Incremental computation of the set of period sets

Eric Rivals

Incremental computation of the set of period sets

Eric Rivals[1] [0000−0003−3791−3973]

LIRMM, Université Montpellier, CNRS, Montpellier, France `rivals@lirmm.fr`

Abstract. Overlaps between words are crucial in many areas of computer science, such as code design, stringology, and bioinformatics. A self overlapping word is characterized by its periods and borders. A period of a word u is the starting position of a suffix of u that is also a prefix u, and such a suffix is called a border. Each word of length, say $n > 0$, has a set of periods, but not all combinations of integers are sets of periods. Computing the period set of a word u takes linear time in the length of u. We address the question of computing, the set, denoted Γ_n, of all period sets of words of length n. Although period sets have been characterized, there is no formula to compute the cardinality of Γ_n (which is exponential in n), and the known dynamic programming algorithm to enumerate Γ_n suffers from its space complexity. We present an incremental approach to compute Γ_n from Γ_{n-1}, which reduces the space complexity, and then a constructive certification algorithm useful for verification purposes. The incremental approach defines a parental relation between sets in Γ_{n-1} and Γ_n, enabling one to investigate the dynamics of period sets, and their intriguing statistical properties. Moreover, the period set of a word u is key for computing the absence probability of u in random texts. Thus, knowing Γ_n is useful to assess the significance of word statistics, such as the number of missing words in a random text.

Keywords: string · overlap · period · autocorrelation · combinatorics

1 Introduction

Considering finite words, we say that a word u overlaps a word v if a suffix of u equals a prefix of v of the same length. A pair of words u, v can have several overlaps of different lengths. For instance, over the alphabet $\{a, b\}$, let $u := ababba$ and $v := abbabb$, then u overlaps v with the suffix-prefix $abba$, and with a. It appears that the longest overlap contains all other overlaps: to find all overlaps from u to v, it suffices to study the overlaps of $abba$ with itself.

For a word u, a suffix that equals a prefix of u is called a *border*, and the length of u minus the length of a border, is called a *period*. Computing all self-overlaps of a word u is computing all its borders or all its periods, which can be done in linear time (see the algorithm for computing the border array in [24, Chap. 1]). This well-studied problem was also solved for preprocessing the pattern in the seminal Knuth-Morris-Pratt pattern matching algorithm [10]): the borders serve to optimally shift the window along the text when seeking the pattern. For instance the word *ababaaba* has length $n = 8$ and set of periods: $\{0, 5, 7\}$ (zero being the trivial period - the whole word matches itself).

One can easily see that distinct words of the same length can share the same set of periods, even if one forbids a permutation of the alphabet. For a word u, let us denote

by $P(u)$ its period set (which we abbreviate by PS). In this work, we investigate algorithms to enumerate all possible period sets for any words of a given length n. This set is denoted Γ_n for $n > 0$ and is non trivial if the alphabet contains at least two symbols. Brute force enumeration can consider all possible words of length n and compute their period set, but this approach becomes computationally unaffordable for $n > 30$. Currently, only a dynamic programming algorithm exists to enumerate Γ_n, but it suffers from high space complexity [22].

The notion of overlap in strings is crucial in many areas and applications, among others: combinatorics, bioinformatics, code design, or string algorithms. Interest in Γ_n sparkled mostly in the 80's, when researchers started to evaluate the average behavior of pattern matching algorithms, or that of filtering strategies for sequence alignment, text comparisons or clustering. A powerful filtering when comparing two texts, is to list their k-long substrings (a.k.a. k-mers), for appropriate values of k, and then compute e.g. a Jaccard distance between their set of k-mers, to see whether the two texts are similar enough to warrant a costly alignment procedure [27].

In a different area, testing Pseudo-Random Number Generators can also be translated into a question on vocabulary statistics. Indeed, in random real numbers written as sequence of digits, all substrings of a given length, say k, should ideally have an almost equal number of occurrences (i.e., should not significantly deviate from the theoretical expectation). Empirical tests, named *Monkey Tests*, were developed for such generators [14,17,11]. It turns out that the absence probability of a word/string in a random text is essentially controlled by the period set of the word [7]. Hence, the need for enumerating Γ_n appears in diverse domains of the literature [18,19].

In network communication, the so-called prefix-free, bifix-free, and cross-bifix-free codes are used for synchronization purposes. Their design require to select a set of words that are mutually non overlapping – a long studied topic. Nielsen published in 1973 a construction algorithm [15] together with a note on the expected duration of a search for a fixed pattern in random data [16], thereby linking explicitly pattern matching and code design. Improved algorithms for such code design were recently published [2,1].

Many aspects of periodicity of words were and are still extensively studied in combinatorics, e.g. [4,6]. For instance, the periods of random words were investigated in [9] and the Fine and Wilf (FW) Theorem was generalized to three or more words [3]. Algorithms for constructing extremal FW-words relative to a subset of periods were gradually improved in [25,26], a question that is related to the ones we investigate in Section 5. Last, the sequence formed by the cardinality of Γ_n, which is denoted κ_n, has an entry in the OEIS, and some known lower and upper bounds [7,22], which helped closing a conjecture related to κ_n in 2023 [23].

Plan. In Section 2, we introduce a notation, definitions, and review some known results. In Section 3, we propose an incremental approach to compute Γ_n from Γ_{n-1}, which uses $O(n)$ memory. In Section 4, we give an algorithm to build a word that is a witness for a period set. In Section 5, we propose to explore the dynamics of period sets when n increases, before exploring the distribution of period sets according to their basic period and concluding in Section 6.

2   Related works, notation and preliminary results.

2.1   Notation

Here we introduce a notation and basic definitions.

For two integers $p, q \in \mathbb{N}_{>0}$, we denote the fact that p divides q by $p \mid q$ and the opposite by $p \nmid q$. We consider that strings and arrays are indexed from 0. We use $=$ to denote equality, and $:=$ to denote a definition. The cardinality of a discrete set X is denoted by $\sharp(X)$.

An *alphabet* Σ is a finite set of *letters*. A finite sequence of elements of Σ is called a *word* or a *string*. The set of all words over Σ is denoted by $\Sigma^{\star}$, and ε denotes the empty word. For a word x, $|x|$ denotes the *length* of x. Let n be an integer. The set of all words of length n is denoted by Σ^n. Given two words x and y, we denote by $x.y$ the *concatenation* of x and y. For a word w, we define the powers of w inductively: $w^0 := \varepsilon$ and for any $n > 0$, $w^n := w.w^{n-1}$. A word u is *primitive* if there exists no word v such that $u = v^k$ with $k \geq 2$.

Let $u := u[0..n-1] \in \Sigma^n$. For any $0 \leq i \leq j \leq n-1$, we denote by $u[i-1]$ the ith symbol in u, and by $u[i..j]$, the substring starting at position i and ending at position j. In particular, $u[0..j]$ denotes a prefix and $u[i..n-1]$ a suffix of u.

Let $x, y \in \Sigma^*$ and let j be an integer such that $0 \leq j \leq |x|$ and $j \leq |y|$. If $x[n-j..|x|-1] = y[0..j-1]$, then the *merge* of x and y with offset j, which is denoted by $x \oplus_j y$, is defined as the concatenation of $x[0..n-j-1]$ with y. I.e., $x \oplus_j y := x[0..n-j-1].y$.

2.2   Periods, period set, and the set of period sets

Let us first define the concepts of period, period set, basic period, and set of period sets, and then recall some useful results.

Definition 1 (Period/border). *The string $u = u[0..n-1]$ has period $p \in \{0, 1, \ldots, n-1\}$ if and only if $u[0..n-p-1] = u[p..n-1]$, i.e. for all $0 \leq i \leq n-p-1$, we have $u[i] = u[i+p]$. Moreover, we consider that $p = 0$ is a period of any string of length n. The substring $u[p..n-1]$ is called a* border.

Zero is also called the trivial period. The *period set* of a string u is the set of all its periods and is denoted by $P(u)$. Formally, for any length $n > 0$, any word u in Σ^n, $P(u) := \{p \in \mathbb{N} : p \text{ is a period in } u\}$. The *weight* of a period set is its cardinality. The smallest non-zero period of u is called its *basic period*. When $P(u) = \{0\}$, we consider that its basic period is the string length n.

