The language (and series) of Hammersley-type processes*

Cosmin Bonchiş[1,2], Gabriel Istrate[1,2], and Vlad Rochian[1]

[1] Department of Computer Science, West University of Timişoara, Bd. V. Pârvan 4, Timişoara, Romania. Corresponding author's email: gabrielistrate@acm.org
[2] e-Austria Research Institute, Bd. V. Pârvan 4, cam. 045 B, Timişoara, Romania.

Abstract. We study languages and formal power series associated to (variants of) Hammersley's process. We show that the ordinary Hammersley process yields a regular language and the Hammersley tree process yields deterministic context-free (but non-regular) languages. For the extension to intervals of the Hammersley process we show that there are *two* relevant formal languages. One of them leads to the same class of languages as the ordinary Hammersley tree process. The other one yields non-context-free languages.
The results are motivated by the problem of studying the analog of the famous Ulam-Hammersley problem for heapable sequences. Towards this goal we also give an algorithm for computing formal power series associated to the variants of the Hammersley's process, that have the formal languages studies in this paper as their support. We employ these algorithms to settle the nature of the scaling constant, conjectured in previous work to be the golden ratio. Our results provide experimental support to this conjecture.

1 Introduction

The Physics of Complex Systems and Theoretical Computing have a long and fruitful history of cooperation: for instance the celebrated Ising Model can be studied combinatorially, as some of its versions naturally relate to graph-theoretic concepts [21]. Methods from formal language theory have been employed (even in papers in physics venues) to the analysis of dynamical systems [13,22]. Sometimes the cross-fertilization goes in the opposite direction: concepts from the theory of *interacting particle systems* [12] (e.g. the voter model) have been useful in the analysis of gossiping protocols. A relative of the famous TASEP process, the so-called *Hammesley-Aldous-Diaconis* (HAD) process, has provided [1] the most illuminating solution to the famous *Ulam-Hammersley problem* [18] concerning the scaling behavior of the longest increasing subsequence of a random permutation.

In this paper we contribute to the literature on investigating physical models with discrete techniques by bringing methods based on formal language theory (and, possibly, noncommutative formal power series) to the analysis of several variants of the HAD process: We define formal languages (and power series) encoding all possible trajectories of such processes, and completely determine (the complexity of) these languages.

* This work was supported by a grant of Ministery of Research and Innovation, CNCS - UEFISCDI, project number PN-III-P4-ID-PCE-2016-0842, within PNCDI III.

The main process we are concerned with was defined in combinatorially in [9], and in more general form in [4], where it was dubbed *the Hammersley tree process*. It appeared naturally in [9] as a tool to investigate a version of the Ulam-Hammersley problem that employs the concept (due to Byers et al. [7]) of *heapable sequence*, an interesting variation on the concept of increasing sequence. Informally, a sequence of integers is heapable if it can be successively inserted into the leafs of a (not necessarily complete) binary tree satisfying the heap property. The Ulam-Hammesley problem for heapable sequences is open, the scaling behavior being the subject of an intriguing conjecture (see Conjecture 3 below) involving the golden ratio [9]. Methods based on formal power series can conceivably rigorously establish the true value of this constant. We also study a (second) version of the HAD tree process, motivated by the Ulam-Hammersley problem (and a similar conjecture, see Conjecture 7 below), for random intervals [3].

The outline of the paper is the following: In section 2 we give more details on the combinatorial and probability-theoretic motivations of the problem we are interested in. Section 3 presents concepts and notations we will use in the sequel. In Section 4 we presents and prove our main result: we precisely identify Hammersley's language for every $k \geq 1$. The language turns to be regular for $k = 1$ and deterministic context-free but non-regular for $k \geq 2$. The result is then extended to (the analog of) Hammersley's process *for intervals* in Section 5. In this case, it turns that there are *two* natural ways to define the associated formal language. The "effective" version yields the same language as in the case of permutations. The "more useful" one yields (as we show) non-context-free languages that can be explicitly characterized. We then proceed by presenting (Section 6) algorithms for computing the associated power series. They are applied to determining true value of scaling constant (believed to be equal to the golden ratio) for the Ulam-Hammersley problem for heapable sequences. In a nutshell, **the experimental results tend to confirm the identity of this constant to the golden ratio**; however the convergence is slow, as the estimates based on the formal power series computations we undertake (based on small values of n) seem quite far from the true value. The paper concludes (Section 7) with several discussions and open problems.

2 Motivation

The following combinatorial concept was introduced (for $k = 2$) in [7] and further studied in [9,15,10,4,5,3]:

Definition 1. *A sequence $X = X_0, \ldots, X_{n-1}$ is min k-heapable if there exists some k-ary tree T with nodes labeled by (exactly one of) the elements of X, such that for every non-root node X_i and parent X_j, $X_j \leq X_i$ and $j < i$. In particular a 2-heapable sequence will simply be called heapable [7]. Max heapability is defined similarly.*

Example 2. Let $X = [5, 1, 4, 2, 3]$. X is max 2-heapable: A max 2-heap for X is displayed in Figure 1. On the other hand sequence $Y = [2, 4, 1, 3]$ is obviously **not** max 2-heapable, as 4 cannot be a descendant of 2 in a max-heap.

Heapability can be viewed as a relaxation of the notion of increasing sequence, thus it is natural to extend to heapable sequences the framework of the Ulam-Hammersley problem [18], concerning the scaling behavior of the longest increasing subsequence

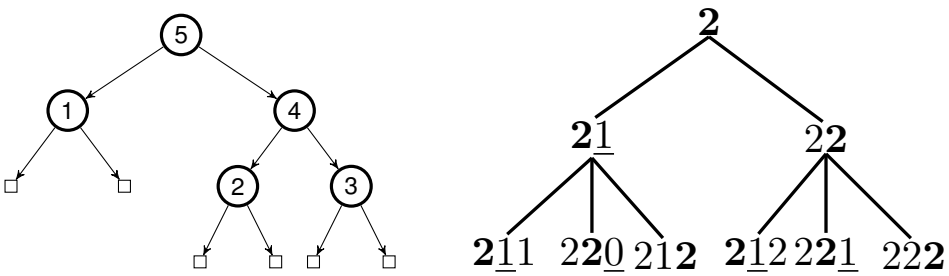


Fig. 1: (a). Heap ordered tree for sequence X in Example 2. (b). Words in Hammersley's tree process ($k = 2$). Insertions are bolded. Positions that lost a lifeline are underlined.

