alpha \implies x_\alpha \, \varepsilon \, z_\alpha$). Now, by definition of imperfect fairs - each imperfect x_α is constructed by concatenating some fair substring at least twice, so $\exists y_\alpha \in S_{PF} : x_\alpha = y_\alpha \cdot y_\alpha$. One can assemble this into defining f as composition map $z_\alpha \mapsto y_\alpha \mapsto x_\alpha$. Like this every z_α is mapped to a unique x_α and f is injective.

- $g : S_{PF} \to S_{IF}$ is injective - following on the previous point, with index $\forall \beta \in I(S_{PF})$ and $\forall x_\beta \in S_{PF}$ we have $y_\beta = x_\beta \cdot x_\beta$. Again, by definition of imperfectly fair, $y_\beta \in S_{IF}$. So g can be defined as $x_\beta \mapsto y_\beta$. Like this every x_β is mapped to a unique y_β and g is injective.

□

[199] Recall that if NOT operation is defined one can also derive OR and AND from $x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)$.

Lemma 65 - Greater cardinality of unfair countable.

Let $(R_{<\omega_1}, P, \pi)$ be a replicata, s.t. $R_{<\omega_1} = S_{UF} \uplus S_{FC} \uplus S_{UC}$. Then, unfair countable strings have greater cardinality than fair countable:

$$|S_{FC}| < |S_{UC}|$$

Proof. (i) We want to show that there exists no injective map from S_{UC} to S_{FC}. Suppose, for the sake of contradiction, that there exists an injective function $f : S_{UC} \to S_{FC}$.

(ii) Recall that if some s is unfair countable, then, by construction, $s \in S_{UC}$ iff $\exists w \in S_{FC} : w \; \varepsilon \; s$. All unfair countable must contain at least one fair substring. Otherwise, s is finite.

(iii) Given that f is injective, it must map every string from the $dom(f)$ to some unique value in the image $ran(f)$. Namely, $\forall s_x, s_y \in S_{UC}$ we have that - if $s_x \neq s_y$, then $f(s_x) \neq f(s_y)$.

(iv) Let $u, v \in S_{PF}$ and $u \neq v$. Without loss of generality, the above means that if f is injective, then it maps all pairs of (u, v) found in any unfair formula in S_{UC} one-to-one to a fair counterpart in S_{FC}.

(v) In terms of pigeonhole principle, if f is injective, then each $s_\alpha \in S_{UC}, \alpha \in I(S_{UC})$ ends up having a unique pigeonhole $f(s_\alpha) \in S_{FC}$.

(vi) For example, take $s_x = (u \cdot v)$ and $s_y = ((u)(v))$. We can only map them to the best candidate "pigeonhole" for (u, v) pair in S_{FC} - the one without parentheses - $f(s_x) = \text{“}u \cdot v\text{”} = f(s_y)$. Then this will lead to a contradiction and our initial assumption that f injective must be wrong. So to complete the proof, It remains to clarify what we mean when we say *map to the best candidate "pigeonhole"*.

(vii) Above we assume that such map exists by proposing the plain encoding of the (u, v) pair in form of “$u \cdot v$”. But maybe we can do better with our argument. We can show that there are not enough candidates.

(viii) Let

$$\begin{aligned} S_p = \{&\text{“}(u)v\text{”}, ..., \text{“}(u)v(u)v\text{”}, ...\text{“}((u)v)\text{”}, ... \\ &\text{“}u(v)\text{”}, ..., \text{“}u(v)u(v)\text{”}, ..., \text{“}(u(v))\text{”}, ... \\ \text{“}(u)(v)\text{”}, ..., &\text{“}(u)(v)(u)(v)\text{”}, ..., \text{“}((u)(v))\text{”}, ...\} \end{aligned}$$

be a set of "pigeons" in need of unique pigeonholes. Note that $|S_p| = |S_{UF}| = \aleph_0$ as it is constructed by replacing $\{0, 1\}$ with $\{u, v\}$ in those finite SEFs that contain both 0 and 1.

(ix) Now let us reiterate our initial assumption that f is injective (but in line with the argument about existence of the best pigeonhole candidate). Essentially, we have arrived at - f is injective iff one can map every pigeon (formula) in S_p to its respective and unique pigeonhole. For this to be possible there must exist such an encoding that would allow writing up $\{u, v\}$ as a set of fair formulas for all possible $\{u, v\}$ pigeonholes. Assume this is as well possible. It is important to note that such encoding would still mean sticking to the language of $\{0, 1\}$ for fair strings plus concatenation.

(x) Observe that the following pigeons will share the pigeonhole:

$$f(\text{“}(u)v\text{”}) = f(\text{“}u(v)\text{”}) = f(\text{“}(u)(v)\text{”}) = ... = h_1$$

$$f(\text{“}(u)v(u)v\text{”}) = f(\text{“}u(v)u(v)\text{”}) = f(\text{“}(u)(v)(u)(v)\text{”}) = ... = h_2$$

$$...$$

and, in fact, we can continue this list by taking $u_n \in S_{PF}, n \in \mathbb{N}$ to replace (u, v) pair with countable $(u_1, .., u_n)$ tuple. It follows that countably many pigeons will have to share the pigeonhole.

(xi) Also in the above list we have omitted one line:

$$f(\text{“}((u)v)\text{”}) = f(\text{“}(u(v))\text{”}) = f(\text{“}((u)(v))\text{”}) = ... = h_\omega$$

which can also be nicely interpreted that those pigeons never get the pigeonhole since if we try to encode $z =$ “$((u)(v))$” with countably many concatenations, then it will violate the definition for imperfectly fair strings, i.e.: $z \notin S_{PF} \wedge z \notin S_{IF}$.

In conclusion, there exists no injective $f : S_{UC} \to S_{FC}$, since (as it turns out) one cannot match all $s_\alpha \in S_{UC}$, so that each formula gets a unique ”pigeonhole” assigned, and one cannot do it at least countably many times. □

All the work we did in solidifying our understanding about the structure $R_{<\omega_1}$ is a preparation work for it to be used as a model. But we need to do few more observations before we can start this next chapter.

Corollary 66 - Cardinalities of $R_{<\omega_1}$ partition.

Let $(R_{<\omega_1}, P, \pi)$ be a replicata, s.t. $R_{<\omega_1} = S_{UF} \cup S_{FC} \cup S_{UC}$. Then, the following holds:

1. $|S_{UF}| = \aleph_0 = \omega$
2. $|S_{FC}| = |S_{PF}| = |S_{IF}| = \aleph_1 = \omega_1$

3. $|S_{UC}| = \aleph_2 = \omega_2$

Proof. Rather than implicitly relying on the law of excluded middle[200], to show (1-3) we would use the fact that there is partition $R_{<\omega_1}$, which we have arranged in particular strict order - $|S_{UF}| < |S_{FC}| < |S_{UC}|$

(1) There are countably many unfair finite SEF equivalence classes $|S_{UF}| = \aleph_0$ as according to *Theorem 23.*

(2) Discussed earlier in the paper *lemma 45* about the uncountability of fairs. Specifically, follows from uncountability of perfect fairs as shown in *lemma 59* and that $|S_{PF}| = |S_{IF}|$ as shown in *lemma 64.* hence $|S_{UF}| < |S_{FC}|$

(3) $|S_{FC}| < |S_{UC}|$ has been just shown in the previous *lemma 65.*

We conclude by assigning respective cardinalities in ascending order $|S_{UF}| = \aleph_0, |S_{FC}| = \aleph_1, |S_{UC}| = \aleph_2$. □

The following terminology about *forcing* would be important for our discussion in the next chapter of the paper. So let us recall some definitions and specifically the countable chain condition (ccc) on partial order (as discussed in [6] in the context of *forcing conditions*[201]).

Definition 64 - Forcing "prerequisites".

Let M be a transitive model of ZFC, the *ground model.* In M, let us consider a nonempty partially ordered set $(P, <)$.

(i) We call $(P, <)$ a *notion of forcing* and the elements of P *forcing conditions.*

(ii) We say that p is *stronger* than q if $p < q$.

(iii) If p and q are conditions and there exists r s.t. both $r \leq p$ and $r \leq q$, then p and q are *compatible*; otherwise they are *incompatible.*

(iv) A set $D \subset P$ is *dense* in P if for every $p \in P$ there is $q \in D$ such that qp.

(v) A set $W \subset P$ is an *antichain* if its elements are pairwise incompatible.

Using the above notion of antichain one can formulate the countable chain condition, which would allow to somehow restrict set P as "not too large" and even more. The original idea asks - if X is a dense linearly ordered set and every disjoint collection of open intervals in X is at most countable

[200] which invokes *AC* - see [19], but we work in $ZF + DC$

[201] See chapter *14. Forcing* on p. 201-203 in [6]

(X satisfies ccc), then is P isomorphic to real line? This question is called the Suslin's Problem. In topology[202] (specifically, order theory) - where W is called *strong downwards antichain* - an antichain in which no two distinct elements have a common lower bound in P, that is - $\forall x, y \in W : x \neq y \implies \exists z \in P \wedge z \leq x \wedge z \leq y$, which is essentially the same as point *(v)* in the above *Definition 64*).

Definition 65 - Countable Chain Condition (ccc).

A partially ordered set $(P, <)$ (forcing notion) is said to satisfy the *Countable Chain Condition*, or to be *ccc*, if every (strong) antichain in P is at most countable.

Next we show how to construct partial order on $R_{<\omega_1}$. We try to align towards contiguity ideas in *Definition 56*, so that later one partial order can be refined and extended by the other.

Definition 66 - Substring-Partition Partial order (sppo) on $R_{<\omega_1}$.

Let $(R_{<\omega_1}, P_\varepsilon, \pi)$ be a replicata, s.t. $R_{<\omega_1} = S_{UF} \cup S_{FC} \cup S_{UC}$, where:

- S_{UF} be a set of all *unfair finite* SEF equivalence classes in $R_{<\omega_1}$,
- $S_{FC} = S_{PF} \cup S_{IF}$ be a set of all *fair countable* SEF equivalence classes in $R_{<\omega_1}$, which can be partition into a set of *perfect fairs* S_{PF} and *imperfect fairs* S_{IF},
- S_{UC} be a set of all *unfair countable* SEF equivalence classes in $R_{<\omega_1}$.

Then, one can define partial order $P_\varepsilon = (R_{<\omega_1}, <)$ using substring relation ε acting on SEFs in $R_{<\omega_1}$ and benefitting from the above partition of $R_{<\omega_1}$:

(i) **by substring**: $\forall s_x, s_y \in R_{<\omega_1} : s_x \; \varepsilon \; s_y => s_x < s_y$

(ii) **by partition (kind)**:

  (a) perfect fairs are less than imperfect fairs: $s_x \in S_{PF} \wedge s_y \in S_{IF} => s_x < s_y$

  (b) fairs are less than unfairs: $s_x \in S_{FC} \wedge s_y \notin S_{FC} => s_x < s_y$

  (c) unfair finite are less than unfair countable: $s_x \in S_{UF} \wedge s_y \in S_{UC} => s_x < s_y$

This is called *substring-partition partial order* (sppo).

[202] Another typical idea sharing the same purpose of working with sets that are "not too large" also comes from topology, and it is obviously the notion of *compactness*.

Theorem 67 - $R_{<\omega_1}$ with sppo satisfies ccc.

Let $(R_{<\omega_1}, P_\varepsilon, \pi)$ be a replicata with partial order P_ε defined by substring relation on SEFs and partition ordering $S_{PF} < S_{IF} < S_{UF} < S_{UC}$ (see *Definition 66*). Then, $R_{<\omega_1}$ satisfies countable chain condition.

Proof. (Assume $ZF + DC$). Partial order is defined according to *Definition 66*. Note that (a-c) points in (ii.) are not superfluous (or illustrative), but they are stronger than the consequence of substring relation ε applied on SEFs alone: $\forall s_x, s_y \in R_{<\omega_1} : s_x \ \varepsilon \ s_y => s_x < s_y$. The requirement $s_x \ \varepsilon \ s_y$ does not have to be always met, but we can still use partition information to always induce linear order[203]: $S_{PF} < S_{IF} < S_{UF} < S_{UC}$. This situation simplifies the rest of the proof.

Now given the induced linear order by $R_{<\omega_1}$ partition, one can exclude antichains across multiple partitions and consider each disjoint subset individually. We have that $R_{<\omega_1}$ is ccc iff partial order $P_\varepsilon = (X, <)$ is ccc, where $X \in \{S_{PF}, S_{IF}, S_{UF}, S_{UC}\}$. So we can break our proof as following:

- $(S_{PF}, <)$ is ccc: follows from the fact that every perfectly fair is a part of some GCGA table (essentially a filter) of almost adjoint perfectly fairs - see the *Lemma 59* and *Lemma 45*. Furthermore, uncountability of S_{PF} implies that each perfectly fair string is dense, i.e.: $\forall s_y \in S_{PF}, \exists s_x \in S_{PF} : s_x \ \varepsilon \ s_y \implies s_x < s_y$, where s_x is a shared prefix somewhere backwards on the context that was formed by scanning the input to construct GCTA table (Recall that the statement that such table exists is equivalent to DC - see *Theorem 44*.)

- $(S_{IF}, <)$ is ccc: by definition $\forall s_x \in S_{IF}, \exists s_y \in S_{IF} : s_y \ \varepsilon \ s_x \wedge s_y = u \cdot u$, where $u \in S_{PF}$. Hence, there is always a perfect fair that would be less than any imperfect fair - $\forall s_x \in S_{IF} : \exists u \in S_{PF}$, s.t. $u < s_x$. So it is not possible to form even a countable antichain in S_{IF} for sppo P_ε on S_{IF}.

- $(S_{UF}, <)$ is ccc: Given that $|S_{UF}| = \aleph_0$ it follows trivially that any subset of S_{UF} cannot form an uncountable antichain.

- $(S_{UC}, <)$ is ccc: Comes from two points:

  (a) By construction, each unfair countable formula $s_x \in S_{UC}$ is formed by replacing some finite substring if unfair finite formula $s_y \in S_{UF}$. Hence, for every s_y there is some finite s_x "template" (or layout) formula, which would be always lesser $s_x < s_y$ (by definition of P_ε - see *Definition 66*)

[203] Note that if we want to define a lattice then such linear order would need some adoption like fixing "(0)" as bottom and "(1)" as top

(b) Note that every $s_x \in S_{UC}$ is also inherently dense (upwards and downwards) as every fair substring s_v must be also a substring of some greater imperfectly fair s_k, which in turn, must be a substring of some unfair countable $s_u \in S_{UC} : s_k \ \varepsilon \ s_u$, resulting in $s_x < s_u$ (and vice versa).

□

The final evidence demonstrating that $R_{<\omega_1}$ is ccc wraps up this subsection. In concluding remarks, sets of ω_0 and ω_1 cardinalities are referred to as smaller sets, while sets of ω_2 and greater cardinality are considered by us as not so small or simply much bigger sets.

6.9 Fair Boolean Algebras

Next results will be required for the construction of a replicated model in the *A replicated Set Theory model deciding CH* section. Let's begin by reviewing some notation for boolean operations (depending on the context).

Definition 67 - Bitwise Boolean Operations on Binary Strings.

Let B_α be a set of all binary strings of non-zero length $\alpha \in Ord$:

$$B_\alpha = \{s : s \in \{0,1\}^\alpha\}$$

Then, boolean operations $\neg, \vee, \wedge, \oplus, \overline{\wedge}$ on any two binary strings (aligned by their length and index positions) can be defined in a bitwise manner. Namely, $\forall u, v, z \in B_\alpha$ and $\forall i \in I(\alpha)$:

$$\begin{aligned}
z = \neg u &\Leftrightarrow z(i) = \neg u(i), && \text{(NOT)}\\
z = u \vee v &\Leftrightarrow z(i) = u(i) \vee v(i), && \text{(OR)}\\
z = u \wedge v &\Leftrightarrow z(i) = u(i) \wedge v(i), && \text{(AND)}\\
z = u \oplus v &\Leftrightarrow z(i) = u(i) \oplus v(i), && \text{(XOR)}\\
z = u \,\overline{\wedge}\, v &\Leftrightarrow z(i) = u(i) \,\overline{\wedge}\, v(i), && \text{(NAND)}
\end{aligned}$$

where logical operations on individual bits[204] $u(i)$ and $v(i)$ are:

- **NOT** - *logical NOT*, *logical negation* or equivalently *binary inverse* (INV);
- **OR** - *logical OR* or equivalently *binary sum*;

[204] Also see *cursor functions* in *Definition 3* (much earlier in the paper) for notation to access individual bits

- **AND** - *logical AND* or equivalently *binary multiplication*;
- **XOR** - *exclusive logical OR*, *symmetric difference* or equivalently *exclusive OR*;
- **NAND** - *negated logical AND* or *NOT AND*.

We refer to such string-boolean operations (when applied on the whole binary string in a bulk) as *bitwise boolean operations*.

We want to adhere to the following convention:

- we use natural number as subscript and index if α is finite $\alpha = n, n \in \mathbb{N}$ (also $|B_n| \in \mathbb{N}$);
- when the subscript α is not specified, it is assumed that $\alpha = \omega$, and in this context, $\mathbb{B} = B_\omega$.

Furthermore, throughout the paper we are interchangeably using the following equivalent notation for binary operations, which, ideally, deserves some comment as well:

(a) $\vee, \wedge, \neg$ in the context of binary strings (and their Boolean algebras);

(b) $+, \cdot, -$ in the context of abstract Boolean algebras;

We also use (a) in formulas of formal logical language (e.g. FOL or Set Theory). For example, in this sense notation (a) for binary operations is used in *Lindenbaum algebra*, which can be defined [6] on the equivalence of all sequences of FOL language[205] . So, for the sake of clarity, let us recall a detailed definition of what *Boolean algebra* is from [6].

Definition 68 - Boolean Algebra.

A Boolean algebra is a set B with at least two elements[206], 0 and 1, endowed with binary operations $+$ and $\cdot$, and a unary operation $-$:

$$(B, +, \cdot, -, 0, 1)$$

The above Boolean operations satisfy the following axioms:

$$\begin{aligned}
u + v &= v + u, & u \cdot v &= v \cdot u, && \text{(commutativity)}\\
u + (v + w) &= (u + v) + w, & u \cdot (v \cdot w) &= (u \cdot v) \cdot w, && \text{(associativity)}\\
u \cdot (v + w) &= u \cdot v + u \cdot w, & u + (v \cdot w) &= (u + v) \cdot (u + w), && \text{(distributivity)}\\
u \cdot (u + v) &= u, & u + (u \cdot v) &= u, && \text{(absorption)}\\
u + (-u) &= 1, & u \cdot (-u) &= 0. && \text{(complementation)}
\end{aligned}$$

[205] FOL language includes propositional logic, which was mentioned in *Theorem 55* as corresponding to a *binary mess*.

[206] in the context of binary strings we used $\mathbf{0}$ = "(**0**)" and $\mathbf{1}$ = "(**1**)"

Every Boolean algebra gives rise to a Boolean ring [6]. The opposite is not true as XOR is not a universal logical gate. But as we will see, there is a logical operation that (if the set is closed under that operation) does give rise to a Boolean Algebra, defined on the same set.

Lemma 68 - NAND Universality.

All boolean operations including $\wedge, \vee, \neg$ (AND, OR, NOT) can be expressed via NAND $\overline{\wedge}$.

Proof. This is a known fact in logic and computer science. This property is called universality, when all other logical operations or gates can be expressed by the few or one gate (as in case of NAND $\overline{\wedge}$ operation). The proof follows directly from the logical formulas below $\forall a, b \in \{0, 1\}$:

$$\begin{aligned} \neg a &= a \mathbin{\overline{\wedge}} a && \text{(NOT)} \\ a \wedge b &= \neg(a \mathbin{\overline{\wedge}} b) && \text{(AND)} \\ a \vee b &= \neg a \mathbin{\overline{\wedge}} \neg b && \text{(OR)} \\ a \oplus b &= (a \mathbin{\overline{\wedge}} (a \mathbin{\overline{\wedge}} b)) \mathbin{\overline{\wedge}} (b \mathbin{\overline{\wedge}} (a \mathbin{\overline{\wedge}} b)) && \text{(XOR)} \end{aligned}$$

and so on. Recall that the combination of logical gates AND and NOT are also universal. □

Corollary 69 - NAND Closure.

If B be a set of at least two elements $\{0, 1\}$ closed under boolean operation NAND $\overline{\wedge}$ (or, equivalently, AND $\cdot$ and NOT $-$). Then, B is also closed under $\wedge, \vee, \neg$ (AND, OR, NOT) operations.

Proof. Follows from *Lemma 68.* □

Theorem 70 - Boolean Algebra on Binary Strings.

Let B_α be a set of all binary strings of non-zero length $\alpha \in Ord$ and $\alpha \leq \omega$:

$$B_\alpha = \{s : s \in \{0, 1\}^\alpha\},$$

which is closed under at least one bitwise boolean operation $\overline{\wedge}$ (or, equivalently, two operations $\neg$,). Then, there exists a Boolean algebra defined on those binary strings:

$$(B_\alpha, \wedge, \vee, \neg, \mathbf{0}, \mathbf{1})$$

Proof. Let **0** and **1** be binary strings of length α, consisting either only of zeros, or, only of ones, respectively. Note that **0** and **1** strings are necessarily included in every B_α, given that the set contains all binary strings of non-zero length α. It also means that the cardinality must be $|B_\alpha| = 2^\alpha$. We claim that if B_α is closed under $\wedge, \vee, \neg$ and satisfies all axioms in *Definition 68* for each α, then B_α is a Boolean algebra $(B_\alpha, \wedge, \vee, \neg, \mathbf{0}, \mathbf{1})$.

We are going to proof this by induction on length α:

1. $\alpha = 1$: B_1 is a trivial Boolean algebra of two (we ignore zero length case).

2. $\alpha < \omega$: A finite case is true by induction on $n \in \mathbb{N}$ with previous cases as base. Assume $\alpha = n$ is true and their exists a Boolean algebra B_n. Let us show $\alpha = n + 1$.

   (i) Define bottom and top algebras $B_b = \{\text{“}s \cdot 0\text{”}\}$ and $B_t = \{\text{“}s \cdot 1\text{”}\}$ by concatenation of $\{0\}$ and $\{1\}$ bits to the binary strings in B_n, so that:
   $$B_{n+1} = B_b \cup B_t$$

   (ii) Concatenation above can be described with isomorphism $f : \{0,1\}^n \times \{0,1\} \to \{0,1\}^n$, s.t. $B_b = f(B_n, 0)$ and $B_t = f(B_n, 1)$.

   (iii) Observe that both B_b and B_t remain valid Boolean algebras, since axioms in *Definition 68* are idempotent to the last $n+1$ bit of the string remaining the same (when bitwise boolean operation are applied on the last bit). In other words, if the last bit is treated as a constant c, we have that B_n being valid Boolean algebra implies that $f(B_n, c)$ is also an algebra, since f is an isomorphism.

   (iv) We will use the concept of isomorphisms to show that the set B_{n+1}, constructed by concatenating a bit to strings in B_n, forms a Boolean algebra under the operations $\wedge, \vee$, and $\neg$.

   (iv) **Merging B_b and B_t using isomorphism f:**

      i. **Isomorphic Representation:** The set B_{n+1} can be described as $B_{n+1} = B_b \cup B_t$, which corresponds to $\{0,1\}^n \times \{0,1\}$. Here, each string in B_{n+1} can be uniquely identified as a tuple (s, b) where $s \in B_n$ and $b \in \{0,1\}$, representing the concatenation of the binary string s with the bit b.

      ii. **Isomorphism Construction:** Define the function $f : B_n \times \{0,1\} \to B_{n+1}$ by $f(s, b) = s \cdot b$. This function is an isomorphism because it is a bijection (one-to-one and onto) and preserves the structure of the Boolean operations when applied element-wise. Furthermore, f preserves monotonicity induced by $B_b < B_t$ - meaning that $\forall u, v \in B_{n+1}$ we have $f(u) \leq f(v)$ whenever $u \leq v$, since we can decide on the

corner case $u \in B_b$ and $v \in B_b$ implies $f(u) \leq f(v)$ (or by comparing the last $n+1$ bit).

iii. **Boolean Operations via Isomorphism:** For any $s, t \in B_n$ and $a, b \in \{0,1\}$, define the operations on B_{n+1} as:

$$f(s,a) \wedge f(t,b) = f(s \wedge t, a \wedge b)$$

$$f(s,a) \vee f(t,b) = f(s \vee t, a \vee b)$$

$$\neg f(s,a) = f(\neg s, \neg a)$$

These operations are well-defined due to the closure and Boolean operations properties of B_n and $\{0,1\}$.

iv. **Algebra Properties:** Since both B_n and $\{0,1\}$ are Boolean algebras, the product $B_n \times \{0,1\}$, under these operations, satisfies the axioms of a Boolean algebra:

- **Closure:** Shown by the definition of operations.
- **Associative, Commutative, Distributive:** Inherited from B_n and $\{0,1\}$.
- **Identity and Inverse Elements:** Follow directly from the identities and inverses in B_n and $\{0,1\}$.

3. $\alpha = \omega$: By constructing the set B_{n+1} through the isomorphism $f : B_n \times \{0,1\} \to B_{n+1}$ and establishing that it satisfies the Boolean algebra properties under the defined operations, we conclude that B_{n+1} forms a Boolean algebra. The induction can similarly be carried forward to any finite n and extended to the countably infinite case $\alpha = \omega$, arguing by the infinite extension of finite Boolean algebras, given proper handling of limits and closure under operations.

   However, an alternative line of argument just for the special case $\alpha = \omega$ would be in the context of $(R_{<\omega_1}, P, \pi)$ replicata:

   (i) $B_\omega = \mathbb{B} = \pi(R_{<\omega_1})$

   (ii) Elements of $\mathbb{B}$ correspond one-to-one to Cantor Set and $|\mathbb{B}| = 2^{\aleph_0}$.

   (iii) $BPI(R_{<\omega_1}) \implies SRT(R_{<\omega_1})$, where SRT stands for Stone Representation Theorem[207]:
   Every boolean algebra is isomorphic to an algebra of sets or $g : B_\omega \to S(B_\omega)$, where $S(B_\omega)$ is a dual Stone space for B_ω.

   (iv) As a compact totally disconnected Hausdorff space, the Cantor set is an example of a Stone space. The latter implies that $\mathbb{B} = S(B_\omega)$ and B_ω is corresponding Boolean algebra on the set of $\{0,1\}^\omega$ strings.

[207] see *Lemma 56* and [6] p. 81

Hence, the theorem is proven, establishing that B_α is a Boolean algebra for each $\alpha \in Ord$ where $\alpha \leq \omega$. □

Corollary 71.

Each Boolean algebra on binary strings of length $\alpha \in Ord$ and $\alpha \leq \omega$: $(B_\alpha, \wedge, \vee, \neg, \mathbf{0}, \mathbf{1})$ isomorphic to an abstract Boolean algebra $(B_\alpha, +, \cdot, -, 0, 1)$, where $|B_\alpha| = 2^\alpha$.

Proof. Let $h : B_\alpha \to B_\alpha$ be an automorphism provided by identity. *Definition 67* implies that boolean bitwise operations on binary strings $\wedge, \vee, \neg$ are equivalent to $+, \cdot, -$. So after rewriting the formulas, all axioms in *Definition 68* are satisfied, and the structure remains the same. □

Definition 69 - Partial Order.

A *partial order* (p.o.) is a binary relation $\leq$ over a set P that is *reflexive*, *antisymmetric*, and *transitive*. That is, for any elements $a, b, c \in P$, the following conditions hold:

- **Reflexivity:** $a \leq a$
- **Antisymmetry:** If $a \leq b$ and $b \leq a$, then $a = b$
- **Transitivity:** If $a \leq b$ and $b \leq c$, then $a \leq c$

We also tend to abuse the notation. If partial order $\leq$ is defined on a set X and $P = X$ we note this fact interchangeably: either as $(P, \leq)$ or $(X, \leq)$, or even as $P(X, \leq)$. But sometimes we just write $P_\leq$ to mean the same as $\leq$. If $P_\leq$ is defined on $Y \supset X$, then we write $P_\leq(X)$ to mean that partial order is restricted to subset or $(X, \leq)$.

Furthermore, let us also recall from [6] that one can use Boolean-algebraic operations on B to define a partial order $P(B)$ based on the following correspondence:

$$u \leq v \Leftrightarrow u - v = 0$$

We can formulate this as a lemma.

Lemma 72 - Partial Order on Boolean Algebras.

If B is a Boolean algebra. Then, there exists a corresponding partial order $P(B)$.

Proof. 1. Assume B is a Boolean algebra. Define $P(B)$ as the set of elements of B.

2. Define the partial order $\leq$ on $P(B)$ by $u \leq v$ iff $u \wedge v = u$ (which is equivalent to $u - v = 0$ in a Boolean algebra, where $u - v$ is defined as $u \wedge \neg v$).

3. This relation is reflexive ($u \leq u$), antisymmetric (if $u \leq v$ and $v \leq u$, then $u = v$), and transitive (if $u \leq v$ and $v \leq w$, then $u \leq w$).

4. Hence, $P(B)$ equipped with $\leq$ forms a partial order.

□

The converse is not necessarily true, unless partial order is also bounded distributive *lattice* with complementation (each element is its own complement).

Definition 70 - Lattice on Partial Order.

A partially ordered set $(L, \leq)$ is a *lattice* if it satisfies the following criteria:

(i) **Existence of Join and Meet**: For every two elements $a, b \in L$, there exist unique elements called *supremum* (join or the least upper bound) and *infimum* (meet or the greatest lower bound), of a and b, denoted by:

(a) $sup(a, b) := a \curlyvee b$, also defined as $a = a \curlyvee b \Leftrightarrow a \leq b$

(b) $inf(a, b) := a \curlywedge b$, also defined as $b = a \curlywedge b \Leftrightarrow a \leq b$

Thus $\curlywedge$ and $\curlyvee$ binary operations on L.

(ii) **Monotonicity**: The binary operations $\curlyvee$ and $\curlywedge$ are monotone with respect to the partial order $\leq$, meaning:

$$a_1 \leq a_2 \text{ and } b_1 \leq b_2 \implies a_1 \curlyvee b_1 \leq a_2 \curlyvee b_2 \text{ and } a_1 \curlywedge b_1 \leq a_2 \curlywedge b_2.$$

(iii) **Closure under Finite Subsets**: For every finite, non-empty subset $A \subseteq L$, both $\sup(A)$ and $\inf(A)$ exist, thereby ensuring the lattice operations close over finite subsets.

The following additional properties extend the basic lattice framework:

(iv) **Boundedness**: A *bounded lattice* is characterized by the existence of both a greatest element (*maximum* or *top*, denoted $\top$) and a least element (*minimum* or *bottom*, denoted $\bot$), satisfying $x \in L : \bot \leq x \leq \top$. In algebraic terms, this extends $(L, \curlyvee, \curlywedge)$ to $(L, \curlyvee, \curlywedge, \bot, \top)$, where $\bot$ and $\top$ serve as identity elements for $\curlyvee$ and $\curlywedge$ respectively.

(v) **Completeness**: The lattice L is termed a complete lattice if for every non-empty subset A of L, regardless of whether A is finite or infinite, both $\sup(A)$ and $\inf(A)$ exist.

(vi) **Distributivity**: For any $x, y, z \in P$, we have:

$$x \curlywedge (y \curlyvee z) = (x \curlywedge y) \curlyvee (x \curlywedge z) \quad \text{(distributivity of } \curlywedge \text{ over } \curlyvee\text{)}.$$

$$x \curlyvee (y \curlywedge z) = (x \curlyvee y) \curlywedge (x \curlyvee z) \quad \text{(distributivity of } \curlyvee \text{ over } \curlywedge\text{)}.$$

(vii) **Complementation**: For every element $a \in L$, there exists a complement denoted $\neg a$, s.t. $a \curlywedge \neg a = \bot$ and $a \curlyvee a = \top$. This complement can be defined in terms of the partial ordering relation as the set-theoretic complement.

Lemma 73 - Boolean Algebra on Lattice.

If $(L, \curlyvee, \curlywedge, \bot, \top)$ is a bounded lattice that satisfies complementation and distributivity (*Definition 70*), then $B(L, +, \cdot, -, 0, 1)$ is a Boolean algebra that can be defined $\forall a, b \in L$ as:

(a) $a + b = a \curlyvee b$

(b) $a \cdot b = a \curlywedge b$

(c) $-a = \neg a$

(d) $0 = \bot$

(e) $1 = \top$

Proof. The proof is straightforward check of Boolean algebra axioms in *Definition 68*. Since the bounded distributive lattice $(L, \curlyvee, \curlywedge, \bot, \top)$ with complementation satisfies the properties of associativity, commutativity, distributivity, identity, and complements with the defined operations, it forms a Boolean algebra $B(L, +, \cdot, -, 0, 1)$. □

To benefit from the above proof, we will extend partial order P_ε in the discussed way.

Definition 71 - Fair Partial Order.

Let $(R_{<\omega_1}, P_\varepsilon, \pi)$ be a replicata and $P_{FC} := P_\varepsilon(S_{FC})$ be called *fair* p.o., defined by a sppo P_ε, when restricted to the fair string subset $S_{FC} \subset R_{<\omega_1}$.

Lemma 74 - Fair Lattice on Partial Order.

Let $(R_{<\omega_1}, P_\varepsilon, \pi)$ be a replicata. If $\leq$ is a fair p.o. $P_\varepsilon(S_{FC})$ restricted to the fair string subset $S_{FC} \subset R_{<\omega_1}$, then there exists a *fair* lattice

$$(L_F, \curlyvee, \curlywedge)$$

that can be defined on $P_\varepsilon(S_{FC})$.

Proof. We want to proof that if one takes $\leq$, which is defined as $P_\varepsilon(S_{FC})$ (see *Definition 66*) and use $\leq$ to define $(L_F, \curlyvee, \curlywedge)$ as specified by (i-iii) properties in *Definition 70*, then it is a lattice. We will do so, by showing that $(L_F, \curlyvee, \curlywedge)$ satisfies property (i), which implies (ii) and (iii):

(i) Recall, that p.o. $\leq$ is dense on both S_{PF} and S_{IF} (as discussed in the proof of *Theorem 67*). This, coupled with $S_{FC} = S_{PF} \cup S_{IF}$, implies existence of meets and joins for all $x, y \in S_{FC}$.