Note that two definitions of period set exist in the litterature: the one above, also used in [7] (even if the set was encoded in a binary string called autocorrelation), where $P(u)$ is a subset of $\{0, 1, \ldots, n-1\}$, and the one from [13] in which it equals $P(u) \cup \{n\}$ (where the length n is included in $P(u)$). We will use the first one.

Let $\Gamma_n := \{Q \subset \{0, 1, \ldots, n-1\} \mid \exists u \in \Sigma^n : Q = P(u)\}$ be the set of all period sets of strings of length n. We denote its cardinality by κ_n. The period sets in Γ_n can be partitioned according to their basic period; thus, for $0 \leq p < n$, we denote by $\Gamma_{n,p}$ the subset of period sets whose basic period is p, and by $\kappa_{n,p}$ the cardinality of this subset. A surprising result from Guibas and Odlyzko's characterization of period sets [7] is the

alphabet independence of Γ_n: Any alphabet of size at least two gives rise to the same set of period sets, i.e., to Γ_n. In the sequel, we assume $\sharp(\Sigma) > 1$. Circa 20 years later, Halava et al. gave a simpler proof of the alphabet independence result by solving the following question [8]. For any period set Q of Γ_n, let v be a word over an alphabet Σ with $\sharp(\Sigma) > 1$ such that $P(v) = Q$; then compute a binary word u such that $P(u) = Q$. Indeed, they gave a linear time algorithm to compute such a word u from v.

For the most important properties on periods, we refer the reader to [7,12,13] and Appendix F.1. Below we recall the two characterizations of period sets from [7] and from [13], and state the version for strings of the famous Fine and Wilf (FW) Theorem [5]. We refer the reader to [22,23] for more properties on period sets and on Γ_n.

2.3   Characterizations of period sets

For a given word length $n > 0$, the question of characterization is: among all possibles subsets of $\{0, 1, \ldots, n-1\}$, recognize those that are period sets for at least one word of length n.

In 1981, Guibas and Odlyzko (GO for short) proved a double characterization of period sets: one is based on two rules, the Forward Propagation Rule (FPR) and the Backward Propagation Rule (BPR), the second is the recursive predicate Ξ (which is given in extenso in Appendix F.2).

First, let us formulate the FPR and BPR in terms of sets (rather than in term of binary vector as in [7]). Let P a subset of $\{0, 1, \ldots, n-1\}$.

Definition 2 (FPR). *P satisfies the FPR iff for all pairs p, q in P satisfying $0 \leq p < q < n$, it follows that $p + i(q - p) \in P$ for all $i = 2, \ldots, \lfloor (n-p)/(q-p) \rfloor$.*

Definition 3 (BPR). *P satisfies the BPR iff for all pairs p, q in P satisfying $0 \leq p < q < 2p$, and $(2p - q) \notin P$, it follows that $p - i(q - p) \notin P$ for all $i = 2, \ldots, \min(\lfloor p/(q-p) \rfloor, \lfloor (n-p)/(q-p) \rfloor)$.*

We can now state GO's characterization theorem

Theorem 1. *Let P a subset of $\{0, 1, \ldots, n-1\}$. The four following statements are equivalent: (1) P is the period set of a binary word of length n.*
(2) P is the period set of a word of length n.
(3) Zero belongs to P and P satisfies the forward and backward propagation rules.
(4) P satisfies predicate Ξ.

The equivalence of (1) and (2) yields the abovementioned alphabet independence of Γ_n. The authors also noticed that the FPR and BPR are local properties [7, Lemma 3.1], which we use later on. Finally, predicate Ξ is a recursive procedure to check whether P belongs to Γ_n in $O(n)$ time: it considers two cases depending on whether the basic period is $\leq \lfloor n/2 \rfloor$ (case a) or not (case b) (cf. Appendix F.2).

Besides this characterization, [7] studied the set of distinct strings over a given alphabet that share the same period set (a.k.a. its *population*). They also reported values of κ_n, the cardinality of Γ_n, for $n < 55$, exhibited lower and upper bounds for $\log(\kappa_n)/\log(n)^2$, and conjectured its convergence when $n \to \infty$. Much later, improved lower and upper bounds were provided and served to close the conjecture [22,23].

An alternative characterization of PS appears in 2002 as Theorem 8.1.11 in [13, Chap. 8]; it comprises four equivalent statements, of which the first three "are proved in [7]" (see notes in [13, p. 310]; the first two are identical to (1) and (2) in Theorem 1). We thus recall only statement (iv) of this characterization in the next theorem.

Theorem 2 ([13]). *Let $P := \{0 = p_0 < p_1 < \ldots < p_s = n\}$ be a set of integers and let $d_h := p_h - p_{h-1}$ for $1 \leq h \leq s$. Then P is a period set (i.e., $P \in \Gamma_n$) if and only if for each h such that $d_h + p_h \leq n$, one has: (a) $p_h + d_h \in P$ and (b) if $d_h = kd_{h+1}$ for some integer k, then $k = 1$.*

Note that in this formulation n belongs to P (see our remark about two definitions of period sets), but as when $h = s$ one has $d_s + p_s > n$, one sees that conditions (a) and (b) are in fact only required for $1 \leq h < s$.

Given a period set $P(w)$ of some word w, the proof shows that there exists a binary word having the period set $P(w)$. It is a proof of existence, it is not constructive, but the authors also give an example on how to build a binary string for $Q := \{0, 11, 14, 17, 18\}$ knowing that Q is the period set of the word $w = abcabcadefgabcabca$. Knowing that $Q \in \Gamma_n$, an algorithm for solving this problem was given by [8].

In this work, we address a different question, termed *constructive certification*: Given Q a subset of $\{0, 1, \ldots, n-1\}$, build a (binary) string u such that $P(u) = Q$ iff $Q \in \Gamma_n$, or return the empty string otherwise.

2.4   Additional properties of periods

In [7], the authors give a version for strings of the famous Fine and Wilf Theorem [5], a.k.a. the periodicity lemma. A nice proof was provided by Halava and colleagues [8].

Theorem 3 (Fine and Wilf). *Let p, q be periods of $u \in \Sigma^n$. If $n \geq p + q - \gcd(p, q)$, then $\gcd(p, q)$ is a period of u.*

We can reformulate Theorem 3 as a condition that must be satisfied by a period set P.

Theorem 4. *Let $P \in \Gamma_n$. Let p, q be periods of P such that $\gcd(p, q) \notin P$, and define $FW(p, q) := p + q - \gcd(p, q)$. Then $FW(p, q)$ must be strictly larger than n.*

We call $FW(p, q)$ the *Fine and Wilf (FW) limit* of (p, q), and the fact that $FW(p, q)$ must be larger than n, the *FW condition*.

2.5   Dynamic programming algorithm to enumerate Γ_n

In 2001, further investigation of Γ_n led to a dynamic programming programming algorithm to enumerate all period sets in Γ_n: it converts the recursive approach of predicate Ξ into a dynamic program that stores all Γ_i for $0 < i \leq \lfloor 2n/3 \rfloor$ [21,22]. With some practical improvements, the range was reduced to $0 < i \leq \lfloor n/2 \rfloor$. However, as κ_n is exponential in n, this induces a large memory usage, which remains a serious drawback. Hence, the quest for memory efficient algorithms. The authors also demonstrated that Γ_n equipped with set inclusion is a lattice, but this did not help to improve Γ_n enumeration.

3   Incremental enumeration of Γ_n

Rationale of the incremental approach

In their seminal work [7], GO manipulate period sets, not as sets, but as a binary strings of length n, where position i is set to 1 if i is a period. For example, the binary encoding of period set $\{0,3,5\}$ for length $n = 6$ is 100101. They call these binary strings correlations, and even autocorrelations to emphasize that it encodes all self-overlaps a string [7, p. 21]. The rule based characterization (statement (3) of Theorem 1) implies a *special substring* property [7]. Indeed, noting the locality of the forward and backward propagation rules, the authors state in Lemma 3.1 in [7]: If a binary string v satisfies the forward and backward propagation rules, then so does any prefix or suffix of v.