(LIS) of a random permutation. This extension can be performed in (at least) two ways, equivalent for LIS but no longer equivalent for heapable sequences: the first way, that of studying the length of the longest heapable subsequence, was dealt with in [7], and is reasonably simple: with high probability the length of the longest heapable subsequence of a random permutation is $n - o(n)$. On the other hand, by Dilworth's theorem [8] the length of the longest increasing subsequence of an arbitrary sequence is equal to the number of classes in a partition of the original sequence into *decreasing* subsequences. Thus it is natural to call *the Hammersley-Ulam problem for heapable sequences* the investigation of the scaling behavior of the number of classes of the partition of a random permutation into a minimal number of (max) heapable subsequences. This was the approach we took in [9]. Unlike the case of LIS, for heapable subsequences the relevant parameter (denoted in [9] by $MHS_k(\pi)$) scales logarithmically, and the following conjecture was proposed:

Conjecture 3. For every $k \geq 2$ there exists $\lambda_k > 0$ s.t., as $n \to \infty$, $\frac{E[MHS_k(\pi)]}{ln(n)}$ converges to λ_k. Moreover $\lambda_2 = \frac{1+\sqrt{5}}{2}$ **is the golden ratio.**

The problem was further investigated in [4,5], where the existence of the constant λ_k was proved. The equality of λ_2 to the golden ratio is less clear: authors of [4] claim it is slightly less than ϕ. Some non-rigorous, "physics-like" arguments, in favor of the identity $\lambda_2 = \phi$ was already outlined in [9], and is presented in [10], together with experimental evidence. Here we bring more convincing such evidence.

The intuition for Conjecture 3 relies on the extension from the LIS problem to heapable sequences of a correspondence between LIS and an interactive particle system [1] called the *Hammersley-Aldous-Diaconis (shortly, Hammersley or HAD) process*. The validity of correspondence was noted, for heapable sequences, in [9]. The generalized process was further investigated in [4], where it was called *the Hammersley tree process*. We can describe a combinatorial version of the process in [4] that is totally adequate for our purposes as follows: individuals appear (at integer times $t \geq 1$). Each individual is initially endowed with a value X_t and k "lifelines". The appearance of a new individual X_{t+1} subtracts a life from the smallest individual $X_a > X_{t+1}$ (if any) still alive at moment t. Intuitively lifelines in the HAM_k process will correspond to free slosts in the corresponding tree.

To recover the connection with random permutations we will assume from now on that the X_i's in process Ham_k are independent random numbers in $(0,1)$. The proposed value for λ_2 arises from a conjectural identification of the "hydrodynamic limit" of Hammersley's tree process (in the form of a compound Poisson process). Combinatorially we can describe this purported limit as follows: first encode the process by words over the alphabet Σ_k obtained by discarding the value information from particles and only record the number of lifelines. The insertion process can be described by chosing a random position in the current word w, inserting a k there and subtracting one from the first nonzero digit to the right of the insertion place (Fig. 1 (b).)

Example 4. If we run process Ham_2 on sequence X from Example 2, the outcome is a multiset of particles $1, 2$ and 3, each with multiplicity 2, encoded by the word 22200. Live particles correspond to the "free slots" depicted (as small rectangles) in Figure 1(a).

As $n \to \infty$ a "typical" sample word from Hammersley's process Ham_2 will have approximately $c_0 n$ zeros, $c_1 n$ ones and $\sim c_2 n$ twos, for some constants $c_0, c_1, c_2 > 0$[3]. Moreover, conditional on the number of zeros, ones, twos, in a typical word these digits are "uniformly mixed" throughout the sequence. Experimental evidence presented in [10] seems to confirm the accuracy of this picture.

A proof of the existence of constants c_0, c_1, c_2 was attempted in [9] based on subadditivity (Fekete's lemma). However, part of the proof in [9] is incorrect. While it could perhaps be fixed using more sophisticated tools (e.g. the subadditive ergodic theorem [20]) than those in [9], an alternate approach involves analyzing the asymptotic behavior of process Ham_k using (noncommutative) power series ([19,6]). Since the probability of word w in the process Ham_k is $1/|w|!$ times its multiplicity, this motivates:

Definition 5. *Given $k \geq 1$,* Hammersley's power series of order k *is the formal power series $F_k \in \mathbf{N}(<\Sigma_k>)$, $F_k(w) =$ the multiplicity of w in the process Ham_k.*

Understanding and controling the behavior of formal power series F_k may be the key to obtaining a rigorous analysis that confirms the picture sketched above. Though that we would very much want to accomplish this task, in this paper we resign ourselves to a simpler, language-theoretic, version of this problem:

Definition 6. *The* Hammersley language of order k, L_H^k, *is defined as the support of F_k, i.e. the set of words over Σ_k s.t. there exists a* trajectory *of HAM_k that yields w.*

The Ulam-Hammersley problem has also been studied [11] for sets of random intervals, generated as follows: to generate a new interval I_n first we sample (independently and uniformly) two random x, y from $(0,1)$. Then we let I_n be the interval $[min(x,y), max(x,y)]$. In fact the problem was settled in [11], where the scaling of LIS was determined to be $\lim_{n\to\infty} \frac{E[LIS(I_1,\ldots,I_n)]}{\sqrt{n}} = \frac{2}{\sqrt{\pi}}$.

In [3] we considered the extension of heapability to partial orders. Several results were proved; in particular, the greedy algorithm for partitioning a permutation into a minimal number of heapable subsequences extends to interval orders. This justifies an extension of the Ulam-Hammersley problem from increasing to heapable sequences of intervals. In [3] we conjectured the following scaling law:

[3] Nonrigorous computations predict that $c_0 = c_2 = \frac{\sqrt{5}-1}{2}, c_1 = \frac{3+\sqrt{5}}{2}$.

Conjecture 7. For every $k \geq 2$ there exists $c_k > 0$ such that, if R_n is a sequence of n random intervals then $\lim_{n\to\infty} \frac{E[\#Heaps_k(R_n)]}{n} = c_k$. Moreover $c_k = \frac{1}{k+1}$.

Remarkably, it was already noted in [3] that the connection between the Ulam-Hammersley problem and particle systems extends to the interval setting as well. The corresponding model is defined as follows:

Definition 8. *The* interval Hammersley process with k lifelines *is the stochastic process defined as follows: Particles arrive at integer moments; they have a value in the interval* $(0, 1)$*, and a number of lifelines. The process starts with a single particle with exactly* k *lifelines. Given the state* Z_{n-1} *of the process after step* $n-1$*, we choose, independently, uniformly at random and with repetitions two random reals* $X_n, Y_n \in (0, 1)$*. Then we perform the following operations:*

- *First a new particle with* k *lifelines is inserted with value* $min(X_n, Y_n)$*.*
- *Then the smallest (if any) live particle whose value is higher than* $max(X_n, Y_n)$ *loses one lifeline, yielding state* Z_n*.*

The state *of the process at a certain moment* n *comprises a recording of all real numbers chosen along the trajectory:* $(X_0, Y_0, \ldots, X_{n-1}, Y_{n-1})$*. Each number is endowed (in case it represented a new particle) with an integer in the range* $0 \ldots k$ *representing the number of lifelines the given particle has left at moment* n*.*

3 Preliminaries

For $k \geq 1$ define alphabet $\Sigma_k = \{1, 2, \ldots, k\}$. Define as well $\Sigma_\infty = \cup_{k\geq 1}\Sigma_k$. Given x, y over Σ_∞ we use notation $x \sqsubseteq y$ to denote the fact that x is a prefix of y. The set of (non-empty) prefixes of x is be denoted by $Pref(x)$.