(ii) Monotonicity is implied by (i).

(iii) It follows by an induction argument from (i) that every non-empty finite subset of a lattice has a least upper bound and a greatest lower bound.

□

Lemma 75 - Fair Boolean Algebra.

If $(L_F, \curlyvee, \curlywedge)$ is fair lattice on $P_\varepsilon(S_{FC})$, then it can be extended to complete Boolean algebra with restricted $\curlyvee, \curlywedge$ as $\dot{\curlyvee}, \dot{\curlywedge}$, namely:

$$(B_F, \dot{\curlyvee}, \dot{\curlywedge}, \neg, \bot, \top),$$

where:

- $\neg$ is defined as a bitwise boolean operation(see *Definition 67*);
- $\bot$ and $\top$ are countable binary strings, consisting either only of zeros, or, only of ones, respectively.
- $\dot{\curlywedge}$ is restricted as $\dot{\curlywedge} := x \curlywedge y$, where either $x \in S_{PF} \wedge y \in S_{IF}$ or $x \in S_{IF} \wedge y \in S_{PF}$.
- $\dot{\curlyvee}$ is restricted as $\dot{\curlyvee} := x \curlyvee y$, where either $x \in S_{PF} \wedge y \in S_{IF}$ or $x \in S_{IF} \wedge y \in S_{PF}$.

We refer to such complete Boolean algebra as fair.

Proof. The result follows if we apply *Lemma 73*. Lemma 74 implies that $(L_F, \curlyvee, \curlywedge)$ already satisfies properties (i - iii) in *Definition 70*. This means we need to show the remaining properties (iv - vii):

(iv) We extend the L_F as $B_F := P_\varepsilon(S_{FC}, \leq) \cup \{\bot, \top\}$, so that $\forall x, y \in B_F : \bot \leq x \leq \top$.

(v) According to discussion in *Theorem 67* P_ε is dense on S_{PF} as well as S_{IF}. Note that the lattice is bounded with $\bot$ and $\top$ and $S_{FC} = S_{PF} \cup S_{IF}$ implies that if $X \subset S_{FC}$ then $inf(X)$ and $sup(X)$ are defined by induction.

(vi) $\dot{\curlyvee}, \dot{\curlywedge}$ are restricted versions of $\curlyvee, \curlywedge$ essentially reinterpreting $P_\varepsilon(S_{FC})$ within symmetry induced by the linear order $\bot < S_{PF} < S_{IF} < \top$. Given that $S_{FC} = S_{PF} \cup S_{IF}$, let's fix x and y, so that $x \in S_{PF} \wedge y \in S_{IF}$ implies $(x \dot{\curlywedge} y) = x$ and $(x \dot{\curlyvee} y) = y$ (or vice versa if $x \in S_{IF} \wedge y \in S_{PF}$). The latter guarantees distributivity.

(vii) Follows from the above extension as we defined bitwise boolean $\neg$ operation.

□

6.10 Quasi-Large Cardinals

Working with $PIT(R_{<\omega_1})$ offers a nice segue to many crucial proofs, as highlighted in the literature. This approach not only aids in establishing important results but also highlights potent implications for *fair determinacy* and the *partition of* $R_{<\omega_1}$, as presented in *Corollary 48* and *Lemma 50* respectively.

Let us recall notions about partition (in the context of Ramsey Theorem) and cardinal compactness [6].

The symbol $\kappa \longrightarrow (\lambda)^n_m$ (read: κ *arrows* λ) denotes the following *partition property*:

Definition 72 - Generalized Ramsey partition property (arrow notation).

Every partition of $[\kappa]^n$ into m pieces has a homogeneous set of size λ. In other words, every $F : [\kappa]^n \longrightarrow m$ is constant on $[H]^n$ for some $H \subset \kappa$ such that $|H| = \lambda$.

Using arrow notation Ramsey Theorem will be expressed as following.

Theorem 76 - Ramsey Theorem.

$\aleph_0 \longrightarrow (\aleph_0)^n_m$, where $n, m \in \omega$

When $m = 2$ we simply write $\kappa \to (\lambda)^n$.

If the cardinal satisfies certain compactness theorem for infinitary languages, then it is called "weakly compact"[208]. A weakly compact cardinal is uncountable and fulfills the partition property $\kappa \longrightarrow (\kappa)^2$. Such cardinals are also inherently inaccessible, a property derived assuming ZFC, where strong limit cardinals, as discussed in various lemmas and definitions, play a crucial role.

[208] See *Definition 9.8* and *Lemma 9.9* in [6]

Let us assume $ZF + DC$ instead and continue with a hypothetical scenario of this paper where the continuum, denoted as $2^{\aleph_0}$, equals $\aleph_2$—an uncountable, regular cardinal that does not qualify as a strong limit cardinal yet exhibits properties similar to those of a weakly compact cardinal, such as the partition and tree properties—we encounter a highly unconventional and non-standard situation in set theory. This hypothesis, which negates the CH by positing $2^{\aleph_0}$ as $\aleph_2$ or larger, adheres to standard properties of being uncountable and regular but not a strong limit cardinal, as $\aleph_1 < \aleph_2$ but $2^{\aleph_1} \geq \aleph_2$.

If $\aleph_2$ possessed the partition property of a weakly compact cardinal, it would suggest that for some functions $f : [\aleph_2]^2 \to 2$, there exists a subset H of cardinality $\aleph_2$ where f is constant on $[H]^2$. Additionally, if $\aleph_2$ exhibited the tree property, it would mean every tree of height $\aleph_2$ with levels smaller than $\aleph_2$ contains a branch of length $\aleph_2$. These adjustments would necessitate significant changes to standard set theory, introducing new combinatorial principles and potentially altering the structure of the set-theoretic universe.

To capture this scenario semantically, we consider a model N of $ZF + DC$ in which $\aleph_1$ is a measurable cardinal, yet not a large cardinal in the traditional sense (since it is accessible). This situation demonstrates that certain large cardinal properties can occur at accessible cardinals within $ZF + DC$, potentially consistent with the existence of large cardinals but not necessarily requiring them.

We formalize this notion with the following definition.

Definition 73 - Quasi-Large Cardinal.

Let N be a model of $ZF + DC$. An ordinal $\gamma \in \mathrm{Ord}^N$ is called a *quasi-large cardinal* in N if γ is an accessible cardinal in N that satisfies large cardinal properties typically associated with inaccessible cardinals. For example, γ in N carries a normal measure.

This definition captures the essence of a cardinal that, while accessible, exhibits characteristics of large cardinals. It allows us to discuss cardinals that behave like large cardinals without necessitating their existence, aligning with the nuances possible in ZF without the Axiom of Choice.

Now, let us take a closer look at another remarkable consequence of *partition of* $R_{<\omega_1}$ (*Lemma 50*).

Definition 74 - Weakly Compact Cardinal.

A cardinal κ is *weakly compact* if it is uncountable and satisfies the partition property $\kappa \longrightarrow (\kappa)^2$.

Theorem 77 - Weak compactness of ɕ.

Let $\mathfrak{c} = 2_0^{\aleph}$ be cardinality of the continuum. We state that $\mathfrak{c}$ is a (quasi) weakly compact cardinal. Namely, $\mathfrak{c} \longrightarrow (\mathfrak{c})^2$.

Proof. Follows from $\mathfrak{c}$ being a quasi-measurable cardinal (see *Theorem 79* below). □

Finally, we will look at defining a two-valued measure on $R_{<\omega_1}$. Recall[209] from [6]:

Lemma 78 - κ-complete Ultrafilter.

Let κ be the least cardinal with the property that there is a nonprincipal σ-complete ultrafilter on κ, and let U be such an ultrafilter. Then U is κ-complete.

This lemma allows for a more succinct definition of a *measurable cardinal*.

Definition 75 - Measurable cardinal.

An uncountable cardinal κ is *measurable* if there exists a κ-complete nonprincipal ultrafilter U on κ.

Also note further results from [6] that in ZFC:

- *every measurable cardinal is inaccessible*;
- *every measurable cardinal is weakly compact*;
- *every measurable cardinal carries a normal measure.*

In terms of the above *Definition 73*, one can easily derive the notion *quasi-measurable* cardinal (assuming that it exists in a large enough model).

Definition 76 - Quasi-measurable cardinal.

An uncountable cardinal κ is *quasi-measurable* if it is quasi-large (see *Definition 73*) and there exists a κ-complete nonprincipal ultrafilter U on κ.

Theorem 79 - Normal measure on $\mathfrak{c}$.

Let $\mathfrak{c} = 2^{\aleph_0}$ be cardinality of the continuum. We state that $\mathfrak{c}$ is quasi-measurable. Furthermore, there exists a two-valued measure $\mu : \mathfrak{c} \to \{0, 1\}$ defined as null-value for $S \subset \mathfrak{c}$, namely:

$$\mu(x) = \begin{cases} 0 & \text{if } x \in S; \\ 1 & \text{otherwise.} \end{cases}$$

[209] Also see *Theorem 10.1 (due to Ulam)* and *Lemma 10.2* on p.126-128 in [6]

Proof. Assume $ZF + DC$. One approach for the proof would be to invoke $Ultrafilter(R_{<\omega_1})$ given that $|R_{<\omega_1}| = \mathfrak{c}$ (see *Theorem 55*). This will also allow to define the two-value measure on $R_{<\omega_1}$ (such as μ).

Another approach would be even more straight-forward. We can simply invoke *Theorem 46 of Perfect Determinacy* to define μ for $S_{PF} \subset R_{<\omega_1}$. This will result in the existence of the σ-complete $Ultrafilter(R_{<\omega_1})$. In fact such ultrafilter will be a normal measure carried by $\mathfrak{c}$.

Either way, it is obvious that if there exists a model of $ZF + DC$ in which $R_{<\omega_1}$ is constructable, it implies the existence of the quasi-measurable cardinal size of continuum $\mathfrak{c}$. □

7 A replicated Set Theory model deciding CH

It must be mathematical common knowledge by now that CH is independent of ZF. Specifically, an immediate and foundational example of the Set Theory (ZF) model satisfying CH would be Gödel's model L [6] - a constructable universe of all transitive sets which includes all ordinals. Obviously, only if one also accepts the axiom of constructability $V = L$, that every set in the universe V is constructible by following Gödel's approach. Paul Cohen result, in its turn, famously shows the independence of CH by creating a counter example model using forcing, when, assuming $con(ZF)$, $ZF + CH$ fails. Of course, another important[210] independence result produced by Cohen is the independence of AC, namely: $con(ZFC) \implies con(ZF + \neg AC)$.

On one hand, there is a strong sentiment among many modern mathematicians that there may be a missing axiom. It can be either coming from assuming existence of large cardinals. Or, such missing axiom can be MM^{++}, which, assuming $con(ZF)$, if considered in addition to the existing Set Theory ZFC decides CH [29]. On the other hand, assuming any additional axiom may still seem like a strong stance to adopt, even though we cannot hope to obtain a complete (and unique) theory with respect to forceability which extends $ZFC + CH$[211].

In this section we are going to put together all the main points that we have discussed until now. The result would be a $\neg CH$ model built with bijective representation of sets as infinite binary strings. Such arguably "simple" illustration [212] will rest on a special example of partial order that can be constructed from the infinite binary strings using substring relation (see *Definition 66* of sppo in previous section).

We would also look into whether or not our approach (to define such model) is consistent with the main generic forcing results. In fact, we would ask if CH can be decided for all $ZF + DC$ models. As already mentioned, forcing is a technique, which was initially proposed and used by Paul Cohen. It has evolved into a rich theory and became a main tool to obtain independence proofs [6].

[210] The following quote from [21] seems appropriate to explain the significance of it: "A mathematician of the present generation hardly considers the use of the Axiom of Choice a questionable method of proof. As a result of algebra and analysis going abstract and the development of new mathematical disciplines such as set theory and topology, practically every mathematician learns about the Axiom of Choice (or at least of its most popular form, Zorn's Lemma) in an undergraduate course. He also probably vaguely knows that there has been some controversy involving the Axiom of Choice, but it has been resolved by the logicians to a general satisfaction."

[211] Specifically, see the discussion and counterexamples on p.7 in [30] of Woodin Theorem 3.2.1 [31] on Ω-consistency of T extending theory $ZFC+$"$\exists$ *class many Wooding cardinals*" iff $T + CH \vdash \phi(p)$, where $\phi(p)$ is a Σ^2_1 statement with real parameter p.

[212] "If the Continuum Hypothesis fails then there should be a "simple", "definable" evidence for this failure." [32]

7.1 Constructing the replicated model

For all our purposes we assume N is either the whole universe V of $ZF+DC$ or a large enough portion[213] of the cumulative hierarchy $\{V_\alpha : \alpha \in Ord\}$, so that N is a well-founded transitive model of $ZF + DC$.

Let us check the plausibility of the following working scenario of M being a transitive submodel of N:

1. **assumption**: if ZF is consistent, then $ZF + DC$ is also consistent;
2. **independence**: ZFC is independent of $ZF + DC$;
3. **conclusion**: $M \models ZFC$ and is a transitive *submodel* of N: $M \subseteq N$.

Firstly, as usual step (1), we suppose that if ZF is consistent, then $ZF + DC$ is also consistent. In modern set-theoretic practice, one typically strengthens "ZF is consistent" to "there is a transitive model of ZF". Using standard reflection or constructibility arguments, one then obtains a transitive model N such that:

$$N \models ZF + DC$$

Secondly, as step (2), we rely on independence. It is known that ZFC is independent of $ZF + DC$ (See Problem 5.26 on p. 83 in [21], where a model is constructed in which the Axiom of Choice fails but Dependent Choice still holds[214].)

Finally, as conclusion for our scenario: from N one can form a transitive submodel $M \subseteq N$ satisfying

$$M \models ZFC$$

This arises, for instance, by taking the constructible universe inside N (often denoted L^N), which ensures the full AC holds in M even though it may fail in N. So the idea of constructing such or even similar submodel M of N seems plausible.

Next, we will complete the outline the main scenario in which we will operate as well as the structure for the rest of the section.

Let's begin by addressing the fundamental query: What is meant by a replicated model? Conceptually, a replicated model M is any transitive Set Theory model, such that, if $(P, \leq)$ is a notion of forcing in M (ground model), then it can be mapped by embedding $h : M \to N$ into superset

[213] Here, by large enough portion we mean that all ZF axioms plus DC can been "seen" as a part of the "smaller" $V_\alpha \subset V$ but still large enough transitive set.

[214] Problem 5.26 on p.83 in [21] describes such model of ZF "Let N be the class of all sets which are hereditarily definable from a countable sequence of ordinals. In N, the Axiom of Choice fails, and the Principle of Dependent Choices is true. Thus the Axiom of Choice is independent of the Principle of Dependent Choices."

model $N \supset M$ that satisfies existence of a replicata construct $N \models R_{<\omega_1}$ (see *Definition 61*) and $ran(h) \subseteq R_{<\omega_1}$. The fact that restriction of such embedding $h{\restriction}_P$ can be mapped into a replicata (existing in N) is called *replication*, meaning that if one takes the Set Theory language as it is interpreted in M, then formulates generic forcing notion on it, so that forcing language can be also interpreted on $R_{<\omega_1}$ structure (which is merely a collection of string replicators or SEF-language formulas built with sets living in N). One can also say that with such arrangement $R_{<\omega_1}$ offers to host a replica of $P \in M$ inside N, although M is already a submodel of N by construction.

Definition 77 - A replica host.

Let $(R_{<\omega_1}, P_\varepsilon, \pi)$ be a replicata and N a transitive model of Set Theory ZF. Also let $M \subset N$ be a submodel of N. Then, we call the Set Theory model N a *replica host* for M iff the following holds:

1. $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ with $R_{<\omega_1} \subset N$ (constructed in N);
2. M is a transitive model of Set Theory language, s.t. $M \subset N$;
3. $(P, \leq)$ is a generic notion of forcing in M (ground model);
4. There exists an embedding $h : M \to N$ into superset model (since M is a submodel of N), s.t. $h(P) \subseteq R_{<\omega_1}$;
5. Furthermore, we require restriction $h{\restriction}_P$ to be order preserving on $R_{<\omega_1}$.

Again, by the above definition since M is a submodel of N, there exists an embedding $h : M \to N$. This makes M kind of *inner-model*, but this is of minor importance as long as both M and N are transitive.

Definition 78 - A replicated model.

Let N be a transitive model of Set Theory ZF with submodel M. If N serves as replica host for M (as specified in Definition 77), then M is called a *replicated model*.

In the context of the above definition we will mean the same by interchangeably saying that:

- M is a *replicated model*;
- forcing replicas from M are interpreted in the replicata $R_{<\omega_1}$;
- $P \in M$ is a replicated forcing over $R_{<\omega_1}$;

- replicated forcing of M is hosted by N.

Given the above definitions and explanations, a complete outline for our scenario and the rest of the section looks as following:

(i) $\exists N : N \models ZF + DC$ (assuming ZF is consistent)

(ii) M is a submodel of N, s.t. $M \models ZFC$;

(iii) We want to show that N is a replica host for submodel M, so that $P \in M$ is a replicated forcing over $R_{<\omega_1}$:

  (a) $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$

  (b) $(P, \leq)$ is a generic notion of forcing in M (ground model);

  (c) $h : M \to N$ is an embedding, s.t. $h(P) \subseteq R_{<\omega_1}$ and restriction $h\restriction_P$ is order preserving on $R_{<\omega_1}$;

(iv) We claim that Martin Maximum (MM) as well as stronger statement MM^{++} hold in M. Notably [29], MM decides cardinality of the continuum (implying $\neg CH$) and, in fact, confirms Goedel's conjecture $2^{\aleph_0} = \aleph_2$.

Let us break up the scenario into coverage per subsections.

- Item (i) of the above scenario is already discussed in this section, and we have fixed N as a transitive model of $ZF+DC$. The following subsection - *Countable binary strings and replicators in* $R_{<\omega_1}$ - is focused on showing that $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$. Namely, that one can construct a replicata in $ZF + DC$ model. It culminates with *Theorem 85 - Constructability of replicata*, claiming that one can construct replicata in N. This goes in line with work in previous section, where most of our results were obtained primarily assuming $ZF + DC$.

- Item (ii) in the above scenario is covered by section - *A replicated model of Set Theory.* We define M as boolean valued model on complete fair algebra in $R_{<\omega_1}$ and show that $M \models ZFC$. Essentially, M is also a submodel of N, which implies most of the item (iii) in our scenario.

- The story is concluded with item (iv) in the last subsection - *Replicating Martin Maximum.*

7.2 Countable binary strings and replicators in $R_{<\omega_1}$

In general, we assume that all coding is done in Set Theory language. We assume that $\mathbb{B}$ and $R_{<\omega_1}$ consists of ZF sets, but will also work with their high-level view as strings. In literature strings are often defined as sequences in [5, 6]. Unfortunately, ambiguity may arise from many options in notation

like $\{0,1\}^\omega$, $\langle 1,0,1,..\rangle$ or $Seq_\omega(0,1)$[215], all of which mean sequence as a tuple set but without clarity about its ultimate representation in V (even if we call them strings). Specifically, we extend previous *Definition 46* (where we define strings from sequences) and will now use two kinds of $\{0,1\}^\omega$ string representation as a set:

1. *flat string* - implemented as some mapping $f : I \to \{0,1\}$, where $I \in Ord$ is an index set $|I| < \omega_1$ [216];

2. *recursive string* - implemented by using recursive Kuratowski notation[217].

This means that all $R_{<\omega_1}$ strings can be seen essentially seen either as functions or as nested sequences, both kinds are "stored" as sets in V at their low-level representation. Especially, and also, if we define any functions between strings and sets.

Of course, we will continue to use figure brackets to note such string functions as sets $\{0,1\}^\omega$. In that case, we implicitly mean flat representation. Otherwise, when we care about specific representation[218], then we will explicitly write:

- either $\{0,1\}^\omega$) to mean *flat string* represented by a function and packed as a flat countable set like $\{\langle 0,1\rangle, \langle 1,0\rangle, \langle 2,1\rangle, \langle 3,0\rangle, ..\}$ to represent "$1,0,1,0,..$";

- or $kur(\{0,1\}^\omega)$ to mean *recursive string* - the countable sequence of $\{0,1\}$, explicitly represented as the ZF set with recursive Kuratowski notation.

Depending on our needs (what kind of model we intend to construct), the replicata structure $R_{<\omega_1}$ within some model N usually consists of infinite strings encoded as tuples of ordinals. In fact, since we need the whole replicata as part of the model universe, which means that both the range (set of strings $\mathbb{B} \in V^N$) and the domain[219] (SEFs $R_{<\omega_1} \in V^N$) are included into V^N as part of π definition. If we are to restrict the construction process of the model only to the use of binary strings of countable length, then we define such structure inside the model as $R_{<\omega_1} = \{b : b = flat(\{0,1\}^\omega), |b| = \omega\}$[220].

[215] note that $Seq(\{0,1\})$ notation often means all finite partial functions $\omega \to \{0,1\}$ which is a very different structure than what we understand under countable binary string, hence the ω index

[216] again, see *Definition 46*

[217] as defined in *Definition 119*

[218] Also see *Definition 82*

[219] Remember that one can always obtain $R_{<\omega_1} = \pi^{-1}(\mathbb{B})$

[220] Since we work in ZF, it means that here and similar context we actually take as the alphabet (base) of the string some unique ordinal mappings from shorthand like $0 \mapsto \{\}, 1 \mapsto \{\{\}, \{\{\}\}\}$ and so on - see **??** below

Furthermore, we target the construction of $R_{<\omega_1}$ so that its cardinality is equal to $|\mathbb{R}| = \mathfrak{c}$, which follows from Cantor's Theorem.

Definition 79 - Axiom of string concatenation $SC(\kappa)$.

Let κ be a cardinal. If x and y are two strings of cardinality $|x| = |y| = \kappa$, then $\exists z : z = x \cdot y, |z| = \kappa$.

In terms of $ZF + DC$, the above *Axiom of string concatenation*[221] or $SC(\kappa)$ means that there exists string concatenation process that can be expressed in terms of concatenation of sequences - see *Definition 47*, which is applicable for both flat (functions on sequences to $\{0, 1\}$) and recursive (nested) strings provided by Kuratowski notation. Specifically, $SC(\omega)$ indicates that only the existence of concatenated strings of countable cardinality are implied by the axiom. It follows that, after encoding, the underlying sets are also of countable cardinality.

Definition 80 - Axiom of string replication $SR(\kappa)$.

Let κ be a cardinal. If x is a string of cardinality $|x| \leq \kappa$, then $\exists y : y =$ "$(x), |y| = \kappa$, where parentheses $(\dots)$ is a replication operator that can be recursively applied to a finite or possibly infinite string x by concatenating the same substring x to itself in order to produce a new string y.

Once again, the above definition is applicable to both $flat(\{0, 1\}^\omega)$ and $kur(\{0, 1\}^\omega)$ representations. For $flat(\{0, 1\}^\omega)$ strings we have already used a similar *Definition 46*. For $kur(\{0, 1\}^\omega)$ strings, replication process is equally applicable via concatenation. Replication is applied to the substring between two paired parentheses (round brackets). It resembles algorithmic string copying over and over again - that would be infinitely many times or in an *infinite loop*. Compare provided definitions with the ones offered earlier. Namely, *Definition - 122*, etc. Also in case of $SR(\omega)$ we mean to consider only at most countable strings as operands and replication outcome.

The phrase *applied recursively* does mean both recursive implementation of the loop and the ability to nest parentheses inside expressions to generate a new string y. The expression on the right hand is called *string enumeration formula (SEF)* and can be defined as a formal language[222]. We will leave the exercise of proving this for the reader. But it is enough to mention, that the same $SR(\kappa)$ can be recursively applied to define x and so on[223].

[221] Also see future instances of how string concatenation would be defined in this paper - specifically *Definition 116.*

[222] see earlier discussion in the paper

[223] But, again, greater ordinality is more relevant for the discussion in the future section where this result is a separate axiom.

Furthermore, we want to extend *production* function as it was given in *Definition 61*, simply by noting that it works correctly with both string representations ($flat(\{0,1\}^\omega)$ and $kur(\{0,1\}^\omega)$) and, in general, can be defined on all valid SEF expressions. ZF[224]. Recall, that production function is able to evaluate SEF expressions - binary strings with replication operator and produce a corresponding binary string as an output. Note that for our current purposes it is enough to consider only countable binary strings and countable SEF expressions that produce them.

Definition 81 - Extended Production function ($ZF + DC$).

Let B_{SEF} be a set of all valid countable binary SEF expressions, which can be taken as a domain of a function. Again, a valid countable binary SEF expression has either $flat(\{0,1,(,)\}^\omega)$ or $kur(\{0,1,(,)\}^\omega)$ underlying representation of a string, s.t. any included parentheses are always paired with opening round bracket coming first. If $\mathbb{B}$ is a set of all countable binary strings, namely either $\mathbb{B} = \{b : b = flat(\{0,1\}^\omega)\}$, or $\mathbb{B} = \{b : b = kur(\{0,1\}^\omega)\}$, depending on representation. Then $\exists \pi^\star : B_{SEF} \to \mathbb{B}$, which is called *extended production* function.

By now, we have a very good understanding of how countable binary strings can be represented as $flat(\{0,1\}^\omega)$, which, in fact, was already discussed in the previous section - *Accessibility, Choice and Fair Determinacy* - of this paper. We will now focus on $kur(\{0,1\}^\omega)$. So, for the next lemma we would need to take a closer look at the recursive version of the Kuratowski notation that we use as set representation of strings (again, see the *Definition 119*).

According to the usual Kuratowski definition, an ordered pair or tuple (a, b) is represented as a set: $\{\{a\}, \{a, b\}\}$. Such Kuratowski pairs are typically used to construct Cartesian products and relations, not to encode sequences or strings. Such definition doesn't extend naturally to sequences of more than two elements. On the contrary, our recursive version of the Kuratowski notation can do exactly the opposite - represent strings[225].

Lemma 80 - Recursive countable string representation (ZF).

Countable binary strings can be represented as ZF sets using recursive Kuratowski notation k. Which means that all $kur(\{0,1\}^\omega)$ are ZF sets.

Proof. Let M be a model of ZF. If we apply recursive Kuratowski notation to represent countable binary strings, then $kur(\{0,1\}^n) = \{y : y = k(x_1, .., x_n), x_i \in \{0,1\}, i \leq n\}$ for some $n \geq \omega$. This means that $k(x_1, .., x_n)$

[224] also see much earlier *Definition - 24*

[225] See an example of *String concatenation encoded as sets* in *Table 6*

is just a function $k : X \to Y$ that takes an n-tuple (or sequence) $\langle x_n \rangle$ of $\{0,1\}$ from the domain X and maps it recursively to a pair

$$\big\{\{k(x_1,..,x_{n-1})\},\{k(x_1,..,x_{n-1}),x_n\}\big\}$$

in the image Y. This means that $k(x_1,..,x_n)$ always produces a pair (x_{n-1},x_n) which is a set in M iff $X \subset M$. Hence, either by *Axiom of Pairing* or by *Separation Schema* of ZF, $Y \subset M$. It remains to show that $X \subset M$ for any (at least) countable $kur(\{0,1\}^n)$ or for $n < \omega_1$.

Since M is a model of ZF, then it must be transitive and inductive. Let $\mathbb{N}$ be the smallest subset of M, s.t. $\mathbb{N} = \bigcap\{I : I \text{ is inductive}\}$[226]. We can encode natural numbers like $0 = \{\}, 1 = \{0\}, 2 = \{0,1\}, 3 = \{0,1,2\} \ldots$, so that the cardinality of each set is finite. Since M must contain all ordinals Ord[227], let us also extend $\mathbb{N}$ for the purpose of indexing of countable strings, so that $\Theta = \{\alpha : \alpha \in Ord \wedge \alpha < \omega_1\}$ includes all countable ordinals. Obviously, $\Theta \subset Ord$ implies that our index set Θ must be both transitive and inductive.

Now, being equipped with the above encoding for $\{0,1\} \subset M$ and for our index set $\Theta \in M$, let us show that every $\langle x_n \rangle \in X$ is also a member of M. The finite case $n \in \mathbb{N}$ is trivial - we can always map all finite sequences to some finite ordered pair $\big\{\{k(x_1,..,x_{n-1})\},\{k(x_1,..,x_{n-1}),x_n\}\big\}$ in the image of k. At closer look, so is the remaining case $n \in \Theta$. It is enough to observe for any $\iota \in \Theta : \iota \leq n$ we also have $\langle x_\iota \rangle \in M$ by applying the *Axiom of Infinity*. Indeed, assume $\langle x_\iota \rangle \notin M$. Then ι must be either finite, but we have shown all finite indexes are already in M. So ι must be an infinite ordinal. Since M is both transitive and inductive, it must contain all ordinals including such as $\forall \iota \in \Theta : \iota \in Ord \implies \iota \in M$. Hence, the image of recursive Kuratowski notation for countable binary strings is always a ZF set. □

Lemma 81 - Countable Replication.

The following holds in $ZF + DC$:

1. $CC \implies CUT$
2. $CUT \implies SC(w)$
3. $SC(w) \implies SR(w)$

Proof. 1. $CC \implies CUT$, where CUT stands for *Countable Union Theorem* which says that *a union of at most countable sets is a countable set*[228]. For the details of the proof please see [9].

[226] See *exercise 1.2* in [6]

[227] As shown by Gödel using smallest L model of ZF[6]

[228] or alternatively *Any countable union of countable sets is a countable set*

2. To show $CUT \implies SC(w)$, it is enough to interpret $x \cdot y$ as a union of two countable sets $kur((z_1, .., z_{n+m-1})) \cup kur((z_1, .., z_{n+m-1}, z_{n+m}))$, where $|x| = n$, $|y| = m$ and k is a recursive Kuratowski morphism. Here we define $\langle z_{n+m} \rangle$ as a countable union of countably many applications of k to represent the remaining sequence $(x_1, .., x_n, y_1, .., y_{m-1})$ mapped to $(z_1, .., z_n, z_{n+1}, .., z_{n+m-1})$ under "inwards" recursion. In our Kuratowski representation $kur(\langle z_{n+m} \rangle)$ translates into $\{k(z_1, .., z_{n+m-1})\} \bigcup \{k(z_1, .., z_{n+m-1}), z_{n+m}\}$. Assuming CUT means that such set $kur(\langle z_{n+m} \rangle)$ exists iff it is a result of a countable union of countably many applications of k. It remains to show that we can merge x and y together into concatenated sequence by defining a map $\mu : \mu(x, y) = z = x \cdot y$. The later is achieved by defining a big enough index set $I = \{\alpha : \alpha \in Ord \cap \alpha < n + m < \omega_1\}$, so that the result of concatenation is just a function $z : I \to \{0, 1\}$ with explicit representation $kur(\langle z_{n+m} \rangle) = k(z_1, .., z_{n+m})$.

3. $SC(w) \implies SR(w)$: we start by observation that replication operator is a shorthand for concatenation of the same string countably many times. Indeed, $y = (x)$ is an equivalent or a shorthand of the formula $y = x \cdot x \cdot \ldots \cdot x \wedge |y| = \alpha$, where x, y are strings represented as sets and $\alpha \in Ord : \omega \leq \alpha < \omega_1$. Assuming CUT there exists $\langle z_\alpha \rangle$ s.t. $\bigcup_{\alpha \in I} z_\alpha = k(x_1, .., x_\alpha)$, where I is a big enough index set $I = \{\alpha : \alpha \in Ord \cap \alpha < \omega_1\}$ and k is a recursive Kuratowski notation for $x = x_1 = ... = x_\alpha$. In that case $y = (x)$ can be interpreted as a formula in set theoretical language for countable union of sets in recursive Kuratowski notation. If both strings and their representations as sets are countable $|x| = |y| = \aleph_0$, then there exists a countable string replication and $SR(w)$ holds.

□

In general, CUT is required to show exactly this - if x and y are countable, then so is their union, intersection and difference. So we need to assume CUT both for concatenation and for replication of countable strings. Given the above result, we can see that $ZF + CC$ is indeed sufficient for our needs.

Obviously, if $SR(w)$ holds then string replication can be represented as a set not only for a single pair of brackets per formula, but at least for countably many recursively nested replicators following the same logic of countable unions as discussed in previous lemma but with more complex encoding in recursive Kuratowski notation. A production function[229] defined over the domain of such expressions or SEFs is discussed in the next theorem.

Theorem 82 - Countable Replication Theorem (CRT).

---

[229] see *Definition 81*

There exists an extended production function $\pi^\star : B_{SEF} \to \mathbb{B}$ iff one can produce countable strings by replication (meaning that axiom $SR(\omega)$ holds). Furthermore, production function π defined on the whole domain B_{SEF} is surjective (many formulas can be evolved into the same countable binary string).

Proof. Recall from *Definition 81* that B_{SEF} is a set of all valid countable binary SEF expressions, which can be taken as a domain of a function. A valid countable binary SEF expression is a $kur(\{0,1,(,)\}^\omega)$, s.t. any included parentheses are always paired with opening round bracket coming first. $\mathbb{B}$ is a set of all countable binary strings, namely $\mathbb{B} = \{b : b = kur(\{0,1\}^\omega)\}$.

$\exists \pi, \pi : B_{SEF} \to \mathbb{B} \implies SR(\omega)$: this direction is trivial. Given production function $\pi^\star : B_{SEF} \to \mathbb{B}$, one can produce countable strings which would be equivalent to any replication result simply by taking the mapped countable binary string in the image of the function, i.e. $\forall \phi \in B_{SEF} : b = \pi^\star(\phi) \wedge b \in \mathbb{B}$. By applying *Replacement Schema* of ZF [6], we can state that $b = \pi^\star(\phi)$ is a set or a string having a set representation using recursive Kuratowski morphism. Which means that we can compute countable SEF expressions into countable infinite binary strings. Hence $SR(\omega)$ holds.