As the rule based characterization of period sets – statement (3) of Theorem 1 – also requires that an autocorrelation has its first bit equal to one, or equivalently that zero belongs to a period set, one gets the following theorem ([22, Thm 1.3]):

Theorem 5. *Let v be an autocorrelation of length n. Any substring $v_i \ldots v_j$ of v with $0 \le i \le j < n$ such that $v_i = 1$ is an autocorrelation of length $j - i + 1$.*

Applying Theorem 5 to a prefix of v, one gets for any $n > 0$: The prefix of length $(n-1)$ from an autocorrelation of length n is an autocorrelation of length $(n-1)$. In terms of period sets, this statement can be reformulated as:

Corollary 1. *If P is a period set of Γ_n, then $P \setminus \{n-1\}$ belongs to Γ_{n-1}.*

First, this means that, knowing Γ_n, it is easy to compute Γ_{n-1}. It suffices to consider each element of Γ_n in turn (or in parallel) and to possibly remove the period $(n-1)$ from it (i.e., if $(n-1)$ belongs to it) to obtain an element of Γ_{n-1}. With this procedure one can obtain the same element of Γ_{n-1} twice, and one must keep track of this to avoid redundancy.
Conversely, we get the Lemma that underlies the incremental approach for computing Γ_n:

Lemma 1. *Let Q be a period set of Γ_n. Then Q can only be of two alternative forms: either P or $P \cup \{n-1\}$, for some P in Γ_{n-1}.*

Incremental algorithm framework

Lemma 1 suggests an approach for computing Γ_n using Γ_{n-1}. Consider each P from Γ_{n-1}, and check whether the candidate sets P and $P \cup \{n-1\}$ are period sets of Γ_n. Algorithm 1 presents a generic incremental algorithm for Γ_n, where certify denotes the certification function used. In general, a certification function takes as input n and any subset Q of $\{0, 1, \ldots, n-1\}$, returns True if and only if Q belongs to Γ_n.

The recursive predicate Ξ from [7] (cf. Appendix F.2) is indeed a certification function: it does exactly what is required for any subset of $\{0, 1, \ldots, n-1\}$, in $O(n)$ time [7]. Thus, using predicate Ξ, Algorithm 1 correctly computes Γ_n from Γ_{n-1}. We will discuss alternative certification functions below.

Besides its simplicity, the main advantage of Algorithm 1, compared to the dynamic programming enumeration algorithm of [21], is its space complexity. Here, the computation considers each period set P from Γ_{n-1} in turn (and independently from the

---

Algorithm 1: IncrementalGamma(length $n > 1$; set Γ_{n-1})

Output: Γ_n: the set of period sets for length n;

1 $G := \emptyset$;   // G: variable to store Γ_n
2 **for** *all $P \in \Gamma(n-1)$* **do**
3 | **if** *certify(P, n)* **then** insert P in G ;
4 | $Q := P \cup \{n-1\}$ // build extension P with period $n-1$;
5 | **if** *certify(Q, n)* **then** insert Q in G ;

6 **return** G;

---

others), executes twice the certification function for P and Q; this implies that the memory required, besides storage of P, Q, is the one used by the certification function. With the predicate Ξ, it is linear in n, so $O(n)$ space. The time complexity is proportional to κ_n (i.e., the cardinality of Γ_n) times the running time of the certification function, which yields the following theorem. Of course, the set Γ_{n-1} must be available on disk space before hand.

Theorem 6. *Provided that Γ_{n-1} is available on external memory, then*

1. *Algorithm 1 using any certification function correctly computes Γ_n from Γ_{n-1}.*
2. *Using the predicate Ξ as certification function, it runs in $O(n\kappa_n)$ time and $O(n)$ space.*

Moreover, it is worth noticing that Algorithm 1 is embarrassingly parallelizable. Note that the output contains κ_n period sets, whose cardinality is bounded by n and sometimes equal n. So, the output size is bounded by $n\kappa_n$.

Alternative certification functions

In our incremental setup, that is when computing Γ_n using Γ_{n-1}, we know that P belongs to Γ_{n-1} (line 2 in Algorithm 1). Hence, the candidate sets, denoted P and Q, are not **any subset** of $\{0, 1, \ldots, n-1\}$, but already satisfy some constraints for length $(n-1)$. Therefore, finding alternative certification functions is interesting. Here, we discuss two alternative functions, and in Section 4, we exhibit a constructive certification algorithm, which not only certifies a candidate set, but also computes a witness, i.e., a word whose period set is the candidate set, only if the answer is positive.

To simplify Algorithm 1 by improving how certifications are done for the candidate sets P and $Q := P \cup \{n-1\}$, we can take advantage of two facts. First, the only period that can be added is $n-1$. This limits the cases for which we need to check some conditions. Second, P and Q are not independent, and if $n-1$ is compulsory at length n, because it is generated by the FPR from smaller periods, then only candidate set $Q := P \cup \{n-1\}$ may belong to Γ_n, but not P.

Beides Predicate Ξ, a second certification function, we call it *rule based*, exploits statement (3) of Theorem 1 and uses procedures to verify if the forward and backward propagation rules are satisfied. Algorithm 3 gives the code for computing Γ_n using the rule based certification (see explainations in Appendix B.1).

A third certification approach, which exploits the characterization of Theorem 2, can perform the certification of both P and Q in $O(\sharp(P))$ time. Some details are also given in Appendix B.2).

4   Constructive certification of a period set

Let Q be subset of $\{0, 1, \ldots, n-1\}$. We say that a word u *realizes Q* if $P(u) = Q$.

The certification functions used in Section 3 yield a True/False answer, but no witness for a period set. As there is no ressources providing Γ_n for many word lengths, checking the output of an enumeration algorithm, remains difficult. Hence, we need a constructive certification function, which given length $n > 0$ and set Q, provides a word realizing Q if only if $Q \in \Gamma_n$, and the empty string otherwise. Given the alphabet independence of Γ_n, we restrict the search to binary words. We present an algorithm called *binary realization* (see Algorithm 2) solving this question, and demonstrate its linear complexity.

Using Algorithm 2 as a certification function in Algorithm 1, for each $R \in \Gamma_n$, we get a word u realizing R. Then, computing the period set of u allows us to check that $P(u) = R$. Let us define the notion of *nested set*.

Definition 4. *Let $n > 0$, P be a subset of $\{0, 1, \ldots, n-1\}$, and q be an element of P. We denote by P_q, the* nested set of P starting at period q:

$$P_q := \{(r - q) \text{ for each } r \in P \text{ such that } r \geq q\}.$$

By construction P_q starts with 0; moreover, if we choose $q = 0$ then $P_q = P$.

4.1   Binary realization of a subset of $\{0, 1, \ldots, n-1\}$

Algorithm 2 computes a word u that realizes a set P for length $n > 0$, or returns the empty word ε if P is not a valid period set of Γ_n. For legibility, the preliminary checks on P are not written in Algorithm 2: they include checking that P is a subset of $[0, \ldots, n-1]$, is ordered, and has zero as first period. The word u is written over the alphabet $\{\mathtt{a}, \mathtt{b}\}$.

The algorithm considers elements of P backwards, starting with largest integer first, since P is ordered. At each execution of the for loop, it considers the current integer $P[i]$ as a period and builds a suffix of u of length $n - P[i]$ (variable lg). In fact, it considers a potential larger and larger nested sets, and computes a suffix of u for this length. At the end of the for loop, the variable *suffix* contains a string of length lg realizing the nested set. Note that algorithm uses three variables (whose names start with *prev*) to store the length, the inner period, and the suffix obtained with the previous period.

The base case is processed before the loop and consider the nested set for $P[k-1] = max(P)$ for the length $n - max(P)$ without any period. Hence, the suffix $\mathtt{a}.\mathtt{b}^{(\text{prevLg -1})}$ is a realization for nested set $\{0\}$ for length $n - max(P)$.