A k*-ary (min)-heap* is a k-ary tree, non necessary complete, whose nodes have labels $t[\cdot]$ respecting the min-heap condition $t[parent(x)] \leq t[x]$.

For $w \in \Sigma^*$, we denote by $|w|_a$ the number of copies of character a in w.

Definition 9. *Let* $t \geq 1$*, and let* $a_1, b_1, \ldots, a_t > 0$ *and* $b_t \geq 0$ *be integers. We will use notation* $[a_0, b_0, \ldots, a_t, b_t]$ *as a shorthand for the word* $1^{a_0}0^{b_0} \ldots 1^{a_t}0^{b_t} \in \Sigma_1^*$ *(where* $0^0 = \epsilon$*, the null word).*

Definition 10. *Given* $k \geq 1$*, word* $w \in \Sigma_k$ *is called* k-dominant *if the following inequality holds* **for every** $z \in Pref(w)$*:* $|z|_k - \sum_{i=0}^{k-2}(k-i-1)\cdot|z|_i > 0$*. We call the left-hand side term* ***the structural difference*** *of word* z*.*

Observation 1 *1-dominant words are precisely those that start with a 1. 2-dominant words are those that start with a 2 and have, in any prefix,* strictly *more twos than zeros .*

Finally, we will use the following concepts:

Definition 11. *Let* w *be a word that is an outcome of the process* Ham_k*. An* increment of w *is a position* p *in* w *(among the* $|w| + 1$ *possible positions: at the beginning of* w*, at the end of* w *or between two letters of* w*) such that no nonzero letters of* w *appear to the right of* p*. The number of increments of word* w *is denoted by* $\#inc_k(w)$*. It is nothing but 1 plus the number of trailing zeros of* w*.*

Definition 12. *Let* L *be an alphabet that contains* Σ_k *for some* $k \geq 1$*. Given a word* $w \in L^*$ *we denote by* $s(w)$ *the sum of the digit characters of* w*.*

4 Main result

Theorem 13. *For every $k \geq 1$ $L_H^k = \{w \in \Sigma_k^* \mid w \text{ is } k\text{-dominant.}\}$.*

Corollary 14. *Language L_H^1 is regular. For $k \geq 2$ languages L_H^k are deterministic one-counter languages but not regular.*

Proof. For $k = 1$ the result is trivial, as $L_H^1 = 1\Sigma_1^*$. The claim $L_H^k \in$ is deterministic one-counter language for $k \geq 2$ follows from Theorem 13, as one can construct a one-counter pushdown automaton P_k for the language on k-dominant words. To prove that L_H^k, $k \geq 2$ is not regular is a simple exercise in formal languages. Details are given in the Appendix.

□

The proof of Theorem 13 proceeds by double inclusion. Inclusion "$\subseteq$" is proved with the help of several easy auxiliary results:

Lemma 15. *Every word in L_H^k starts with a k.*

Proof. Follows easily by appealing to the particle view of the Hammersley process: the particle with the smallest label x stays with k lives until the end of the process, as no other particle can arrive to its left.

Lemma 16. *L_H^k is closed under prefix.*

Proof. Again we resort to the particle view of the Hammersley process: let $w \in L_H^k$ be a word and $u = x_0 \dots x_{n-1}$ be a trajectory in [0,1] yielding w. A non-empty prefix z of w corresponds to the restriction of u to some segment $[0, l]$, $0 < l < 1$. This restriction is a trajectory itself, that yields z.

Lemma 17. *Every word in L_H^k has a positive structural difference.*

Proof. Let $w \in L_H^k$ and let t be a corresponding trajectory in the particle process.

Let λ be the number of times a particle arrives as a local maximum (without subtracting a lifeline from anyone). For $i = 1, \dots, k$ let λ_i be the number of time the newly arrived particle subtracts a lifeline from a particle currently holding exactly i lives. $\lambda, \lambda_1, \dots, \lambda_k \geq 0$. Moreover, $\lambda > 0$, since the largest particle does not take any lifeline.

By counting the number of particles with i lives at the end of the process, we infer: $|z|_0 = \lambda_1, |z|_1 = \lambda_2 - \lambda_1, \cdots |z|_{k-1} = \lambda_k - \lambda_{k-1}$. Finally, $|z|_k = \lambda + \sum_{i=0}^{i-2} \lambda_i .(*)$

Simple computations yield $\lambda_{i+1} = |z|_0 + \ldots + |z|_i$, for $i = 0, \dots, k-1$. Relation (*) and inequality $\lambda > 0$ yield the desired result.

Together, Claims 15, 16 and 17 establish the fact that any word from L_H^k is k-dominant, thus proving inclusion "$\subseteq$". To proceed with the opposite inclusion, for every k-dominant word w we must construct a trajectory of the process HAM_k that acts as a witness for $w \in L_H^k$.

We will further reduce the problem of constructing a trajectory T_z to the case when z further satisfies a certain simple property, explained below:

Definition 18. *k-dominant word u is called critical if* $|u|_k - \sum_{i=0}^{k-2} (k-1-i) \cdot |u|_i = 1$.

The above-mentioned reduction has the following statement:

Lemma 19. *Every k-dominant* **critical** *z is witnessed by some trajectory T_z.*

Proof. By induction on $|z|$. The base case, $|z| = 1$:, is trivial, as in this case $z = k$.

Inductive step: Assume the claim is true for all the critical words of length strictly smaller than z's. We claim that w_1, the word obtained from z by deleting the last copy of k and increasing by 1 the value of the letter immediately to the right of the deleted letter, is critical.

Indeed, it is easy to see that the structural difference of w_1 is 1. Clearly the deleted letter could not have been the last one: otherwise deleting it would yield a prefix of z that has structural constant equal to zero. Also clearly, the letter whose value was modified in the previous constraint could not have been a k, by definition, and certainly is nonzero after modification. So w_1's construction is indeed correct. As $|w_1| = |z| - 1$, w_1 satisfies the conditions of the induction hypothesis.