$\exists \pi^\star, \pi^\star : B_{SEF} \to \mathbb{B} \impliedby SR(\omega)$: for other direction we start by showing that $\mathbb{B} \subset B_{SEF}$. Indeed, if one omits all parentheses in a SEF expression, then one gets just a countable binary string, meaning that $\forall \phi \in B_{SEF} : \phi \in \mathbb{B} \iff |\phi|_(= |\phi|_) = 0$[230]. Next, we rely on the result obtained earlier[231] that $|\mathbb{B}| = |B_{SEF}| = \mathfrak{c}$. Having $\mathbb{B} \subset B_{SEF} \wedge |\mathbb{B}| = |B_{SEF}|$ also means that if $\pi^\star$ exists then it must be surjective, since $\pi^\star$ is clearly not an identity mapping. Now, if one assumes the ability to compute any countable binary SEF expression into a countable binary string, then such computation would be exactly what is achieved by $SR(\omega)$. This allows us to define a correspondence $\forall \phi \in B_{SEF} : \exists b \in \mathbb{B} \wedge \phi \mapsto b$, which is precisely the production function[232].

□

Finally, we want to notice that similar set theoretical claims equally hold for $flat(\{0,1\}^\omega)$ and $kur(\{0,1\}^\omega)$ representations. This follows from the below lemma.

Definition 82 - Coding of recursive and flat strings.

Let $z = \{0,1\}^\omega$. Usually we assume that every string has flat representation

[230] For $|x|_y$ notation - see *Definition 116*

[231] See *lemma 19* and *theorem 20* where B_{SEF} is essentially the same set as $\mathcal{L}_{SEF}$

[232] note that here and the rest of the proof we do not need to rely on any additional choice beside CC

as tuple (unless explicitly specified differently), i.e. we assume $x = flat(z)$. Also, let $y = kur(z)$. Then, $c_r : z \to y$ is *recursive string coding* morphism and $c_f : z \to x$ is *flat string coding* morphism. For notational convenience, we say that both representation functions are aliases to respective coding morphism and are, in fact, idempotent. Meaning that for $z = \{0,1\}^\omega$ we have $flat(flat(z)) = flat(z)$ and $kus(kusr(z)) = kur(z)$.

Lemma 83 - String Representation Equivalence.

Let $c_r : z \to y$ and $c_f : z \to x$ be respectively *recursive* and *flat* string coding morphisms (*Definition 82*), then there exists a bijection between their images x and y.

Proof. Beforehand, we want to make sure that x, y are both ZF sets, i.e. $x, y \in V$. The observation that $x \in V$ follows from ZF axioms since every flat string is merely a function. Other observation $y \in V$ follows from *lemma 80*. Now, since each representation coding must be one-to-one and onto with the defined domain of all binary sequences such as $z = \{0,1\}^\omega$, c_r and c_f codings themselves are recoverable. Hence, there exists an identity bijection $id = c_r^{-1} = c_f^{-1}$. □

7.3 Full Boolean-Valued Models

Consider *full Boolean-valued models* as described on p. 208 in [6].

Lemma 84 - Full Boolean-valued model of ZFC.

Assume $N \models ZF + DC$, B is a complete Boolean algebra in N and V^B is a Boolean-valued submodel of N. If F is an ultrafilter on a complete Boolean algebra B, then the Boolean-valued submodel V^B satisfies not only ZF but also the axiom of choice.

Proof. Consider V^B as the Boolean-valued model constructed from B. Assume F is an ultrafilter on B. Clearly ZF holds in $V^B \subset N$ - assuming it is constructed according to [6]. We aim to demonstrate that V^B also upholds the axiom of choice under this configuration, and thus V^B is a transitive model of ZFC.

1. **Boolean-Valued Models and Ultrafilters:**

   - A Boolean-valued model A is considered full if for any formula $\varphi(x, x_1, \ldots, x_n)$ and for all $a_1, \ldots, a_n \in A$, there exists an $a \in A$ satisfying:

$$[\![\varphi(a, a_1, \ldots, a_n)]\!] = [\![\exists x\, \varphi(x, a_1, \ldots, a_n)]\!]$$

- An ultrafilter F on B allows us to define an equivalence relation $\equiv$ on A by:
$$x \equiv y \iff [\![x = y]\!] \in F.$$
- Furthermore, a binary relation E on $A/\equiv$ is defined as:
$$[x]E[y] \iff [\![x \in y]\!] \in F.$$

2. **Properties of the Equivalence Relation:**
   - The relation $\equiv$ is an equivalence relation due to the properties of F as a filter.
   - The definition of E remains independent of the choice of representatives, supported by the properties of F as an ultrafilter.
3. **Two-Valued Model A/F:**
   - The structure $A/F = (A/\equiv, E)$ forms a two-valued model, inheriting the logical properties of V^B.
4. **Transforming V^B to a Two-Valued Model:**
   - By employing the ultrafilter F, we transform V^B into a two-valued model V^B/F.
   - If V^B/F adheres to the ZFC axioms, so does V^B since the transformation preserves the logical structure.
5. **Verifying the Axiom of Choice in V^B/F:**
   - Construct a choice function σ in V^B/F ensuring for any set x, $\sigma(x)$ selects an element from x.
   - The ultrafilter F guarantees that for each x, a y exists such that $[\![y \in x]\!] \in F$, thereby validating $\sigma(x)$ as a well-defined function.
   - Since $[\![\sigma(x) \in x]\!] = 1$ within V^B/F, the axiom of choice holds in this model.

Given that V^B/F satisfies the axiom of choice and that the ultrafilter-based transformation preserves the ZFC axioms, V^B fully satisfies the axiom of choice. Consequently, V^B supports all the axioms of ZFC, including the axiom of choice.

□

7.4 A replicated model of Set Theory

We assume that the universe V^N of our model of interest is "large enough" to contain $R_{<\omega_1}$ set, which has the size of continuum and consists of $\{0,1\}^\omega$ strings (similar to Cantor space). Those strings can be also expressed as formulas in "dialect" of Set Theory language (so-called String Enumeration Formulas or SEF) and manipulated by a production function π. We also have P_ε poset defined on $R_{<\omega_1}$. All together those concepts are part of the replicata structure.

Lemma 85 - Constructability of replicata.

Let N be a transitive model of $ZF + DC$. If there exists a replicata $(R_{<\omega_1}, P_\varepsilon, \pi)$, s.t. $R_{<\omega_1} \in V^N$, then $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ and $N \models P_\varepsilon$ is ccc.

Proof. Our proof consists of summarizing previous results and grouping them together as following:

1. $\pi^\star$ **is well-defined**: If $N \models ZF{+}DC$, then $N \models \pi^\star$ is a production function.
   - (i) DC is known[233] to be stronger than CC, but weaker than AC, i.e. $N \models DC \implies N \models CC$ [9, 21, 25].
   - (ii) $N \models SR(\omega)$ follows from *Lemma 81 - Countable Replication*.
   - (iii) Extended production function $\pi^\star$ is well-defined for both $flat(\{0,1\}^\omega)$ and $kur(\{0,1\}^\omega)$ representations on countable binary strings:
     - if $B_{SEF} \subset \{0,1,(,)\}^\omega$ is a domain of $\pi^\star$ that consists of all valid SEFs, then B_{SEF} is also a ZF set;
     - if $\mathbb{B} = \{0,1\}^\omega$ is a codomain of $\pi^\star$ that consists of all countable binary strings, then $\mathbb{B}$ is also a ZF set;
     - For every $s_x \in B_{SEF}$, there exists a unique $x \in \mathbb{B}$. This follows from *Theorem 82 - Countable Replication Theorem (CRT)* as each valid SEF formula can be evaluated into its replication-free from by applying replication. Hence, $\pi^\star$ is a well-defined function in $ZF + DC$, and it satisfies *Definition 81*;
     - Finally, *Lemma 83* guarantees that $\pi^\star$ is well-defined for both recursive and flat representations.
2. **Constructability of** $R_{<\omega_1}$: If $N \models ZF + DC$, then $N \models R_{<\omega_1}$ and $N \models \pi$.
   - (i) $R_{<\omega_1} = \{s_x : s_x \in B_{SEF}\}$ is defined as a set of labels of SEF equivalence classes $[s_x]$, where $[s_x] = \{\varphi \in R_{<\omega_1} : \pi^\star(\varphi) = \pi^\star(s_x)\}$ is an equivalence class of formulas that point to the same image ($\pi^\star$ is constant). $R_{<\omega_1}$ is $ZF + DC$ set and $N \models R_{<\omega_1}$.

[233] Also see discussion of *Definition 41*

(ii) Furthermore, *Lemma 49 - Bijectivity of* π implies that restriction $\pi = \pi^\star{\restriction}_{R_{<\omega_1}}$ is bijective. It follows that $N \models \pi^\star \implies N \models \pi$.

3. **Partial order** P_ε **on** $R_{<\omega_1}$: If $N \models ZF{+}DC$, then $N \models P_\varepsilon(R_{<\omega_1}, \leq)$.

   (i) P_ε is a sppo defined on $R_{<\omega_1}$ (*Definition 66 - Substring-Partition Partial order (sppo) on* $R_{<\omega_1}$).

   (ii) Furthermore, *Theorem 67 -* $R_{<\omega_1}$ *with sppo satisfies ccc* implies that $N \models P_\varepsilon$ is ccc.

□

An immediate corollary of $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ is the negation of CH.

Corollary 86 - Failure of CH **in** N**.**

$N \models (R_{<\omega_1}, P_\varepsilon, \pi) \implies N \models \neg CH$

Proof. Follows from *Theorem 66 - Cardinalities of* $R_{<\omega_1}$ *partition.* □

Although this result is not novel, as it simply demonstrates the independence of CH from ZF in another manner (without applying forcing), it raises an intriguing question: does CH fail in all "large enough" models of $ZF + DC$? If so, what property (in addition to "largeness" as $R_{<\omega_1} \subset N$) characterizes such a class of $ZF + DC$ models?

Certainly, there are several $ZF + DC$ models where CH holds (trivially due to CH being independent of ZF). For example, we can group them as:

- Cohen's Forcing Extension: Let V be our starting model of $ZFC{+}CH$ (such as L), then use proper forcing to preserve DC and even CH;
- model of ZF, where HOD satisfies CH (DC is often preserved in HOD models);
- model of $ZF + DC$ with CH using Symmetric Extensions, and so on.

These examples typically rely on the $V = L$ assumption or involve adding no more than ω_1 reals to the model universe. Nonetheless, identifying DC as the crucial axiom to decide CH in "large enough" models would be groundbreaking.

One approach to show this would be to rely on results obtained in the context of axioms stating the existence of certain large cardinals. The nature of this has been heavily studied by many, prominently by R. Solovay [33], H. Woodin [34–37]. It seems that some of the results discussed in the subsection *6.10 Quasi-Large Cardinals* can be potentially used to construct such forcing. But for now we would follow on the different line of thought to obtain our result, mainly by M. Foreman and M. Magidor and S. Shelah [38] as well as

D. Asperó and R. Schindler [29]. We will construct a model M of ZFC as a submodel of N to show that, in fact, our construction implies existence of forcing conditions equivalent to MM.

Lemma 87 - Boolean-valued submodel for replicated forcing.

Let N be a transitive model of $ZF + DC$. If $(B_F, \dot{\Upsilon}, \dot{\lambda}, \neg, \bot, \top)$ is a fair algebra in N, where $S_{FC} \subset R_{<\omega_1}$ is a subset of fair countable strings and B_F is a complete Boolean algebra extended over the fair lattice $P_\varepsilon(S_{FC}, \leq)$ (as defined in *Lemma 75*). Then, there exists a Boolean-valued model V^{B_F} of ZFC, s.t. $N \models B_F$ and $M = V^{B_F}$ is a submodel of N.

Proof.

1. We start by defining $B_{R_{<\omega_1}}$. From previous result we have $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$. Let $B_{R_{<\omega_1}} = B(P_\varepsilon(R_{<\omega_1}))$ be a complete boolean algebra on sppo defined with replicata $R_{<\omega_1}$. From earlier results, *Lemma 56 - Boolean Prime Ideal on* $B(P(R_{<\omega_1}))$ and *Theorem 55 - Restricted Prime Ideal Theorem* $BPI(X)$, we have that such Boolean algebra $B_{R_{<\omega_1}}$ has prime ideal $N \models BPI(B_{R_{<\omega_1}})$ as well as that every filter can be extended to an ultrafilter on $R_{<\omega_1}$ - $N \models UltrafilterTheorem(B_{R_{<\omega_1}})$.

2. Observe that $B_{R_{<\omega_1}}$ is a complete Boolean algebra. Furthermore, fair algebra B_F is a complete subalgebra of $B_{R_{<\omega_1}}$: $B_F \subset B_{R_{<\omega_1}}$ as B_F is complete on itself (*Lemma 75*).

3. Let $V^{B_{R_{<\omega_1}}}$ be a Boolean-valued model constructed on $B_{R_{<\omega_1}}$ and $V^{B_{R_{<\omega_1}}} \models ZFC$ (follows from *Lemma 84*). Clearly, $V^{B_{R_{<\omega_1}}} \subset N$.

4. Similarly, V^{B_F} is a Boolean-valued model constructed on $S_{FC} \subset R_{<\omega_1}$. Let $U_{R_{<\omega_1}}$ be an ultrafilter on $R_{<\omega_1}$ and $U_F \subset U_{R_{<\omega_1}}$ be an ultrafilter on the set of fairs S_{FC}, s.t. $U_F = \{x : x \in U_{R_{<\omega_1}} \wedge x \in S_{FC}\}$. If we define the choice function on U_F (instead of $U_{R_{<\omega_1}}$) in *Lemma 84*, then $V^{B_F} \models ZFC$. Clearly, $V^{B_F} \subset N$.

□

To sum up, the compatibility of the scenario considered above is meaningful only if N satisfies AC for the sets and operations defined in M. However, N may still not satisfy AC globally (for all of V). In general, M cannot be a submodel of N if N globally violates AC in ways that conflict with satisfaction of ZFC axioms in M. Our conclusion is that if $M = V^{B_F}$ is large enough, then AC will not fail in M.

7.5 Properties of Stationary Sets

Before we can start looking in detail if $ZF + DC$ implies MM, we want to highlight a few more known results. Recall from [6] definition of a stationary

set[234]:

Definition 83.

Let κ be a regular uncountable cardinal. A set $C \subseteq \kappa$ is a closed unbounded subset of κ if C is unbounded in κ and if it contains all its limit points less than κ. A set $S \subseteq \kappa$ is stationary if $S \cap C \neq \emptyset$ for every closed unbounded subset C of κ.

Observe that the set of all perfectly fair strings forms a stationary subset in $R_{<\omega_1}$.

Lemma 88 - Stationary Set of Perfect Fairs.

Let $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ and $S_{PF} \subset R_{<\omega_1}$ be a set of perfectly fair strings. Then,

$$N \models S_{PF} \text{ is a stationary set}$$

Proof. 1. We start by noticing that the set of all fair countable binary SEFs S_{FC} is closed and unbounded. S_{FC} *is unbounded* since $|S_{FC}| = |S_{PF}| = |S_{IF}| = \aleph_1 > \aleph_0$, which follows from *Lemma 45*, *Lemma 64*, *Lemma 65*. This implies that S_{FC} is also a closed unbounded subset of $\aleph_2$, since it contains all limit points of cardinality $\aleph_1 < \aleph_2$.

2. Now let us show that $S_{PF} \subseteq S_{FC}$ is stationary[235]. Note that for every perfectly fair $x \in S_{PF}$, there exists some imperfectly fair $y \in S_{IF}$ so that $x \; \varepsilon \; y$, which implies that $x \subset y$ as according to[236] *Definition 46*. Since $S_{FC} = S_{PF} \cup S_{IF}$, we have that every closed unbounded subset $\forall C \subset S_{FC}$ will have a non-empty intersection with S_{PF}: $S_{PF} \cap C \neq \emptyset$. Hence, S_{PF} is a stationary set.

□

Corollary 89 - Perfect stationary subset.

Every stationary set S contains S_{PF} as a subset, i.e. $S \supset S_{PF}$.

Proof. Assume the opposite. Then S and S_{PF} are either disjoint or partially overlap. Let $z \in S_{PF} : z \notin S \cap S_{PF}$. In the proof of the previous lemma we made an observation that for every perfectly fair $x \in S_{PF}$, there exists some imperfectly fair $y \in S_{IF}$, so that $x \; \varepsilon \; y$, which implies that $x \subset y$. However, since S_{FC} must contain all clubs, there exists a club $C \in S_{FC}$, which is not intersected by z. But then S can not be a stationary set. Hence, we arrived at contradiction. □

[234] Also see *Lemma 98*

[235] as according to *Definition 83*

[236] or simply via string-to-set isomorphism in *Definition 120*

Next we will provide two results from [6] about preservation of stationary sets but without proof[237]:

Lemma 90.

Let κ be a regular uncountable cardinal. Let $V[G]$ be a generic extension of V by a κ-c.c. notion of forcing. Then every closed unbounded subset $C \subset \kappa$ in $V[G]$ has a closed unbounded subset $D \in V$. Consequently, if $S \in V$ is stationary in V, then S remains stationary in $V[G]$.

However, if the notion of forcing $\mathbb{P}$ is more restricted and satisfies the countable chain condition, then one can obtain a more general result (see *lemma 31.2* from [6]):

Lemma 91 - Preservation of stationary sets.

If $\mathbb{P}$ satisfies the countable chain condition, then for every uncountable λ, every closed unbounded set $C \subset [\lambda]^{\omega} \in V[G]$ has a subset $D \in V$ that is closed unbounded in V. Hence, every stationary set $S \subset [\lambda]^{\omega}$ remains stationary in $V[G]$.

To summarize[238]:

- Closed unbounded sets, or *club sets*, are significant in the context of large cardinals and their preservation under various forcing conditions.
- A subset S of a limit ordinal λ (usually ω_1) is called *stationary* if it intersects every closed unbounded subset of λ.
- The notion of forcing P is *stationary set preserving* if every stationary set S remains stationary in the generic extension $V[G]$.
- The preservation of stationary sets under forcing indicates the robustness of these sets against perturbations in the set-theoretical universe. It also underlines the continuity of certain set properties despite the potential addition of new sets or changes in the structure of the extended universe $V[G]$ by forcing.
- The countable chain condition (ccc) on the forcing notion $\mathbb{P}$ ensures that the forcing does not add new countable sequences of ordinals, which is crucial for maintaining stationary subsets of λ in the extended universe $V[G]$.

[237] Please see *lemma 22.25* and *lemma 31.2* in [6] for the proof

[238] Recall *Definition 64 - Forcing "prerequisites"*, the above *Definition 83* and *Martin's Maximum* chapter in [6] as well as the results on preservation of stationary sets

7.6 Replicating Martin Maximum

Definition 84 - Martin's Maximum (MM).

If $(P, <)$ is a stationary set preserving notion of forcing and if D is a collection of $\aleph_1$ dense subsets of P, then there exists a D-generic filter G on P.

Theorem 92 - Martin's Maximum Model.

Assuming that ZF is consistent. If $\exists N : N \models ZF + DC$ and M is a submodel of N, s.t. $M \models ZFC$, then N is a replica host for submodel M, so that $P \in M$ is a replicated forcing over $R_{<\omega_1}$:

1. $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$
2. $(P, \leq)$ is a generic notion of forcing in M (ground model);
3. $h : M \to N$ is an embedding, s.t. $ran(h) \subseteq R_{<\omega_1}$ and restriction $h{\restriction}_P$ is order preserving on $R_{<\omega_1}$;

Furthermore,

$$M \models \mathsf{MM}$$

Proof.

1. $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ follows from *Lemma 85.*
2. $N \models B_F$ and $M = V^{B_F}$ is a submodel of N follows from *Lemma 87.*
3. Let $h : M \to N$ be an embedding that corresponds to the fact that M is submodel of N. Notice that fair algebra $B_F \subset B_{R_{<\omega_1}}$, which implies $ran(h) \subseteq R_{<\omega_1}$.
4. Let $(P, \leq)$ be a candidate for the generic notion of forcing in M (as a ground model). Keep in mind that we might later choose to extend $(P, \leq)$ by pretending that it aligns not just with $P_\varepsilon(B_F)$ but with large $P_\varepsilon(B_{R_{<\omega_1}})$. Explicit requirement that restriction $h{\restriction}_P$ is order preserving on $R_{<\omega_1}$ follows from the fact that $h : M \to N$ is already a model embedding - an isomorphism that preserves relations like P. Since $M = V^{B_F}$, it is reasonable to assume that we can choose $(P, \leq) = h^{-1}(P_\varepsilon(B_F))$, so that all dense subsets of $P_\varepsilon(B_F)$ can be matched with dense subsets in $(P, \leq)$ in M.
5. Next, we will verify the $M \models \mathsf{MM}$ claim: namely, that $(P, \leq)$ is stationary set preserving forcing notion and if D is a collection of $\aleph_1$ dense subsets of P, then there exists a D-generic filter G on P:
   (a) $\boldsymbol{M \models P}$ **is ccc**: Since $(P, \leq)$ in M corresponds to $P_\varepsilon(B_F)$ in N via embedding h, it means that $(P, \leq)$ is also a well-founded p.o.

relation (a consequence of $M = V^{B_F}$ being a full Boolean-valued model over ultrafilter) and is *ccc* (recall that *Theorem 67 - $R_{<\omega_1}$ with sppo satisfies ccc* implies that $N \models P_\varepsilon$ is ccc).

(b) A set of conditions $\boldsymbol{G \subset P}$ **is generic over** $\boldsymbol{M}$:

(i) $\boldsymbol{G}$ **is a filter on** $\boldsymbol{P}$: Notice that if $h(G)$ is a filter on $P_\varepsilon(B_F)$ (in N), then the pre-image G must be also a filter on $(P, \leq)$ (when viewed from inside the model $M = V^{B_F}$). Let us first construct the image $H = h(G)$ as such a filter in N, and then show that G is a filter on P in M. At this point of the proof we need to do it in the most explicit manner. Now let $f : S_{FC} \to S_{UC}$ be an injective map, s.t. for each fair string there exists a corresponding unfair string $\forall s \in S_{FC} : f(s) =$ "(s)". Also let $F \subset S_{FC}$ be a filter[239] on S_{FC} as according to *Definition 58*:

A. F is not empty: for simplicity we require $S_{PF} \subset F$ so that its cardinality is as large as $|F| = |S_{PF}| = |S_{FC}|$.

B. If $p, q \in F$, there is some $r \in F$ such that $r \leq p$ and $r \leq q$: in case of P_ε this follows from $r \;\varepsilon\; p \oplus q$.

C. If $p \in F$, $q \in P$ and $p \leq q$, then $q \in F$: for P_ε this is translated into if $p \;\varepsilon\; q$, then $q \in F$.

We know that F exists on $P_\varepsilon(B_F)$, since B_F is a complete Boolean algebra[240]. Next, one can define H to be a filter on $P_\varepsilon(R_{<\omega_1})$ build from F:

$$H = \{\text{“}(s)\text{”} : s \in F\}$$

With such construction, on one hand, $H \notin M$, but, on the other hand, $\forall s \in F : f(s) \in H$ and $s \;\varepsilon\; f(s) \implies s \leq f(s)$. We have that both F and H are filters on P_ε in N if P_ε is taken large enough, e.g. over the whole $R_{<\omega_1}$. Furthermore, we have that $G = h^{-1}(H)$ is also a filter on P. After this point, we have extended $(P, \leq)$ to be large and matched by $P_\varepsilon(B_{R_{<\omega_1}})$.

(ii) **if** $\boldsymbol{D}$ **is dense in** $\boldsymbol{P}$ **and** $\boldsymbol{D \in M}$**, then** $\boldsymbol{G \cap D \neq \emptyset}$: Suppose there is D, which consists of all $\aleph_1$-dense subsets in $(P, \leq)$ and $D \in M$. We can show that such D exists by observing that it can correspond one-to-one to $h(D) \in P_\varepsilon(B_F)$, which is $\aleph_1$-dense in $P_\varepsilon(B_F)$. Indeed, according to *Lemma 75*, fair algebra B_F is a complete Boolean algebra constructed around a fair lattice on $P_\varepsilon(S_{FC})$, where $S_{FC} =$

[239] Here we follow sppo definition to match $p \leq q$ with $p \;\varepsilon\; q$, but it is also possible to work with the reverse P_ε without loss of generality

[240] see *lemma 75*

$S_{PF} \cup S_{IF}$ is a disjoint union of two $\aleph_1$-dense subsets. Note that D is $\aleph_1$-dense in P iff the corresponding set $h(D)$ is $\aleph_1$-dense in $P_\varepsilon(B_F)$.

(c) **$(P, \leq)$ is stationary set preserving forcing notion**: Recall from [6], that a notion of forcing P is *stationary set preserving* if every stationary set $S \subset \omega_1$ remains stationary in $V^{P_\varepsilon(B_F)}$. A stationary set $S \subset \omega_1$ [241] is a set that intersects every closed unbounded subset of ω_1. *Lemma 88* implies that S_{PF} is such a stationary set, since S_{PF} intersect every subset of S_{IF} (by definition of imperfect fairs) and S_{UC} by definition of unfair countable. Every subset of B_F is closed and unbound. Furthermore, since each *imperfectly fair* SEF is a concatenation of *perfectly fair* strings (basically, a subset of S_{PF}), we have that:

(i) any large enough subset $S \subset S_{PF}$ with $|S| \geq \aleph_1$ is dense and stationary in $R_{<\omega_1}$, hence S_{FC} contains all stationary subsets, which are also ω-closed.

(ii) $h(G)$ intersects every dense set in $h(D)$, hence, respectively G is D-generic in M;

(iii) if $M[G]$ is a generic extension of M with filter G, then $M = V^{P_\varepsilon(B_F)} \subset M[G] \subset V^{P_\varepsilon(B_{R_{<\omega_1}})} \subset N$, where $V^{P_\varepsilon(B_{R_{<\omega_1}})}$ is a full Boolean algebra model of ZFC constructed around larger $R_{<\omega_1} \supset S_{FC}$ (by General forcing theorem and embedding $M \subset N$ as well as *corollary 66 - Cardinalities of $R_{<\omega_1}$ partition*).

6. Finally, we want to show that if $M[G]$ is a generic extension of M with filter $G \in M[G]$, then all stationary subsets in M are preserved in $M[G]$. This follows from (5.a) and (5.b.i). Namely, that $M \models P$ is ccc and $\forall s \in F : f(s) \in H$ and $s \; \varepsilon \; f(s) \implies s \leq f(s)$, where $H = h(G)$.

7. Furthermore, it follows from (1) that since replicata $(R_{<\omega_1}, P_\varepsilon, \pi)$ is constructable in any model N of $ZF + DC$ (without any additional specific requirement except "largeness", i.e. $R_{<\omega_1} \in V^N$), it is evident that for any large enough forcing notion[242] P_ε, s.t. $S_{PF} \subset P_\varepsilon$ and $P_\varepsilon$is ccc, we have that P_ε is stationary set preserving and not only $M \models \mathsf{MM}$, but also $M[G] \models \mathsf{MM}$ and $N \models \mathsf{MM}$ (since $M \subset M[G]$, but also $M[G] \subset N$ as it has been constructed within N structure).

□

Let us take a moment to discuss the $M \models \mathsf{MM}$ implications.

[241] here ω_1 is a regular uncountable cardinal as well as the limit ordinal

[242] See *lemma 89* and *lemma 91*

- **Failure of the Continuum Hypothesis (CH):** MM forces $2^{\aleph_0} = \aleph_2$.[243]
- **Stationary set reflection:** Every stationary subset of $\omega_2 \cap \mathrm{cof}(\omega)$ reflects to some ω_1-size ordinal.[244]
- **Strong Δ-system lemma:** Under MM, every family of $\aleph_1$ countable sets has an $\aleph_1$-sized Δ-system.[245]
- **Non-existence of ω_2-Aronszajn trees:** MM gives the tree property at ω_2.[246]
- **Determination of Σ^1_2 sets of reals:** MM (via PFA) yields $\mathrm{AD}^{L(\mathbb{R})}$ and hence determinacy for all projective sets, in particular Σ^1_2.[247]
- **Bounded Proper Forcing Axiom (BPFA):** MM $\Rightarrow$ PFA $\Rightarrow$ BPFA.[248]
- **Large-cardinal-like behaviour:** MM produces precipitous ideals, failure of $\square$, strong reflection, etc.[249]
- **Well-ordering of the reals:** Under BPFA (hence under MM) the reals admit a Δ^1_2 well-order of type ω_2.[250]
- **Canonical models:** Iteration techniques preserve MM and let one build canonical models with many combinatorial properties simultaneously.[251]

7.7 Goedel's conjecture

We can extend our previous conclusion to a more general statement:

Corollary 93 - General failure of CH in large $ZF + DC$ models.

If $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$, then CH fails in any large enough model N of $ZF + DC$ s.t. $R_{<\omega_1} \in V^N$.

Proof. Follows from *Corollary 86* as well as *Theorem 92*. □

[243] Foreman–Magidor–Shelah [39].
[244] Foreman [40].
[245] Todorčević [41].
[246] Baumgartner [42].
[247] Steel [43].
[248] Caicedo [44].
[249] Foreman–Magidor–Shelah [39].
[250] Caicedo–Veličković [45].
[251] Foreman [46].

The following theorem effectively captures largeness of the any corresponding transitive $ZF + DC$ model, which is necessary and sufficient to imply Goedel's conjecture and decide CH.

Theorem 94 - Largness criteria.

Let N be a transitive model of $ZF+DC$. Then the following are equivalent:

1. $N \models 2^{\aleph_0} = \aleph_2$ (Goedel's conjecture)
2. $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$

Proof. (1) $\rightarrow$ (2): $N \models 2^{\aleph_0} = \aleph_2$ means that V^N is large enough to contain set $\mathbb{R}$ with cardinality of the continuum equal to $\aleph_2$, which is consistent with replicata construct. If V^N can contain a cantor set $\mathbb{B} = \pi(R_{<\omega_1})$ of countable binary strings bijective to $\mathbb{R}$, then $N \models (R_{<\omega_1}, P_\varepsilon, \pi)$ by *Lemma 85*.

(2) $\rightarrow$ (1): Follows from $N \models \mathsf{MM}$, since it contains the replicata structure $(R_{<\omega_1}, P_\varepsilon, \pi)$ with the inner model $M \models \mathsf{MM}$ (by *Theorem 92*), which means V^N is large enough to contain $V^{M[G]}$ (in case of large enough forcing), but also in general $\mathsf{MM} \implies 2^{\aleph_0} = \aleph_2$ [6, 29]. □

Corollary 95.

Any set of cardinality of the continuum $2^{\aleph_0}$ can be well-ordered by ω_2.

Proof. $2^{\aleph_0} = \aleph_2$ implies[252] that there is a bijection between respective cardinals $f : \omega_2 \rightarrow 2^{\aleph_0}$. Hence, any set of cardinality of the continuum $2^{\aleph_0}$ can be well-ordered by definition. □

We conclude this subsection by trying to make the strongest possible claim about consistency of $ZF + DC$ models.

The scenario that we are aiming at:

(i) If there was a trivial proof showing that in any model of $ZF + DC$, the set of reals is bijective with the Cantor set and that this set must have cardinality $2^{\aleph_0} = \aleph_2$, it would have significant implications for models of $ZF + DC$ (including smaller models carefully built using of forcing by adding only $\aleph_1$ many reals).

(ii) If every model of $ZF + DC$ inherently has $2^{\aleph_0} = \aleph_2$, this would mean that models created by forcing to make $2^{\aleph_0} = \aleph_1$ or $2^{\aleph_0}$ equals to any cardinal other than $\aleph_2$ are inconsistent.

[252] without assuming full AC

Earlier, in lemma 85, we assumed that our model already contains a large enough structure like $R_{<\omega_1} \in N$ mainly to make our live and logistics around construction of inner models easier. However, if we focus on the idea that it is very straightforward (almost trivial) to construct a replicata $(R_{<\omega_1}, P_\varepsilon, \pi)$ in any $ZF + DC$ model (as this process does not need any special assumptions except DC - see *Theorem 44*), we arrive at very striking conclusion.

Theorem 96 - Inconsistency of smaller $ZF + DC$ models.

Let N be a transitive model of $ZF + DC$, s.t.

$N \not\models 2^{\aleph_0} = \aleph_2$. Then, N is inherently inconsistent.

Proof. Can be show in several steps:

1. Indeed, let N be such a transitive model of $ZF+DC$, where CH holds and $2^{\aleph_0} = \aleph_1$ (meaning that N is small).

2. Assume N is consistent, meaning that there is no statement ϕ, s.t. $N \models \phi$ and $N \models \neg\phi$. So, let ϕ be equal to $2^{\aleph_0} = \aleph_1$.

3. Now, consider some trivial enough proof that is true in all transitive models of $ZF + DC$:

   (a) Let $\mathbb{B}$ be a Cantor Set containing all countable binary strings and $|\mathbb{B}| = \mathfrak{c} = 2^{\aleph_0}$.

   (b) Invoke DC to construct a set of labels with SEFs $R_{<\omega_1}$, consisting of disjoint union of:

   - S_{UF} all unfair finite labels;
   - S_{FC} all fair countable labels (equal to their exact string values $\pi(S_{FC})$);
   - S_{UC} all unfair uncountable labels.

   (c) $R_{<\omega_1}$ is the main part of replicata, together with poset P and production function π. The latter is a bijective map that evaluates SEFs formulas into countable binary strings $\mathbb{B} = \pi(R_{<\omega_1})$, which have the cardinality of continuum by Cantor's theorem.

   (d) It follows from *Theorem 66 - Cardinalities of $R_{<\omega_1}$ partition* that $|R_{<\omega_1}| > \aleph_1$ and $2^{\aleph_0} = \aleph_2$. In fact, by *Theorem 92* we also have that $2^{\aleph_0} = \aleph_2$.

4. We arrived at contradiction with our initial assumption at step (1).

□

8 Replication Schema

In this section we cover topics like transfinite strings, which are simply contiguous sequences defined on the proper class of all cardinals Ord. We also look at generalization of replicata as a proper class Λ, which has a transitive model. Finally, we discuss the connection between Large Cardinals such as supercompact cardinal and N^* model of $ZF + DC_\lambda + I$, where such Λ replicata class exists.