In the for loop the key variable is the *innerPeriod*, which equals the offset $P[i+1] - P[i]$, which is the basic period of the current nested set. If innerPeriod $<$ prevIP then the FPR is violated and the algorithm returns ε (line 7). Two cases are considered depending on whether the current length is smaller twice the previous length (case 1) or not (case 2). Because of the notion of period, the suffix must start and end with a copy of *prevSuffix*. The construction of the suffix depends on the case. In case 1, the two copies of *prevSuffix* are concatenated or overlap themselves, and some additional conditions are required (line 9). These conditions are dictated by the characterization of period set from [7] (see the predicate Ξ in Appendix F.2). Whenever one is not

---

Algorithm 2: Binary Realization

Input: $n > 0$: integer; P: a subset of $[0, 1, \ldots, n-1]$ including 0, in a sorted array
Output: a **binary** string realizing P at length n xor the empty string otherwise;

1   $k := \sharp(P)$;   `// k: cardinality of P`

2   **if** $k = 1$ **then return** $\texttt{a.b}^{(\texttt{n}-1)}$ `// trivial case where` $P = \{0\}$`;`

    `// processing the largest period and init. variables`

3   prevLg $:= n - P[k-1]$; prevIP $:=$ prevLg; prevSuffix $:= \texttt{a.b}^{(\mathrm{prevLg}-1)}$;

4   **for** *i going from $k-2$ to 0* **do**

5   lg $:= n - P[i]$;

6   innerPeriod $:= P[i+1] - P[i]$;

7   **if** *innerPeriod $<$ prevIP* **then return** ε ;

8   **if** $lg \leq 2\times$ *prevLg* **then**   `// condition for case 1`

9   **if** *(innerPeriod $=$ prevIP) OR ((prevIP $\nmid$ innerPeriod) AND ((innerPeriod $=$ prevLg) OR (prevSuffix has period innerPeriod)))* **then**

             `// suffix := a prefix of prevSuffix concat. with prevSuffix`

10   suffix $:=$ prevSuffix$[0..$innerPeriod$-1]$. prevSuffix;

11   **else return** ε `// invalid case for length lg;`

12   **else**   `// condition for case 2`

        `// suffix := prevSuffix newsymbols prevSuffix`

13   m $:=$ lg $-2\times$ prevLg;

14   newPrefix $:=$ prevSuffix $.\texttt{a}^{\texttt{m}}$;

15   **if** *newPrefix is not primitive* **then**

16   newPrefix $:=$ prevSuffix $.\texttt{a}^{(\texttt{m}-1)}\texttt{b}$;

        `// Invariant: newPrefix is primitive`

17   suffix $:=$ newPrefix . prevSuffix;

      `// update variables`

18   prevLg $:=$ lg; prevIP $:=$ innerPeriod; prevSuffix $:=$ suffix;

19   **return** *suffix*;

---

satisfied, Algorithm 2 returns the empty word as expected. In case 2, the two copies of *prevSuffix* must be separated by *m* additional symbols (to be determined). One builds a *newPrefix* that starts with *prevSuffix* followed by $\texttt{a}^m$, and one checks whether *newPrefix* is primitive. This *newPrefix* is the part that ensures the suffix will have *innerPeriod* as a period. The primitivity is required, since *newPrefix* may have a proper period, but this period shall not divide *innerPeriod*. If primitivity is not satisfied, then changing the last symbol $\texttt{a}$ of *newPrefix* by $\texttt{b}$ will make it primitive. This is enforced by Lemma 3 from [8], which states that for any binary word w, $w\texttt{a}$ or $w\texttt{b}$ is primitive. So, we know that at least one of the two forms of *newPrefix* is primitive as necessary. It can be that both are primitive and suitable. Finally, we build the current *suffix* by concatenating *newPrefix* with *prevSuffix*.

Complexity First, in case 1, checking the condition "*prevSuffix has period inner-Period*" can be done in linear time in $|prevSuffix|$ (which is $\leq n$). Overall, this can be executed $\sharp(P)$ times. Second, the primitivity test performed in case 2 takes a time

proportional to the length of the string *newPrefix*. However, the sum of these lengths, for all iterations of the loop, is bounded by n. Other instructions of the for loop take constant time. Overall, the time complexity of Algorithm 2 by $O(\sharp(P) \times n)$. However, when Algorithm 2 is plugged in Algorithm 1 it processes special instances: either P or $Q := P \cup \{n-1\}$, with $P \in \Gamma_{n-1}$. Then, the time taken by all verifications of condition "*prevSuffix has period innerPeriod*" for all cases 1 is bounded by n, due to the properties of periods that generate more than two repetitions in a string (see Lemma 4 and Lemma 2 from [8]). For the instances processed in Algorithm 1, the time complexity of Algorithm 2 is $O(\sharp(P) + n)$ or $O(n)$.

Remark we can modify Algorithm 2 to build, instead of a binary word, a realizing word that maximizes the number of distinct symbols used in it.

4.2 Examples of binary realization

We consider the case of the period set $P := \{0, 3, 6, 8\}$ from Γ_9, which does not belong to Γ_{10}, and show the traces of execution for both lengths $n := 9$ and $n := 10$. The table below shows the trace for $n := 9$. The operator $\oplus_j$ merges the two strings with an offset of length j if the corresponding prefix and suffix are equal, for any appropriate integer j. So when $n = 9$ and $i = 0$, the merge $v := w \oplus_3 w$ with $w = abaaba$ is feasible since w has period 3. When $n = 10$, the merge $w := y \oplus_3 y$ with $y = abab$ is not possible since $a \neq b$. The trace for $n = 10$ is in Appendix C.

period	length	inner period	case	suffix	valid
8	9-8 = 1	9-8 = 1	2	$z := a$	true
6	9-6 = 3	8-6 = 2	2	$y := z.b.z = aba$	true
3	9-3 = 6	6-3 = 3	2	$w := y.y = abaaba$	true
0	9-0 = 9	3-0 = 3	1	$v := w \oplus_3 w = (aba)^3$	true

5   Fate and dynamics of period sets

The incremental algorithm and Lemma 1 induces a *parental* relationship between sets in Γ_{n-1} and Γ_n. Any period set P occurs first in $\Gamma_{max(P)+1}$. When the length increases from $n-1$ to n, a period set P in Γ_{n-1} faces three cases: (1) P may remain as is in Γ_n, (2) P has an *extension* with period $n-1$ (i.e. $P \cup \{n-1\} \in \Gamma_n$), or (3) P *dies*, i.e., is neither in case (1) nor case (2). Note that cases (1) and (2) are not exclusive from each other. Thus, P at length $n-1$ can be the parent of at most two period sets in in Γ_n. One can thus investigate the dynamics of period sets when the word length n increases starting with $n = 1$.

Example 1. For instance, $\{0, 3, 6\}$ is born in Γ_7 and still belongs to Γ_8 and Γ_9; its extension $\{0, 3, 6, 7\}$ also belongs to Γ_8. From a dynamic view point, $\{0, 3, 6\}$ is the *parent* of both $\{0, 3, 6\}$ and $\{0, 3, 6, 7\}$ in Γ_8. On the contrary, $\{0, 4, 6\}$ belongs Γ_7, but dies at $n = 8$, since the pair $(4, 6)$ satisfies the FW condition at $n = 8$ (i.e. it would require to add $gcd(4, 6)$ as a new basic period. Last, $\{0, 2, 4, 6\}$ belongs to Γ_7, Γ_8, and generates $\{0, 2, 4, 6, 8\}$ in Γ_9 because the extension with period 8 is required by the FPR, but it never dies.

With these definitions at hand, consider the parental relation when the word length n increases starting from $\Gamma_1 := \{\{0\}\}$: it forms an infinite directed tree whose nodes are period sets and arcs represent the parental relation. The tree is rooted with period set $\{0\}$, is structured in successive layers corresponding to PS for successive word lengths, and the outdegree of each node can be 0, 1, or 2. When a period set dies, a branch of the tree becomes a dead end.

For each period set P, one may ask at how many consecutive word lengths it exists. We show that its fate depends only on the periods in P, and define below two variables that give the limit of its existence, and give algorithms to compute these.

Definition 5 (Recursive FW limit and next extension). *Let $n \in \mathbb{N}_{>0}$ and $P \in \Gamma_n$. The recursive FW limit of P, denoted $rfw(P)$, is the smallest length k at which any two periods of P satisfies the FW condition, and infinity otherwise. The next extension of P, denoted $e(P)$, is $\min_{j>max(P)}\{j = 2p_j - p_i \text{ with } : p_j < p_i \in P\}$ if $\sharp(P) > 1$, and $e(\{0\}) = +\infty$ otherwise.*

If $\sharp(P) > 1$, as the word length increases, the current periods of P will induce, by the FPR, new compulsory periods larger than $max(P)$; the minimum among those is $e(P)$, meaning that when the word length reaches $e(P)+1$ then P will necessarily be extended (i.e., possiblity (2) but not (1)), unless P dies. When the word length reaches $rfw(P)$, then at least one pair of periods will satisfy the FW condition, and thus P must die at that length. Of course, if $\sharp(P) = 1$ or all non zero periods in P are multiple of its basic period, then $rfw(P) = +\infty$. Thus, any given period set P ceases to belong to *Gamma_l* if $l := min(e(P)+1, rfw(P))$ (i.e. case (1) is forbidden), it dies at length $rfw(P)$, and it is extended at length $e(P)+1$ if $e(P)+1 < rfw(P)$.