By the induction hypothesis, w_1 can be witnessed by some trajectory T. We can construct a trajectory for z by simply following T and then inserting the last k of z into w_1 in its proper position (thus also making the next letter assume the correct value).

We now derive Theorem 13 from Lemma 19. The key observation is the following fact: every k-dominant word z is a prefix of a *critical word*, e.g. $z' = z(k-2)^\lambda$ where $\lambda = |z|_k - \sum_{i=0}^{k-2}(k-i-1)\cdot|z|_i - 1 \geq 0$.

By Lemma 19, z' has a witnessing trajectory $T_{z'}$. Since the existence of a trajectory is closed under taking prefixes, Theorem 13 follows.

5 Extension to interval heapability

To prove a similar result for the interval Hammersley process we need to "combinatorialize" the process from Definition 8, that is, to replace that definition (which employs (random) real values in $(0, 1)$) with an equivalent stochastic process on words.

The combinatorialization introduce some technical complications with respect to the case of permutations. Specifically, for permutations the state of the system could be preserved, with no real loss of information by a string representing only the number of lifelines of the given particles, but **not** their actual values. This enables (as we will see below in Section 6) an algorithm for computing the associated formal power series.

To accomplish a similar goal in the case of random intervals we apparently need to take into account the fact that at each step we choose *two* random numbers in Definition 8, even though only one of them is inserted a particle into. The proper discretization in this case seems to require an extra symbol $\diamond$ (that will mark the position of real values that were generated but in which no particle was inserted), and is accomplished as follows:

Definition 20. *Process $Ham_{k,INT}$ is the stochastic process on $(\Sigma_k \cup \{\diamond\})^*$ defined as follows: The process starts with the two-letter word $Z_1 = k\diamond$. Given the string representation Z_{n-1} of the process after step $n-1$, we choose, independently, uniformly at random and with repetitions two positions X_n, Y_n into string Z_{n-1}. X_n, Y_n may happen to be the same position, in which we chose randomly an ordering of X_n, Y_n.*

Then we perform (see Figure 5) the following operations:

- *First, a new particle with k lifelines is inserted into Z_{n-1} at position $min(X_n, Y_n)$.*
- *Then a $\diamond$ is introduced in position $max(X_n, Y_n)$ (immediately after the previously introduced k, if $X_n = Y_n$).*
- *Then the smallest (if any) live particle occurring **after the position of the newly inserted** $\diamond$ loses one lifeline, yielding string Z_n.*

A result that was easy for the process Ham_k but deserves some discussion in the case of the interval process is the following:

Proposition 21. *Consider the string $w_n \in \Sigma_k^*$ obtained by taking a random state of the Hammersley interval process with k lifelines at stage n and then "forgetting" the particle value information (recording instead only the value in $\Sigma_k \cup \{\diamond\}$). Then w_n has the same distribution as a sample from process $Ham_{k,INT}$ at stage n.*

Proof. The crux of the proof is the following

Lemma 22. *The ordering of the values $X_0, Y_0, X_1, Y_1, \ldots, X_{n-1}Y_{n-1}$ inserted in the first n steps in the Hammersley interval process (disregarding their number of lifelines) is that of a random permutation with $2n$ elements.*

Proof. X_i, Y_i have the same distribution, tone is he minimum of two random uniformly distributed variables in $(0, 1)$, the other one is the maximum. Thus to simulate the process $Ham_{k,INT}$ for n steps one needs $2n$ random numberr in (0,1) which yields a random permutation of size $2n$.

This discussion motivates extending the investigation of the problems in this paper to the interval Hammersley process $Ham_{k,INT}$ as well. We do this via the following:

Definition 23. *Denote by $L^k_{H,INT}$, called* the language of the interval Hammersley process*, the set of words (over $\Sigma_k \cup \{\diamond\}$) generated by the process $Ham_{k,INT}$.*

Due to the presence of diamonds, words in the previous language are not "physical", because diamonds do not necessarily correspond to actual particles. The motivation for Definition 23 was that we need diamonds in the "extended words" to obtain (see Appendix below) an algorithm for computing multiplicities. Ultimately, though, we are interested in words, without diamonds as it is their behavior that identifies the conjectured limit constant. This motivates the following variant of the previous definition:

Definition 24. *The* effective language of the interval Hammersley process*, $L^{k,eff}_{H,INT}$, is the set of strings in Σ_k^* obtained by deleting all diamonds from some string in $L_{H,INT}$.*

Finally define $F_{k,INT}$ to be the formal power series of multiplicities in the interval Hammersley process. The probability of word w is $\frac{1}{|(w|!)^2} \cdot F_{k,INT}(w)$ for all $w \in L^k_{H,INT}$. In the previous equation the factorial is squared due to the independent choice of *two* positions into a string z to determine the next string w generated by the process. If z has length $n-1$ this leads to n^2 choices.

In spite of the fact that the dynamics of process $Ham_{k,INT}$ is quite different from that of the ordinary process Ham_k (a fact that will be reflected in the coefficients of the

power series), and the conjectured scaling behavior is not at all similar (for $k \geq 2$), our next result shows that this difference **is not visible** on the actual trajectories: the effective language of the Interval Hammersley process coincides with that of the "ordinary" process. Indeed, we have:

Theorem 25. *For every $k \geq 1$, $L^{k,\mathit{eff}}_{H,INT} = L^k_H = \{w \in \Sigma^*_k \mid w \text{ is } k\text{-dominant.}\}$.*

Proof. It is immediate that $L^k_H \subseteq L^{k,\text{eff}}_{H,INT}$. Indeed, every trajectory of the process Ham_k is a trajectory of the process $Ham_{k,INT}$ as well: simply restrict at every stage the two particles to choose the same slot.

For the opposite inclusion we prove, by induction on $|t|$, that the outcome w of every trajectory t of the interval Hammersley process belongs to L^k_H. The case $|t| = 0$ is trivial, since $w = k$.

Definition 26. *Given a word w over Σ_k, word z is a* left translate of w *if z can be obtained from z by moving a k in w towards the beginning of w (we allow "empty moves", i.e. $z = w$).*

Lemma 27. *L^k_H is closed under left translates. That is, if $w \in L^k_H$ and z is a left translate of w then $z \in L^k_H$.*

Proof. By moving forward a k the structural constants of all prefixes of w can only increase. Thus if these constants are positive for all prefixes of w then they are positive for all prefixes of z as well.