8.1 Iterative Powerset

This brief subsection contains a definition that would become handy on later stages of discussing the replication schema. An iterative higher-order powerset operation is denoted $P^\alpha(X)$ for an ordinal α. This is a generalization of the standard powerset operation $P(X)$, where $P(X)$ refers to the set of all subsets of X. The operation $P^\alpha(X)$ applies the powerset operation iteratively according to the ordinal α. Here is a formal definition of $P^\alpha(X)$, where $\alpha \in Ord$:

Definition 85 - Iterative Powerset.

Let X be any set and α be an ordinal. The operation $\mathcal{P}^\alpha(X)$ is defined inductively on α as follows:

- **Base case (when $\alpha = 0$):**

  $$\mathcal{P}^0(X) = X.$$

  This means the 0th powerset is simply the original set X itself.

- **Successor ordinal $\alpha + 1$:** If α is an ordinal and $\mathcal{P}^\alpha(X)$ has been defined, then:

  $$\mathcal{P}^{\alpha+1}(X) = P(\mathcal{P}^\alpha(X)).$$

  This means that $\mathcal{P}^{\alpha+1}(X)$ is the powerset of $\mathcal{P}^\alpha(X)$, i.e., the set of all subsets of $\mathcal{P}^\alpha(X)$.

- **Limit ordinal λ:** If λ is a limit ordinal (i.e., there is no immediate predecessor to λ), then:

  $$\mathcal{P}^\lambda(X) = \bigcup_{\alpha<\lambda} \mathcal{P}^\alpha(X).$$

  This means that at a limit ordinal λ, the iterative powerset operation results in the union of all the previous stages $\mathcal{P}^\alpha(X)$ for $\alpha < \lambda$.

8.2 Expected Length of Strings

We seek to define a transfinite string as a sequence or function $s : Ord \to X$ that maps ordinals to values in some set X (see *Definition 90*). If the range of such a function can be fixed as some an alphabet or the *coding base*, i.e. $X = B$, then such a function can be seen as being part of some B-coding space (see *Definition 93*). Before proceeding, we must first clarify what is meant by the notion of transfinite string *length*.

Let s_x and s_y be two of such strings. One way to compare them is to compare their respective lengths, i.e. $|s_x|$ and $|s_y|$. Now, as per our definition, transfinite string is a function that maps Ord to B. So this turns out very convenient as ordinals are well-ordered and hence are excellent for such kind of comparison. Furthermore, we want to avoid potential confusion around concepts of cardinality and ordinality when it comes to strings. For example, when we are saying that $s_x < s_y$ but $|s_x| = |s_y|$, we do mean that both strings have the same cardinality but different ordinality, or, even more so, if $s_x \ \varepsilon \ s_y$ implies $s_x < s_y$. Hence, we always map cardinality of the transfinite string to the cardinality of its domain.

Definition 86 - Length of the transfinite string.

Let B be a coding base. If $s : \lambda \to B, \lambda \in Ord$ is a transfinite string, then $|s| = |\lambda|$

Finally, we call such length *expected* to underpin the fact that the exact representation and what we call "length" may depend on the coding of strings — particularly when it comes to considering all possible other "unorthodox" representations of strings as sets in foundational frameworks such as ZF. In this subsection, we aim to systematically clarify any possible confusion surrounding the notion of transfinite string length, beginning by revisiting the construction of ordinals and the cumulative set-theoretic universe.

Construction of Ordinals in ZF In ZF set theory, ordinals are defined using the von Neumann construction:

- The *empty set* is the first ordinal: $0 := \emptyset$.
- The *successor* of an ordinal α is defined as $\alpha + 1 := \alpha \cup \{\alpha\}$.
- A *limit ordinal* is an ordinal λ such that $\lambda = \bigcup_{\alpha<\lambda} \alpha$, and it has no immediate predecessor.
- Every ordinal α is a transitive set well-ordered by $\in$, and equals the set of all smaller ordinals: $\alpha = \{\beta \in Ord \mid \beta < \alpha\}$.

The class of all ordinals, Ord, is strictly well-ordered by $\in$ and serves as a canonical indexing tool for transfinite processes such as induction and recursion.

Cumulative Hierarchy of Sets The cumulative hierarchy[253] is a transfinite sequence $\{V_\alpha\}_{\alpha \in Ord}$ defined by transfinite recursion:

- $V_0 := \emptyset$
- $V_{\alpha+1} := \mathcal{P}(V_\alpha)$
- $V_\lambda := \bigcup_{\beta<\lambda} V_\beta$, for limit ordinals λ

The universe of all sets is then given by $V := \bigcup_{\alpha \in Ord} V_\alpha$. Each stage V_α contains all sets that can be constructed before stage α, and this hierarchy indicates the growth of set-theoretic complexity at each ordinal stage.

Ordinals vs. Cardinals

- Every cardinal can be viewed as an *initial ordinal*, i.e. an ordinal κ which has no smaller ordinal of the same cardinality.
- A *successor cardinal* is one of the form κ^+, the next cardinal above κ.
- A *limit cardinal* is a cardinal λ that is not a successor cardinal (i.e. there is no κ with $\kappa^+ = \lambda$); equivalently, λ is a limit point in the sequence of all cardinals.

Regular vs. Singular

- A cardinal λ is *regular* if its cofinality $\mathrm{cf}(\lambda)$ equals λ.
- A cardinal λ is *singular* if $\mathrm{cf}(\lambda) < \lambda$.

Successor Ordinal vs. Limit Ordinal

- A *successor ordinal* is any ordinal of the form $\alpha + 1$.
- A *limit ordinal* is an ordinal λ such that $\lambda \neq \alpha + 1$ for any α.

To better understand how different types of ordinals arise as potential lengths of transfinite strings, we can classify them into the following natural hierarchy:

- **Case 0:** λ is a *successor ordinal*, i.e. $\lambda = \gamma + 1$, but not necessarily a new cardinal.

[253] see p. 64 in [6]

- If γ is finite, then λ is also a finite cardinal and hence regular.
- If γ is infinite, then λ is strictly larger as an ordinal, but $|\lambda| = |\gamma|$. Thus, λ is not a new cardinal.

- **Case 1:** λ is a *limit ordinal*, but not a new cardinal.
  - An ordinal λ is a *limit ordinal* if and only if $\lambda \neq 0$ and $\lambda = \bigcup_{\alpha<\lambda} \alpha$.
  - May satisfy $|\lambda| = |\alpha|$ for some $\alpha < \lambda$, i.e. not necessarily an initial ordinal.
  - Example: $\omega \cdot 2$ is a limit ordinal with cardinality $\aleph_0$.
- **Case 2:** λ is a *successor cardinal*, i.e. $\lambda = \kappa^+$ for some cardinal κ.
  - Then λ is an initial ordinal strictly greater in cardinality than κ.
  - $\lambda = \kappa^+ = \mathcal{P}(\kappa)$, where $|\kappa| < |\lambda|$.
- **Case 3:** λ is a *limit cardinal*, i.e. a cardinal with no immediate predecessor.
  - $\lambda = \aleph_\alpha$ for some limit ordinal α.
  - If $\mathrm{cf}(\lambda) = \lambda$, then λ is regular; otherwise, singular.
  - Example: $\aleph_\omega = \sup\{\aleph_\alpha : \alpha < \omega\}$ is singular with $\mathrm{cf}(\aleph_\omega) = \omega$.
- **Case 4:** λ is an *inaccessible cardinal*.
  - λ is regular, uncountable, and a strong limit: for all $\kappa < \lambda$, $2^\kappa < \lambda$.
  - Inaccessibles are limits of smaller regular cardinals, and not reachable by the usual set-theoretic operations.
  - Their existence is not provable in ZF, but follows from the assumption of an inaccessible cardinal.

This classification provides a systematic foundation for reasoning about the types of ordinals that can occur as transfinite string lengths. It also highlights the subtle difference between ordinal indexing and cardinal measurement:

- Cofinality $\mathrm{cf}(\lambda)$ tells us the *minimal size* of an unbounded subset of a limit ordinal λ.
- Cardinality $|\lambda|$ measures the *overall size* of λ.
- Without full choice, these concepts are still definable for well-orderable sets and often coincide for strings indexed by ordinals.

So for the rest of this paper, we will use the term *expected length* to indicate that:

1. The length of a transfinite string is defined as an arbitrarily large cardinal (which can be viewed as initial ordinal, depending on the context);

2. Strings are naturally ordered not only by length as cardinality of the domain, but also by the ordinality of the domain, which can be further extended by the substring ε relation;

3. The length can also be explicitly encoded as part of the string representation.

To comment on the last point: one can imagine a general string encoding where the length is included alongside the string value—akin to metadata. For example, a set A representing a string in *flat* or *kur* format can be re-encoded as an ordered pair (A, λ), where λ represents the length of the string. It is trivial to show that such representations, where sets are built from strings indexed by ordinals, can be modeled by a transitive structure.[254] In this setup, each set $A \in Ord$ can be mapped to (A, λ), enriching the model with explicit length annotations.

8.3 Closed Unbounded Filters

To further refine our tools, we introduce $\mathcal{P}_\kappa(\lambda)$, a construct that provides a more nuanced way of handling subsets with restricted cardinalities.

Definition 87 - The κ-small subsets of λ.

Let κ be a regular uncountable cardinal and there exists some set A, s.t. $|A| = \lambda$ and $\lambda \geq \kappa$. Then, the set $[\lambda]^{<\kappa} = \mathcal{P}_\kappa(\lambda)$ is defined as the collection of all subsets of λ whose cardinalities are strictly less than κ:

$$\mathcal{P}_\kappa(\lambda) = \{X \subset \lambda : |X| < \kappa\}.$$

Here, $|X|$ denotes the cardinality of the subset A. The condition $|X| < \kappa$ implies that there exists an injection from X into some cardinal smaller than κ, but no bijection between X and κ. The cardinality of $\mathcal{P}_\kappa(\lambda)$ is $|A|^{<\kappa}$ [6].

Now recall from [6] that $\mathcal{P}_\kappa(\lambda)$ helps with generalization from $(\kappa, <)$ to $(\mathcal{P}_\kappa(\lambda), \subset)$. This permits definition of closed unbounded filters[255] on $\mathcal{P}_\kappa(\lambda)$. We will list several known facts about club filters on $\mathcal{P}_\kappa(\lambda)$ without proof as lemma.

[254] This motivates the intriguing idea of constructing an inner model of ZFC using only string-encoded sets built from ordinals—a direction we intend to explore later in the paper.

[255] See p.100-101 in [6]

Lemma 97 - Properties of clubs on $\mathcal{P}_\kappa(\lambda)$.

Let A be a set and $\mathcal{P}_\kappa(A)$ the collection of subsets of A of cardinality less than κ ($|A| \geq \kappa$ see *Definition 87*). The following definitions and results hold:

1. **Unbounded Sets:** A set $X \subseteq \mathcal{P}_\kappa(A)$ is unbounded if for every $x \in \mathcal{P}_\kappa(A)$, there exists $y \supseteq x$ such that $y \in X$.
2. **Closed Sets:** A set $X \subseteq \mathcal{P}_\kappa(A)$ is closed if for any chain $x_0 \subseteq x_1 \subseteq \cdots \subseteq x_\xi \subseteq \cdots$ (for $\xi < \alpha$ with $\alpha < \kappa$) of sets in X, the union $\bigcup_{\xi<\alpha} x_\xi$ is in X.
3. **Closed Unbounded Sets:** A set $C \subseteq \mathcal{P}_\kappa(A)$ is closed unbounded (club) if it is both closed and unbounded.
4. **Stationary Sets:** A set $S \subseteq \mathcal{P}_\kappa(A)$ is stationary if $S \cap C \neq \emptyset$ for every club set $C \subseteq \mathcal{P}_\kappa(A)$.
5. **Club Filter:** The club filter on $\mathcal{P}_\kappa(A)$ is the filter generated by club sets.
6. **Cardinality and Isomorphism:**
   (a) If $|A| = |B|$, then $\mathcal{P}_\kappa(A)$ and $\mathcal{P}_\kappa(B)$ are isomorphic, and, club and stationary sets in $\mathcal{P}_\kappa(A)$ correspond to those in $\mathcal{P}_\kappa(B)$.
   (b) If $|A| = \kappa$, then the set $\kappa \subseteq \mathcal{P}_\kappa(\kappa)$ is a club, and the club filter on κ is the restriction of the club filter on $\mathcal{P}_\kappa(\kappa)$.
7. **κ-complete:** The club filter on $\mathcal{P}_\kappa(A)$ is κ-complete.

Also let us recall[6] the definition of the normal filter.

Definition 88 - Normal filter.

Let $\mathcal{F}$ be a filter on a cardinal κ. The filter $\mathcal{F}$ is *normal* if it is closed under diagonal intersections:

$$\text{If } X_\alpha \in \mathcal{F} \text{ for all } \alpha < \kappa, \text{ then } \bigtriangleup_{\alpha<\kappa} X_\alpha \in \mathcal{F}. \tag{2}$$

The club filter is κ-complete and normal, and contains all complements of bounded sets[256]. It is the smallest such filter on κ.

[256] Again, please consult p.96 in [6] for the proofs.

Lemma 98.

Let κ be a regular, uncountable cardinal, and let $\mathcal{F}$ be a normal filter on κ that contains all final segments of the form

$$\{\alpha \in \kappa : \alpha_0 < \alpha < \kappa\},$$

for some fixed $\alpha_0 < \kappa$. Then $\mathcal{F}$ contains all club subsets of κ.

8.4 Transfinite Strings

So far, we have considered two representational conventions in Set Theory language for countable binary strings, namely $flat(\{0,1\}^\omega)$ and $kur(\{0,1\}^\omega)$. Recall that by *Definition 82* strings are usually flat and conversion morphisms are idempotent, i.e. $flat(flat(s)) = flat(s)$ and $kur(kur(s)) = kur(s)$ where s is some string. As for the kur function, we mean that a string is represented by a transfinite pair—a pair that is nested transfinitely many times. Next, we will look at generalization of this idea for transfinite sequences. Furthermore, it turns out that if one can interpret both representations in set theory language as two different (unique) "coding types" bijective to $\{0,1\}$, then one can also encode some additional information on top of those types.

The central idea behind folding of replication schema[257] is to abstract away from $\{0,1\}$ and look at sequences built up from $flat(s)$ and $kur(s)$ representations of a string s as generalization of binary strings. Let us capture these ideas of encoding additional information with two kinds of string representation more formally.

Recall, that through this section and in this paper in general, V denotes a proper class of all sets and Ord is a class of all ordinals (ordinal numbers).

Definition 89 - Coding base.

Assume the following:

- let *pairing function* $\tau : V \times V \to V$ map sets $\forall a, b \in V$ to an ordered pair (a, b) as defined in:

  $$\tau(a, b) = \{\{a\}, \{a, b\}\} = (a, b)$$

  where τ is a subclass defined on the proper class V as a formula in Set Theory language (trivially by invoking Axiom of Pairing);

[257] The idea of folding is exposed in *Definition 100* and replication schema is a *Lemma 110* about existence of the transitive model for replicata as a proper class. However, the concept of folding is to be covered much later in the paper.

- furthermore, since one can always index any ordered pair using *binary index* $I_2 = \{0,1\},\{0,1\} \subset Ord$ for any two finite pairs $x = \tau(a,b) = (a,b)$ and $y = \tau(c,d) = (c,d)$ (where $\{a,b,c,d\} \subset V$), it means that there is always a trivial bijection such that $a \mapsto c$ and $b \mapsto d$ over the same indexes $x_0 = y_0$ and $x_1 = y_1$.

Then, the map $B : I_2 \to \{a,b\}$ from the binary index $I_2 = \{0,1\},\{0,1\} \subset Ord$ to the preimage of the pairing function $\tau(a,b)$ defined on sets $\forall a,b \in V$ is called a *coding base*. For the sake of simplicity, we use the following notation to mean the same:

$$\text{if } B(I_2) = \tau^{-1}(a,b), \text{ then } B(0) = \tau(a,b)_0 = a \text{ and } B(1) = \tau(a,b)_1 = b$$

Next, we want to extend the *Definition 46 - Strings and substrings from sequences* to take Ord as the index class: $I = Ord$. This gives the most general definition of any string x as some function defined on ordinals or a transfinite sequence $(x_\alpha)_{\alpha \in Ord}$, and such string is called *flat representation* noted as $flat(x)$.

Definition 90 - Transfinite strings and substrings.

Let $(x_\alpha)_{\alpha \in Ord}$ be a transfinite sequence defined on all Ord and range over X, then x is a *transfinite string*. For simplicity (and for the rest of the paper), we call such transfinite string x simply *string*. If $k : Ord \to Ord$ is a contiguous function[258], then transfinite subsequence $(y_\gamma)_{\gamma \in Ord} := (x_{k(\alpha)})_\alpha$ is a called a *substring* y of the *string* x. We note this as $y \; \varepsilon \; x$.

Again, in case of defining the Replication Schema construct, we are mostly interested in and mean transfinite strings, when we talk about strings. Also, for most such strings (defined as transfinite sequences) we have each $x_\alpha \in ran(B)$ (for some coding base B).

Definition 91 - B-strings.

If B is the coding base, then a string with range in set B is called a *B-string*.

Before we can proceed to definition of $flat(x)$ and $kur(x)$ representations of string x, we would benefit from another notion derived from and defined over the class of all ordinals Ord.

Definition 92 - Partial ordinals.

Let $\lambda \in Ord$ and $\mathcal{P}(\lambda)$ be set containing all subsets of ordinal λ. Then every element of $\mathcal{P}(\lambda)$ is called *partial ordinal* or simply *partial*.

[258] See *Definition 44*

Lemma 99 - Bijectivity of partials and binary strings.

Assume the following:

- Let $\lambda \in Ord$ and $\mathcal{P}(\lambda)$ be set containing all subsets (and hence partials) of ordinal λ.
- $I_2 = \{0,1\}, \{0,1\} \subset Ord$ is a binary index set.
- For a given subset (partial) $S \in \mathcal{P}(\lambda)$, define the indicator function[259] $\chi_S(\eta)$ (where $\eta \in \lambda$) as:

$$\chi_S(\eta) = \begin{cases} 1 & \text{if } \eta \in S \\ 0 & \text{if } \eta \notin S \end{cases}$$

- Now, define function $g : \mathcal{P}(\lambda) \to \{0,1\}^\lambda$ such that for each partial $x \in \mathcal{P}(\lambda)$ there exists a transfinite binary sequence $y : \lambda \to \{0,1\}$, which is a binary string.

Then, g is bijective.

Proof. Sufficient to note that lemma is a special case of Cantor's Theorem but for subsets of ordinals. Specifically, one can explicitly define g for all $\forall S \in \mathcal{P}(\lambda)$ as $g(S) = \{(\eta, \chi_S(\eta)) : \forall \eta < \lambda\}$. □

Definition 93 - String Coding space.

The structure (X, g, λ, B) is called B-string *coding space* or just *string space* iff the following is true:

(i) $|\lambda|$ is the *expected length* of every B-string in X, i.e. $X = \{x : |x| = |\lambda| \wedge dom(x) = \lambda\}$ (see *Definition 86*);

(ii) B is the coding base - binary[260] case is defined as $B : I_2 \to \{a, b\}$ (see *Definition 89*);

(iii) $g : \mathcal{P}(\lambda) \times B \to V$ is a generation function that maps partials in $\mathcal{P}(\lambda)$ to their respective string representation as sets using the coding base B;

(iv) $X \subseteq \{g(S, B) : \forall S \in \mathcal{P}(\lambda)\}$ is the resulting set of B-strings following specific representation.

[259] Also see *Definition 120*

[260] Certainly, considering larger B allows for a much more general definitions, which we choose not to explore much at the moment.

We fix length of each string in the coding space as the cardinal $|\lambda|$ for some arbitrary large ordinal $\lambda \in Ord$. However, as explained in *subsection 8.2 - Expected Length of Strings*, the "length" of the string may not necessarily be equal to the cardinality of a set, but rather encoded as meta information. This is especially the case if we use many levels of nesting for the encoding as in *kur* representation. So it is good to fix this information as the part of the external structure - in our case the coding space itself. This is why such externally fixed string length is called the *expected length*. Technically, we want to keep our definitions abstract enough but still required for different encoding "implementations" to work in the same way[261].

Next, we define homomorphism and isomorphism notions.

Definition 94 - String space representation homomorphism.

Let (X, g, λ, B_x) and (Y, h, κ, B_y) be two B-string coding spaces s.t. $|X| \leq |Y|$. Then function $f : X \to Y$ is called a B-string space representation homomorphism iff $\forall x \in X, \exists y \in Y : a \; \varepsilon \; x \wedge b \; \varepsilon \; y \wedge f(a) = b$.

The above definition of homomorphism is very general as it tries to capture both structural and encoding similarity. Studying such complexity goes far beyond the main focus of the paper. We would be rather interested in much more at hand structural similarity.

Definition 95 - String space representation isomorphism.

Let (X, g, λ, B_x) and (Y, h, κ, B_y) be two B-string coding spaces. Then function $f : X \to Y$ is called a B-string space representation isomorphism iff the following holds:

(i) $\lambda = \kappa$

(ii) $|B_x| = |B_y|$

(iii) $|X| = |Y|$, meaning that f is bijective

(iv) $Y = f(X)$

Definition 96 - Canonical flat representation.

Let (X, e, λ, B) and $(Y, h, \lambda, id(I_2))$ be two isomorphic B-string coding spaces with $f : X \to Y$ being the respective isomorphism. Let $id(I_2) = \tau^{-1}(0, 1)$ be the identity to binary index set $I_2 = \{0, 1\}$ and the function $h(S, id(I_2)) = g(S)$, where g is specified as in *Lemma 99*, with the same explicit binary

[261] For example, Kuratowski and flat encoding must be interchangeable on the high level of our set abstraction that we model

encoding. Namely, h explicitly maps every partial in $\mathcal{P}(\lambda)$ to the set of tuples (using χ_S):

$$\forall S \in \mathcal{P}(\lambda) : h(S, id(I_2)) = g(S) = \{(\eta, \chi_S(\eta)) : \forall \eta < \lambda\}$$

where χ_S is also defined as the indicator function in *Lemma 99*.

Then, the space $(Y, h, \lambda, id(\{0,1\}))$ is called the *canonical flat representation* and function f is called the *canonical flat isomorphism*.

Furthermore, if $|Y| = |\mathcal{P}(\lambda)|$, we just have $Y = g(\mathcal{P}(\lambda))$ as the string space for canonical flat representation.

Definition 97 - Canonical Kuratowski representation.

Let (X, e, λ, B) and $(Y, h, \lambda, id(I_2))$ be two isomorphic B-string coding spaces with $f : X \to Y$ being the respective isomorphism. Similar to previous definition, we have the identity to binary index set $id(I_2)$ as coding base and g is again defined as in *Lemma 99*. Let the function $h(S, id(I_2)) = k(g(S))$, where k is recursively defined as in *Definition 119*, s.t. all $x_\alpha \in I_2, \alpha < \lambda$.

Then, h explicitly maps every partial in $\mathcal{P}(\lambda)$ to the set of nested tuples (using transfinite Kuratowski notation):

$$\forall S \in \mathcal{P}(\lambda), \exists x = g(S) : h(S, id(I_2)) = k(x_1, .., x_\lambda)$$

Furthermore, the space $(Y, h, \lambda, id(\{0,1\}))$ is called the *canonical Kuratowski representation* and function f is called the *canonical Kuratowski isomorphism*.

Let us also proof some special case lemma for binary code bases.

Lemma 100 - Isomorphic coding spaces.

Let (X, g, λ, B_x) and (Y, h, κ, B_y) be two B-string coding spaces such that $\lambda = \kappa, |B_x| = |B_y|, |X| = |Y|$. If $|X| = |\mathcal{P}(\lambda)|$, then there exists isomorphism $f : X \to Y$ and two spaces are isomorphic.

Proof. Without loss of generality assume $B_x = B_y = id(I_2)$. Given that $|Y| = |X| = |\mathcal{P}(\lambda)|$, we have $dom(g) = dom(h) = \mathcal{P}(\lambda)$. Both g and h are generation functions and injective by definition of the string space. This implies that both $g : \mathcal{P}(\lambda) \to X$ and $h : \mathcal{P}(\lambda) \to Y$ are also bijective. Now consider the composition $f = h \circ g^{-1}$, which will be the expected bijective $f : X \to Y$. □

Corollary 101.

Let $A = (X, g, \lambda, id(I_2))$ and $B = (Y, h, \lambda, id(I_2))$ be the canonical flat and

Kuratowski representations respectively, so that $|Y| = |X| = |\mathcal{P}(\lambda)|$. Then, both A and B are isomorphic.

Lemma 102 - Flat representation.

Let $h : \mathcal{P}(\lambda) \times B \to V$ be a generation function as in *Definition 96*. Then, B-string coding space (X, h, λ, B) has a *flat representation* iff there exists a canonical flat string representation $(Y, h, \lambda, id(I_2))$ and the following holds:

(i) X and Y are isomorphic with $f : X \to Y$ being an isomorphism;

(ii) $Y = \{h(S, id(I_2)) : \forall S \in \mathcal{P}(\lambda)\}$;

(iii) $X = \{h(S, B) : \forall S \in \mathcal{P}(\lambda)\}$;

(iv) if coding base B is replaced with $id(I_2)$, then $X' = \{h(S, id(I_2)) : \forall S \in \mathcal{P}(\lambda)\} = Y$

Proof. Follows from *(iv)* and *Lemma 100*. □

A very similar lemma can be stated for the Kuratowski representation. The only thing that is different would be the generation function h being defined as in *Definition 97*. We provide this lemma without the proof due to its similarity.

Lemma 103 - Kuratowski representation.

Let $h : \mathcal{P}(\lambda) \times B \to V$ be a generation function as in *Definition 97*. Then, B-string coding space (X, h, λ, B) has a *Kuratowski representation* iff there exists a canonical Kuratowski string representation $(Y, h, \lambda, id(I_2))$ and the following holds:

(i) X and Y are isomorphic with $f : X \to Y$ being an isomorphism;

(ii) $Y = \{h(S, id(I_2)) : \forall S \in \mathcal{P}(\lambda)\}$;

(iii) $X = \{h(S, B) : \forall S \in \mathcal{P}(\lambda)\}$;

(iv) if coding base B is replaced with $id(I_2)$, then $X' = \{h(S, id(I_2)) : \forall S \in \mathcal{P}(\lambda)\} = Y$

Next, we need to define a replication operation.

Definition 98 - Replication Operation.

Let X and Y be two isomorphic coding spaces such that:

- X is a part of the $flat$ representation space (X, g, λ, B);

- Y is a part of the *kur* representation space (Y, h, λ, B);
- g is a $flat$ generation map (as in *Definition 96*);
- h is a *kur* generation map (as in *Definition 97*);
- B is a coding base $B : \{0,1\} \to \{a,b\}$ with $B(0) = \tau(a,b)_0 = a$ and $B(1) = \tau(a,b)_1 = b$.

Then, *replication operation* $\mathcal{R}_\lambda^{\{f,k\}}(a,b)$ over two sets a and b is defined as a shorthand for either $X = \mathcal{R}_\lambda^f(a,b)$ or $Y = \mathcal{R}_\lambda^k(a,b)$, where $\{f,k\}$ are labels for $flat$ and *kur* representations respectively.

Definition 99 - Replication Blueprint.

Let λ be a limit ordinal, then a canonical flat representation $(P, h, \lambda, id(I_2))$, where $|P| = |\mathcal{P}(\lambda)|$, is called a *replication blueprint*.

Furthermore, if one takes canonical flat representation $(P, h, \lambda, id(I_2))$ as blueprint, then one can observe that it is possible to arrange chains of isomorphisms if wired together between infinitely many coding spaces. Such chains, if defined over sequence of coding bases $B_{\varsigma(\gamma)}, \gamma \in Ord$, which is determined by $\varsigma \in P$ and encoded in particular way, can form sequences of replication code. For example, if $\varsigma =$ "001..", then the corresponding sequence will be

$$\mathcal{R}_\lambda^{\{f,k\}}(0,1) \to \mathcal{R}_\lambda^{\{f,k\}}(\mathcal{R}_\lambda^{\{f,k\}}(0,1),1) \to \mathcal{R}_\lambda^{\{f,k\}}(0,\mathcal{R}_\lambda^{\{f,k\}}(\mathcal{R}_\lambda^{\{f,k\}}(0,1),1)) \to \dots$$

but same can be specified more formally (see the next *Definition 100*). Also, it seems that one can manipulate the choice of $\{f,k\}$ labels which brings us one step closer to exposing the ideas of encoding additional information with two kinds of string representations as two types of coding spaces.

Rather than representing a chain of $\mathcal{R}_\lambda^{\{f,k\}}(a,b)$ isomorphisms as a chain of functions (and invoking the Replacement schema), we choose to fold each of such unique chains directly into a specific sequence or actually a string of isomorphisms.

Definition 100 - Folding of Replication Isomorphisms into Strings.

Assuming that:

- B is a coding base (usually $id(I_2)$);
- $\lambda \in Ord$ is the expected length and P is a partially ordered index set (usually a coding space), where both λ and P are usually determined by the replication blueprint;

- x and y are the respective B-strings from the index set P, where $|x| = |y| = \lambda$;
- $\mathcal{R}_\lambda^{\{f,k\}}(a, b)$ is replication operation over two sets a and b;
- $r : 0 \mapsto f, 1 \mapsto k$ is a helping map on I_2 (where $\{f, k\}$ are labels for $flat$ and kur representations respectively).

Then, we say that one can *fold* a chain of replication isomorphisms (each defined using $\mathcal{R}_\lambda^{\{f,k\}}(a, b)$) into a corresponding sequence, namely, a *replication string*. The resulting string can be mapped (using subindexes x and y) one-to-one to a member of the collection $F_{x,y\in P}$ as the following transfinite induction[262]:

- **Base case (when $\alpha = 0$):**

$$F_{x,y\in P}(0) = \mathcal{R}_\lambda^{r(x(0))}(B(0), B(1))$$

  This means the domain of 0th isomorphism from the sequence is equivalent to the blueprint coding space P.

- **Successor ordinal $\alpha+1$:** If α is an ordinal, $\alpha < \lambda$ and $F_{x,y\in P}(0)$ has been defined, then:

$$F_{x,y\in P}(\alpha + 1) = \begin{cases} \mathcal{R}_\lambda^{r(x(\alpha+1))}(F_{x,y\in P}(\alpha), B(1)), & \text{if } y(\alpha+1) = 0 \\ \mathcal{R}_\lambda^{r(x(\alpha+1))}(B(0), F_{x,y\in P}(\alpha)), & \text{otherwise} \end{cases}$$

- **Limit ordinal λ:** If λ is a limit ordinal (i.e., there is no immediate predecessor to λ), then:

$$F_{x,y\in P}(\lambda) = \bigcup_{\alpha<\lambda} F_{x,y\in P}(\alpha).$$

At the limit ordinal λ, the union of the entire collection of isomorphism strings between coding spaces for a given blueprint yields a set. This set constitutes a coding space whose string representation has cardinality at least as large as that of the indexing set P, as established above. However, this assertion demands further clarification and a more precise notation. We'll return to the subject of replication folding in later sections. It will remain central to exposing of the deeper interconnection between Large Cardinals and Replication Schema.

We conclude this subsection with the reference to the *Replication Schema - Lemma 110*, which comes shortly after.

[262]Here, x is the "type" subindex and y is the subindex for the isomorphism "sequential" code.

8.5 Transfinite Replicata

We now consolidate the key points about replication discussed earlier. The core idea of a *replication scheme* is a generalization of the concept of *replicata* (see *Definition 61*), applied to *Transfinite strings and substrings* (see *Definition 90*) and later extended to string coding spaces. Although the definition of replicata is already broad—covering arbitrary ordinals κ and λ—we aim to further generalize and validate it to ensure it fully accommodates transfinite strings as intended.

Definition 101 - Transfinite Replicata.

Assume:

1. (X, g, λ, B) is B-string *coding space* (see *Definition 93*), where:
   (i) B is a coding base (usually $id(I_2)$);
   (ii) $g : \mathcal{P}(\lambda) \times B \to V$ is a generation function that maps subsets in $\mathcal{P}(\lambda)$ to their respective string representation as sets using the coding base B (usually $g = flat$);
   (iii) $X \subseteq \{g(S, B) : \forall S \in \mathcal{P}(\lambda)\}$ is the resulting set of B-strings following specific representation (if g is $flat$, then $X = [B]^{|\lambda|}$).
2. $|\lambda|$ is the *expected length* of every B-string in X, i.e. $X = \{x : |x| \leq |\lambda| \wedge dom(x) = \lambda\}$ (see *Definition 86*).
3. $\kappa \in Ord$ is the maximum *replication degree*, s.t. $\kappa < \lambda$ and $|\kappa| = |\lambda|$.
4. If $B^\star \supset B$, then $Str(B^\star) = \{f : Ord \to B^\star\}$ is a proper class of all transfinite $B^\star$-strings with canonical $(flat)$ representation, so that $Str_{|\lambda|}(B^\star) = \{\ s : |s| = |\lambda| \wedge s \in Str(B^\star)\}$ (by transfinite induction on Ord).
5. π is the production function over the subset of transfinite strings $g(Str_{|\lambda|}(\Sigma^\star))$ (where $\Sigma^\star \supset B$ is an extended λ-SEF alphabet - see below), containing only syntactically valid B-string enumeration formulas or simply λ-SEFs.
6. Also, since SEFs are special kind of strings with extended alphabet, we say that if s_x is a λ-SEF, then its length never exceeds the produced λ-string, i.e. $|s_x| \leq |\pi(s_x)|$.
7. λ-SEF is considered syntactically valid iff:
   - it is build over the alphabet $\Sigma^\star \supset \{B(0), B(1), (,)\}$, which is extended with ordinals $\alpha \leq \kappa$, that can be placed between doubled parentheses. For example, countable degree of replication in SEF

"(x)" becomes "$(\omega(x)\omega)$", when translated into λ-SEF language, to indicate the *transfinite replication degree*; but, then again, for simplicity and where it is clear from the context we keep abusing the exponential notation such as "$(x)^{\alpha}$" to mean the degree of replication;

- all parentheses are paired and ordinal indicates the degree of replication after the closing parenthesis[263];
- it is trivial to show that such notation allows to scan the $\Sigma^{\star}$ string of the potential formula and always find the correct pair which results in syntactical proof of every λ-SEF[264].