A challenging open question is to compute how many sets dies at length n, without enumerating Γ_n. The sequence of the numbers of dying period sets at length n is $0, 0, 0, 0, 0, 1, 1, 2, 1, 3, 2, 8$ for $n := [1, 12]$.

6 Conclusion and exploration of Γ_n: distributions of period sets with respect to basic period and to weight

The key element of a period set is its basic period, which defines the first level of periodicity in a word. How period sets in Γ_n are distributed according to their basic period is non trivial. Enumerating Γ_n allows inspecting this distribution. The left plot in Figure 1 displays $\kappa_{n,p}$, the counts of period sets for all possible basic periods p, in Γ_{60}. In predicate Ξ [7], one separates period sets depending on the basic period being $\leq \lfloor n/2 \rfloor$ (case **a**) or larger than $\lfloor n/2 \rfloor$ (case **b**). The smooth decrease of counts beyond $\lfloor n/2 \rfloor$ is explained by the combinatorial property that links number of period sets in case **b** and the number of binary partitions of an integer (see Lemma 5.8 in [22]). However, the distribution of counts for period sets in case **a**, still requires some investigation and statistical modeling. Here, we observe that between basic period 1 and 30, $\kappa_{n,p}$ reaches local maxima when p divides the string length n (e.g. see the peaks at $p = 10, 12, 15, 20,$ or 30, which all correspond to period sets of case **a**).

Other works have investigated combinatorial parameters that control the number of periods of a word [6]. Thanks to enumeration of Γ_n, one can study the distribution of
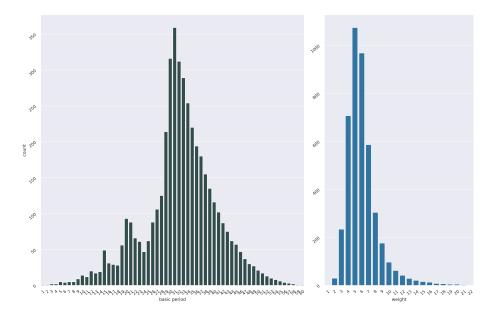
Fig. 1: Distribution in Γ_{60} of the number of period sets by basic period (left) and by weight (right), for string length of $n := 60$. Beyond basic period 30, the counts decrease smoothly with the basic period. Between basic period 1 and 30 the counts increase to a local maximum when the basic period reaches $\lfloor n/x \rfloor$ for $1 < x \leq 12 =$ (e.g. basic periods 10, 12, 15, 20, 30). The distribution by weight (right) is limited to weight below 22; it is unimodal and right skewed towards low weights.

period sets with respect to weight and how it evolves with n. The right plot of Figure 1 displays the number of period sets with equal weight (i.e. same number of periods) for $n = 60$. This distribution is right skewed and illustrates the constraints imposed by multiple periods. Similar figures to Fig. 1 for other string lengths are shown in Appendix A.

Conclusion. We provide algorithms to enumerate Γ_n incrementally with low space requirement, with several certification functions, and an algorithm for binary realization of a period set. We define a parental relation between period sets for distinct word lengths (which makes up a tree), and propose a way to study their dynamics as n increases. Many questions remain open – besides that on the number of dying PS mentioned in Section 5. Can one speed up enumeration by exploiting the tree and a dynamic update of the recursive FW limit and next extension of a PS? Can we exploit the tree to unravel how population sizes of PS evolve with n? Extending the notions presented here to generalizations of words, like partial words, degenerate strings, or multidimensional words opens avenues of future work. As seen in Section 5, the number of dying PS is not monotonically increasing in function of n; thus understanding the sequences of κ_n and $\kappa_{n,p}$ is both stimulating and challenging (see also Figures 1, 2-4).
Ressource: we provide files containing the period sets of Γ_n for $n = 1, \dots, 60$ at [20].

References

1. Bajic, D., Loncar-Turukalo, T.: A simple suboptimal construction of cross-bifix-free codes. Cryptography and Communications **6**(6), 27–37 (2014). https://doi.org/10.1007/s12095-013-0088-8

2. Bilotta, S., Pergola, E., Pinzani, R.: A new approach to cross-bifix-free sets. IEEE Transactions on Information Theory **58**(6), 4058–4063 (2012). https://doi.org/10.1109/TIT.2012.2189479

3. Castelli, M., Mignosi, F., Restivo, A.: Fine and wilf's theorem for three periods and a generalization of sturmian words. Theoretical Computer Science **218**(1), 83–94 (Apr 1999). https://doi.org/10.1016/s0304-3975(98)00251-5, http://dx.doi.org/10.1016/S0304-3975(98)00251-5

4. Ehrenfeucht, A., Silberger, D.: Periodicity and unbordered segments of words. Discrete Mathematics **26**(2), 101–109 (1979). https://doi.org/10.1016/0012-365x(79)90116-x, http://dx.doi.org/10.1016/0012-365X(79)90116-X

5. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proc. Amer. Math. Soc. **16**, 109–114 (1965)

6. Gabric, D., Rampersad, N., Shallit, J.: An inequality for the number of periods in a word. International Journal of Foundations of Computer Science **32**(05), 597–614 (Jun 2021). https://doi.org/10.1142/s0129054121410094

7. Guibas, L.J., Odlyzko, A.M.: Periods in strings. J. of Combinatorial Theory series A **30**, 19–42 (1981). https://doi.org/10.1016/0097-3165(81)90038-8

8. Halava, V., Harju, T., Ilie, L.: Periods and binary words. J. Comb. Theory, Ser. A **89**(2), 298–303 (2000). https://doi.org/10.1006/JCTA.1999.3014, https://doi.org/10.1006/jcta.1999.3014

9. Holub, S., Shallit, J.O.: Periods and borders of random words. In: Ollinger, N., Vollmer, H. (eds.) 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France. LIPIcs, vol. 47, pp. 44:1–44:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.STACS.2016.44

10. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM Journal of Computing **6**, 323–350 (1977)

11. Leopardi, P.: Testing the Tests: Using Random Number Generators to Improve Empirical Tests. In: Ecuyer, P.L., Owen, A.B. (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2008, pp. 501–512. Springer Berlin Heidelberg (2009), https://doi.org/10.1007/978-3-642-04107-5_32

12. Lothaire, M. (ed.): Combinatorics on Words. Cambridge University Press, second edn. (1997)

13. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2005), http://www.cambridge.org/gb/knowledge/isbn/item1172552/?site_locale=en_GB, uRL: http://www-igm.univ-mlv.fr/~berstel/Lothaire/index.html

14. Marsaglia, G., Zaman, A.: Monkey tests for random number generators. Computers and Mathematics with Applications **26**(9), 1–10 (1993)

15. Nielsen, P.: A note on bifix-free sequences (corresp.). IEEE Transactions on Information Theory **19**(5), 704–706 (Sep 1973). https://doi.org/10.1109/tit.1973.1055065, http://dx.doi.org/10.1109/TIT.1973.1055065

16. Nielsen, P.: On the expected duration of a search for a fixed pattern in random data (corresp.). IEEE Transactions on Information Theory **19**(5), 702–704 (Sep 1973). https://doi.org/10.1109/tit.1973.1055064, http://dx.doi.org/10.1109/TIT.1973.1055064

17. Percus, O.E., Whitlock, P.A.: Theory and Application of Marsaglia's Monkey Test for Pseudorandom Number Generators. ACM Transactions on Modeling and Computer Simulation **5**(2), 87–100 (April 1995)

18. Rahmann, S., Rivals, E.: Exact and efficient computation of the expected number of missing and common words in random texts. In: Proc. of CPM 2000. p. 375–387. Springer Berlin Heidelberg (2000). https://doi.org/10.1007/3-540-45123-4_31,