Now assume that the induction hypothesis is true for all trajectories of length less than n. Let t be a trajectory of length n, let t' be its prefix of length $n-1$, let w be the yield of t and z be the yield of t'. By the induction hypothesis $z \in L^k_H$. Let y be the word obtained by applying the Hammersley process to z, deleting a life from the same particle as the interval Hammersley process does to z to obtain w. It is immediate that w is a left translate of y (that is because in the interval Hammersley process we insert a particle to the left of the position where we would in Ham_k). Since $y \in L^k_H$, by the previous lemma $w \in L^k_H$.

The previous result contrasts with our next theorem:

Theorem 28. *For $k \geq 1$ the language $L^k_{H,INT}$ is* **not** *context-free.*

Proof. Define the language $S_k = L^k_{H,INT} \cap \{k\}^* \diamond^* \{k-1\}^* \diamond^*$.

Lemma 29. $S_k = \{k^{c+d+e} \diamond^{c+e} (k-1)^c \diamond^{c+d} \mid c, d, e \geq 0\}$.

Proof. The direct inclusion is fairly simple: let $w \in S_k$. define c to be the number of letters $(k-1)$ in w. Since there are no stars in between the $(k-1)$'s, all such letters must have been produced by removing one lifeline each by some k's. Thus the number of stars in between the k's and $(k-1)$'s is $c+e$, with e being the number of pairs $(k, \diamond)$ that did not kill any particle that will eventually become a $k-1$.

On the other hand the number of k's is obtained by tallying up c (for the c letters that become $k-1$, needing one copy of k each), e (for the pairs $(k, \diamond)$ where $\diamond$ belongs to the first set of diamonds) and d (for d pairs $(k, \diamond)$ with $\diamond$ in the second set of diamonds).

For the reverse implication we outline the following construction:
First we derive $k^e \diamond^e$. Then we repeat the following strategy c times:

- We insert a k at the beginning of the $k-1$ block (initially at the end of the word) and the corresponding $\diamond$ at the end of the word.
- With one pair $k, \diamond$ (with $\diamond$ inserted in the first block) we turn the k into a $k-1$.

Finally we insert k pairs $(k, \diamond)$, with $\diamond$ in the second block.

The theorem now follows from the following

Lemma 30. *S_k is not a context-free language.*

Proof. An easy application of Ogden's lemma (see the Appendix).

In fact we can give a complete characterization of $L^k_{H,INT}$ similar in spirit to the one given for language L^k_H in Theorem 13:

Theorem 31. *Given $k \geq 1$, the language $L^k_{H,INT}$ is the set of words w over alphabet $\Sigma_k \cup \{\diamond\}$ that satisfy the following conditions:*

1. *$|w|_\diamond = |w|/2$. In particular $|w|$ must be even.*
2. *For every prefix p of w, (a). $|p|_\diamond \leq |p|/2$ and (b). $s(p) + (k+1)|p|_\diamond \geq k|p|$.*

Proof. The inclusion $\subseteq$ is easy: given $w \in L^k_{H,INT}$, conditions 1. and 2 (a). hold, as the process Ham^k_{INT} inserts a digit (more precisely a k) before every diamond.

As for condition 2 (b)., each $\diamond$ takes at most one life of a particle. The total number of lifelines particles in p are endowed with at their birth moments is $k(|p| - |p|_\diamond)$. These lifelines are either preserved (and are counted by $s(p)$), or they are lost, in a move which (also) introduces a $\diamond$ in p. Thus $k(|p| - |p|_\diamond) \leq |p|_\diamond + s(p)$, which is equivalent to b.

The inclusion $\supseteq$ is proved by induction on $|w|$. What we have to prove is that every word that satisfies conditions 1-2 is an output of the process $Ham_{k,INT}$.

The case $|w| = 2$ is easy: the only word that satisfies conditions 1-2 is easily seen to be $w = k\diamond$, which can be generated in one move.

Assume now that the induction hypothesis is true for all words of lengths strictly less than 2n, and let $w = w_1 \ldots w_{2n}$ be a word of length $2n$ satisfying conditions 1-2.

Lemma 32. $w_{2n} = \diamond$.

Proof. Let $p = w_1 \ldots w_{2n-1}$. By condition 2a, $|p|_\diamond \leq (2n-1)/2$, hence $|p|_\diamond \leq n-1$. Since $|w|_\diamond = n$, the claim follows.

Lemma 33. $w_1 = k$.

Proof. Let $q = w_1$. Since $|q|_\diamond \leq 1/2$, w_1 must be a digit. Since $s(q) \geq k|q| = k$, the claim follows.

Let now r be the largest index such that $w_r = k$. Let s be the leftmost position $s > r$ such that $w_s = \diamond$. Let t be the leftmost position $t > s$ such that $w_t \neq \diamond$, $t = 2n+1$ if no such position exists.

Consider the word b obtained from w by a. deleting positions w_r and w_s. b. increasing the digit at position w_t by one, if $t \neq 2n+1$. Note that, if $t \neq 2n+1$ then $w_t \neq k$, by the definition of index r. Also, $|b| = 2n-2 < 2n$.

w is easily obtained from b by inserting a k in position r and a diamond in position s, also deleting one lifeline from position t if $t \neq 2n+1$. To complete the proof we need to argue that b satisfies conditions 1-2a.b. Then, by induction, b is an output of the process $Ham_{k,INT}$, hence so is w.

Condition 1 is easy to check, since $|b| = 2n-2$, and b has exactly one $\diamond$ less than w, i.e. n-1 $\diamond$'s. As for 2.(a)-(b), let p be a prefix of b. There are four cases:

- **Case 1:** $1 \leq |p| < r$**:** In this case p is also a prefix of w, and the result follows from the inductive hypothesis.
- **Case 2:** $r \leq |p| < s-1$**:** In this case $p = w_1 \dots w_{r-1} w_{r+1} \dots w_{|p|+1}$. Let $z_1 = w_1 \dots \dots w_{|p|+1}$ be the corresponding prefix of w.
  The number of diamonds in p is equal to the number of diamonds in z_1. Since z_1 does **not** end with a diamond (as $|p| < s-1$), the number of diamonds in z_1 is equal to that of its prefix u of length $|p|$. By the induction hypothesis $|p|_\diamond = |u|_\diamond \leq |u|/2 = |p|/2$. So condition 2 (a). holds. On the other hand $s(p) + (k+1)|p|_\diamond = (s(z_1) - k) + (k+1)|z_1|_\diamond \geq k|z_1| - k = k|p|$, so 2 (b). holds as well.
- **Case 3:** $s-1 \leq |p| < t-2$**:** In this case $p = w_1 \dots w_{r-1} w_{r+1} \dots w_{s-1} w_{s+1} \dots w_{|p|+2}$. Let $z_2 = w_1 \dots \dots w_{|p|+2}$ be the corresponding prefix of w of length $|p|+2$ and z_3 the prefix of w of length $s-1$.
  The number of diamonds in p is equal to the number of diamonds in z_2 minus one. By the induction hypothesis, this is at most $|z_2|/2 - 1$, which is at most $(|p|+2)/2 - 1 = |p|/2$. Thus condition 2 (a). holds. Now $s(p) + (k+1)|p|_\diamond =$
  $(s(z_2) - k) + (k+1)(|z_2|_\diamond - 1) = (s(z_3) - k) + (k+1)(|z_3|_\diamond + |z_2| - |z_3| - 1)$
  $\geq k|z_3| - k + (k+1)(|p| + 2 - |z_3| - 1) = (k+1)(|p|+1) - |z_3| - k =$
  $= (k+1)|p| - |z_3| + 1 > k|p| + 1 + (|p| - |z_3|) > k|p|$