8. $dom(\pi) = R_{\kappa}^{\lambda}$ is a set of equivalence class labels of λ-SEFs ($R_{\kappa}^{\lambda} \subset Str_{|\lambda|}$), where κ is the greatest replication degree and the length of each λ-SEF $s_x \in R_{\kappa}^{\lambda}$ does not exceed λ, i.e. $|s_x| \leq |\lambda|$.

9. $ran(\pi) \subseteq X$ is a subset of transfinite B-strings of the coding space X.

Then, we say that the space $(R_{\kappa}^{\lambda}, P_{\varepsilon}, \pi)$ is called *transfinite replicata* or simply λ-replicata iff the following holds:

(i) String $x \in ran(\pi)$ is called *perfectly λ-fair* iff x does not contain any λ-substring $\nexists y \in ran(\pi) : y \ \varepsilon \ x$ that can be expressed as the concatenation of at least two identical substrings, i.e., $y = z \cdot z$ where $|y| = |\lambda|$.

(ii) Similarly, string $x \in ran(\pi)$ is called imperfectly λ-fair iff x does not contain any λ-substring $\nexists y \in ran(\pi) : y \ \varepsilon \ x$ that can be expressed as the concatenation of κ-many substrings or, equivalently, to have a replication degree $\kappa \leq \lambda$, i.e., $y = (z)^{\kappa}$, where $|y| = |\lambda|$.

(iii) String $x \in ran(\pi)$ is called λ-unfair iff it has fixed length $|x| = |\lambda|$ and is neither (perfectly or imperfectly) λ-fair[265]

(iv) P_{ε} is a *transfinite substring partial order* (tspo) generalized from *Substring-Partition Partial order (sppo) on $R_{<\omega_1}$ - Definition 66* by considering transfinite strings of length $\lambda \geq \omega_1$ for kind partitions.

[263] For example, if y is a variable in SEF $x =$ "$(y)^{\kappa}$", then $\pi(x)$ is string where y will be concatenated κ-many times.

[264] As a side note, very early on, when contemplating the design of the SEF language we said that every syntactically valid formula or expression is arbitrary mapped to or evaluated as truth. For example, all elements of $dom(\pi)$ are valid and hence true. And, on the contrary, one can otherwise, say that if such language is λ-recognizable, then it is also λ-decidable, which is more of interest for the large theory of computability of infinite languages

[265] Note that all the lesser β-unfairs ($\forall \beta : \beta < \lambda$) can still be substrings of both λ-unfair or imperfectly λ-fair.

Lemma 104 - Bijectivity of production in λ-replicata.

Let $(R_\kappa^\lambda, P_\varepsilon, \pi)$ be a λ-replicata, where $\kappa < \lambda$ is the maximum replication degree. Then, the production function π is bijective.

Proof. To establish bijectivity of π, we analyze the cardinality of R_κ^λ and the properties of π.

First, note that R_κ^λ contains all λ-strings, i.e., all transfinite strings of length λ. Since the set of all subsets of λ, $\mathcal{P}(\lambda)$, can be mapped to λ-strings (e.g., via indicator functions), it follows that $|R_\kappa^\lambda| \geq |\mathcal{P}(\lambda)|$.

Next, observe that π is injective by definition. For each $s_x \in R_\kappa^\lambda$, where $|s_x| \leq |\lambda|$, the production function maps s_x to a unique string $x = \pi(s_x)$ of length $|x| = |\lambda|$. This ensures that no two distinct λ-SEFs in R_κ^λ produce the same string under π.

Finally, since R_κ^λ is constructed to represent all possible λ-strings, its cardinality cannot exceed $|\mathcal{P}(\lambda)|$. Thus, $|R_\kappa^\lambda| = |\mathcal{P}(\lambda)|$.

Combining these observations, π is a bijection between R_κ^λ and the set of λ-strings, as it is both injective and surjective. This completes the proof. □

Before we proceed with considering a respective model for λ-replicata, we need to review our axiomatic assumptions about Dependent Choice. For $\lambda < \omega_1$ it was sufficient to assume DC. That allowed us to construct an ω-replicata in *Lemma 85* with countable replication (see *Lemma 81*). In general, a similar principle would be necessarily for constructing replicata if $\lambda \geq \omega_1$. In that sense, when we say λ-replicata is constructed it means we assume the necessary stronger version of Dependent Choice called DC_λ (see *Definition 42*). The latter allows us to use λ-replication in λ-replicata (from λ-strings).

Also note that the alternative definition of *perfectly λ-fair* string is simply a special case of *imperfectly λ-fair* strings for $k = 2$. However, it is important to have the production function π to behave as a bijection (see *Lemma 104*).

Bijectivity of π ensures the essential properties required for λ-replicata. This was demonstrated over the α-tranches, where each α-tranche corresponds to a new cardinality $|\alpha| = |\mathcal{P}(\lambda)|$, built from the previous one. Replicata are defined to "exist" or "seat" on such cardinalities. Using transfinite induction, this was shown for each limit cardinal $\aleph_{\omega_\alpha} = \lambda$. Furthermore, the proof of *Theorem 104* depends on the tspo P_ε being well-defined for every λ-replicata. To clarify this, we aim to construct a binary relation ε on $P_\varepsilon \subseteq R_\kappa^\lambda$ and demonstrate that it forms a partial order (see *Definition 69*). This will involve partitioning SEFs by their degree of replication, as outlined in *Definition 66*. Before presenting this, we will introduce a lemma on well-founded partial orders to refine the definition and restrict tspo P_ε for perfect λ-fairs to be strictly well-founded.

Lemma 105 - Well-Foundedness Under Finite Differences.

Let λ be a (possibly infinite) index set (e.g., $\lambda = \omega_1$). Let $A \subseteq \{0,1\}^\lambda$ be a set of λ-strings closed under "being almost disjoint" property as well as "being almost adjoint":

1. For all $x, y \in A$, the bitwise XOR $z = x \oplus y$ has a finite symmetric difference (see *Lemma 57*) with z containing only the concatenation of "(0)" (all-0) infinite substrings except for finitely many segments of $0, 1$ mix-ins (i.e., $x \oplus y$ differs from the all-0 sequence in only finitely many coordinates).

2. A contains all x and y for which $z = x \oplus y$ has a finite symmetric difference, as well as x and y for which $z = \neg(x \oplus y)$.

Define a function rank: $A \to$ Ord (the class of all ordinals) by *transfinite recursion*:

$$\begin{aligned}\text{rank}(x) \;=\; \min\big\{\, \alpha \;:\; \exists\, y \in A \text{ with } \text{rank}(y) < \alpha \\ \text{and } (x \oplus y) \text{ is nonzero in only finitely many coordinates}\big\}.\end{aligned}$$

Then the induced ordering $\prec$ on A given by $x \prec y \iff \text{rank}(x) < \text{rank}(y)$ is a well-founded partial order: there are no infinite descending chains with respect to $\prec$.

Proof. We establish two main claims:

1. *Well-definedness of* rank(x) *by transfinite recursion.*

2. *Well-foundedness of the induced partial order.*

(1) Well-definedness. We employ ordinal (transfinite) recursion. Suppose we are assigning rank(x) to each $x \in A$. At stage α, we say

$$\begin{aligned}\text{rank}(x) \;=\; \alpha \quad \text{iff} \quad & \alpha \text{ is the least ordinal for which there exists some } y \in A \\ & \text{with } \text{rank}(y) < \alpha \text{ and } x \oplus y \text{ has finite symmetric difference} \\ & \text{- finitely comparable binary representation of a natural} \\ & \text{number uniquely corresponding to } y\end{aligned}$$

Because all ranks rank(y) for $y \in A$ are by definition assigned at earlier stages (i.e., $< \alpha$), and ordinals are well-ordered, there is always a *least* α (if any) satisfying the stated property. The set of candidate α is nonempty (we can always take an ordinal larger than all rank(y) for y that differ from x finitely). Thus rank(x) is assigned consistently for every $x \in A$. This shows rank is *well-defined*.

(2) Well-foundedness. Consider the induced partial order defined by

$$x \prec y \quad \iff \quad \text{rank}(x) < \text{rank}(y).$$

Suppose, for contradiction, that there is an infinite strictly descending chain

$$x_0 \succ x_1 \succ x_2 \succ \cdots \quad \implies \quad \text{rank}(x_0) > \text{rank}(x_1) > \text{rank}(x_2) > \cdots .$$

But an infinite strictly decreasing sequence of ordinals cannot exist, because the class of ordinals is *well-ordered by* $<$. Hence no such descending chain can occur in $(A, \prec)$.

Thus $(A, \prec)$ is *well-founded*. □

Lemma 106 - Almost adjoint (disjoint) λ-fair.

Let $z = x \oplus y$, where $|x| = |y| = \lambda$. Then, z is finite symmetric difference (or its inverse) iff x and y are both perfectly fair λ-strings.

Proof. Consider extending the definition of perfectly fair string to $\lambda \geq \omega$ and the proof follows. □

Corollary 107 - Closure of perfectly λ-fairs.

Let N be a model of $ZF + DC_\lambda$, s.t. $N \models (R^\lambda_\kappa, P_\varepsilon, \pi)$ to be a λ-replicata and $\kappa \leq \lambda^+$, then $S_{PF} \subset \pi(R^\lambda_\kappa)$ is closed under the property of being almost adjoint or almost disjoint.

It seems that it is possible to further refine the partial order relation on P_ε. One way of looking at it is that λ-fair strings have comparably rich structure, similar to λ-unfairs, as every λ-fair contains other λ-fair substrings that are intermixed or interleaved with smaller δ-fair unless it is most λ-perfect (when it consists only of such $\delta < \lambda$ substrings). Another way is to think of interleaves in large λ-fair as form of partial order by reverse contiguity (see - *Definition 56*). Clearly same principles are reflected in λ-unfair strings which are build from SEF formulas using a mix of λ-fair and δ-fair (where $\delta < \lambda$), but simply have substrings with greater degree of replication.

Let us next define P_ξ which acts as a restriction of P_ε by behaving like $\prec$ in *Lemma 105 - Well-Foundedness Under Finite Differences* on $S_{PF} \subset \pi(R^\lambda_\kappa)$ for given λ-replicata.

Lemma 108 - Well-founded partial order on λ-replicata.

Let N be a model of $ZF + DC_\lambda$, s.t. $N \models (R^\lambda_\kappa, P_\varepsilon, \pi)$ to be a λ-replicata and $\kappa \leq \lambda^+$, then $P_\xi \subseteq P_\varepsilon$ is a well-defined partial order (tspo), which is built from P_ε, such that it is:

- λ-cc
- well-founded

Note that the alternative approach to endowing λ-replicata with a well-founded partial order would be to restrict P_ε with equivalence relation over the the corresponding ultrafilter U_α. But for most purposes we will stick to our P_ξ construction with well-foundedness provided by ranks over finite symmetrical difference and its inverse.

The last point allows us to make a rather interesting observation. It is possible to define a well-founded replicata class on all ordinals that would have an isomorphic transitive model.

Definition 102 - Well-founded Replicata at Limit Cardinal.

Let $\Lambda(\lambda) = (R_\lambda^{<\lambda^+}, P_\xi, \pi)$ be a shorthand for well-founded λ-replicata at the limit cardinal λ with replication degree also $\kappa = \lambda$.

Definition 103 - Well-founded Replicata Class.

If N is a model of $ZF + DC_\lambda$, s.t. $N \models \Lambda(\lambda)$ *is a well-founded replicata* and λ is a limit cardinal, then we call Λ a well-founded replicata class defined on the transitive class of all ordinals Ord.

Lemma 109 - Transitive Collpase of Replicata Class.

If N is a model of $ZF + DC_\lambda$, s.t. $N \models \Lambda$ *is a well-founded replicata class*, then there exists a transitive model $(M, \in)$ of $ZF + DC_\lambda$ isomorphic to tspo on Λ.

Proof. Since $\Lambda(\lambda) = (R_\lambda^{<\lambda^+}, P_\varepsilon, \pi)$ for each $\lambda \in Ord$, then one can extend tspo $P_{\varepsilon^*} = (\Lambda, \varepsilon^*)$ to be well-founded[266] on the whole replicata class Λ. There exists a transitive model $(M, \in)$ isomorphic to (Λ, ε^*) by Mostowski Collapsing Theorem. □

Now a much stronger and succinct statement can be made about the replicata class. We are going to construct such class by putting together previous arguments.

Lemma 110 - Replication Schema.

There exists a proper class of replicata Λ.

[266] Similar to p. 69 in [6].

Proof. The lemma follows naturally if there exists such a $ZF + DC_\lambda$ model N as in *Definition 103*, where $N \models \Lambda(\lambda)$ *is a replicata* and $\lambda \in Ord$ is a limit cardinal. Such a model exists for every such λ, so that:

$$\Lambda(\lambda) = (R_\lambda^{<\lambda^+}, P_\varepsilon, \pi)$$

from *Transitive Collpase of Replicata Class - Lemma 109.*

Now, we aim to put all these replicata structures together and demonstrate how they jointly form a proper class Λ while still exhibiting the defining properties of the replicata structure across all its elements.

We define this class structure as:

$$\Lambda = (R_\Lambda, P_{\varepsilon^*}, \pi)$$

where:

$$R_\Lambda = \bigcup_{\forall \lambda \in Ord} R_\lambda^{<\lambda^+}$$

The construction proceeds as follows:

1. **Proper Class Formation:** Since for each $\alpha \in Ord$, there exists a smallest limit cardinal λ such that $\alpha < \lambda$, we can associate a replicata structure $\Lambda(\lambda)$ with each such λ. The union R_Λ extends over all ordinals, ensuring that Λ is not merely a set but a proper class, as it cannot be bounded by any set-sized collection.

2. **Preservation of Replicata Properties:** The properties defining a replicata—such as well-foundedness, transitive substring partial ordering (tspo), and the existence of a suitable bijection π—must be shown to hold for the entire class Λ.

    - *Well-Foundedness:* Since each $R_\lambda^{<\lambda^+}$ is well-founded by construction, and well-foundedness is preserved under unions of chains of well-founded structures, R_Λ remains well-founded.
    - *tspo Preservation:* To demonstrate that R_Λ preserves the tspo property, note that each tranche $R_\lambda^{<\lambda^+}$ satisfies tspo. Given the construction of R_Λ as a union over λ-tranches, tspo is preserved globally, as no contradictions arise from the ordering relations defined on each tranche.
    - *Bijection π:* The bijection π is defined tranche-wise. Since π is trivially bijective per tranche and the tranches are disjointly indexed by distinct limit cardinals λ, the overall mapping π remains bijective over R_Λ.

3. **Closure Under Relevant Operations:** The class Λ must be closed under operations relevant to its defining properties. By construction, each $\Lambda(\lambda)$ is closed under the operations defined in *Transitive Collpase of Replicata Class*. The union R_Λ inherits this closure, as all such operations are performed within each tranche and extend coherently over the entire union.

4. **Proper Class Verification:** The final step is to confirm that Λ is indeed a proper class. Assume for contradiction that Λ is a set. Then R_Λ would be bounded by some cardinal κ, contradicting the construction that for every ordinal α, there exists a corresponding limit cardinal $\lambda > \alpha$. Hence, Λ must be a proper class.

Therefore, the union of all replicata $\Lambda(\lambda)$ over all $\lambda \in Ord$ forms a proper class $\Lambda = (R_\Lambda, P_{\varepsilon^*}, \pi)$, completing the proof. □

8.6 Nontrivial elementary embeddings

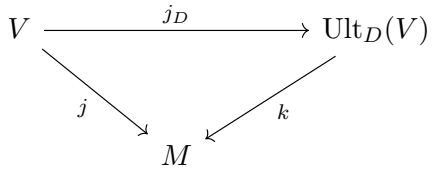
Consider the following lemma on relation between nontrivial elementary embedding and existence of normal measures quoted from [6] without proof.

Lemma 111 - Nontrivial elementary embedding.

Let $j : V \to M$ be a nontrivial elementary embedding, let κ be the least ordinal moved, and let D be the ultrafilter on κ such that $X \in D \iff \kappa \in j(X) \quad (X \subset \kappa)$. Let $j_D : V \to \mathrm{Ult}_D(V)$ be the canonical embedding of V into the ultrapower $\mathrm{Ult}_D(V)$. Then there is an elementary embedding

$$k : \mathrm{Ult}_D(V) \to M$$

such that $k(j_D(a)) = j(a)$ for all a. The following diagram commutes:



It is important to note that the above result relies on D being a σ-complete ultrafilter. The symbol Ult denotes the transitive collapse[267] of the ultrapower, and $\mathrm{Ult}_D(V)$ being an inner model. In fact, [6] shows that if j is a nontrivial elementary embedding of the universe, then there exists a (quasi) measurable cardinal, and vice versa.

On one hand, the construction of a κ-complete ultrafilter from an elementary embedding yields the above commutative diagram. Hence, for each

[267] by the Mostowski Collapsing Theorem

normal measure $D = \{X \subset \kappa : \kappa \in j(X)\}$, there is always a corresponding κ-complete ultrafilter. Note that the measure $D = \{X \subset \kappa : \kappa \in j(X)\}$ defined from the elementary embedding is indeed normal: Let f be a regressive function on some $X \in D$. Then $(jf)(\kappa) < \kappa$, and if $\gamma = (jf)(\kappa)$, then $f(\alpha) = \gamma$ for almost all α.

On the other hand, as evident from the above commutative diagram, if $\mathrm{Ult}_D(V)$ is an inner model, then there exists j via k — the nontrivial elementary embedding of the universe into M.

8.7 Inaccessible cardinals and Ultrafilters

Recall [6], that by definition, a (strongly) inaccessible cardinal κ is an uncountable regular strong limit cardinal. This means, in particular, that κ is a strong limit cardinal: for every $\lambda < \kappa$, we have $2^\lambda < \kappa$. In other words, not only does κ have uncountable cofinality (making it regular), but also no "small" powerset can reach κ. Inaccessible cardinals are precisely those large cardinals with such strong limit properties.

So far we have discussed (weak) limit cardinals that turned out to be quasi-large[268]. Interestingly enough we will now look into "progression" of how the world of quasi-large connects with the world of large cardinals. Namely, we want to look at what immediate consequences the assumption of an additional axiom will have for our previous results.

Let I be an axiom stating that *there exists an inaccessible cardinal*. Again, we remind here several known and relevant results about inaccessible cardinals[269]:

Lemma 112.

The following holds true for ZFC:

- If κ is an inaccessible cardinal, then V_κ is a model of ZFC.
- If ZFC is consistent, then so is $ZFC + I$.

But most importantly it cannot be proved in ZFC that inaccessible cardinals exist. It is consistent in ZF (or $ZF{+}DC_\lambda$, or $ZF{+}UltrafilterTheorem$, etc.) that one has a cardinal κ carrying a "normal, κ-complete ultrafilter,"[270] yet κ is not (and cannot be proved) to be strongly inaccessible in that model. Equivalently, a model of $ZF + DC_\lambda$ like N can fail to satisfy the statement "There is an inaccessible cardinal" even while it has quasi-measurable cardinals. So one has to always assume an additional axiom in order to work with large cardinals.

[268] Again, see *Definition 73*

[269] See *Inaccessibility of Inaccessible Cardinals* on p.167 in [6]

[270] Also see *Definition 106* in the upcoming subsection

Next, we make few observations about ultrafilters. Let $\lambda \in Ord$ be some large enough fixed limit cardinal, so that $\Lambda(\lambda)$ is a replicata in N. Until now, we may have guessed the possibility of existence of the ultrafilter on such replicata $\Lambda(\lambda)$ as indicated by the restricted form of *Ultrafilter Theorem* to the case of $\lambda = \omega_1$ (see, respectively, result reference in *Theorem 55* and the proof of the equivalent result in *Lemma 56*). Yet we did not attempt to explain neither how it can be "constructed", nor if *Ultrafilter Theorem* can be shown in full.

In fact, one of immediate corollaries of *Lemma 108* is that tspo P_ε implies significant amount of structure including the existence of ultrafilter U on each λ-replicata.

Lemma 113 - Ultrafilter on replicata.

Let N be a model of $ZF + DC_\lambda$ and λ be a limit cardinal, s.t. $N \models \Lambda(\lambda)$ is a replicata. Then, $\Lambda(\lambda)$ carries a nontrivial ultrafilter U.

Proof. We start by invoking λ-fair determinacy (which can be generalized from *Theorem 47* and follows from DC_λ). This allows us to play games and choose between all kinds of λ-fair and $< \lambda$-fair.

Next, we construct the filter by identifying two disjoint sets:

(i) **measure-one set**: let X_1 be the union of all λ-fair, λ-unfair and include δ-fair: $\forall \delta < \lambda$ (which includes ω-fair);

(ii) **measure-zero set**: let X_0 contain all finite unfair, s.t. $|X_0| = \aleph_0$.

Observe that X_1 can be partially ordered by tspo $P_\varepsilon \supset X_1$ in such a way that it is a filter. In fact, if we take $U = X_1$, then it will be a maximum and nontrivial filter, since every set $x \notin X_1$ must be in X_0. □

Corollary 114 - Ultrafilter Theorem.

Let N be a model of $ZF + DC_\lambda$ and λ be a limit cardinal. If F is a filter on $X \subset \lambda$, then $\forall \lambda \in Ord$ filter F can be extended to the ultrafilter $U \subseteq \lambda$ in model N.

Proof. Assume that $\forall \lambda \in Ord$ one can construct α-tranches $\alpha < \lambda$ as in *Lemma 104*. Note that for each α-tranche there is an ultrafilter U_α on $\mathcal{P}_\lambda(\lambda)$ (by the *Lemma Ultrafilter on replicata*). Essentially, as long as one can pick λ large enough to include (cover) $X \subset \lambda$, one can always extend a filter on X to such ultrafilter. □

Evidently, even stronger claims about structure induced by tspo P_ε can be made. Especially if we start to work in the context of a larger model. Let us suppose that N^* is a model of $ZF + DC_\lambda + I$, which is formed as a

natural (not elementary) extension of $N \models ZF + DC_\lambda$ that we have worked with so far, namely $N^* \models \Lambda$ *is a replicata class.* Next, we will show the existence of a measurable cardinal in N^*.

Theorem 115 - N^* implies measurable cardinals.

Let N^* be a model of $ZF + DC_\lambda + I$, then $N^* \models \exists$ measurable cardinal.

Proof. By definition $N^* \models \Lambda$ *is a replicata class.* Let $\Lambda(\lambda) \subset \Lambda$ be a replicata such that $\kappa = |\Lambda(\lambda)|$ is the smallest inaccessible cardinal. It follows from *Lemma Ultrafilter on replicata* that κ carries a nontrivial ultrafilter U (via respective bijection). Notably, such U can also act as zero-one measure on the whole replicata set and specifically on κ. Then, the corresponding cardinal κ must be a measurable cardinal (by *Definition 75*). □

8.8 Fine and Normal Measures

Finally, we will touch upon the fine and normal measures on $\mathcal{P}_\kappa(A)$ (see *Definition 87*). Again, let A be a set of cardinality greater than or equal to κ. Recall the following definitions and results from [6].

Definition 104 - Fine measure.

For each $x \in \mathcal{P}_\kappa(A)$, let $\hat{x} = \{y \in \mathcal{P}_\kappa(A) : x \subset y\}$, and consider the filter on $\mathcal{P}_\kappa(A)$ generated by the sets $\hat{x}$ for all $x \in \mathcal{P}_\kappa(A)$; that is, the filter

$$\{X \subset \mathcal{P}_\kappa(A) : X \supset \hat{x} \text{ for some } x \in \mathcal{P}_\kappa(A)\}. \tag{3}$$

We call U a *fine measure* on $\mathcal{P}_\kappa(A)$ if U is a κ-complete ultrafilter on $\mathcal{P}_\kappa(A)$ that extends the κ-complete filter (3); i.e., $\hat{x} \in U$ for all $x \in \mathcal{P}_\kappa(A)$.

Note that if κ is a regular cardinal, then the filter (3) is κ-complete.

Definition 105 - Strongly compact cardinal.

An uncountable regular cardinal κ is *strongly compact* if for any set S, every κ-complete filter on S can be extended to a κ-complete ultrafilter on S.

Lemma 116.

The following are equivalent for any regular cardinal κ:

(i) For any set S, every κ-complete filter on S can be extended to a κ-complete ultrafilter on S.

(ii) For any A such that $|A| \geq \kappa$, there exists a fine measure on $\mathcal{P}_\kappa(A)$.

(iii) The language $L_{\kappa,\omega}$ satisfies the compactness theorem.

Definition 106 - Normal measure.

A fine measure U on $\mathcal{P}_\kappa(A)$ is *normal* if $f : \mathcal{P}_\kappa(A) \to A$ is such that $f(x) \in x$ for almost all x, then f is constant on a set in U.

When we say "for almost all x", we mean for $x \in U$ (that is, a measure-one subset of $\mathcal{P}_\kappa(A)$). Also, when we say "f is constant on a set in U", we mean there exists some member $Y \in U$ on which f takes a single constant value. It does not refer to being constant on each individual $x \in \mathcal{P}_\kappa(A)$ (since each x is itself a subset of A), nor does it require that f is constant on every set in U. Rather, the requirement is that there is at least one "large" set $Y \in U$ (where "large" means Y is in the ultrafilter U) such that for all $x \in Y$, $f(x) = \beta$ for some fixed $\beta \in A$. This is the essence of normality for a fine measure: any regressive function (one where $f(x) \in x$) must "collapse" to a constant value on a measure-one set.

Definition 107 - Supercompact cardinal.

An uncountable cardinal κ is *supercompact* if for every A such that $|A| \geq \kappa$, there exists a normal measure on $\mathcal{P}_\kappa(A)$.

We will also reference here an important characterization[271] lemma for λ-supercompact cardinals.

Lemma 117 - Characterization of λ-supercompact.

Let $\lambda \geq \kappa$. A normal measure on $\mathcal{P}_\kappa(\lambda)$ exists if and only if there exists an elementary embedding $j : V \to M$ such that

(i) $j(\gamma) = \gamma \quad$ for all $\gamma < \kappa$;

(ii) $j(\kappa) > \lambda$;

(iii) $M_\lambda \subset M$, i.e., every sequence $\langle a_\alpha : \alpha < \lambda \rangle$ of elements of M is a member of M.

A cardinal κ is called λ-*supercompact* if it satisfies *Lemma 117*.

Next, since we have finished listing known definitions and results from [6], let us focus on showing that fine measure exists on each λ-replicata.

Theorem 118 - Fine measure on replicata.

Let N^* be a model of $ZF + DC_\lambda + I$. If $N^* \models \exists$ a replicata $\Lambda(\lambda)$, then $\Lambda(\lambda)$ carries a fine measure D.

[271] See p.375 in [6]

Proof. Let δ, γ, κ be regular uncountable cardinals s.t. $\delta^+ \leq \kappa$. Also, let λ be the smallest limit cardinal s.t. for some set A (of cardinality $\gamma = |A|$), we have $\kappa \leq \gamma \leq \lambda$ and, respectively, $|\mathcal{P}(\kappa)| \leq |\mathcal{P}(A)| \leq |\mathcal{P}(\lambda)|$, where $\kappa < \lambda$. By *Definition 102*, we have that $\Lambda(\lambda) = (R_\lambda^{<\lambda^+}, P_\varepsilon, \pi)$ is a λ-replicata (as modeled by N^*). Following from *Lemma 113*, we have a non-trivial ultrafilter U on $\Lambda(\lambda)$. We want to show that there exists κ-complete ultrafilter $D \subseteq U$, which is also a fine measure on $\mathcal{P}_\kappa(A)$.

Recall that $R_\lambda^{<\lambda^+}$ is collection of "$< \lambda^+$"-strings with replication degree λ. It does mean that if one constructs a smaller sub-collection $R_\delta^{<\delta^+}$, then it can be arranged that those "$< \delta^+$"-strings are padded (with a constant, e.g. zero) to match the greater length of "$< \lambda^+$"-strings and illustrate the inclusion. Observe that such smaller replicata will have the following cardinality[272]: $|\Lambda(\delta)| = |R_\delta^{<\delta^+}| = |\mathcal{P}_{\delta^+}(\delta)| = |\mathcal{P}(\delta)|$. This allows to observe the following inclusion chain:

$$\mathcal{P}(\delta) \subseteq \mathcal{P}_\kappa(\delta) \subset \mathcal{P}_\kappa(\kappa) \subset \mathcal{P}_\kappa(\gamma) \subset \mathcal{P}_\kappa(\lambda) \subset \mathcal{P}_{\lambda^+}(\lambda) \subseteq \mathcal{P}(\lambda) \tag{4}$$

Now let F be a club filter generated by club sets on $\mathcal{P}_\kappa(A)$. Following from *Lemma 97*, such filter F is κ-complete. Furthermore, assume F is constructed as the κ-complete filter (3) in *Definition 104*. Namely:

$$F = \{X \subset \mathcal{P}_\kappa(A) : X \supset \hat{x} \wedge (\exists x) x \in \mathcal{P}_\kappa(A)\}$$

where $\hat{x} = \{y \in \mathcal{P}_\kappa(A) : x \subset y\}$. The question presents itself if F can be extended to some κ-complete ultrafilter D on $\mathcal{P}_\kappa(A)$.

Recall that substring relation is defined[273] as subsequence with additional structure (induced by concatenation or by contiguity), but, in an essence, ε is just a special kind of subset $\subset$ relation. Hence, one can try to construct a filter like F, but using strings and substrings and map them one-to-one to sets and subsets.

Consider for the moment the case that we actually ignore, when A is a "smaller" set as in $|A| = \delta$ and $\kappa = \delta^+$. If G is a non-trivial κ-complete ultrafilter on $R_\delta^{<\delta^+}$ (by *Lemma 113*), where $|R_\delta^{<\delta^+}| = |R_\delta^{<\kappa}| = |\mathcal{P}_\kappa(\delta)|$, then G would extend the above F. To show this, let us map $x \in \mathcal{P}_\kappa(A)$ to perfect κ-string $s_x \in S_{PF}$. Then, y becomes imperfect κ-string $s_y \in S_{IF}$ and X becomes the corresponding unfair κ-string s_X (from disjoint S_{UF}), which is not a club but contains other κ-fair strings (those κ-fair strings would correspond to closed and unbounded).

Finally, let us revisit the main case when $|A| > \delta$. It turns out that we have chain of ultrafilter extensions (as a consequence of the above powerset inclusion chain (4) for corresponding replicata): G is extended by D and D is extended by U. We have U as a non-trivial λ-complete ultrafilter on

[272] Also see *Corollary* **??**

[273] See *Definition 90*

$R_\lambda^{<\lambda^+}$. The similar correspondence between the equivalence classes of SEFs for each replicata and club filter F built with $\hat{x}$ holds also for R_λ^λ and $\mathcal{P}_{\lambda^+}(\lambda)$. Now one can invoke DC_λ and select $\mathcal{P}_\kappa(\lambda)$ as a subset of $\mathcal{P}_{\lambda^+}(\lambda)$. Then, as long as one considers SEF κ-strings mapped from $\forall x : x \in \mathcal{P}_\kappa(\lambda)$, one can get $D = \{s_Y : s_Y \in U \wedge |s_Y| < \kappa \wedge s_Y \supset s_X\}$, where s_X is mapped from $X \in F$. By *Definition 104*, D is a fine measure on $\mathcal{P}_\kappa(A)$. □

Corollary 119 - Strongly compact cardinal.

D is a fine measure on $\mathcal{P}_\kappa(A)$ for all $A : |A| \geq \kappa$. Consequently, $N^* \models$ ∃ strongly compact cardinal.

Proof. In the context of N^*, result follows from *Theorem 118* when applied to the construct of α-tranches as in *Lemma 104*. □

We would need a small technical lemma for the upcoming proof.

Lemma 120 - Inverse of perfectly λ-fair.

Let s_x and s_y be two perfectly λ-fair strings. Then, $s_x \wedge s_y =$ "(0)" iff $s_x = \neg s_y$.

Proof. Follows from *Definition 63*, as any s_x and s_y (by being perfectly λ-fair strings)must be closed under non-empty intersections as almost disjoint(adjoint) unless one is the inverse of the other. □

Theorem 121 - Normal measure on replicata.

Let N^* be a model of $ZF + DC_\lambda + I$. If $N^* \models \exists$ a replicata $\Lambda(\lambda)$, then $\Lambda(\lambda)$ carries a normal measure D.

Proof. Let $N^* \models \Lambda(\lambda)$, s.t. $\Lambda(\lambda) = (R_\lambda^\lambda, P_\varepsilon, \pi)$ for some smallest limit cardinal λ (which is large enough to contain $\Lambda(\lambda)$). We have a regular uncountable cardinal $\kappa < \lambda$, s.t. D is a nontrivial κ-complete ultrafilter on $\mathcal{P}_\kappa(\lambda)$ and fine measure as in *Theorem 118*.

Use the one-to-one correspondence between the equivalence classes of SEFs for each replicata and sets in $\mathcal{P}(\lambda)$ to define a bijective map $e : R_\lambda^\lambda \to \mathcal{P}(\lambda)$ (since $|R_\lambda^\lambda| = |\mathcal{P}(\lambda)|$). Also, one can invoke DC_λ and select $\mathcal{P}_\kappa(\lambda) \subset \mathcal{P}(\lambda)$. Let $E = e^{-1}(\mathcal{P}_\kappa(\lambda))$ be a subset of equivalence classes of SEFs, i.e. $E \subset R_\lambda^\lambda$

Again, recall that substring relation ε is just a special kind of subset $\subset$ (see *Definition 90*). Let $U \supseteq D$ be the measure on $\mathcal{P}(\lambda)$ constructed as in *Lemma 113*. We want show that normality (as specified in *Definition 106 - Normal measure*) on U using the replicata structure, so that normality of D follows.