19. Rahmann, S., Rivals, E.: On the distribution of the number of missing words in random texts. Combinatorics, Probability and Computing **12**(01) (Jan 2003). https://doi.org/10.1017/s0963548302005473, http://dx.doi.org/10.1017/S0963548302005473

20. Rivals, E.: Sets of period sets for words of length n. Zenodo (Sept 2024). https://doi.org/10.5281/zenodo.13826260, data set

21. Rivals, E., Rahmann, S.: Combinatorics of periods in strings. In: Proc. of ICALP 2001. p. 615–626. Springer Berlin Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_51,

22. Rivals, E., Rahmann, S.: Combinatorics of periods in strings. Journal of Combinatorial Theory, Series A **104**(1), 95–113 (Oct 2003). https://doi.org/10.1016/s0097-3165(03)00123-7, http://dx.doi.org/10.1016/S0097-3165(03)00123-7

23. Rivals, E., Sweering, M., Wang, P.: Convergence of the Number of Period Sets in Strings. In: Etessami, K., Feige, U., Puppis, G. (eds.) 50th International Colloquium on Automata, Languages, and Programming (ICALP 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 261, pp. 100:1–100:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/https://doi.org/10.4230/LIPIcs.ICALP.2023.100, https://drops.dagstuhl.de/opus/volltexte/2023/18152

24. Smyth, W.F.: Computating Pattern in Strings. Pearson - Addison Wesley (2003)

25. Tijdeman, R., Zamboni, L.: Fine and wilf words for any periods. Indagationes Mathematicae **14**(1), 135–147 (Mar 2003). https://doi.org/10.1016/s0019-3577(03)90076-0, http://dx.doi.org/10.1016/S0019-3577(03)90076-0

26. Tijdeman, R., Zamboni, L.: Fine and wilf words for any periods ii. Theoretical Computer Science **410**(30–32), 3027–3034 (Aug 2009). https://doi.org/10.1016/j.tcs.2009.02.004, http://dx.doi.org/10.1016/j.tcs.2009.02.004

27. Ukkonen, E.: Approximate string-matching with q-grams and maximal matches. Theor. Comp. Sci. **92**(1), 191–211 (Jan 1992)

A   Distributions of the number of period sets by basic period and by weight

Like in Figure 1, we explore how period sets are distributed according to their basic period, and according to their weight for other string lengths. We plot these distributions for $n = 48$, $n = 55$, and $n = 59$ in Figures 2, 3, and 4, respectively. We choose these values because they differ in their number of divisors $48 = 2^4 \times 3$, $55 = 5 \times 11$ and 59 is prime. In essence, both plots for Γ_{48}, Γ_{55}, and Γ_{59} look very similar to those for Γ_{60}. Even for a prime string length, $n = 59$, the distribution of number of period sets in case **a**, shows a maximum at $\lfloor n/2 \rfloor$ and local maxima at $\lfloor n/3 \rfloor$, $\lfloor n/4 \rfloor$ etc.
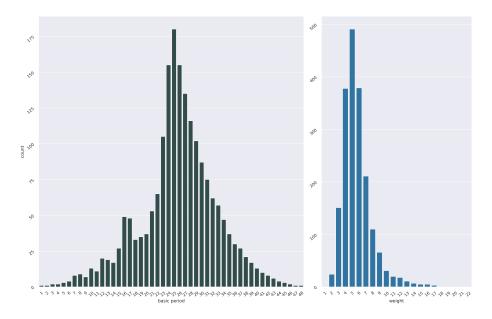

Fig. 2: Distribution in Γ_{48} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 48$.

B   Alternative certification functions and variants of incremental algorithm

B.1   Incremental algorithm with rule based certification

Here, we detail an alternative version of the incremental algorithm, which uses the rule based certification function derived from Theorem 5.1 from [7] (see also below). This is related to subsection 6. Algorithm 3 presents the pseudo-code; it uses two functions
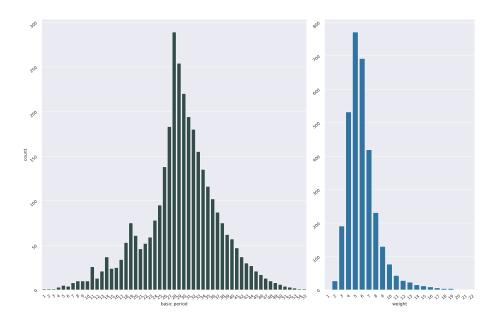
Fig. 3: Distribution in Γ_{55} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 55$.
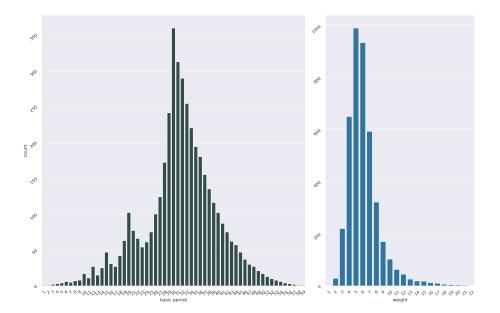

Fig. 4: Distribution in Γ_{59} of the number of period sets by basic period (left) and by weight (right), i.e., for string length of $n := 59$.

named *checkFPR* and *checkBPR*, which check if a set of integers satisfies respectively, the Forward and Backward Propagation Rules.

In our case, as the candidate sets include a period set of Γ_{n-1}, they necessarily satisfy the first condition (i). Regarding the FPR, since P belongs to Γ_{n-1}, P satisfies the FPR up to position $n-2$ included; and thus, only period $n-1$ can be required by the FPR. For each possible pair (p,q) considered in the FPR, we only need to check if the FPR formula yields $(n-1)$. Second, for the same reason, when considering the candidate set $P \cup \{n-1\}$, we are sure that the FPR is satisfied.

Let R be any subset of $\{0,1,\ldots,n-1\}$ containing zero and assuming that R is sorted in increasing order, then we have that checking the FPR and BPR takes $O(n\log(n))$ time.

Algorithm 3 differs from Algorithm 1 in two aspects. First, it can indicate for which reason the candidate set is not a period set if the check fails. Second, it also computes the set of "dying" period sets of Γ_{n-1}, that is the period set of Γ_{n-1} that are not in Γ_n, nor cannot be extended at length n. We will define these notions in Section 5. Of course, dying period sets could also be computed within Algorithm 1, which uses the predicate Ξ (but for simplicity and to avoid redundancy, was not mentioned earlier).

Altogether the time complexity of Algorithm 3 is bounded by $O(n\log(n) \times \kappa_n)$, which may not be optimal.

Definition 6. *A dying period set P is a period set of Γ_{n-1} such that neither P nor $P \cup \{n-1\}$ belong to Γ_n.*

Algorithm 3: IncrementalGamma with rule based certification

Input: $n > 1$: integer; Γ_{n-1}: the set of period sets for length $n-1$
Output: Γ_n: the set of period sets for length n; D: the set of dying PS at length n;

1   $G := \emptyset$;   `// G: variable to store` Γ_n
2   $D := \emptyset$;   `// D: variable to store dying PS`
3   **for** $P \in \Gamma_{n-1}$ **do**
4      $Q := P \cup \{n-1\}$ `// build extension of P with period` $n-1$;
5   **if** *checkFPR(P, n)* **then** `//` $n-1$ `is required by FPR at length n`
6   **if** *checkBPR(Q, n)* **then** insert Q in G;
7   **else** insert P in D `// otherwise P is dying at length n`;
8   **else**
9   **if** *checkBPR(P, n)* **then** insert P in G ;
10   **else**
11   **if** *checkBPR(Q, n)* **then** insert Q in G ;
12   **else** insert P in D `// otherwise P is dying at length n`;

13   **return** G *and* D;

B.2   An alternative combinatorial certification

A third certification function exploits the characterization of Theorem 2, where the necessary conditions are expressed in function of the periods p_h, their differences d_h,

for $0 < h \leq s$, and of n. In the incremental approach, the word size increases from $n-1$ to n. Here we sketch why the certifications of candidate sets P and $Q := P \cup \{n-1\}$ can be combined in $O(n)$ time.