  so condition 2 (b). is established as well. In the previous chain of (in)equalities we used the fact (valid by the very definition of t) that for all $s \leq i < t$, $w_i = \diamond$.
- **Case 4:** $t-2 \leq |p| \leq 2n$**:**] In this case $p = w_1 \dots w_{r-1} w_{r+1} \dots w_{s-1} w_{s+1} \dots (w_t + 1) \dots$. Furthermore, p ends with $w_{|p|+2}$ (if $|p|+2 \neq t$) and with $w_{|p|+2} + 1$ (if $|p|+2 = t$). Let $z_4 = w_1 \dots \dots w_{|p|+2}$ be the prefix of w of length $|p|+2$.
  - $|p|_\diamond = |z_4| - 1 \leq |z_4|/2 - 1 = (|p|+2)/2 - 1 = |p|/2$.
  - On the other hand $s(p) + (k+1)|p|_\diamond = (s(z_4)) - k + 1) + (k+1)(|z_4| - 1) \geq k|z_4| - k + 1 - k - 1 = k \cdot (|p|+2) - 2k = k|p|$.

  so conditions 2 (a)-(b). are proved in this last case as well.

6 Computing Hammersley's power series and Applications

In this section we return to the power series perspective on the Ulam-Hammersley problem for heapable sequences. We outline a simple algorithm (based on dynamic programming) to compute the coefficients of Hammersley's power series F_k.

Justifying correctness of algorithm ComputeMultiplicity is simple: a string w can result from any string z by inserting a k and deleting one life from the closest non-zero letter of z to its right. After insertion, the new k will be the rightmost element of a

maximal block of w of consecutive k's. The letter it acts upon in z cannot be a k (in w), and cannot have any letters other than zero before it.

The candidates in w for the changed letter are those letters l succeeding the newly inserted k such that $0 \leq l \leq k-1$ and the only values between k and l are zeros. Thus these candidates are the following: (a)letters in w forming the maximal block B of zeros immediately following k (if any), and (b).the first letter after B, provided it has value 0 to $k-1$. Since we are counting multiplicities and all these words lead to distinct candidates, the correctness of the algorithm follows.

A simple adaptation of the algorithm ComputeMultiplicity yields another algorithm (called $ComputeMultiplicity_{INT}$) for computing the coefficients of the formal power series $F_{k,INT}$ associated to the Interval Hammersley process $Ham_{k,INT}$. Details are presented in the Appendix.

For $k = 1$ the algorithm ComputeMultiplicity simplifies to a recurrence formula: Indeed, in this case there are no candidates of type (b). Assuming that we use the notation from Definition 9, we derive: $F_1([a_1, b_1, \ldots, a_s, b_s]) = \sum_{\substack{i=1: \\ a_i>1}}^{s} \sum_{\substack{j,l\geq 0 \\ j+1+l=b_i}} F_1([a_1, \ldots, a_i-1, j, 1, l, a_{i+1}, \ldots, b_s]) + \sum_{\substack{i=1: \\ a_i=1}}^{s} \sum_{\substack{j,l\geq 0 \\ j+1+l=b_i}} F_1([a_1, \ldots, a_{i-1}, b_{i-1}+j, 1, l, a_{i+1}, \ldots, b_s])$ if $b_s > 0$, otherwise $F_1([a_1, b_1, \ldots, a_s, 0]) = \sum_{\substack{i=1: \\ a_i>1}}^{s-1} \sum_{\substack{j,l\geq 0 \\ j+1+l=b_i}} F_1([a_1, b_1, \ldots, a_i-1, j, 1, l, a_{i+1}, \ldots, a_s, 0]) + \sum_{\substack{i=1: \\ a_i=1}}^{s-1} \sum_{\substack{j,l\geq 0 \\ j+1+l=b_i}} F_1([a_1, \ldots, a_{i-1}, b_{i-1}+j, 1, l, a_{i+1}, \ldots, b_s]) + F_1([a_1, \ldots, a_s - 1, 0])$.

w	1	10	11	100	101
$F_1(w)$	1	1	1	1	2
w	110	111	1000	1001	1010
$F_1(w)$	2	1	1	3	5
w	1011	1100	1101	1110	1111
$F_1(w)$	3	5	3	3	1

w	2	21	22	211	212	220	221
$F_2(w)$	1	1	1	1	2	1	1
w	222	2111	2112	2120	2121	2122	2201
$F_2(w)$	1	1	3	2	3	3	1
w	2202	2210	2211	2212	2220	2221	2222
$F_2(w)$	3	1	1	2	2	1	1

Fig. 2: The leading coefficients of formal power series (a). F_1. (b). F_2.

In spite of this, we weren't able to solve the recurrence above and compute the generating functions F_1 or, more generally, F_k, for $k \geq 1$. An inspection of the coefficients obtained by the application of the algorithm is inconclusive: We tabulated the leading coefficients of series F_1 and F_2, computed using the Algorithm 3 in Figures 2 (a). and (b). The second listing is restricted to 2-dominant strings only. No apparent closed-form formula for the coefficients of F_1, F_2 emerges by inspecting these values.

6.1 Application: estimating the value of the scaling constant λ_2.

The computation of series F_2 allows us to tabulate (for small value of n) the values of the distribution of increments, a structural parameter whose limiting behavior determines the value of the constant λ_2 (conjectured, remember, to be equal to $\frac{1+\sqrt{5}}{2}$).

That increments are useful in computing λ_2 is seen as follows: consider a word w of length n that is a sample from the Ham_k process. Increments of w are those positions where the insertion of a k does not remove any lifeline, thus increasing the number of

heaps in the corresponding greedy "patience heaping" algorithm [9] by 1. If w has t increment positions then the probability that the number of heaps will increase by one (given that the current state of the process is w) is $t/(n+1)$.