For convenience, to match the normality notation in definition, let $A = \lambda$. Also let B be some subset of measure-one sets in $\mathcal{P}_\kappa(A)$. We define a family of functions $F = \{f_\alpha : \mathcal{P}_\kappa(A) \to A\}$, where $\alpha \in I(F)$, and require that for each such function we have $f_\alpha(x) \in x$. Furthermore, by bijectivity of e we have a proxy family $G = \{g_\alpha : g_\alpha(e(x)) = e(f_\alpha(x)) \wedge \alpha \in I(F) \wedge \forall x \in B\}$. Essentially, G contains κ-strings each being a substring of λ-string.

□

Corollary 122 - Supercompact cardinal.

D is a normal measure on $\mathcal{P}_\kappa(A)$ for all $A : |A| \geq \kappa$. Consequently, $N^* \models$ $\exists$ supercompact cardinal.

8.9 N^* implies MM^{++}

Recall the definition for MM^{++} from [29].

Definition 108 - Martin's Maximum $^{++}$ (MM^{++}).

Given a forcing P that preserves stationary subsets of ω_1, a collection $\{D_i : i < \omega_1\}$ of dense subsets of P, and a collection $\{\tau_i : i < \omega_1\}$ of P-names for stationary subsets of ω_1, then there exists a filter $g \subset P$ such that for every $i < \omega_1$:

(i) $g \cap D_i \neq \emptyset$

(ii) $(\tau_i)_g = \{\xi < \omega_1 : \exists p \in g \text{ such that } p \Vdash_P \xi \in \tau_i\}$ is stationary.

We can now state one of our main results that we intend to show more formally as a theorem.

Theorem 123 - MM^{++} and Supercompactness.

Let N^* be a sufficiently large model of $ZF + DC_\lambda + I$ such that it satisfies existence of a proper class of replicata Λ. Then, also the following holds in N^*:

(i) $N^* \models \exists$ supercompact cardinal

(ii) $N^* \models \mathsf{MM}^{++}$

Proof. The construction of N^* leads naturally to a key result regarding the large cardinal hierarchy and its consequences for forcing axioms. Specifically, we observe that the existence of a supercompact cardinal within N^* (see *Corollary 122*) suffices to establish MM^{++}, as originally formulated in [29].

□

The supercompactness property within N^* alone is sufficient to establish MM^{++}. Thus, part *(ii)* directly implies *(i)*, rendering additional intermediate arguments redundant. This aligns with the established literature on large cardinal implications for forcing axioms (see [29]).

Although the above result superseded our prior approach, which involved demonstrating that an intermediate model N (of $ZF+DC$) satisfies MM^{++}, it remains of interest to revisit the original approach. The main motivation for such follow up will come from the fact N^* requires much stronger additional axiomatic assumptions $DC_\lambda + I$ rather than DC. Even though it might be impossible to connect the world of large cardinals with forcing axioms without I axiom.

9 In search of String Theory for SEF

Most of this section is still very much WORK IN PROGRESS, so any results are to be tentatively perceived as drafts only as they are provided without proof. For most purposes it is safe for the reader to ignore this section completely for now.

9.1 Meta-level of FOL in Model Theory

Next, we will give a more formal definition for the theory around SEF[274] in terms of *first order logic* (FOL). We will aim to show that $ZF + SEF$ is *consistent relative to* ZF[275]. In order to make arguments about models for the SEF theory we will employ some notations and definitions from [47].

In the discourse of *model theory* we rely on FOL language simply referenced as $\boldsymbol{\mathcal{L}}$-language[276], which is defined together with $\boldsymbol{\mathcal{L}}$-terms, $\boldsymbol{\mathcal{L}}$-formulas, $\boldsymbol{\mathcal{L}}$-theories and most importantly models for logical interpretion. Next to being FOL language on its own, model theory concerns itself with other languages of non-logical nature that may accompany various mathematical objects of interest (also called domains or universes) and try to capture all structural information that can be reflected with functions, relations and constants. Notation for such languages is simply $\boldsymbol{\mathcal{L}}$ or sometimes $\boldsymbol{\mathcal{L}}_x$, where x is a name of a particular FOL theory in mind. Given that the language of set theory is also FOL, model theory can natively operate with it.

Pure formal systems with FOL have a number of limitations. This comes from inability to pass functions as values into formula variables. That's why this is called first-order rather than higher-order logic. Ultimately it is impossible to properly work with infinity in most classical FOL systems[277]. For model theory that means that all the quantifier are restricted to the domain of the model.

Recall that a language $\boldsymbol{\mathcal{L}}$ is a set of non-logical symbols[278] such that each symbol $c_n \in \boldsymbol{\mathcal{L}}$ (endowed with some finite arity $n \in \mathbb{N}$) is either a *function* or a *relation*. Other symbols can be just *constants*. $\boldsymbol{\mathcal{L}}$ is usually written in displayed form as the following definition.

Definition 109.

---

[274] that we used to proof the $\neg CH$ cases

[275] If T is some theory (consisting of axioms, e.g. like ZF) and A is another axiom, then we say that $T + A$ is consistent relative to T (or that A is consistent with T assuming that T itself is consistent) - see p. 163 of [6] for *relative consistency*

[276] here we primarily follow notations and main definitions from [47]

[277] That's why set theory has a separate language, although most of the statements are specified in FOL

[278] from *Models of Set Theory* in [6] as well as [47, 48]

Let $\boldsymbol{\mathcal{L}}$ be a *signature* or a *language*[279] given by:

1. a set of function symbols $\boldsymbol{\mathcal{F}}$ and positive integers n_f for each $f \in \boldsymbol{\mathcal{F}}$;
2. a set of relation symbols $\boldsymbol{\mathcal{R}}$ and positive integers n_R for each $R \in \boldsymbol{\mathcal{R}}$;
3. a set of constant symbols $\boldsymbol{\mathcal{C}}$.

Such languages can be often naturally *satisfied* by their respective models[280]. A model of $\boldsymbol{\mathcal{L}}$ is a pair $\boldsymbol{\mathcal{A}} = (A, I)$, where A is the universe of $\boldsymbol{\mathcal{A}}$ and I is the interpretation function which maps the symbols of $\boldsymbol{\mathcal{L}}$ to appropriate relations, functions, and constants in A. Non-logical symbols of the language might be different up to interpretation between concrete models. Sometimes when we consider a structure $\boldsymbol{\mathcal{A}} = (A, I)$ as a model for the set of non-logical symbols of the language $\boldsymbol{\mathcal{L}} = (\boldsymbol{\mathcal{F}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{C}})$, we might write the structure by explicitly indicating the interpretation mapping for occurrences of each non-logical symbol of $\boldsymbol{\mathcal{L}}$ in the structure $\boldsymbol{\mathcal{A}}$ with additional super-script index as $(A, \boldsymbol{\mathcal{F}}^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{R}}^{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{A}}})$[281].

For example, the signature of the set theory like ZF is just $\{\in\}$[282]. Recall that in set theory language (for theories like ZF) we often use many more non-logical symbols such as

$$=, \emptyset, \omega, \omega_1, \aleph_0, \subset, \mathcal{P}, \cup$$

which are merely definable elements (formulas encoded as sets like classes), relations, functions and constants. Since those symbols are not central for the main discourse in set theory, they can be seen merely as conservative extensions.

In $\boldsymbol{\mathcal{L}}$-language we will also use logical *connectivity* symbols like $\neg, \wedge, \exists$ together with $\vee, \forall$[283] to show syntactical truth $\top$ (or falsehood $\bot$)[284] of formulas under consideration. And, we will use square brackets for FOL $\boldsymbol{\mathcal{L}}$-formulas to avoid potential confusion or overlap with replicator brackets[285].

In general[286], given:

(1) some language $\boldsymbol{\mathcal{L}}$ - a set of non-logical symbols

[279] here signature[48] or language[47] have the same meaning modulo literature on model theory

[280] also see - "Tarski's definition of truth" in [6]

[281] unless interpretation is already clear from the discourse

[282] or, just the relation $\{\in\}$, where $=$ is meerly a constant

[283] $\vee, \forall$ can be implemented as shorthand for formulas using only $\neg, \wedge, \exists$[47]

[284] both $\top, \bot$ are constant terms for FOL

[285] at this point, we consider only basic alphabet for SEFs, without "[1+]" extension and so on

[286] and by following *Models of Set Theory* in [6] and *Definable Sets and Interpretability* in [47]

(2) with some theory $\boldsymbol{\mathcal{T}}$ - a sequence of $\boldsymbol{\mathcal{L}}$-formulas (in FOL model theory language or $\boldsymbol{\mathcal{L}}$-language),

(3) some model $\boldsymbol{\mathcal{M}}$ of $\boldsymbol{\mathcal{L}}$ - an $\boldsymbol{\mathcal{L}}$-structure for this language,

(4) and some $\boldsymbol{\mathcal{L}}$-formulas ϕ and φ,

we assume that:

(a) proofs are finite;

(b) (syntactic provability $\vdash$) when we mean that some theory $\boldsymbol{\mathcal{T}}$ *proves* a FOL formula ϕ, we write $\boldsymbol{\mathcal{T}} \vdash \phi$, or even shorter $\vdash_{\boldsymbol{\mathcal{T}}} \phi$;

(c) (semantic entailment $\models$) when we mean that some theory $\boldsymbol{\mathcal{T}}$ *satisfies* a FOL formula ϕ, we write $\boldsymbol{\mathcal{T}} \models \phi$, or even shorter $\models_{\boldsymbol{\mathcal{T}}} \phi$;

(d) when we mean that some model $\boldsymbol{\mathcal{M}}$ *satisfies* $\boldsymbol{\mathcal{T}}$, we also mean that $\forall\varphi.[[\varphi \in \boldsymbol{\mathcal{T}}] \wedge [\boldsymbol{\mathcal{M}} \models \varphi]]$ or, we sometimes write $\varphi \models_{\boldsymbol{\mathcal{M}}} \top$, or even shorter $\models_{\boldsymbol{\mathcal{M}}} \varphi$;

(e) to show that $\boldsymbol{\mathcal{T}} \models \phi$, we give an informal mathematical proof that $\boldsymbol{\mathcal{M}} \models \phi$ whenever $\boldsymbol{\mathcal{M}} \models \boldsymbol{\mathcal{T}}$;

(f) to show that $\boldsymbol{\mathcal{T}} \not\models \phi$, we usually construct a counterexample.

(g) (soundness) $\boldsymbol{\mathcal{T}} \vdash \phi \implies \boldsymbol{\mathcal{T}} \models \phi$;

(h) by Goedel's Completeness Theorem[287], if $\boldsymbol{\mathcal{T}}$ is an $\boldsymbol{\mathcal{L}}$-theory and ϕ is an $\boldsymbol{\mathcal{L}}$-sentence, then $\boldsymbol{\mathcal{T}} \models \phi \iff \boldsymbol{\mathcal{T}} \vdash \phi$;

(i) (corollary) every consistent set of sentences has a model - namely, $\boldsymbol{\mathcal{T}}$ is consistent iff $\boldsymbol{\mathcal{T}}$ is satisfiable;

(j) (compactness) - $\boldsymbol{\mathcal{T}}$ is satisfiable if and only if every finite subset of $\boldsymbol{\mathcal{T}}$ is satisfiable.

To sum up, model theory is a FOL meta language that can interoperate between other "languages" treating them and their objects as non-logical symbols on the meta-level. This helps to understand relations between those objects by translating respective theories into FOL to argue about object properties, etc. As such approach suites our needs well, we will do the same.

To better understand how we are going to work with *model theory*, it is useful to go (again) through some of its key concepts and results.

Absoluteness. Here and later in the paper our plan is to work with multiple theories on a meta-level of model theory. *Absoluteness of quantifier-free formulas in FOL* is an important tool to study which properties are absolute in different models:

[287] In other words, syntactic notion of provability $\vdash$ in first-order logic is equivalent to the semantic notation of logical entailment $\models$

Definition 110.

We define and use *absoluteness* in following sense:

(a) an $\boldsymbol{\mathcal{L}}$-formula ϕ is said to be *absolute* to some class of models (structures), if it has the same truth value in each of the members of that class;

(b) (example) quantifier-free $\boldsymbol{\mathcal{L}}$-formulas are absolute;

(c) if the truth of a formula in each substructure N of a structure M follows from its truth in M, the formula is *downward absolute*;

(d) if the truth of a formula in a structure N implies its truth in each structure M extending N, the formula is *upward absolute*;

(e) (example of downward absoluteness) universal sentences [288] that are true in a structure are also true in every substructure of the original structure;

(f) (example of upward absoluteness) existential sentences are upward absolute from a structure to any structure containing it;

(g) (in set theory) one usually begins with the fixed model of set theory and only considers other transitive models containing the same ordinals as the fixed model.

Firstly, we rely on the fact that (in set theory) $\boldsymbol{\Delta}_0$[289] formulas of FOL are *absolute* for all transitive models. On the meta-level of the model theory and its $\boldsymbol{\mathcal{L}}$-language of the FOL such formulas are called quantifier-free formulas (see *Proposition 1.1.8* in [47]).

Boundness. Secondly and at risk of repeating ourselves, model theory $\boldsymbol{\mathcal{L}}$-language [47] is significantly limited by the capabilities of FOL (contrary to the higher order logic). For example, statements like "every bounded subset has a least upper bound" cannot be expressed as a FOL formula because we cannot quantify over subsets. Another consequence is coming from the direct limitation of the model theory itself, namely - quantification is always restricted to the elements of the model structure (i.e. the actual universe of the model). Basically, if we discuss a model $\boldsymbol{\mathcal{M}}$ with some universe M and use a quantifier $\forall x.[P(x)]$ we actually always mean $\forall x \in M.[P(x)]$.

[288] with only universal quantifiers

[289] Here and everywhere else in the paper we use bold Greek symbols $\boldsymbol{\Sigma}$, $\boldsymbol{\Pi}$ and $\boldsymbol{\Delta}$ to reference parts of the *Lévy hierarchy* of formulas in the formal language of the ZF set theory

Definition 111.

Absoluteness depends on boundness (or "freeness") of variables in $\mathcal{L}$-formulas:

(a) we say that a variable ν is to occur freely in an $\mathcal{L}$-formula ϕ if it is not inside $\exists\nu$ or $\forall\nu$ quantifier; otherwise, we say that it is *bound*;

(b) (example) quantifier-free $\mathcal{L}$-formulas have no bound variables;

(c) an $\mathcal{L}$-sentence ϕ is an $\mathcal{L}$-formula with no free variables;

(d) if a theory (or a fragment) $\mathcal{T}$ consists only of quantifier-free $\mathcal{L}$-formulas, it is called *model-free*; otherwise, we say that $\mathcal{T}$ is *bound*;

(e) (example) model-free $\mathcal{L}$-theories are absolute;

(f) bound theory can have semantic interpretation (entailment) only after being bound to a model.

Axiomatization. By the very definition of semantic entailment, no theory interpretation is possible without an existing model. This means that theory can contain axioms which can not be proven syntactically, since they are stated in a semantic fashion with propositions and objects which are true or false only depending on the properties of those objects (or depending on what they "mean" in a particular model). Ability to describe object properties in form of FOL statements ($\mathcal{L}$-formulas) is called to *axiomatize*. In model theory objects are studied as part of domains in $\mathcal{L}$-structures. We can also simply say that $\mathcal{L}$-structures are axiomatized by $\mathcal{L}$-theories (or axioms in those theories). Note that this is different from *axiomatizable* which is a reductionism applicable to the theory itself (and not the objects).

Definition 112.

A collection of objects that can have all their properties described (axiomatized) as FOL statements ($\mathcal{L}$-formulas) is called *axiomatization*. Or more precisely:

(a) *axiomatized* in a language $\mathcal{L}$ means described uniquely up to isomorphism by a single $\mathcal{L}$-sentence;

(b) (example) a collection of objects and all their properties can be *axiomatized* if there is a one-to-one correspondence between those properties and FOL statements ($\mathcal{L}$-formulas);

(c) (example) multiple FOL statements can be uniquely represented by a single conjugation (CNF), hence finite number of properties can be always axiomatized;

(d) (example) any finite collection of finite structures can always be axiomatized in FOL;

(e) (example) some, but not all, infinite collections of finite structures can also be axiomatized by a single $\mathcal{L}$-sentence;

(f) An $\mathcal{L}$-theory T is said to be *finitely axiomatizable* if there is a finite set of $\mathcal{L}$-sentences Σ such that $\Sigma \vdash T$;

(g) *recursively axiomatizable theory*

Initially, when we wanted to discuss a new theory and its axioms, we did not want to come up with entirely new formulation for everything. If that can be possible we wanted to reuse well-working concepts like classes from set theory language. In our first attempt to write this, when we started talking about *κ-String Theory* (which is coming in the next subsection and will be later extended by *SEF Theory*), we simply wanted to mention that some classes or formulas (like cardinals) and axioms (like union axiom) were "borrowed" from ZF. "Borrowed" formulas means that we have initially translated some ZF formulas using FOL of $\mathcal{L}$-language [47] "implicitly" to describe SEF theory.

However and in general, many such formulas may not necessarily be absolute. For example, many cardinality related statements are not absolute. Specifically, $Y = \mathcal{P}(X)$, $|Y| = |X|$, *α is a cardinal*, etc. It means that we also rely on the fact that given a transitive model M statements like "*α is a cardinal*" can be interpreted for specific quantifier-free formulas[290] in such model M[291]. Namely, If $\alpha \in M$ and if *alpha* is a cardinal, then $M \models$ *α is a cardinal* and so on.

Now, to clarify this notion of "borrowing" formulas - we can do better. Let us define a notion of *primordial theory*. Such theory typically consist mainly of convenient definitions, abbreviations or other aliases and constants. In that sense, it is almost consistent or naturally conservative as it does not try to proof anything about itself yet. Although we don't always know whether such primordial theory is consistent *a priori* (before it has been extended by a normal theory and interpreted in a model), we certainly know that it is never complete by definition.

Definition 113 - Primordial theory.

Primordial theory is a sequence of $\mathcal{L}$-formulas Υ, which are either absolute or at most conservative. Meaning that Υ consists of two fragments:

(a) *absolute* - a sub-sequence of $\mathcal{L}$-formulas A each is quantifier-free or otherwise absolute;

[290] or $\mathbf{\Delta}_0$ formulas in *Lévy hierarchy*

[291] see exercise *12.6* in [6]

(b) *conservative* - a sub-sequence of $\boldsymbol{\mathcal{L}}$-formulas Θ, s.t. if there is an $\boldsymbol{\mathcal{L}}$-theory T_1 with a conservative extension $T_2 \supset T_1$, then T_2 is Θ-conservative over T_1.

According to the above definition any primordial theory has a requirement to remain absolute after satisfiable extension, which includes binding to a model for semantic interpretation. Note that before such binding to a model no semantic interpretation is possible (see *bound theory* above). To clarify "otherwise absolute", by this we mean that every primordial theory can be accompanied by a model placeholder - a collection of items (but not really yet fully a set or even a class). We assume that upon binding with the model, each formula of a primordial theory with universal quantifiers (over "former" collection) can be either rewritten to iterate over the collection of formulas by replacing them with conjugation of consistent formulas or shown to be absolute in the model by other means of quantifier-elimination specific to the model.

A few examples of a primordial theory would be:

- **Trivial cases**:
  - Every empty theory $\boldsymbol{\mathcal{T}}_0$ is trivially primordial.
  - Every theory that consists only of absolute $\boldsymbol{\mathcal{L}}$-formulas is primordial.
  - Every theory that consists only of definitions is primordial.
- **Finite collection of quantifier-free $\boldsymbol{\mathcal{L}}$-formulas**: take a collection of finitely many quantifier-free $\boldsymbol{\mathcal{L}}$-formulas ϕ in $\boldsymbol{\mathcal{L}}$-language which will form a primordial theory $\boldsymbol{\mathcal{T}}_0$ iff $\forall\phi \in \boldsymbol{\mathcal{T}}_0.\nexists\varphi \in \boldsymbol{\mathcal{T}}_0.[\varphi = [\boldsymbol{\mathcal{T}}_0 \vdash \phi \wedge \boldsymbol{\mathcal{T}}_0 \vdash \neg\phi]]$[292]. Observe that the same can be translated by rewriting such collection "$\forall\phi \in \boldsymbol{\mathcal{T}}_0$" as a single formula which would be a (long) finite conjunction of consistent quantifier-free formulas (also called conjunctive normal form or CNF). Now, if any of ϕ has a universal quantifier, it again can iterate over a collection placeholder before binding, and only then to be rewritten into CNF after satisfiable extension of $\boldsymbol{\mathcal{T}}_0$. Although existence of countably long CNF may be treated with much more meticulous adherence to technicalities (even on the meta-level), for our concerns it is enough to say that when infinite constructs are discussed - we always really on the semantic interpretation of the respective infinity axioms inside the model;
- **Definition of a class in set theory**. Semantically (on a meta-level of the $\boldsymbol{\mathcal{L}}$-language), classes can be described as equivalence classes of

[292] in "pseudo set theoretical language"

logical formulas[49]: If $\mathcal{A}$ is a $\boldsymbol{\mathcal{L}}$-structure interpreting ZF[293], then the object language "class-builder expression" $\{x \mid \phi\}$ is interpreted in $\mathcal{A}$ by the collection of all the elements from the domain of $\mathcal{A}$ on which $\lambda x\phi$ holds; thus, the class can be described as the set of all predicates equivalent to ϕ (which includes ϕ itself). In particular, one can identify the "class of all sets" with the set of all predicates equivalent to $x = x$.

- **Quantifier elimination in Presburger arithmetic**[294] implies decidability of the theory. This means that such theory must have some initial primordial subset of definitions and other statements minus actual results. Results will include $\boldsymbol{\mathcal{L}}$-sentences (theorems and proofs) about the properties of the theory itself such as consistency, completeness and decidability.

Below we use such concepts such as *Boolean algebra*, *Lindenbaum algebra* and *realization of FOL* (Propositional logic), and later on (principal) *ideal* and *filter*, *prime ideal* and *ultrafilter*[295]. Let us also refine what we mean when we say that an $\boldsymbol{\mathcal{L}}$-formula is *satisfied by a model* (and so on) by using *realization* map. We say that on a meta-level our $\boldsymbol{\mathcal{L}}$-language can itself be interpreted (not only by a two-values Boolean algebra but) in any other Boolean algebra model $\boldsymbol{\mathcal{B}}_A := (B, \vee, \wedge, \neg, \bot, \top)$.

Lemma 124 - FOL generic realization.

Assume that:

- $\boldsymbol{\mathcal{L}}$-language is reduced to a FOL signature $(\vee, \wedge, \neg)$;
- P is a collection of all propositional variables of all finite subsets of $\boldsymbol{\mathcal{L}}$-formulas of $\boldsymbol{\mathcal{L}}$-language;
- $\boldsymbol{\mathcal{A}} = (\{\bot, \top\}, \vee, \wedge, \neg, \bot, \top)$ is a two-values algebra model;
- $f : P \to \{\bot, \top\}$ be a FOL *realization* map, where $\{\bot, \top\}$ is a domain of $\boldsymbol{\mathcal{A}}$;
- $\boldsymbol{\mathcal{B}} := (B, \vee, \wedge, \neg, \bot, \top)$ is any model with the Boolean algebra signature.

[293] Note that depending on formulation pure axioms of the ZF set theory do not formalize the notion of classes, so each formula with classes must be reduced syntactically to a formula without classes in formal theorem provers. For example, one can reduce the formula $A = \{x \mid x = x\}$ to $\forall x(x \in A \leftrightarrow x = x)$

[294] This theory is often applied in automatic theorem provers. The signature of Presburger arithmetic contains only the addition operation and equality (without the multiplication) and the axioms include a schema of induction[50]

[295] for more detailed discussion please see p.117-122 in [26]

Then, for any $\boldsymbol{\mathcal{L}}$-formula φ we have that $\boldsymbol{\mathcal{A}} \models \varphi \implies \boldsymbol{\mathcal{B}} \models \varphi$ iff there exists an isomorphism $g : B \to P$ preserving $\vee, \wedge, \neg$, so that $\boldsymbol{\mathcal{L}}$-language can be also *realized* by a composition $r := f \circ g$.

Proof. Recall, by the definition of formula realization[26] the map $f : P \to \{\bot, \top\}$ realizes every $\boldsymbol{\mathcal{L}}$-formula ϕ and φ iff f can be extended over complexity of $\boldsymbol{\mathcal{L}}$-formulas in a following inductive way:

(i) $f(\neg\varphi) = \neg f(\varphi)$;

(ii) $f(\phi \wedge \varphi) = f(\phi) \wedge f(\varphi)$;

(iii) $f(\phi \vee \varphi) = f(\phi) \vee f(\varphi)$

Furthermore, if any formula φ in $\boldsymbol{\mathcal{L}}$-language is mapped by $f : \varphi \mapsto \top$, then we say f *satisfies* φ. We can trivially check this for $\{\bot, \top\}$: if $\varphi = \bot$, then $\nvdash f(\bot) \to f \not\models \varphi$; if $\varphi = \top$, then $\vdash f(\top) \to f \models \varphi$.

We want to show that the same is true for $r := f \circ g$. Indeed, given that g is $\vee, \wedge, \neg$ preserving, it means that for every $\boldsymbol{\mathcal{L}}$-formula ϕ and φ $\exists u, v \in B$:

a) $r(\neg u) = f(\neg\varphi) = \neg f(\varphi) = \neg r(u)$;

b) $r(u \wedge v) = f(\phi \wedge \varphi) = f(\phi) \wedge f(\varphi) = r(u) \wedge r(v)$;

c) $r(u \vee v) = f(\phi \vee \varphi) = f(\phi) \vee f(\varphi) = r(u) \vee r(v)$

If r is extended over complexity of $\boldsymbol{\mathcal{L}}$-formulas by induction, then we get $\boldsymbol{\mathcal{A}} \models \phi \Leftrightarrow g(u) = \phi \wedge g^{-1}(\phi) = \top \Leftrightarrow \boldsymbol{\mathcal{B}} \models \phi$. □

We call algebra model to be abstract until the domain of the model has been fixed[296]. Otherwise, Boolean algebra is called concrete (as opposite to abstract).

Boolean algebra can be defined from a boolean ring $(B_r, \oplus, \cdot, \mathbf{0}, \mathbf{1})$ as:

1. $\bot := \mathbf{0}$
2. $\top := \mathbf{1}$
3. $x \wedge y := x \cdot y$
4. $x \vee y := x \oplus y \oplus x \cdot y$
5. $\neg x := \mathbf{1} - x$

In general, the categories of Boolean rings and Boolean algebras are equivalent. It means that there is a categorical isomorphism between boolean rings and Boolean algebras. For example, if a boolean ring is implemented by binary strings then there is a corresponding concrete Boolean algebra.

[296] chosen to be a specific object

Definition 114 - Bootstrap-Theory for FOL realisation.

Let $\boldsymbol{\mathcal{L}}$-structure $\boldsymbol{\mathcal{B}} := (B, \vee, \wedge, \neg, \bot, \top)$ be an abstract Boolean algebra with realization r of the $\boldsymbol{\mathcal{L}}$-language[297]. Also let ϕ and φ be $\boldsymbol{\mathcal{L}}$-formulas.

Then $\boldsymbol{\mathcal{T}_0}$ be a sequence of $\boldsymbol{\mathcal{L}}$-formulas corresponding to the following convenient definitions:

1. boolean constants of truth and falsehood:
   1.1. $\top := \varphi \vee \neg\varphi$;
   1.2. $\bot := \varphi \wedge \neg\varphi$;
2. basic operations:
   2.1. Take $\vee, \wedge, \neg$ part of the $\boldsymbol{\mathcal{B}}$ signature and define those operations on any subset of finite formulas recursively for arity $n > 2$, otherwise via truth table;
3. secondary operations:
   3.1. (material conditional) $\phi \to \varphi := \neg\phi \vee \varphi$
   3.2. (material biconditional) $\phi \to \varphi := \neg\phi \vee \varphi$
   3.3. (exclusive or) $\phi \oplus \varphi := \neg[\phi = \varphi]$
   3.4. (logical equivalence) $\phi = \varphi := [\phi \leftrightarrow \varphi]$;
4. restricted logic:
   4.1. $\neg$ and $\wedge$ are sufficient, since $\phi \vee \varphi \leftrightarrow \neg[\neg\phi \wedge \neg\varphi]$;
   4.2. (quantifier) can be restricted to use of only $\exists$, since $\forall x\varphi(x) \leftrightarrow \neg\exists x\neg\varphi(x)$;
   4.3. constant symbols and function symbols can be entirely rewritten using parentheses and predicates.

Definition 115 - Bootstrap-Theory for Classes of objects.

Let $\boldsymbol{\mathcal{T}_0}$ be a sequence of $\boldsymbol{\mathcal{L}}$-formulas that define classes and operations on them following [6]:

1. (*object placeholder* or simply an *object*) is a unique constant in any $\boldsymbol{\mathcal{L}}$-formula, which can be replaced (concretized or implemented) upon this bootstrap-theory extension, for example, by replacing it with *set* or *string* definition - depending on the context;

[297] see *Definition 124*

2. (*class*) If $p_1, .., p_n$ is a $\mathcal{L}$-formula, we call $C = \{x : \varphi(x, p_1, .., p_n)\}$ a class and use curly brackets (braces) as containment notation;

3. (*membership* relation)

   members of the class C are those objects x that satisfy $\varphi(x, p_1, .., p_n)$:

   $$x \in C \leftrightarrow \varphi(x, p_1, .., p_n)$$

4. (*definable*)

   4.1. we say that C is *definable from* $p_1, .., p_n$;

   4.2. if $\varphi(x)$ has no parameters (context or bounded variables) p_i then the class C if *definable*;

5. (*equality*) two classes are considered equal if they have the same elements:

   if $C = \{x : \varphi(x, p_1, .., p_n)\}$ and $B = \{x : \psi(x, q_1, .., q_m)\}$,
   then $C = D \Leftrightarrow \varphi(x, p_1, .., p_n) \leftrightarrow \psi(x, q_1, .., q_m)$

6. (*universal class* or *universe*) the class of all objects: $V = x : x = x$

7. (*class inclusion*) C is a subclass of D: $C \subset D$ iff $\forall x.[x \in C] \rightarrow [x \in D]$

8. basic operations on classes:

   8.1. $C \cap D = \{x : x \in C \wedge x \in D\}$

   8.2. $C \cup D = \{x : x \in C \vee x \in D\}$

   8.3. $C - D = \{x : x \in C \wedge x \notin D\}$

   8.4. $\bigcup C = \{x : x \in S \wedge S \in C\} = \bigcup\{S : S \in C\}$

9. Every object can be considered a class;

10. If S is a class, then the formula $\{x : x \in S\}$ is also a class;

11. A class that is not an object is proper class[298].

Lemma 125.

The following are primordial theories:

(a) Bootstrap-theory for FOL realization (*Definition - 114*)

(b) Bootstrap-theory for Classes of objects (*Definition - 115*)

Proof. Follows from definition of primordial theory, since both (a) and (b) contain only convenient definitions. □

[298] such as *a class of all objects*

9.2 String Theory Model

Let $\mathcal{L}_{SEF} = (\Sigma, \Delta)$ be an infinite String Enumeration Formula (SEF) language with finite alphabet $\Sigma := \{0, 1, (,)\}$[299]. Now we want to endow it with $\boldsymbol{\mathcal{L}}$-structure, so that we can redefine previously discussed statements as $\boldsymbol{\mathcal{L}}$-theory[47] for SEF using FOL. Again, we have to start with the theory behind infinite binary strings and its generalizations such as regular languages in computer science.

Consider $\boldsymbol{\mathcal{B}} = (\mathbb{B}_\kappa, \cdot)$ as an $\boldsymbol{\mathcal{L}}$*-structure* consisting of a class of binary strings $\mathbb{B}_\kappa = \{b : b \in \{0,1\}^\kappa, |b| = \kappa\}$ of infinite length κ equipped with binary string concatenation function $f(x, y) = x \cdot y = \text{“}xy\text{”}_\kappa, \forall x, y \in \mathbb{B}_\kappa$. Observe that $\boldsymbol{\mathcal{B}}$ is a concrete Boolean algebra[300]. It contains only infinite strings, since it is build around the concept of cardinality k ”borrowed” from ZF.

Indeed, the above construction can be generalized for regular languages over a finite alphabet Σ (not just the case of binary strings $\Sigma = \{0, 1\}$). We will do so in two axiomatization steps by defining:

i) an $\boldsymbol{\mathcal{L}}_{str}$-language $(\cdot, \Sigma)$ (a signature of non-logical symbols);

ii) a *basic string theory* $\boldsymbol{\mathcal{T}}_{str}$ (mostly inspired to match ZF);

iii) an $\boldsymbol{\mathcal{S}}_{str}$-structure satisfying $\boldsymbol{\mathcal{T}}_{str}$ (a model $\boldsymbol{\mathcal{S}} := (S_\kappa, \cdot, \Sigma)$ with κ-infinite strings).

The motivation for later is that such model will become useful when dealing with infinite binary strings and SEFs - concepts that must be familiar by now due to prior and somewhat tedious discussion through this paper.

Definition 116 - Basic String Theory (extended list of axioms).

Let $\boldsymbol{\mathcal{L}}_{str} := (\cdot, \Sigma)$ be a language of the *basic string theory*, where:

1. $\cdot$ - is a binary function of string concatenation operation;
2. Σ - is a constant of the alphabet symbols of a regular language, e.g. $\Sigma = \{0, 1\}$;

Then the theory of $\boldsymbol{\mathcal{L}}_{str}$ is a list of axioms $\boldsymbol{\mathcal{T}}_{str}$[301] extending primordial theory-bootstraps:

(a) Bootstrap-theory for FOL realization (*Definition - 114*)

(b) Bootstrap-theory for Classes of objects (*Definition - 115*)

[299] See *Definition 23*

[300] the signature can be extended to match $\vee, \wedge, \neg$ or equivalently $+, \cdot, -$ language

[301] here we simply provide an extended list of axioms under a strong assumption that a much smaller axiomatization is possible

Specifically, we choose *strings* as the name of the object[302] for classes in the above bootstrap.