First, from candidate set P containing $s+1$, we can check whether period $n-1$ can be deduced from smaller periods: if there is an h such that $p_h + d_h = n-1$. This can be done in s constant time computations. If yes, then $n-1$ is a compulsory period at length n and thus only Q must be further examined. If not, both P and Q may belong to Γ_n. Doing the above computations, we determine for which indices h the condition $d_h + p_h = n-1$ is satisfied, and only for those we have to check conditions (a) and (b). Indeed, for all h such that $d_h + p_h < n-1$, we know (a) and (b) are already satisfied since $P \in \Gamma_{n-1}$. However, at length n, the period p_s of P becomes n which changes the value d_s from $n-1$ to n. Hence, we must check condition (b), only for $h = s-1$, which is done in constant time.

In the case of Q, the number of periods is now $s+2$, the new period is $p_s = n-1$, d_s has a new value, and $d_{s+1} = 1$, since the trivial period n is included in Q. The condition (a) is then always satisfied. The condition (b) needs to be verified only for h equal to $s-1$ and s, since only d_s and d_{s+1} have changed. This can be done in two constant time computations. Altogether, with this combined certifications of P and Q, the inner for loop of Algorithm 1 takes $O(s)$ time, which we can bound by $O(n)$.

C   Algorithm Binary realization

C.1   Correctness and complexity of the algorithm

Proof. Let us prove that the Algorithm Binary Realization is correct.

Correction of the base case As we process the last period of P, the nested set is $\{0\}$ for length $n - max(P)$. We must build a suffix without period (i.e., whose basic period is its length). Hence, the word $\texttt{a.b}^{(\text{prevLg}-1)}$ is a binary realization for this set.

Correction of the general case. After setting variables lg and *innerPeriod*, we check the condition (*innerPeriod* $<$ *prevIP*). In a period set, the offset $P[i+1] - P[i]$ decreases when i increases. The condition implies the current nested set is invalid, and we return ε as needed. Another way to formulate this: If the condition is satisfied, then *suffix*, which ends with *prevSuffix*, does not satisfy the FPR, meaning that this set is invalid.

The invariant at the start of the for loop is that *prevSuffix* realizes the nested set $P_{P[i+1]}$ and has *prevIP* as basic period. By construction, we know that $lg = prevLg + innerPeriod$. By construction, *suffix* ends with *prevSuffix* and has basic period *innerPeriod*. Thus, by the invariant, *suffix* will realize $P_{P[i]}$.

Case 1 We build *suffix* by concatenating a prefix of *prevSuffix* of length *innerPeriod* with *prevSuffix* (line 10), and we must ensure that *suffix* has basic period *innerPeriod*. Let us consider the conditions from line 9.

1. If (*innerPeriod* = *prevIP*) then, as *prevSuffix* already has period *prevIP*, *suffix* will inherit from it. Otherwise we know that (*innerPeriod* > *prevIP*).
2. Then, *prevSuffix* has a basic period (*prevIP*) that should not divide *innerPeriod*, which is the length of the prefix of *prevSuffix* that occurs as prefix of *suffix*. Hence,

we require the condition (*prevIP* ∤ *innerPeriod*) to be satisfied. Otherwise, *suffix* would also have *prevIP* as period; then *suffix* would be a binary world, but would not realize *P*.

3. Then, if (*innerPeriod* = *prevLg*) then *lg* = 2× *prevLg* and *suffix* equals /prevSuffix/² and has the desired length and basic period.

4. Otherwise, we check that *prevSuffix* has period *innerPeriod*. If yes, then *suffix* also has period *innerPeriod* by construction (line 10), and thus realizes $P_{P[i]}$. If not, then there is no possible realization of *P* and we return ε (line 11).

Case 2 Here, we know that *lg* is larger than twice *prevLg*. Therefore, we will build a prefix that starts with *prevSuffix* followed by *m* new symbols, such that *suffix* has no period shorter than *innerPeriod*. Hence, we must ensure that *newPrefix* is primitive, otherwise it would have a period that divides *innerPeriod*. By Lemma 3 from [8], for any binary word *w*, *w*a or *w*b is primitive. So, we concatenate a^m to *prevSuffix*, and check if it is primitive (in $O(|\text{newPrefix}|)$ time). If not, we change its last symbol a by b. In both cases, *newPrefix* is primitive. By construction, *suffix* has basic period *innerPeriod* as desired, and thus realizes $P_{P[i]}$.

C.2   Examples of traces of binary realizations

Here is the trace of Algorithm 2 for length *n* := 10, and *P* := {0, 3, 6, 8}, which is not a period set for *n* = 10, i.e., *P* does not belong to Γ_{10}.

period	length	inner period	case	suffix	valid
8	10-8 = 2	10-8= 2	2	$z := ab$	true
6	10-6 = 4	8-6 = 2	2	$y := z.z = abab$	true
3	10-3 = 7	6-3 = 3	1	$w := y \oplus_3 y$	false

The table below illustrates that the merge attempted at the last loop iteration for $P[i] = 3$ is impossible, since a mismatch occurs in the overlap.

pos.	0	1	2	3	4	5	6
y	a	b	a	b	-	-	-
y	-	-	-	a	b	a	b

D   Checking FPR and BPR

Let us state some properties:

1. From the definition of FPR, we can see that checking the FPR for a pair (p, q) of *P* is equivalent to checking the FPR for pair $(0, q)$ in the nested PS P_p.

2. Assume the FPR is satisfied for pair $(0, p)$. Then, it is also satisfied for any pair (hp, jp) with $1 \le h < j < \lfloor n/p \rfloor$ and $hp, jp \in P$, since both periods are multiples of *p*.

From both properties, we get that once the FPR has been checked for the first pair (p,q) taken that has offset $(q-p)$, it is also satisfied for any other pair whose offset equals r or a multiple of r. It follows that, for a set P, one can limit the checking of FPR only to left most pairs whose offsets differ from eachother and are not multiple of another offset. Thus, at least one element, say p, must be an *irreducible period* (as defined in [22]), and q is the closest period to p (i.e., one which gives rise to the smallest offset with respect to p). Since, the number of irreducible periods of a period set of Γ_n is bounded by $\log_2(n)$ [23], the number of such pairs also is. We obtain the bound on the complexity for the general case stated in Lemma 3.

D.1   Checking the FPR and the BPR

Let $n > 0$ and P be a subset of $\{0,1,\ldots,n-1\}$. We assume that P is given as an ordered array. The complexity for checking the FPR or the BPR for P, has, to our knowledge not been previously addressed. For any pair $p < q$, we call their difference $(q-p)$, an offset.

Checking the BPR. Here, we demonstrate a property that relates the BPR to the FW Theorem. Precisely, if BPR is violated for some pair (p,q) at length n, with period $r := p - i(q-p)$ for some i, then the pair $(p-r,q-r)$ violates the FW condition of Theorem 4 in the nested set of length $(n-r)$.

Lemma 2. *Let (p,q) be a pair of integers that violate the BPR, and let $i \geq 2$ such that $r := p - i(q-p) \in P$. Then the pair $(p-r,q-r)$ violates the FW condition for length $(n-r)$.*

Proof. Let $P \in \Gamma_n$. Let p,q in P satisfying $0 \leq p < q < 2p$ be such that $(2p-q) \notin P$. Assume (p,q) violates the BPR. Then, there exists i in $[2,\ldots,\min(\lfloor p/(q-p),\rfloor,\lfloor (n-p)/(q-p)\rfloor)]$, such that $p - i(q-p) \notin P$. If several such integers exist, choose i as their minimum, and define $r := p - i(q-p)$. We will show that the nested period set of P for length $(n-r)$ is not a period set, since two of its periods violates the FW condition, which would require their gcd as an additional period, thereby implying that $P \notin \Gamma_n$, a contradiction. Since i is chosen minimal, we have that $\gcd(p,q)$ is not in P by the definition of the BPR. Note that $p - r = i(q-p)$ and $q - r = (i+1)(q-p)$. Thus, one gets $gcd(p-r,q-r) = q-p$, and the FW limit of $(p-r,q-r)$ equals $2i(q-p)$. Indeed, $FW(p-r,q-r) := p-r+q-r-gcd(p-r,q-r) = 2p-2r = 2i(q-p)$. By hypothesis, we have:

$$
\begin{aligned}
i &\leq (n-p)/(q-p) \\
\Leftrightarrow p + i(q-p) &\leq n \\
\Leftrightarrow r + 2i(q-p) &\leq n \\
\Leftrightarrow FW(p-r,q-r) &\leq n-r
\end{aligned}
$$

meaning that $(p-r,q-r)$ violates the FW condition of Theorem 4 for length $(n-r)$.