Input: $k \geq 1, w \in \Sigma_k^*$
Output: $F_k(w)$

$S := 0$. $w = w_1 w_2 \dots w_n$
if $w \notin L_H^k$
  return 0
if $w ==$ 'k'
  return 1
for i in 1:n-1
  if $w_i == k$ and $w_{i+1} \neq k$
    let $r = min\{l \geq 1 : w_{i+l} \neq 0$ or $i + l = n + 1\}$
    for j in 1:r-1
      let $z = w_1 \dots w_{i-1} w_{i+1} \dots w_{i+j-1} 1 w_{i+j+1} \dots w_{i+r} \dots w_n$
      $S := S + ComputeMultiplicity(k, z)$
    if $i + r \neq n + 1$ and $w_{i+r} \neq k$
      let $z = w_1 \dots w_{i-1} w_{i+1} \dots w_{i+r-1} (w_{i+r} + 1) w_{i+r+1} \dots w_n$
      $S := S + ComputeMultiplicity(k, z)$
  if $w_n == k$
    let $Z = w_1 \dots \dots w_{n-1}$
    $S := S + ComputeMultiplicity(k, z)$
return S

Fig. 3: Algorithm ComputeMultiplicity(k,w)

What we need to show is that (as $n \to \infty$) the mean number of positions that are increments in a random sample w of length n tends to λ_k. Therefore the probability that a new position will increase the number of heaps by 1 is asymptotically equal to $\lambda_k/(n+1)$. The scaling of the expected number of heaps follows from this limit.

In Figure 4 (a). we plot the *exact* probability distribution of the number of increments (from which we subtract one, to make the distribution start from zero) for $k = 2$ and several small values of n. They were computed exactly by employing Algorithm 3 to exactly compute the probability of each string w, and then computing $\#inc_2(w)$. We performed this computation for $2 \leq n \leq 13$. The corresponding expected values are tabulated (for all values $n = 2, \dots, 10$) in Figure 4 (b).

Unfortunately, as it turns out, the ability to exactly compute (for small values of n) the distribution of increments **does not** give an accurate estimate of the asymptotic behavior of this distribution, as the convergence seems rather slow, and not at all captured by these small values of n. Indeed, to explore the distribution of increments for large values of n, as exact computation is no longer possible, we instead resorted to *sampling* from the distribution, by generating 10000 independent random trajectories of length n from process Ham_2, and then computing the distribution of increments of the sampled outcome strings. The outcome is presented (for $n = 100, 100.000, 1.000.000$, together with some of the cases of the exact distribution) in Figure 4 (b). The distribution of increments seems to converge (as $n \to \infty$) to a geometric distribution with parameter

$p = \frac{\sqrt{5}-1}{2} \sim 0.618\cdots$. That is, we predict

$$\forall i \geq 1, \lim_{n\to\infty} Pr_{|w|=n}[\#inc_2(w) = i] = p \cdot (1-p)^{i-1}.$$

The fit between the (sampled) estimates for $n = 1.000000$ and the predicted limit distribution is quite good: every coefficient differs from its predicted value by no more than 0.003, with the exception of the fourth coefficient, whose difference is 0.007. Because of the formula for computing averages, these small differences have, though, a cumulative effect in the discrepancy for the average $E[\#inc_2(w)]$ for $n = 10000000$ accounting for the 0.03 difference between the sampled value and the predicted limit: in fact most of the difference is due to the fourth coefficient, as $4 \times 0.007 = 0.028$.

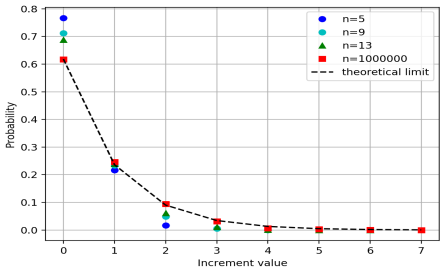
Conclusion 1 *The increment data supports the conjectured value* $\lambda_2 = 1 + p = \frac{1+\sqrt{5}}{2}$.

We intend to present (in the journal version of this paper) a similar investigation of the value of constant c_k in Conjecture 7.

7 Open questions and future work

The major open problems raised by our work concerns the nature and asymptotic behavior of formal power series F_k, $F_{k,INT}$. An easy consequence of Corollary 14 is

Corollary 34. *For* $k \geq 2$ *formal power series* F_k, $F_{k,INT}$ *are* **not N**-*rational.*



n	2	3	4	5	6	7
$E[\#inc_2]$	1.0	1.166	1.208	1.250	1.281	1.307
n	8	9	10	100	100000	1 mil
$E[\#inc_2]$	1.329	1.347	1.363	1.520	1.575	1.580

Fig. 4: (a). Probability distribution of increments, for $k = 2$, and $n = 5, 9, 13, 1000000$. (b). The mean values of the distributions of increments.

Open Problem 1 *Are formal power series* F_1, $F_{1,INT}$ **N**-*rational ?*

Note that Reutenauer [17] extended the Chomsky-Schützenberger criterion for rationality from formal languages to power series: a formal power series is rational if and only if the so-called *syntactic algebra* associated to it has finite rank. We don't know, though, how to explicitly apply this result to the formal power series we investigate in this paper. On the other hand, in the general case, the characterization of context-free languages as supports of **N**-algebraic series (e.g. Theorem 5 in [14]), together with Theorem 28, establishes the fact that series $F_{k,INT}$ is not **N**-algebraic.