Next, we define the notion of what a string is by the following axioms:

1. ***Axiom of string concatenation*** - if x and y are strings, then so is:

$$z = x \cdot y$$

2. ***Axiom of substring***

   2.1. consequently[303], we can also count how many times y can be found in x by using a shorthand $|x|_y$;

   2.2. In fact, if defining of "pure" string-theoretic version of the membership relation is required for strings, then the following statements are equivalent - if x, y, z, s are strings, then $\forall x, y, z, s$:

   - $[z = x \cdot y] \iff [[x \ \varepsilon \ z] \wedge [y \ \varepsilon \ z]]$.
   - $z = x \cdot y$ iff $|z|_x \neq 0$ and $|z|_y \neq 0$.
   - $z = x \cdot s \cdot y \implies s \ \varepsilon \ z$.
   - $|x|_y \neq 0 \iff y \ \varepsilon \ x$

   2.3. in the above formulas we have used "$y \ \varepsilon \ x$" which means "substring of" relation and is semantically different from the set theoretical "member of" relation. We will slightly abuse the set theoretical notation for most cases where it is clear from the context that:

   - if x and y are both strings, then we mean "substring of" and write $y \ \varepsilon \ x$;
   - otherwise[304], we mean "member of" and write $y \in x$;

   2.4. furthermore, even when we say a set of strings without context of ZF, we most certainly mean the class of strings that can be represented as a formula in $\boldsymbol{\mathcal{L}}_s$-language, and would be very similar to a set-theoretical notion of a class[305].

3. ***Axiom of empty string***

   3.1. there exists an empty string;

   3.2. we use the same notation as in set theory to mean $\emptyset := ""_0$. Specifically, $[z = x \cdot s \cdot y] \wedge [s = \emptyset] \implies [z = x \cdot y] \wedge [s \ \varepsilon \ z]$.

   3.3. the following are equivalent definitions:

   - $x = x \cdot y \implies y = \emptyset$

[302] to be used instead of the nominal placeholder
[303] using the axiom of string concatenation
[304] if either one of x and y is a set or both of them are sets
[305] also see *Bootstrap-Theory of Classes* and *Definition 113*

- $x \cdot y = y \implies x = \emptyset$

3.4. also $\emptyset \notin ""_0 \iff \emptyset \notin \emptyset$, but $|\emptyset|_\emptyset = 0$

4. ***Axiom of prefix*** - $z = x \cdot y$ iff x is a prefix of z. Sometimes notated as a prefix quotient $x \backslash z = y$.

5. ***Axiom of suffix*** - (inverse of prefix) $z = x \cdot y$ iff y is a suffix of z. Sometimes notated as a suffix quotient $z/y = x$.

6. ***Axiom of string splitting*** - by analogy with the subset concept of set theory, we can define a "dual" of string concatenation by splitting the string into substrings. We define string splitting by always using some symbol y and call it a weaker version of the axiom. If there exist $\exists y.[y \in \Upsilon \wedge \Upsilon \cap \Sigma = \emptyset]$, then we say that splitting is done using a separator symbol (which is not a part of the alphabet Σ). A stronger split version is when y can be also a string[306].

   6.1. (weaker split) further on we assume a weaker version of the axiom for the basic string theory:

      6.1.1. if $\tau = x \cdot y \cdot z$ and τ, x, z are strings, then we say that τ can be split into x and z by using symbol y, $y \in \Sigma$;

      6.1.2. if $T := \S_y(\tau)$ and τ is a string and $y \in \Sigma$, then T contains a class of all split substrings in τ by y, namely: $\forall x, y, z.[[x \cdot y \cdot z = \tau] \implies [x \in T] \wedge [z \in T]]$;

      6.1.3. we can define a shorthand that when splitter symbol y is not specified, then we assume a default separator (which is not a part of the alphabet Σ). For weak version of the axiom it would be any symbol that is not in Σ. For example, if $\Sigma = \{0, 1\}$, then default separator is an extra comma symbol ",";

      6.1.4. in fact, we can benefit from as many finitely many separators $s \in \Upsilon$ as we need, as long as $\Sigma \cap \Upsilon = \emptyset$;

      6.1.5. in general, it is possible that we can use some explicit $y \in \Sigma$ for splitting.

   6.2. (stronger split) which is provisioned only for comparison background:

      6.2.1. if $\tau = x \cdot y \cdot z$ and τ, x, y, z are strings, then we say that τ can be split into x and z by using y string as a separator;

      6.2.2. the default separator is an empty string $y = \emptyset$;

      6.2.3. if for $T := \S_y(\tau)$ we have $y = \emptyset$, then $T = \S_\emptyset(\tau)$ contains a class of all possible substrings in τ (split by empty string).

[306] it is not a part of these *basic string theory* axioms or $\boldsymbol{\mathcal{T}}_{str}$

7. ***Axiom of class representation***

   7.1. each class of substrings X can be uniquely represented by a comma separated string x (even if the representation is unordered and comes with duplicates);

   7.2. classes X and Y are equal, i.e. $X = Y$ iff each corresponding representation string x and y can be split into the same unique class of substrings $\S(x) \cup \S(y) = X = Y$[307];

   7.3. (Separation schema of classes) if P is a property (with parameter p), then for any class of strings X and p there exists another class $Y = u \in X : P(u, p)$ that contains all those strings $u \in X$ that have property P;

   7.4. (Union of classes) a union over a split of a comma separated string x returns a unique representation of a class (a comma separated string), so that each unique substring occurs only once $\bigcup \S(x) = \{z \in \S(x) : |x|_z = 1\}$.

8. ***Axiom of string infinity***

   8.1. there exists an inductive string - if x is a string and $s \in \Sigma$ is a symbol, then $z = x \cdot s$ or $z = s \cdot x$ is also a string;

   8.2. (corollary) there exists an infinite class of inductive strings called *inductive class*;

9. ***Axiom of power-string***

   9.1. there exists a power-string of a class[308]:

      9.1.1. let $\P(X)$ be a *power-string* operation on classes of strings as defined in *Definition 117*;

      9.1.2. for any class of strings X the exists a class $Y = \P(X)$, the class of all unique concatenations pairs $x \cdot y$ from X.

   9.2. more convenient definitions such as *class product* are possible:

      9.2.1. class product $X \times Y$ can be represented by class of strings $\{x \cdot s \cdot y : x \in X \wedge y \in Y\}$, where $s \in \Upsilon$ is a separator symbol;

      9.2.2. or, equivalently, $X \times Y = \{[x, y] : x \in X \wedge y \in Y\}$[309];

      9.2.3. also from *power-string* we have $X \times Y \subset \P(X \cup s \cup Y)$, where $s \in \Upsilon$ is a separator symbol;

      9.2.4. in general, $X_1 \times \cdots \times X_{n+1} = (X_1 \times \cdots \times X_{n+1}) \times X_{n+1}$, but also $X_1 \times \cdots \times X_n = \{[x_1, ..., x_n] : x_1 \in X_1 \wedge ... \wedge x_n \in X_n\}$[310].

[307] Note that class equality is not the same as string equality

[308] again, such class can be uniquely represented by a comma separated string

[309] we can invoke the separation schema to justify the notation, but it seems superfluous as the previous string definition is expressive enough

[310] together, with high-order split and power-string definitions this allows defining high-order string theory and transitive models, which however will not be covered in this paper

9.3. *relations*[311]:

9.3.1. Let $X^n = \underbrace{X \times \cdots \times X}_{\text{n times}}$ be a product class;

9.3.2. also let a string $x_1 \cdot s \cdot x_2 ... x_{n-1} \cdot s \cdot x_n$, where $s \in \Upsilon$ is a separator symbol, be an example of n-tuple;

9.3.3. then, *n-ary relation* R is class of n-tuples;

9.3.4. or, equivalently, R is a relation on X if $R \subset X^n$;

9.3.5. alternative notation is $R(x_1, ..., x_n)$ instead of $[x_1, ..., x_n] \in R$ or even $x_1 \cdot s \cdot x_2 ... x_{n-1} \cdot s \cdot x_n \in R$;

9.3.6. a binary relation is noted as xRy or $[x, y] \in R$ or $x \cdot s \cdot y$;

9.3.7. if R is binary relation, then the *domain* of R is a class:

$$dom(R) = \{u : \exists v.[u, v] \in R\}$$

9.3.8. and the *range*:

$$ran(R) = \{v : \exists u.[u, v] \in R\}$$

9.4. and *functions*:

9.4.1. let x, y, z be strings and X, Y are classes of strings;

9.4.2. a binary relation f is a function if $[x, y] \in f \wedge [x, z] \in f \rightarrow y = z$;

9.4.3. the unique y such that $[x, y] \in f$ is the value of f at x or in a more standard notation:

$$y = f(x)$$

9.4.4. or, equivalently, $f : x \mapsto y$ or $y = f_x$, etc;

9.4.5. f is a function *on* X if $dom(f) = X$ (if $dom(f) = X^n$, then f is n-ary function on X);

9.4.6. f is a function *from* X *to* Y:

$$f : X \rightarrow Y$$

if $dom(f) = X$ and $ran(f) \subset Y$;

9.4.7. class of strings of all functions from X to Y is Y^X: $Y^X \subset ¶(X \times Y)$;

9.4.8. if $Y = ran(f)$ then f is *onto*;

9.4.9. if $f(x) = f(y) \rightarrow x = y$ then f is *one-to-one*;

9.4.10. function is sometimes called *mapping* or *correspondence*;

9.4.11. the rest of set theory language is also applicable to classes of strings and can be adopted according to [6] including such definitions as:

[311] we try to be as close as possible to the established set theory language [6]

9.4.11.1. *n-ary operation*;
9.4.11.2. *restriction*;
9.4.11.3. *extension*;
9.4.11.4. *composition*;
9.4.11.5. *image*, *inverse image*;
9.4.11.6. *inverse function*;
9.4.11.7. *equivalence relation* (reflexive, symmetric, transitive);
9.4.11.8. *disjoint classes*, *partition of a class*, *equivalence class*, *quotient*;
9.4.11.9. *isomorphism*, *isomorphic*, *automorphism*;

10. ***Axiom of index class***

10.1. there exists an index class for any string or, equivalently, any symbol of any string can be accessed by its index[312];

10.2. any index class can be constructed and defined from logical formulas:

10.2.1. Use FOL definitions from the extended bootstrap-theory;
10.2.2. Define $S(i) := i + 1, i \in \mathbb{Z}$ to be a successor (increment) formula;
10.2.3. Define $D(i) := i - 1, i \in \mathbb{Z}$ to be a predecessor (decrement) formula;
10.2.4. Use binary prefix codes to encode $0 \in \mathbb{Z}$ as well as all ordinals such as $w_0, w_1, .. \in Ord$;
10.2.5. Extend those codes, starting with 0, by applying $S(i)$ and $D(i)$ formulas to obtain binary strings of every index for integers potentially extendable with any ordinals;
10.2.6. Now, a class of such indexes represented by an ordered sequence of unique comma separated strings of $\{0, 1\}$ is called an *index class*[313];

10.3. if x is a string, then there exists a one-to-one correspondence between elements of an index class and each symbol from the sequence of symbols in the string $\exists f_x, f_x : I \to \Sigma$ (sometimes we abuse notation and treat strings as "functions" $x : I \to \Sigma$ meaning a respective function f_x).

11. ***Axiom of string and class cardinality***

11.1. isomorphic strings

[312] This is not equivalent to the axiom of choice in ZF, which is applicable to classes, not symbol indexes

[313] Like this each index class is isomorphic to a linearly ordered sequence and corresponds to a unique ordinal

11.1.1. two strings x and y have the same cardinality $|x| = |y|$ iff each symbol of x can be put in a one-to-one correspondence to each symbol of y;

11.1.2. equivalently, x and y have the same cardinality $|x| = |y|$ iff they have the same index class (namely, for $x : I_x \to \Sigma_x$ and $y : I_y \to \Sigma_y$ we have $I_x = I_y$);

11.2. isomorphic classes

11.2.1. two classes $|X| = |Y|$ have the same cardinality iff there exists a one-to-one correspondence between respective classes meaning that there exists an isomorphic function $f : X \to Y$, where $X = \S(x)$ is a class resulting in splitting representation string x and, similarly, $Y = \S(y)$ results in splitting representation string y (note that the cardinality of the representation strings may not be equal $[|x| \neq |y|] \vee [|x| = |y|]$);

12. ***Axiom of string equality***
two strings x and y are equal $x = y$ iff both have the same index class I (same cardinality) and consist of the same alphabet Σ, so that every symbol matches at the same index $x(i) = y(i), \forall i \in I$;

13. ***Axiom of regularity of representation strings***

13.1. every nonempty class of strings has an $\in$-minimal element;

13.2. or, equivalently, if S is a class of strings, then $\forall S.[S \neq \emptyset \to [\exists x \in S.[S \cap x = \emptyset]]$;

13.3. (corollary) there exists no infinite class with representation string $x_0 \in x_1, x_1 \in x_2, .., x_n \in x_{n+1}, ..$ containing $\in$-relation;

13.4. specifically there exists no $S \in S$;

13.5. and there no "cycles" in representation strings $x_0 \in x_1, x_1 \in x_2, .., x_n \in x_0$;

14. ***Axiom of string arithmetic***

14.1. (string exponent) we can define a shorthand, if $z = x \cdot \cdots \cdot x$ and $|z|_x = n$, then $z = |x^n|$;

14.2. let S be a collection of strings[314], so that:

14.2.1. string concatenation function turns $(S, \cdot)$ into a *monoid* with identity element $\emptyset$, s.t.:

- *Associativity*: $\forall x, y, z.[x \cdot [y \cdot z] = [x \cdot y] \cdot z]$
- *Identity element*: $\forall x.[[x \cdot \emptyset] = x = [x \cdot \emptyset]]$

14.2.2. there is a set of natural numbers as a part of $\mathcal{L}$-structure $(\mathbb{N}, 0, +)$

[314] a model placeholder

14.3. then we can also define arithmetic on S using:

14.3.1. the string length $|x|$

14.3.2. substring counting $|x|_y$

by extending previous substring and empty string axioms as following:

14.3.1. $|\emptyset| = |\emptyset|_\emptyset = 0$

14.3.2. $x = x \cdot y \implies |x|_y = 0$

14.3.3. $x \cdot y = y \implies |y|_x = 0$

14.3.4. $|x|_y = 0 \implies x \neq y$

14.3.5. $\forall x.[x \in \Sigma \implies |x|_x = |x| = 1]$[315]

14.3.6. $\forall x, y.[x, y \in \Sigma \implies |x|_x + |y|_y = |x| + |y| = |x \cdot x| = 2]$

14.3.7. $\forall x.[x \in \Sigma \implies [|x| + \cdots + |x| = |x \cdot \cdots \cdot x| = n] \wedge [n \in \mathbb{N}]]$

14.3.8. $[|x \cdot \cdots \cdot x|_x = |x^n|_x = n] \wedge [n \in \mathbb{N}]$

14.3.9. $[|x|_y = 0] \wedge [z = x \cdot y] \implies |z|_x = |z|_y = 1$

14.3.10. $[|x|_y = 0] \wedge [z = x \cdot y \cdot x \cdot y] \implies |z|_y = 2$

14.3.11. $[|x|_y = 0] \wedge [z = x \cdot y \cdot x \cdot y \cdot x \cdot y] \implies |z|_y = 3$

14.3.12. $[|x_1|_y = \cdots = |x_n|_y = 0] \wedge [z = x_1 \cdot y \cdot x_2 \cdot y \ \ldots \ y \cdot x_n \cdot y] \implies |z|_y = n, n \in \mathbb{N}$

14.4. we can naturally extend the above definitions to classes of $\mathbb{Z}$ and Ord by applying string arithmetic to the axiom of index class.

15. ***Axiom of Choice*** Every family of nonempty classes of strings has a choice function[316].

String arithmetic as mentioned in the previous defintion[317] can be compared to other arithmetical formal systems like Presburger arithmetics or recursive Peano axioms[6]. Namely, string arithmetic allows unary substring-counting with $|x|_y$ over some string concatenation monoid S. Specifically, for Peano addition $a + 0 = a$ we will have something equivalent as in $a + 0 = |x^a \cdot \emptyset|_x = |x^a|_x$ (see the *table 3* below).

Power-string operation is defined as a generalization of a concatenation "product" $X \cdot X$, where X is some class of strings. If we want to run power-string operation recursively, we can start with some $X_0 := \bigcup\{\{s \in \Sigma\} \cup \emptyset\}$ (union of some finite alphabet Σ and an empty string $\emptyset$). Next iteration will start over the union of $X \cdot X$.

[315] but also, in general, $|x|_x = 1$

[316] This axiom is a semantically equivalent alias to *Axiom of Choice* in ZFC, and is supposed to be considered separately from the rest of the *basic string theory* axioms or $\boldsymbol{\mathcal{T}}_{str}$ (just like choice is separate from ZF)

[317] in the *Axiom of string arithmetic* of *Definition - 116*

Table 3: Matching Peano addition with substring-counting addition

#	Peano addition	Substring counting
1	$a+0=a$	$a+0=\|x^a\cdot\emptyset\|_x=\|x^a\|_x$
2	$a+S(b)=S(a)+S(b)$	$a+\|x^b\|_x=\|x^a\|_x+\|x^b\|_x$
3	$a+1=S(a)$	$a+1=\|x^a\|_x+\|x\|_x$
4	$a+2=S(S(a))$	$a+2=\|x^a\|_x+\|x\cdot x\|_x$
5	$a+3=S(S(S(a)))$	$a+3=\|x^a\|_x+\|x\cdot x\cdot x\|_x$
	$\dots$	$\dots$

Definition 117.

Given a class of strings X, we can define *power-string* operation as $\P(X) := \bigcup\{x\cdot y : \forall x,y\in X\}$, so that $\P(X)$ is a subclass of all unique strings in $X\cdot X$ table produced by string concatenation[318].

κ-String Model is a straightforward example of *basic string theory* implementation that works with infinite strings. Given a model theory notion of a language $\boldsymbol{\mathcal{L}}_{str}=(\cdot^{str},c_1^{str},..c_n^{str})$, we can redefine regular languages[319] as an extension of $\boldsymbol{\mathcal{L}}_{str}$-theory, i.e. $\boldsymbol{\mathcal{T}}_{str}$.

Theorem 126 - κ-String Model.

Consider a string concatenation monoid S_κ with strings of fixed cardinality κ. We say that $\boldsymbol{\mathcal{T}}_{str}$ can be satisfied by $\boldsymbol{\mathcal{L}}_{str}$-structure $\boldsymbol{\mathcal{S}}:=(S_\kappa,\cdot,\Sigma)$, where:

- S_κ - is a set of all strings of cardinality κ over a finite alphabet Σ, s.t. $S=\{s : s\in\Sigma^\kappa, |s|=\kappa\}$[320];
- $\cdot$ - is a binary function of string concatenation operation;
- Σ - is a constant of the alphabet symbols of a regular language, e.g. $\Sigma=\{0,1\}$;

Lemma 127 - Lemma of the same length.

The maximum length or simply length of each string in S_κ is fixed by κ: $\forall x,y:[x,y\in S_\kappa]\implies[|x|=|y|=\kappa]$

[318] refer to appendix and see power-string operation illustrated in fig. 8 and finite examples illustrated in python

[319] and ω-regular languages - computer science notions that we used earlier in the paper to argue about computability of SEFs

[320] here if $\kappa=\omega$, this becomes ω-Kleene operator for Σ^κ as with ω-regular languages, however we leave the definition open-ended or abstract enough to accept greater cardinalities

In set theory one can define the cumulative hierarchy of sets[321] V_α. Recall that a class T is transitive if $x \in T \implies x \subset T$. By analogy, we can observe the same for strings:

i.) if $\forall\tau.[[\tau \in T] \wedge [\S(\tau) \subset T]]$, then T is transitive for string τ;

ii.) recursive application of *power-string* operation forms a cumulative hierarchy of strings T_γ, where γ is either ordinal or cardinal.

Again, note that since T is a class and τ is a string, $\tau \in T$ means "member of". Now, let us explain how power-string operates in more detail before formally defining the cumulative hierarchy of strings.

Definition 118.

The cumulative hierarchy of strings is a class of strings T_γ indexed by the class of ordinal numbers $\gamma \in Ord$; s.t., T_γ is the class of all strings having *ranks* less than γ. Thus, there is one class T_γ for each ordinal number γ. T_γ may be defined by *transfinite recursion* as follows:

1. Let T_0 be the empty set: $T_0 := \emptyset$
2. Let T_1 be the union set: $T_1 := \emptyset \cup \Sigma$, where Σ is a finite alphabet (for example, $\Sigma := \{0, 1\}$[322])
3. For any ordinal number $\gamma \in Ord$, let $T_{\gamma+1}$ be the power-string of T_γ: $T_{\gamma+1} := \P(T_\gamma)$
4. For any limit ordinal $\mathfrak{m}$, let $T_\mathfrak{m}$ be the union of all the T_γ so far ($\forall\gamma \in Ord.[\gamma < \mathfrak{m}]$): $T_\mathfrak{m} := \bigcup_{\gamma<\mathfrak{m}} T_\gamma$

Lemma 128.

There exists a surjection between any ordinal γ and a string τ:
$\forall\gamma.\exists\tau.[\gamma \mapsto \tau]$.

Proof. Follows from definition of ranks of strings in T_γ. There is one set T_γ for each ordinal number γ. Now pick any string $\tau \in T_\gamma$ to obtain mapping $\gamma \mapsto \tau$. □

Lemma 129.

Sets T_γ have the following properties[323] (by induction):

[321] Also see *8.2 - Expected Length of Strings*
[322] as in fig. 8
[323] which are very much similar to the cumulative hierarchy of sets V_α

i.) Each T_γ is transitive[324]

ii.) If $\forall\beta,\gamma \in Ord : \beta < \gamma$, then $T_\beta \subset T_\gamma$

iii.) $V_\gamma \subset T_\gamma$[325]

iv.) $\gamma \subset T_\gamma$ [326]

If we can consider strings as another representation for sets[327], that provides some important insight into the nature of fundamental set-theoretical constructs such as V_α. In fact, according to *Lemma 129 (iii)*, for each $\forall V_\gamma, \exists T_\gamma$: $V_\gamma \subset T_\gamma$. This becomes very obvious from the illustration of how *power-string* works and comparison in table 4. Namely:

$$i > 0 : |T_i| = 2^{2^i} - 1 \wedge |V_i| = 2^{2^{i-1}}$$

9.3 Some morphisms between sets and strings

Previous formula makes sense only after mapping sets of cumulative hierarchy V_α into their respective counterparts as binary strings. In other words, a number of morphisms will become handy for our further discussion.

Definition 119 - Kuratowski morphism.

Let S and V be respectively a class of strings (with some finite alphabet Σ) and a class of sets. Then there exists an injective morphism $k : S \to V$, which can be defined using a recursive version of the Kuratowski notation. Such morphism will take each symbol of every α-index string $x \in S, |x| = \alpha, \alpha \in Ord$, so that $x =$ "$x_1, .., x_\alpha$", $x_\alpha \in \Sigma$, and recursively map it to an ordered pair $k(x) \in V$ as following:

$$k(x_1, .., x_\alpha) = \begin{cases} \{\}, & \text{if } \alpha = 0 \\ \{x_1\}, & \text{if } \alpha = 1 \\ \{\{x_1\}, \{x_1, x_2\}\}, & \text{if } \alpha = 2 \\ \{\{k(x_1, .., x_{\alpha-1})\}, \{k(x_1, .., x_{\alpha-1}), x_\alpha\}\}, & \text{otherwise} \end{cases}$$

An important property of Kuratowski notation is that images of any two strings can be easily validated if they are equal or not.

[324] in the sense of *string splitting axiom*

[325] see appendix for fig. 8 and table 4 in *power-string* subsection

[326] ordinals can be encoded as sets using V_α, so this follows from previous $[\gamma \subset V_\gamma] \wedge [V_\gamma \subset T_\gamma] \implies \gamma \subset T_\gamma$

[327] for example, a string can be just a formula in pure set theoretical thinking

Definition 120 - Fraenkel-Cantor morphism.

Consider *Lemma 3.3* in [6], saying that $|A| = \kappa \implies |\mathcal{P}(A)| = 2^{\kappa}$ and the corresponding mapping from its proof. For every $X \subset A$, let f_X be the function

$$f_X(x) = \begin{cases} 1 & \text{if } x \in X, \\ 0 & \text{if } x \in A - X. \end{cases}$$

The mapping $f : X \to f_X$ is in a one-to-one correspondence between $\mathcal{P}(A)$ and $\{0,1\}^A$.

Again, an important consequence of the *Lemma 3.3* (referenced in the above *Definition 120*) is that: given $|A| = \aleph_0 \implies |\mathcal{P}(A)| = 2^{\aleph_0}$. We have already used this fact much earlier in text - see *Lemma 19*.

Next morphism is based on the principle of formal logic which we prefer to state more formally as well.

Principle 2 - Principle of explicit representation.

Implicit definition equals explicit definition.

Here, by explicit definition we mean that when we define ZF classes as formulas we should be able to explicitly write them down[328]. For instance, take a recipe for set construction of cumulative hierarchy V_α and apply transfinite induction. According to this *principle of explicit representation* one should still be able to map each set into an infinite sequence of "{" and "}" braces, or equivalently as $\{0,1\}$. In that case, we can consider a morphism that does exactly this.

Definition 121 - Catalan morphism.

Let V be a class of sets, then as per *principle of explicit representation* we assume existence of the following corresponding sets:

- S is a class of strings with a fixed finite alphabet of "{" and "}"
- $\mathbb{B}$ is a class of binary strings with a fixed finite alphabet of $\{0,1\}$

so that S and $\mathbb{B}$ can admit the following morphisms:

- for every $x \in V$ there is an identity mapping to its explicit representation $e : V \to S$

[328] Including such less common representations for sets as using a variation of a TM with an infinite multitude of tapes as the machine state or even some other formalism

- there exists also a binary encoding $r : S \to \mathbb{B}$ of such representation, which maps "{" and "}" braces into $\{0, 1\}$

Namely, for every $x \in V$ and representation image $e(x)$:

$$r_x(s) = \begin{cases} 0 & \text{if } s = \text{“\{”}, \\ 1 & \text{if } s = \text{“\}”}. \end{cases}$$

Finally, we call *Catalan morphism* a composition of $c := r \circ e$, so that $c : V \to \mathbb{B}$. Note that well-balancing of "{" and "}" braces implies that binary strings in $ran(c)$ must be well-balanced pairs of $\{0, 1\}$. Which means that c is naturally injective as it maps each $x \in V$ into $b \in ran(c)$, where $ran(c)$ is a subset of strings with well-balanced $\{0, 1\}$.

9.4 Strictly-stronger consistency of ZFS

SEF Theory can be defined as an extension over *κ-String Theory*.

Definition 122 - SEF Theory.

Let $\boldsymbol{\mathcal{S}} := (S_\kappa, \cdot, =, \Sigma)$ be a $\boldsymbol{\mathcal{L}}_{str}$-structure and a model for $\boldsymbol{\mathcal{L}}_{str}$ language in which we describe *κ-String Theory*[329] denoted as $\boldsymbol{\mathcal{T}}_{str}$, s.t. $\boldsymbol{\mathcal{S}} \models \boldsymbol{\mathcal{L}}_{str}$ (including $\boldsymbol{\mathcal{S}} \models \boldsymbol{\mathcal{T}}_{str}$). Since we want to describe *SEF Theory*, we will define an extension of $\boldsymbol{\mathcal{L}}_{str}$ language called $\boldsymbol{\mathcal{L}}_{sef}$. Namely, $\boldsymbol{\mathcal{L}}_{sef} \supset \boldsymbol{\mathcal{L}}_{str}$, s.t. if $\boldsymbol{\mathcal{S}} \models \boldsymbol{\mathcal{L}}_{str}$, then $\exists \boldsymbol{\mathcal{S}}_{sef}.[[\boldsymbol{\mathcal{S}}_{sef} \supset \boldsymbol{\mathcal{S}}] \wedge [\boldsymbol{\mathcal{S}}_{sef} \models \boldsymbol{\mathcal{L}}_{sef}]]$.

We describe *SEF Theory* as $\boldsymbol{\mathcal{L}}_{sef}$-theory denoted as $\boldsymbol{\mathcal{T}}_{sef}$. It can be defined as an extension over *κ-String Theory* $\boldsymbol{\mathcal{T}}_{str}$ and satisfied by the respective $\boldsymbol{\mathcal{L}}_{sef}$-structure $\boldsymbol{\mathcal{S}}_{sef}$. Namely, $\boldsymbol{\mathcal{S}}_{sef} \models \boldsymbol{\mathcal{L}}_{sef}$ iff (1) $\boldsymbol{\mathcal{L}}_{sef} \supset \boldsymbol{\mathcal{L}}_{str}$, (2) $\boldsymbol{\mathcal{T}}_{sef} \supset \boldsymbol{\mathcal{T}}_{str}$ and (3) there is a subset of strings $\exists \mathbb{S}_\kappa.[\mathbb{S}_\kappa \subset S_\kappa]$, s.t. each string is a syntactically valid SEF or $\mathbb{S}_\kappa \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$[330], and (4) $\boldsymbol{\mathcal{T}}_{sef} \vdash \mathbb{S}_\kappa \implies \boldsymbol{\mathcal{S}}_{sef} \vdash \boldsymbol{\mathcal{T}}_{sef} \implies \boldsymbol{\mathcal{S}}_{sef} \models \boldsymbol{\mathcal{T}}_{sef}$.

Specifically, (1) and (2) means that $\boldsymbol{\mathcal{T}}_{sef}$ theory extends axioms of $\boldsymbol{\mathcal{T}}_{str}$ theory with the following additional FOL axioms:

17. *Axiom of well-balanace* - $z =$ "$x \cdot y$" $\implies$ $z \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$ is syntactically valid iff x (as every initial prefix of z) never contains more opening brackets than closing brackets and brackets are equally matched. Namely, $\forall x, y \in z.[|x|_{)} \leq |x|_{(}] \wedge [|z|_{(} = |z|_{)}]$.

18. *Axiom of replication* - if alphabet constant $\Sigma \in \boldsymbol{\mathcal{S}}_{sef}$ contains round brackets $\{(,)\} \subset \Sigma$, which are required to act as replication operator, then the following recursive definition for *string enumeration formula* (SEF) is true:

[329] as according to *Definition* **??**

[330] or simply $\mathbb{S}_\kappa \vdash \top$ if $\boldsymbol{\mathcal{T}}_{sef}$ is clear from the context

i.) if x is a string and $|x|_(= |x|_) = 0$, then
   - x is a syntactically valid SEF: $x \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$
   - $\phi := \text{“(”} \cdot x \cdot \text{“)”} = x \cdot .. \cdot x$ is replication of x, s.t. $|x| \leq |\phi| \leq \kappa$
   - ϕ is a syntactically valid SEF: $\phi \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$

ii.) if x, y, z are syntactically valid SEFs, then $\phi = x \cdot \text{“(”} \cdot y \cdot \text{“)”} \cdot z$ is also a SEF, namely $\phi \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$[331]

Further, we are going to show that there exists a model that can satisfy $\boldsymbol{\mathcal{T}}_{sef}$[332].

Lemma 130 - Model of SEF Theory.

Our assumptions are as following:

- $\boldsymbol{\mathcal{T}}_{sef} \supset \boldsymbol{\mathcal{T}}_{str}$
- $\Sigma = \{0, 1, (,)\}$ is an alphabet constant
- $\boldsymbol{\mathcal{S}} := (S_\kappa, \cdot, =, \Sigma)$ be a $\boldsymbol{\mathcal{L}}_{str}$-structure
- $\boldsymbol{\mathcal{S}}_{sef} := (\mathbb{S}_\kappa, \cdot, \pi, =, \Sigma)$ is a $\boldsymbol{\mathcal{L}}_{sef}$-structure

If $\mathbb{S}_\kappa \subset S_\kappa$ and $\mathbb{S}_\kappa \vdash_{\boldsymbol{\mathcal{T}}_{sef}} \top$, then $\boldsymbol{\mathcal{S}}_{sef} \models \boldsymbol{\mathcal{T}}_{sef}$.

Corollary 131.

$\mathbb{T}_\kappa$ is a *transitive model* of $\boldsymbol{\mathcal{T}}_{sef}$.

Now let us show that SEF is *consistent with* ZF, and vice versa. To do so, we follow the narrative in [6][333] as well as some results in model theory[47]. In order to proof (relative) consistency we will rely on the notion of absoluteness of quantifier-free formulas in FOL [334].

We will look at *strictly-stronger consistency theory than SEF and ZF*[335] defined as $ZFS := ZF + SEF$ to show that ZFS is both *consistent relative to* ZF and *consistent relative to* SEF[336]. This can be a viable alternative

[331] note that *ii.* implies *well-balance* and vice versa (if we avoid empty pairs - see *Lemma 130*)

[332] Note that we will not be assuming $ZF + AC$ axioms behind the scene, specifically for *power-string* operation, recursion (transfinite induction) and cardinal arithmetic - all of this is provisioned by *basic string theory* $\boldsymbol{\mathcal{T}}_{str}$ and extended by $\boldsymbol{\mathcal{T}}_{sef}$

[333] Specifically, see p. 163-166 for topics of *Relative Consistency*, *Transitive Models and* $\boldsymbol{\Delta}_0$ *Formulas* and *Consistency of the Axiom of Regularity*

[334] mentioned earlier - see *Definition 113 of Primordial theory*

[335] Recall that if T is consistent relative to S, but S is not known to be consistent relative to T, then we say that S has greater consistency strength than T

[336] Can be abbreviated as $Con(ZF) \implies Con(ZFS)$, etc

to any other approach of examining a version of *pure SEF theory* (already discussed as $\boldsymbol{\mathcal{T}}_{sef}$) in attempt to show that it can be as rich and expressive as ZF[337]. But as we will soon see - ZF and SEF theories are closely interconnected, so it is rather natural to have ZFS as an extension to both.