In algorithmic terms, checking the BPR of a set P can be done by checking the FW condition of Theorem 4 in each nested set of P. Altogether this takes $O(\sharp(P))$ time and space.

Checking the FPR. Some properties are explained in Appendix D and lead to this Lemma.

Lemma 3. *Let P is a subset of* $[0,\ldots,n-1]$*, that is ordered, and has zero as first period. Checking the FPR for P in general takes* $O(n\log(n))time$*.*

E   Fate: computation of the limits of a period set

E.1   Next extension

Algorithm 4 computes the next extension of P. The next extension is a length at which some deducible period needs to be added to P to satisfy the FPR. It equals the added period plus one, and must be larger than the birth length of P (Indeed, $P \in \Gamma_{max(P)+1}$, and thus satisfies the FPR for that length). By definition of the FPR, a period induced by the FPR equals $P[i] + P[i] - P[j]$ for some indexes $0 < j < i < \sharp(P)$. Because, we need the minimum of added periods, we can restrict the computation to pairs of adjacent periods (i.e. that is to case where $j = i-1$), since the offset between periods decreases with their index. Hence, the formula $P[i] + (P[i] - P[i-1])$ for computing the limit induced from current period $P[i]$. Because of this, we can also rule out cases where $P[i]$ is smaller the half the birth length of P (line 6).

Algorithm 4: next extension(period set P (in a sorted array))

   Output: $e(P)$;

1 birthLg := $max(P) + 1$;   // min x s.t. P belongs to $\Gamma(x)$

2 limit := $+\infty$;   // limit to be computed, init. with $+\infty$

3 **for** $i := \sharp(P) - 1$ *to* 1 **do**

4   **if** $P[i] \leq \lfloor \frac{birthLg}{2} \rfloor$ **then** //

5   break;   // avoid such $P[i]$ values whose limit cannot be
           $>$ birthLg

6   **if** $P[i] + (P[i] - P[i-1]) \geq birthLg$) **then** // current limit is beyond
    birthLg
        // update limit with the min of limit and current limit

7   limit := $min($limit$, P[i] + (P[i] - P[i-1]))$;

8 **return** *limit*;

E.2   Recursive FW limit

We exhibit an algorithm to compute what we termed, the recursive FW limit of a PS P (see Algorithm 5). The FW Theorem provides a way to compute a maximal length for any pair of distinct, non trivial periods such that one period is not a multiple of the other. For any p,q in P such that $0 < p < q < n$ and $p \nmid q$, we denote by $FW(p,q)$ the FW limit, that is $FW(p,q) := p + q - gcd(p,q)$. If $p \div q$ we assume that $FW(p,q) := +\infty$. First, the algorithm proceeds with two special cases: if all periods are multiple of the basic period, then it returns $+\infty$. Note this includes the case with basic period equals to one. If P contains only three periods, then it returns $FW(P[1], P[2])$.

    Otherwise, it will compute the limit l and initializes with $+\infty$. It loops over P backwards, to consider longer and longer suffixes starting at a position with period of a word

satisfying P, and builds a list Q of periods restricted to the current suffix. The periods in Q are those of P minus the starting position. It computes $FW(Q[1],Q[2])$ and takes the minimum between l and $P[i] + FW(Q[1],Q[2])$. After terminating the loop, it returns the limit l.

Algorithm 5: RecursiveFWLimit(period set P in a sorted array)

Output: the minimum length at which a pair of periods of P requires a change of basic period (application of FW Theorem);

1 **if** $(P[1] \mid P[i])$ *for all* $1 < i < \sharp(P)$ **then** // If basic period divides all other periods
2   **return** $+\infty$;
3 **if** $\sharp(P) = 3$ **then** // If P contains only two non trivial periods
4   **return** $FW(P[1], P[2])$;
5 limit $:= +\infty$;   // limit to be computed, init. with $+\infty$
6 insert $(P[n-1] - P[n-2])$ in Q ;   // Init Q with the last offset between periods
7 **for** $i := \sharp(P) - 3$ *to* 0 **do**
8   offset $:= P[i+1] - P[i]$;
9    $Q[0] := Q[0] + $ offset;
10   insert offset at first position in Q;
11   limit $:= min($limit$, P[i] + FW(Q[0], Q[1]))$;
12 **return** *limit*;

Complexity. In Algorithm 5, the first special case is processed in $\sharp(P)$ time (lines 1–2), while the second one requires constant time (lines 3–4). The main loop is executed at most $\sharp(P)$ times and all instructions in it take constant time (lines 7–11). Altogether, Algorithm 5 takes $O(\sharp(P))$ time and constant space.

Correctness. The correctness of Algorithm 5 follows from Lemma 2.

F   Properties of periods and characterization of period sets

F.1   Properties of periods

Let us state some known, useful properties of periods, which are detailed in [23].

Lemma 4. *Let p be a period of $u \in \Sigma^n$ and $k \in \mathbb{N}_{\geq 0}$ such that $kp < n$. Then kp is also a period of u.*

Lemma 5. *Let p be a period of $u \in \Sigma^n$ and q a period of the suffix $w = u[p..n-1]$. Then $(p+q)$ is a period of u. Moreover, $(p+kq)$ is also a period of u for all $k \in \mathbb{N}_{\geq 0}$ with $p + kq < n$.*

Lemma 6. *Let p,q be periods of $u \in \Sigma^n$ with $0 \leq q \leq p$. Then the prefix and the suffix of length $(n-q)$ have the period $(p-q)$.*

Lemma 7. *Suppose p is a period of $u \in \Sigma^n$ and there exists a substring v of u of length at least p and with period r, where $r|p$. Then r is also a period of u.*

F.2   Characterization of autocorrelations/period sets [7]

Guibas and Odlyzko have provided two equivalent characterizations of period sets: one is given by predicate Ξ, the other is the rule based characterization given in Section 2. However, they manipulate period sets as binary vectors called *autocorrelation* (or sometimes correlation for short). Remind that an autocorrelation is a binary encoding in a binary string of length n of a period set of Γ_n. We recall in extenso the original predicate Ξ and then their Theorem 5.1, which states the equivalence of characterizations and the alphabet independence.

> *Predicate Ξ:* v satisfies Ξ iff $v_0 = 1$ and, if p is the basic period of v, one of the following conditions is satisfied:
>
> *Case* a: $p \leqslant \lfloor n/2 \rfloor$. Let $r := \mathrm{mod}\,(n,p)$, $q := p + r$ and w the suffix of v of length q. Then for all j in $[1, n-q]$ $v_j = 1$ if $j = ip$ for some i, and $v_j = 0$ otherwise; and the following conditions hold:
>
> 1. $r = 0$ or $w_p = 1$,
> 2. if $\pi(w) < p$ then $\pi(w) + p > q + \gcd(\pi(w), p)$,
> 3. w satisfies predicate Ξ.
>
> *Case* b: $p > \lfloor n/2 \rfloor$. We have $\forall j\colon 1 \leqslant j < p,\ v_j = 0$. Let w be the suffix of v of length $n - p$, then w satisfies predicate Ξ.

Theorem 7. *Let v a binary string of length n. The following statements are equivalent:*

1. *v is the autocorrelation of a binary word*
2. *v is the autocorrelation of a word over an alphabet of size ≥ 2*
3. *$v_0 = 1$ and v satisfies the **Forward** and **Backward Propagation Rules***
4. *v satisfies the predicate Ξ.*

Let $v \in \{0,1\}^n$. We state the original definitions of FPR and BPR.

Definition 7. *v satisfies the FPR iff for all pairs (p,q) satisfying $0 \leq p < q < n$ and $v_p = v_q = 1$, it follows that $v_{p+i(q-p)} = 1$ for all $i = 2, \ldots, \lfloor (n-p)/(q-p) \rfloor$.*

Definition 8. *v satisfies the BPR iff for all pairs (p,q) satisfying $0 \leq p < q < 2p$, $v_p = v_q = 1$, and $v_{2p-q} = 0$, it follows that $v_{p-i(q-p)} = 0$ for all $i = 2, \ldots, \min(\lfloor p/(q-p) \rfloor, \lfloor (n-p)/(q-p) \rfloor)$.*