Open Problem 2 *Are formal power series* F_k **N**-*algebraic ?*

References

1. D. Aldous and P. Diaconis. Hammersley's interacting particle process and longest increasing subsequences. *Probability theory and related fields*, 103(2):199–213, 1995.
2. D. Aldous and P. Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bull.of the A.M.S.*, 36(4):413–432, 1999.
3. J. Balogh, C. Bonchiş, D. Diniş, G. Istrate, and I. Todincã. Heapability of partial orders. *arXiv preprint arXiv:1706.01230*, 2017.
4. A.-L. Basdevant, L. Gerin, J.-B. Gouéré, and A. Singh. From Hammersley's lines to Hammersley's trees. *Probability Theory and Related Fields*, pages 1–51, 2016.
5. A.-L. Basdevant and A. Singh. Almost-sure asymptotic for the number of heaps inside a random sequence. *arXiv preprint arXiv:1702.06444*, 2017.
6. J. Berstel and C. Reutenauer. *Noncommutative rational series with applications*, volume 137. Cambridge University Press, 2011.
7. J. Byers, B. Heeringa, M. Mitzenmacher, and G. Zervas. Heapable sequences and subseqeuences. In *Proceedings of ANALCO'2011*, pages 33–44. SIAM Press, 2011.
8. R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, pages 161–166, 1950.
9. G. Istrate and C. Bonchiş. Partition into heapable sequences, heap tableaux and a multiset extension of Hammersley's process. In *Proceedings of CPM'2015*, volume 9133 of *Lecture Notes in Computer Science*, pages 261–271. Springer, 2015.
10. G. Istrate and C. Bonchiş. Heapability, interactive particle systems, partial orders: Results and open problems. In *Proceedings of DCFS'2016*, volume 9777 of *Lecture Notes in Computer Science*, pages 18–28. Springer, 2016.
11. J. Justicz, E. R. Scheinerman, and P. M. Winkler. Random intervals. *Amer. Math. Monthly*, 97(10):881–889, 1990.
12. T. Liggett. *Interacting Particle Systems*. Springer Verlag, 2004.
13. C. Moore and P. Lakdawala. Queues, stacks and transcendentality at the transition to chaos. *Physica D*, 135(1–2):24–40, 2000.
14. I. Petre and A. Salomaa. Algebraic systems and pushdown automata. *Handbook of Weighted Automata*, pages 257–289, 2009.
15. J. Porfilio. A combinatorial characterization of heapability. Master's thesis, Williams College, May 2015. available from `https://unbound.williams.edu/theses/islandora/object/studenttheses\%3A907`. Accessed: December 2017.
16. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.
17. C. Reutenauer. Séries formelles et algebres syntactiques. *J. Algebra*, 66(2):448–483, 1980.
18. D. Romik. *The surprising mathematics of longest increasing subsequences*. Cambridge University Press, 2015.
19. A. Salomaa and M. Soittola. *Automata-theoretic aspects of formal power series*. Springer Science & Business Media, 2012.
20. W. Szpankowski. *Average Case of Algorithms on Sequences*. John Wiley & Sons, 2001.
21. D. Welsh. *Complexity: Knots, Colourings and Counting*. Cambridge University Press, 1994.
22. H. Xie. *Grammatical Complexity and One-Dimensional Dynamical Systems*. Directions in Chaos vol. 6. World Scientific, 1996.

Appendix

7.1 Insertion in process $Ham_{2,INT}$.

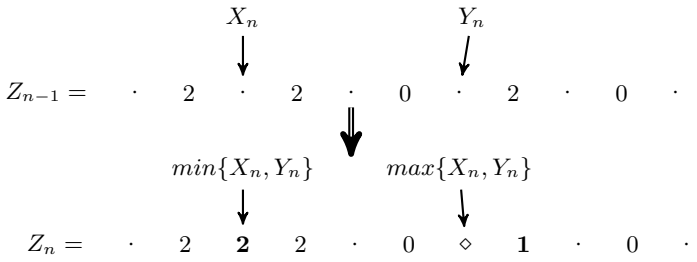


Fig. 5: Insertion in process $Ham_{2,INT}$. Insertion positions are marked with a dot. Positions affected by the insertion are in bold.

7.2 Details for the Proof of Corollary 14

The one-counter PDA has input alphabet $0, 1, 2, \ldots, k$. Its stack alphabet contains two special stack symbols,the bottom symbol Z and another "counting" symbol $*$. Considering the the transition of P_k are informally defined as follows:

- P_k starts with the stack consisting of the symbol Z. If the first letter is not a k, P_k immediately rejects. Otherwise it pushes a $*$ on the stack.
- on reading any subsequent k, P_k pushes a $*$ symbol on stack.
- on reading any symbol $i \in 1 \ldots k-2$, P_k attempts to pop $k-i-1$ stars from the stack. If this ever becomes impossible (by reaching Z), P_k immediately rejects.
- P_k ignores all $k-1$ symbols, proceeding without changing the content of the stack.
- If, while reaching the end of the word, the stack still contains a star, P_k accepts.

It involves applying the pumping lemma for regular languages to words $w_{k,n} = k^{n(k-1)+1}0^n \in L_H^k$. We infer that for large enough n, $w_{k,n} = w_1w_2w_3$, with w_2 nonempty and consisting of k's only, such that for every $l \geq 0$, $w_1w_2^lw_3 \in L_H^k$. We obtain a contradiction by letting $l = 0$, thus obtaining a word z that cannot belong to L_H^k, since $|z|_k \leq (k-1)|z|_0$.

7.3 Proof of Corollary 30

We take a string $s \in S_k$,

$$s = k^{c+d+e} \diamond^{c+e} (k-2)^c \diamond^{c+d}$$

with $c, d, e \geq p$ (where p is the parameter in Ogden's Lemma. We mark all positions of $k-1$. Then $s = uvwxy$, with $uv^iwx^iy \in S_k$ for all $i \geq 0$. The "pumping blocks" v, x cannot consist of more than one type of symbols, otherwise the pumped strings would fail to be a member of $\{k\}^* \diamond^* \{k-1\}^* \diamond^*$.

Therefore no more than two blocks (of the four in s) get pumped. One that definitely gets pumped is the first block of diamonds. Taking large enough i we obtain a contradiction, since the block that fails to get pumped will eventually have smaller length than the (pumped) first block of diamonds.

7.4 Algorithm $ComputeMultiplicity_{INT}$ and its correctness

Input: $k \geq 1, w \in (\Sigma_k \cup \{\diamond\})^*$
Output: $F_{k,INT}(w)$

```
S := 0. w = w_1 w_2 ... w_n
if w ∉ L^k_{H,INT}
    return 0
if w == "k ◇"
    return 1
for i in 1:n
    if w_i == k
        for j in i+1:n
        if w_j ==' ◇'
            let z be the word obtained from w by removing w_i, w_j
            and (possibly) increasing by one first position t > j (if any) with
            0 ≤ w_t < k
            S := S + ComputeMultiplicity_{INT}(k, z)
return S
```

Fig. 6: $ComputeMultiplicity_{INT}(k, w)$

The justification of correctness of the algorithm $ComputeMultiplicity_{INT}$ is just as simple, and very similar to that of Algorithm 3: depending on the chosen positions X_n, Y_n, a move in the interval Hammersley process will insert a k, a $\diamond$ somewhere to its right, and *may* delete a life from the first possible position after the insertion of the $\diamond$.

To revert the move and find all strings z that yield w in one move, one has to consider all digits w_p of w having value k as candidates for the insertion. All digits w_q with value different from k in w, $q > p$, are candidates for the lifeline removal, provided they are the first nonzero digit to the right of a block (starting sometimes after p) consisting only of zeros and diamonds, with at least one diamond included (at a position $p < r < q$).

It is also possible that no lifeline removal takes place from z to w, although a k and a diamond are inserted. This happens when the larger of positions X_n, Y_n has no nonzero digit to its right.