However, there is still a small detail. The corollary of the below stronger consistency proof suggests that both propositions in ZF and in SEF are *mutually inclusive* not only as theories but as languages. Meaning that ZFS can (semantically) act as a *lingua franca* for propositions with inherently different language structures. For example, let $\psi(x_1, ..., x_n)$ be a formula of the ZF set theory language $\boldsymbol{\mathcal{L}}_{zf}$ with $\{\in\}$ structure *versus* another formula $\varphi(x_1, ..., x_n)$ of the SEF theory language $\boldsymbol{\mathcal{L}}_{sef}$ with $\{\cdot, \pi, \Sigma\}$ structure. It is not uncommon for the structure complexity to grow next to the development of the actual theory and the object of interest behind it[338]. Further we will assume that whenever an upstream formula like ψ or φ is brought downstream into $\boldsymbol{\mathcal{L}}_{zfs}$ language, the semantic preserving structure translation is taking place with model relativization.

Theorem 132 - Strictly-stronger consistency of ZFS.

$ZFS := ZF + SEF$ is strictly-stronger consistent than SEF and ZF. Namely, given $\boldsymbol{\mathcal{L}}_{zfs}$ language with structure $\{\in, \cdot, \pi, \Sigma\}$ the following holds:

1. $Con(ZF) \implies Con(ZFS)$
2. $Con(SEF) \implies Con(ZFS)$

Corollary 133.

$\boldsymbol{\mathcal{M}}' \prec \boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{N}}' \prec \boldsymbol{\mathcal{Q}}$ are elementary submodels of $\boldsymbol{\mathcal{Q}}$.

On one hand, the later corollary is merely a useful observation that given the existence of $\boldsymbol{\mathcal{Q}}$ as a model for $\boldsymbol{\mathcal{T}}_{zfs}$ the nature of set Q may remain abstract. On the other hand - each of the concrete elementary submodels $\boldsymbol{\mathcal{M}}'$ and $\boldsymbol{\mathcal{N}}'$ provides an important insight that $\boldsymbol{\mathcal{Q}}$ may behave both as a class of ZF sets and a collection of SEF strings depending on the context.

Corollary 134.

$\boldsymbol{\mathcal{Q}}$ can act as a transitive model for ZFS.

[337] As we know from GSIT $ZF \not\Longrightarrow Con(ZF)$ as well as $SEF \not\Longrightarrow Con(SEF)$

[338] For example, in the hierarchy of algebraic structures one can construct fields from commutative rings and so on

10 Conclusions

In this paper we have developed a new framework for understanding infinite binary strings, resolving the Continuum Hypothesis in a concrete Boolean-valued model, and uncovering deep connections to large-cardinal axioms. Our main findings may be summarized as follows.

String Enumeration Formulas (SEFs)

Primary Results

- **Definition of SEFs.** We introduced a formal language of *String Enumeration Formulas*, built from replicators and nested parentheses, which compactly describe arbitrary infinite binary strings.
- **Categorization of SEFs.** We classified infinite binary strings into cyclic, non-cyclic, and more complex types via their growth patterns in the *Generalized Context Transformation Algorithm (GCTA)* tables.
- **Continuum Access Scheme (CAS).** We exhibited a novel schema using SEFs that accesses uncountable cardinalities, showing that the right-hand side of the GCTA table grows strictly faster (uncountably larger) than the left-hand side, in accordance with Cantor's theorem.
- **Uncountability of SEF languages.** Allowing infinitely nested replication yields a language of SEFs of cardinality $2^{\aleph_0}$, hence uncountable.

Secondary Results

- **Finite SEFs as ω-regular languages.** We showed that the finite fragment of SEFs is ω-regular and admits parsing by standard ω-automata.
- **Production function.** We defined a bijection from equivalence-class labels of SEFs onto the Cantor set of infinite binary strings.
- **Fair vs. unfair SEFs.** We distinguished *fair* (non-cyclic) SEFs from *unfair* (cyclic) ones and analyzed their combinatorial properties.

Determinacy

Primary Results

- **Fair determinacy.** We proved that games on infinite binary strings defined by fair SEFs are determined—Player II has a winning strategy whenever the payoff set arises from a fair formula.

- **Extension of GCTA.** We generalized the GCTA to infinite-length strings, providing a systematic tool for analyzing their structure and determinacy.

A Non-CH Model

Primary Results

- **Boolean Prime Ideal for the continuum.** By building a Boolean-valued model via a complete Boolean algebra of replicated strings, we exhibited a model in which

$$2^{\aleph_0} = \aleph_2,$$

  thus refuting CH.

- **Inner model with replicated hosting.** We introduced the notion of a *replica host* and showed how to embed a transitive inner model into a larger universe so as to realize CH's failure explicitly.

Secondary Results

- **Interactions with large-cardinal axioms.** We discussed how the Boolean-valued construction sits inside $ZF + DC$ and interacts with various choice principles.

- **Role of SEFs in forcing.** We highlighted the utility of SEFs as names for conditions in forcing constructions.

Replication Schema

Primary Results

- **Proper class of replicata.** We proved that there is a proper class of *transfinite replicata*, each a structure of nested SEFs of length a limit cardinal.

- **Existence of supercompact cardinals.** From the replication schema we derived the existence of supercompact (indeed strongly compact) cardinals.

- **Fine and normal measures on replicata.** We showed that each replicatum carries a canonical fine measure (extending closed-unbounded filters) which is in fact normal.

Secondary Results

- **Transfinite strings.** We formalized transfinite strings as functions $\mathrm{Ord} \to \{0,1\}$ of prescribed length.
- **Iterated powerset and club filters.** We introduced the machinery of P^α and closed-unbounded filters on $\mathcal{P}_\kappa(\lambda)$ in service of the schema.
- **Construction of the N^* model.** We explained how to build a model of $ZF + DC_\lambda + I$ (with an inaccessible) that hosts the full proper-class replicata.

Summary of Novel Contributions

- We gave the first complete theory of *String Enumeration Formulas* for infinite binary strings, linking them to ω-regular languages, determinacy, and forcing.
- We constructed a concrete Boolean-valued universe in which $\neg CH$ holds and exhibited the mechanism by which $2^{\aleph_0} = \aleph_2$.
- We introduced the concept of *replicated hosting* and proved the existence of a proper class of transfinite replicata.
- We showed that these replicata carry fine and normal measures, yielding supercompact and strongly compact cardinals, and hence imply the forcing-axiom MM^{++}.

Together, these advances establish a new bridge between infinite-string combinatorics, determinacy, independence phenomena in set theory, and the large-cardinal hierarchy.

In conclusion of this interdisciplinary paper, it is worth pointing out also one of our firstly presented results (in *Context Transformation Algorithm* section), which concerns the context depended transformation technique and has a number of practical applications. Although throughout the rest of the paper our focus was mostly with the consequences of defining and applying CTA to the infinite binary strings, the finite case has possible applications in different areas of computer science such as data compression, cryptography and machine-learning.

Applications in data compression do not require much adaptation specifically for lossless compression of rather sparse (low-entropy) datasets, when finite patterns are easily detectable. A version of CTA can perform well not only as obvious generalization of the running length encoding, but as a part of more sophisticated entropy modeling for computing expected probabilities[339].

[339] which are then used in arithmetic encoding similar to PAQ[51] family of data compression algorithms

In cryptography, an immediate application of the context-depend XOR transformation would be an implementation of memory-bounded hash function[340]. Less obvious application would be pursuing further adaptation of context-depend transformations similar to CTA by using vector or polynomial commitment schemes instead of XOR. Such cryptographic commitments can help to obtain a more optimal and robust verification proofs depending on the requirements.

We speculate that variations of proposed CTA can be successfully employed for supervised and unsupervised machine-learning techniques[341] by extracting features from context sensitive informational settings. For example, instead of computing XOR, value-state rows can contain a feature signature by keeping track of relative informational distances[342] in some metrizable space for each observable context. By the end of the algorithm obtained signatures can be used as feature vectors for machine-learning classification tasks.

[340] For example simply by modifying well-known SHA-256 implementation

[341] such as training of support-vector machines (SVMs), artificial neural-networks (ANN) and specifically deep convolutional neural-networks networks (deep learning)

[342] information distances are then calculated as approximations of relative Kolmogorov complexity[52]

References

1. Penrose, R. *The Road to Reality: A Complete Guide to the Laws of the Universe* ISBN: 9780679776314. https://books.google.ch/books?id=coahAAAACAAJ (Vintage Books, 2007).
2. Cantor, G. & Jourdain, P. *Contributions to the Founding of the Theory of Transfinite Numbers* (Dover Publications, 1915).
3. Gamow, G. *One, Two, Three– Infinity: Facts and Speculations of Science* ISBN: 9780486256641. https://books.google.ch/books?id=EZbcwk6SkhcC (Dover Publications, 1988).
4. Goldrei, D. *Classic Set Theory: For Guided Independent Study* ISBN: 9780412606106. https://books.google.by/books?id=1dLn0knvZSsC (Taylor & Francis, 1996).
5. Kunen, K. *Set Theory: An Introduction to Independence Proofs* ISBN: 9780444854018. https://books.google.ch/books?id=9vG6lAEACAAJ (North-Holland Publishing Company, 1980).
6. Jech, T. *Set Theory* Third. ISBN: 978-3-540-44761-0. https://link.springer.com/book/10.1007/3-540-44761-X (Springer Berlin Heidelberg, 2003).
7. Ferreirós, J. *Labyrinth of Thought: A History of Set Theory and Its Role in Modern Mathematics* ISBN: 9783764383503. https://books.google.ch/books?id=vrQLbbxGNMsC (Birkhäuser Basel, 2008).
8. Kanamori, A. The mathematical development of set theory from Cantor to Cohen. *The Bulletin of Symbolic Logic* **2,** 1–71 (1996).
9. Herrlich, H. *Axiom of Choice* 1st ed. ISBN: 978-3-540-34268-7. https://link.springer.com/book/10.1007/11601562 (Springer Berlin Heidelberg, 2006).
10. Wikipedia contributors. *Continuum hypothesis — Wikipedia, The Free Encyclopedia* [Online; accessed 13-January-2022]. 2021. https://en.wikipedia.org/w/index.php?title=Continuum_hypothesis&oldid=1062726958.
11. Sipser, M. *Introduction to the Theory of Computation, Second Edition* 2005.
12. Staiger, L. in *Handbook of Formal Languages: Volume 3 Beyond Words* (eds Rozenberg, G. & Salomaa, A.) 339–387 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1997). ISBN: 978-3-642-59126-6. https://doi.org/10.1007/978-3-642-59126-6_6.
13. Wikipedia contributors. *Cantor set — Wikipedia, The Free Encyclopedia* [Online; accessed 19-March-2022]. 2022. https://en.wikipedia.org/w/index.php?title=Cantor_set&oldid=1071687681.

14. Kolmogorov, A. & Morrison, N. *Foundations of the Theory of Probability* ISBN: 9781470452995. https://books.google.ch/books?id=BdDBDwAAQBAJ (American Mathematical Soc., 2019).

15. Wikipedia contributors. *Scott's trick — Wikipedia, The Free Encyclopedia* [Online; accessed 26-August-2022]. 2021. https://en.wikipedia.org/w/index.php?title=Scott%27s_trick&oldid=1056369148.

16. Karagila, A. *Lecture Notes: Axiomatic Set Theory* [Online; accessed 30-August-2022]. 2018. http://karagila.org/files/set-theory-2017.pdf.

17. Jessop, S. *What is meant by "dovetailing" — Answer on Stack overflow* [Online; accessed 26-August-2022]. 2011. https://stackoverflow.com/a/5107312.

18. Calude, C. S., Dinneen, M. J. & Shu, C.-K. Computing a Glimpse of Randomness. *Experimental Mathematics* **11,** 361 –370. https://doi.org/ (2002).

19. Gillman, L. Two Classical Surprises Concerning the Axiom of Choice and the Continuum Hypothesis. *The American Mathematical Monthly* **109** (June 2002).

20. Howard, P. & Rubin, J. *Consequences of the Axiom of Choice* ISBN: 9780821809778. https://books.google.ch/books?id=YXaVkHPQED4C (American Mathematical Society, 1998).

21. Jech, T. About the Axiom of Choice. *Studies in logic and the foundations of mathematics* **90,** 345–370. https://api.semanticscholar.org/CorpusID:117003010 (1973).

22. Howard, P. E. & Rubin, J. E. *Consequences of the axiom of choice* in (1998). https://api.semanticscholar.org/CorpusID:118475523.

23. Blass, A. Howard Paul and Rubin Jean E.. Consequences of the axiom of choice, Mathematical Surveys and Monographs, vol. 59. American Mathematical Society, Providence, RI, 1998, viii + 432 pp. *Bulletin of Symbolic Logic* **11,** 61 –63. https://api.semanticscholar.org/CorpusID:123874170 (2005).

24. nLab authors. *dependent choice* https://ncatlab.org/nlab/show/dependent+choice. Revision 7. Sept. 2023.

25. ASPERÓ, D. & KARAGILA, A. DEPENDENT CHOICE, PROPERNESS, AND GENERIC ABSOLUTENESS. *The Review of Symbolic Logic* **14,** 225–249. https://doi.org/10.1017%2Fs1755020320000143 (2020).

26. Halbeisen, L. J. *Combinatorial set theory: With a gentle introduction to forcing* (Springer, 2012).

27. Läuchli, H. Coloring infinite graphs and the Boolean prime ideal theorem. *Israel Journal of Mathematics* **9,** 422–429. https://api.semanticscholar.org/CorpusID:122090105 (1971).

28. Karagila, A. Zornian functional analysis or: How i learned to stop worrying and love the axiom of choice. *arXiv preprint arXiv:2010.15632* (2020).

29. Asperó, D. & Schindler, R. MM^{++} *implies* $(*)$ 2021. arXiv: 1906.10213 [math.LO].

30. Viale, M. Category forcings, MM+++, and generic absoluteness for the theory of strong forcing axioms. *Journal of the American Mathematical Society* **29,** 675–728 (2016).

31. Larson, P. B. in *Handbook of Set Theory* (eds Foreman, M. & Kanamori, A.) 2121–2177 (Springer Netherlands, Dordrecht, 2010). ISBN: 978-1-4020-5764-9. https://doi.org/10.1007/978-1-4020-5764-9_25.

32. Magidor, M. *Can the Continuum Problem be Solved?* 2011. https://youtu.be/3b_wbObaNgA.

33. Solovay, R. M. A Model of Set-Theory in Which Every Set of Reals is Lebesgue Measurable. *Annals of Mathematics* **92,** 1–56. ISSN: 0003486X. http://www.jstor.org/stable/1970696 (2024) (1970).

34. Woodin, H. *The Axiom of Determinacy, Forcing Axioms, and the Nonstationary Ideal* in (1999). https://api.semanticscholar.org/CorpusID:56357189.

35. Woodin, W. H. *The Continuum Hypothesis, Part I* in (2001). https://api.semanticscholar.org/CorpusID:5841256.

36. Woodin, W. H. *The Continuum Hypothesis, Part II* in (2001). https://api.semanticscholar.org/CorpusID:14378920.

37. Woodin, W. H. *Set Theory, Arithmetic, and Foundations of Mathematics: The continuum hypothesis, the generic-multiverse of sets, and the* Ω *conjecture* in (2011). https://api.semanticscholar.org/CorpusID:7900557.

38. Foreman, M., Magidor, M. & Shelah, S. Martin's Maximum, Saturated Ideals, and Non-Regular Ultrafilters. Part I. *Annals of Mathematics* **127,** 1–47. ISSN: 0003486X. http://www.jstor.org/stable/1971415 (2024) (1988).

39. Foreman, M., Magidor, M. & Shelah, S. Martin's Maximum, saturated ideals and nonregular ultrafilters. I. *Annals of Mathematics* **127,** 1–47 (1988).

40. Foreman, M. Stationary reflection and Martin's Maximum. *Journal of Symbolic Logic* **66,** 792–794 (2001).

41. Todorčević, S. A note on the Proper Forcing Axiom. *Contemporary Mathematics* **31,** 209–218 (1984).

42. Baumgartner, J. E. in *Handbook of Set-Theoretic Topology* (eds Kunen, K. & Vaughan, J. E.) 913–959 (Elsevier, 1984).

43. Steel, J. R. PFA implies AD in $L(\mathbb{R})$. *Journal of Symbolic Logic* **65,** 2085–2093 (2000).

44. Caicedo, A. E. The bounded proper forcing axiom, well-ordering the reals and generic absoluteness. *Mathematical Research Letters* **13,** 521–532 (2006).

45. Caicedo, A. E. & Veličković, B. The bounded proper forcing axiom and well-orderings of the reals. *Mathematical Research Letters* **11,** 671–682 (2004).

46. Foreman, M. in *Handbook of Set Theory* 885–1149 (Springer, 2010).

47. Marker, D. *Model Theory : An Introduction* ISBN: 9780387987606. `https://books.google.ch/books?id=QieAHk--GCcC` (Springer New York, 2002).

48. Hodges, W. & Hodges, S. *A Shorter Model Theory* ISBN: 9780521587136. `https://books.google.ch/books?id=S6QYeuo4p1EC` (Cambridge University Press, 1997).

49. Wikipedia contributors. *Class (set theory) — Wikipedia, The Free Encyclopedia* [Online; accessed 24-January-2023]. 2022. `https://en.wikipedia.org/w/index.php?title=Class_(set_theory)&oldid=1125916999`.

50. Wikipedia contributors. *Presburger arithmetic — Wikipedia, The Free Encyclopedia* [Online; accessed 24-January-2023]. 2022. `https://en.wikipedia.org/w/index.php?title=Presburger_arithmetic&oldid=1112785748`.

51. Mahoney, M. V. *Adaptive weighing of context models for lossless data compression* in (2005).

52. Li, M. & Vitanyi, P. *An Introduction to Kolmogorov Complexity and Its Applications* ISBN: 9780387948683. `https://books.google.ch/books?id=LKEmB\_GQ53QC` (Springer New York, 1997).

A Appendix

A.1 CTA code listing

```
"""
Context Transformation Algorithm (c) 2008-2024 <yauhen.yakimovich.
    mail@gmail.com>
require: bitarray==2.7.3
"""
from bitarray import bitarray as ba


def cta(in_bits, cxt_size=4):
    """Context transformation algorithm function."""
    if type(in_bits) != ba:
        in_bits = ba(in_bits)
    out_bits = ba()
    # initiate context lookup by setting current states to zeros
    lookup = ba([False for x in range(2 << cxt_size)])
    cxt = ba([False for x in range(cxt_size)])
    for x in in_bits:
        index = int(cxt.to01(), 2)
        state = lookup[index]
        # output '1' to signal a change of state,
        # otherwise output '0'
        diff = state != x
        out_bits.append(diff)
        # update last remembered state in lookup
        if diff:
            lookup[index] = x
        cxt.pop(0)
        cxt.append(x)
    return out_bits


def inv_cta(in_bits, cxt_size=4):
    """Inverse context transformation algorithm function."""
    if type(in_bits) != ba:
        in_bits = ba(in_bits)
    out_bits = ba()
    # initiate context lookup by setting current states to zeros
    lookup = ba([False for x in range(2 << cxt_size)])
    cxt = ba([False for x in range(cxt_size)])
    for x in in_bits:
        index = int(cxt.to01(), 2)
        state = lookup[index]
        # inverse: output original value and reconstruct
        #          the state lookup table
        if not x:
            # since state was not changed - pick the value
            # from the context state
            v = state
        else:
```

```
            # state was changed by the original transformation
            # - output value will be an inverse of state
            v = not(state)
        out_bits.append(v)
        lookup[index] = v
        # update current context
        cxt.pop(0)
        cxt.append(v)
    return out_bits


def assert_cta():
    """
    Test direct and inverse CTA mapping.

    Output:

        bitarray('10100111011101111101111111111110')
        bitarray('10100111011100001110000100000001')
        bitarray('10100111011101111101111111111110')
    """
    pre_image = '10100111011101111101111111111110'
    image = cta(pre_image)
    print(ba(pre_image))
    print(image)
    print(inv_cta(image))
    assert ba(pre_image) == inv_cta(image)


if __name__ == '__main__':
    assert_cta()
```

appendix/cta.py

A.2 Labels of the finite SEF equivalence classes

E(1,1)				
(0) (1)				
E(2,1)	***E(2,2)***	***E(2,3)***		
0(1) 1(0) (01) (0)1 (10) (1)0	(0(1)) (0)(1) (1(0)) (1)(0) ((0)1) ((1)0)	((0)(1)) ((1)(0))		
E(3,1)	***E(3,2)***	***E(3,3)***	***E(3,4)***	***E(3,5)***
00(1) 01(0) 0(10) 0(1)0 10(1) 11(0) 1(01) 1(0)1 (001) (010) (011) (01)0 (0)10 (0)11 (100) (101) (10)1 (110) (1)00 (1)01	0(1(0)) 0(1)(0) 0((1)0) 1(0(1)) 1(0)(1) 1((0)1) (00(1)) (01(0)) (01)(0) (01)(1) (0(10)) (0(1))0 (0)1(0) (0)(01) (0)(10) (0)(1)0 (10(1)) (10)(0) (10)(1) (11(0)) (1(01)) (1(0))1 (1)0(1) (1)(01) (1)(0)1 (1)(10) ((01)0) ((0)10) ((0)11) ((0)1)0 ((10)1) ((1)00) ((1)01) ((1)0)1 (0(1)0) (1(0)1)	0((1)(0)) 1((0)(1)) (0(1(0))) (0(1)(0)) (0(1))(0) (0((1)0)) (0)(0(1)) (0)(1(0)) (0)(1)(0) (0)((1)0) (1(0(1))) (1(0)(1)) (1(0))(1) (1((0)1)) (1)(0(1)) (1)(0)(1) (1)(1(0)) (1)((0)1) ((01)(0)) ((01)(1)) ((0(1))0) ((0)1)(0) ((0)1)(1) ((0)(01)) ((0)(10)) ((0)(1))0 ((10)(0)) ((10)(1)) ((1(0))1) ((1)0)(0) ((1)0)(1) ((1)(01)) ((1)(0))1 ((1)(10)) (((0)1)0) (((1)0)1)	(0((1)(0))) (0)((1)(0)) (1((0)(1))) (1)((0)(1)) ((0(1))(0)) ((0)(0(1))) ((0)(1(0))) ((0)(1))(0) ((0)((1)0)) ((1(0))(1)) ((1)(0(1))) ((1)(0))(1) ((1)(1(0))) ((1)((0)1)) (((0)1)(0)) (((0)1)(1)) (((0)(1))0) (((1)0)(0)) (((1)0)(1)) (((1)(0))1)	((0)((1)(0))) ((1)((0)(1))) (((0)(1))(0)) (((1)(0))(1))

Figure 4: Ordered list of equivalence class labels of finite SEFs grouped by length and number of brackets

A.3 Concatenation of SEFs

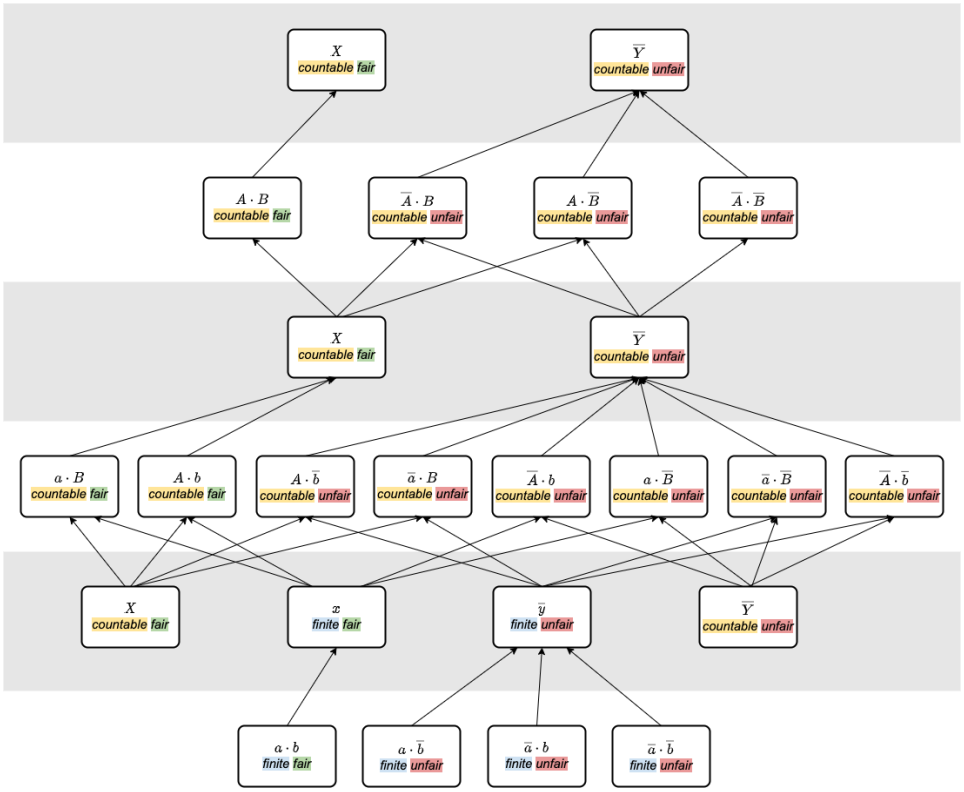


Figure 5: Large Hasse-like diagram for SEF concatenation

SEF concatenation operation "$x \cdot y$" $\mid x, y \in \mathcal{E}_{SEF}$				
kind:= *{ finite, fair, countable, unfair}*	$y := b$ *finite fair*	$y := \overline{b}$ *finite unfair*	$y := B$ *countable fair*	$y := \overline{B}$ *countable unfair*
$x := a$ *finite fair*	$a \cdot b$ *finite fair*	$a \cdot \overline{b}$ *finite unfair*	$a \cdot B$ *countable fair*	$a \cdot \overline{B}$ *countable unfair*
$x := \overline{a}$ *finite unfair*	$\overline{a} \cdot b$ *finite unfair*	$\overline{a} \cdot \overline{b}$ *finite unfair*	$\overline{a} \cdot B$ *countable unfair*	$\overline{a} \cdot \overline{B}$ *countable unfair*
$x := A$ *countable fair*	$A \cdot b$ *countable fair*	$A \cdot \overline{b}$ *countable unfair*	$A \cdot B$ *countable fair*	$A \cdot \overline{B}$ *countable unfair*
$x := \overline{A}$ *countable unfair*	$\overline{A} \cdot b$ *countable unfair*	$\overline{A} \cdot B$ *countable unfair*	$\overline{A} \cdot B$ *countable unfair*	$\overline{A} \cdot \overline{B}$ *countable unfair*

Figure 6: Table for SEF concatenation operation

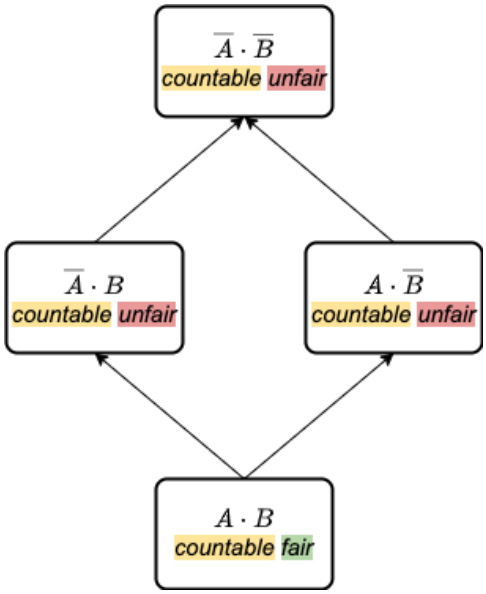


Figure 7: Hasse-like diagram for concatenation of countable SEFs

A.4 Power-string operation

X · X		1	2	3	4	5	6	7
		ø	0	1	00	01	10	11
1	ø	ø	0	1	00	01	10	11
2	0	0	00	01	000	001	010	011
3	1	1	10	11	100	101	110	111
4	00	00	000	001	0000	0001	0010	0011
5	01	01	010	011	0100	0101	0110	0111
6	10	10	100	101	1000	1001	1010	1011
7	11	11	110	111	1100	1101	1110	1111

Figure 8: String concatenation table for power-string operation

Table 4: Compare power-string and powerset operations

#	power-string $\P(X)$	powerset $\mathcal{P}(X)$
0	$T_0 := \emptyset$	$V_0 := \emptyset$
1	$T_1 := \P(T_0)$	$V_1 := \mathcal{P}(V_0)$
...	...	...
i	$T_{i+1} := \P(T_i)$	$V_{i+1} := \mathcal{P}(V_i)$
0	$\lvert T_0\rvert = 1$	$\lvert V_0\rvert = 1$
1	$\lvert T_1\rvert = 3$	$\lvert V_1\rvert = 2$
2	$\lvert T_2\rvert = 7$	$\lvert V_2\rvert = 4$
3	$\lvert T_3\rvert = 31$	$\lvert V_3\rvert = 16$
4	$\lvert T_4\rvert = 511$	$\lvert V_4\rvert = 256$
5	$\lvert T_5\rvert = 131071$	$\lvert V_5\rvert = 65536$
6	$\lvert T_6\rvert = 8589934591$	$\lvert V_6\rvert = 4294967296$
...	...	...
$i > 0$	$\lvert T_i\rvert = 2^{2^i} - 1$	$\lvert V_i\rvert = 2^{2^{i-1}}$
...	...	...

A.5 Power-string example in python

```
"""Implement power-string and string concatenation."""


def cdot(a, b):
    """Concatenation of two strings."""
    return f"{a}{b}"


def K(X=None):
    """
    String concatenation - produce a table `K(X) := X * X`.

    Return a product table T with the results of string
    concatenation K."""
    if X is None:
        X = ["", "0", "1"]
    T = list()
    for i in range(len(X)):
        T.append(["" for k in range(len(X))])
        for j in range(len(X)):
            T[i][j] = cdot(X[i], X[j])
    return T


def uniq(X):
    """Return set as sorted list with unique items."""
    X = list(set(X))
    X.sort()
    X = list(sorted(X, key=len))
```

```
    return X


def Splt(c, S):
    """
    String split operation.

    Return a set of all strings that can be split from S by symbol
     c.

    Example:
        > Splt("","0111")
        ['', '0', '1', '01', '11', '011', '111', '0111']
    """
    if c == "":
        R = list()
        for i in range(len(S)):
            for j in range(len(S)):
                R.append(S[i:j + 1])
    else:
        raise NotImplemented(
            "Split over non-empty sub-string is not implemented.")
    return uniq(R)


def flat(T):
    """Return a union over T elements as a flat list."""
    return uniq([x for row in T for x in row])


def PwrStr(X=None):
    """
    Power string.

    Example:
    > X1 = PwrStr()
    ['', '0', '1', '00', '01', '10', '11']

    > X2 = PwrStr(X1)
    ['', '0', '1',
     '00', '01', '10', '11',
     '000', '001', '010', '011',
     '100', '101', '110', '111',
     '0000', '0001', '0010', '0011',
     '0100', '0101', '0110', '0111',
     '1000', '1001', '1010', '1011',
     '1100', '1101', '1110', '1111']
    """
    X = flat(K(X))
    return X
```

appendix/pwr–str.py

A.6 Set versus String operations

Table 5: Compare set and string operations

set operations	**comment**	**string operations**
x is a *member of* y: $x \in y$	is equivalent to	x is a *substring of* y: $x \ \varepsilon \ y$
y is a *subset of* z: $y \subset z$ or equivalently $\forall x.[x \in y \implies x \in z]$	is equivalent to	y is *included in* z: $y \Subset z$ or equivalently $\forall x.[x \ \varepsilon \ y \implies x \ \varepsilon \ z]$ or equivalently $\forall x.[x \in \S(y) \implies x \in \S(z)]$
s is a *union of* c: $s = \bigcup c$	*set union* is equivalent to *class of strings union*, but not to the *join* which is the opposite of *split*	$[j]$ is a *join of* z: $[j] = \mathbb{J}_{s_1,s_2} z$ or equivalently $[j]$ is an equivalence class, s.t. $\forall j \in [j]$: $\forall x, y.[x, y \in \S_{s_2}(z) \implies x \cdot s_1 \cdot y \ \varepsilon \ j]$, where $s_1, s_2 \in \Upsilon$ are separators
y is a *powerset of* x: $y = \mathcal{P}(x)$	is equivalent to	y is a *power-string of* x: $y = ¶(x)$

A.7 Examples of recursive Kuratowski notation

Given the recursive version of the Kuratowski notation as provided in *Definition 119*, let us illustrate how it encodes string concatenation to represent strings as sets.

Table 6: String concatenation encoded as sets

Tuple/String	**Tuple/String Encoding**	**Kuratowski Notation**
x = (a,b)	"ab"	$\{\{a\},\{a,b\}\}$
y = (c,d)	"cd"	$\{\{c\},\{c,d\}\}$
z = x·y	"abcd"	$\{\{\{\{a\},\{a,b\}\},\{c\}\},$ $\{\{\{a\},\{a,b\}\},\{c\}\},d\}\}$

Again, note that the complexity of the notation for "*abcd*" arises from the recursive structure of the *Definition 119*. The more elements in the sequence, the more nested the notation becomes.

A.8 Acknowledgements

Significant part of the paper is written in informal style [343] with genuine intention to make the discussion a bit more accessible to the wider audience. The disadvantage of this approach is that it often comes at cost of discussing trivialities or even lack of expected rigor in claims. So all of this obviously needs further careful verification and improvement to avoid any potential confusion[344]. However, author firmly believes that all the presented results are overall correct and will contribute to novel developments in the respective fields.

Much love, credit and gratitude goes to all cited sources which have been both inspiring and insightful. This also includes many kinds of online discussion forums and resources on mathematics such as wikipedia and youtube, that have not been cited well enough. Special thanks to the communities of math.stackexchange and mathoverflow forums with all the helpful posts on basic results in Set Theory.

Finally, sincere sense of gratitude goes to a transitive set of all mathematicians - dead, living or future ones. Author wants to thank the audience for their patience. Comments or remarks are welcomed per email correspondence.

[343] including a lot of footnotes and references for non-specialists

[344] please expect version updates of this paper