Information Rate of Some Classes of Non-regular Languages: An Automata-theoretic Approach

Cewei Cui[a], Zhe Dang[a], Thomas R. Fischer[a], Oscar H. Ibarra[b]

[a] *School of Electrical Engineering and Computer Science*
Washington State University, Pullman, WA 99164, USA
[b] *Department of Computer Science*
University of California, Santa Barbara, CA 93106, USA

Abstract

We show that the information rate of the language accepted by a reversal-bounded deterministic counter machine is computable. For the nondeterministic case, we provide computable upper bounds. For the class of languages accepted by multi-tape deterministic finite automata, the information rate is computable as well.

Keywords: automata, formal languages, Shannon information

1. Introduction

A software system often interacts with its environment. The complexity of an average observable event sequence, or behavior, can be a good indicator of how difficult it is to understand its semantics, test its functionality, etc. This is particularly true considering the fact that a modern software system is often too complex to analyze algorithmically by looking at the source code, line by line. Instead, the system is treated as a black-box whose behaviors can be observed by running it (with provided inputs), i.e., testing. One source to obtain all of the system's intended behaviors is from the design, though whether an intended behavior is the system's actual behavior must still be confirmed through testing. Despite this, the problem of computing the complexity of an average intended behavior from the design is important; in particular, the complexity can be used to estimate the cost of testing, even at the design stage where the code is not available yet.

In principle, a behavior is a word and the design specifies a set of words, i.e., a language L. There has already been a fundamental notion shown below, proposed by Shannon [20] and later Chomsky and Miller [4], that we have evaluated through experiments over C programs [23], fitting our need for the aforementioned complexity. For a number n, we use $S_n(L)$ to denote the number of words in L whose length is n. The *information rate* λ_L of L is defined as

$$\lambda_L = \lim_{n\to\infty} \frac{\log S_n(L)}{n}.$$

When the limit does not exist, we take the upper limit, which always exists for a finite alphabet. Throughout this paper, the logarithm is base 2. The rate is closely related to data compression ratio [10] and hence has immediate practical applications [6, 5, 13, 9]. Information rate is a real number. Hence, as usual, when we say that the rate is computable, it means that we have an algorithm to compute the rate up to any given precision (i.e., first n digits, for any n). A fundamental result is in the following.

Theorem 1. *The information rate of a regular language is computable [4].*

The case when L is non-regular (e.g., L is the external behavior set of a software system containing (unbounded) integer variables like counters and clocks) is more interesting, considering the fact that a complex software system nowadays is almost always of infinite-state and the notion of information rate has been used in software testing [23, 24]. However, in such a case, computing the information rate is difficult (sometimes even not computable [14]) in general. Existing results, such as unambiguous context-free languages [15], Lukasiewicz-languages [21], and regular timed languages [2], are limited and mostly rely on Mandelbrot generating functions and the theory of complex/real functions, which are also difficult to generalize.

In this paper, instead of taking the path of Mandelbrot generating functions, we use automata-theoretic approaches to compute the information rate for some classes of non-regular languages, including languages accepted by machines equipped with restricted counters. Our approaches make use of the rich body of techniques in automata theory developed in the past several decades and, as we believe, the approaches themselves can also be applied to more general classes of languages.

We first investigate languages accepted by reversal-bounded nondeterministic counter machines [11]. A counter is a nonnegative integer variable that can be incremented by 1, decremented by 1, or stay unchanged. In addition, a counter can be tested against 0. Let k be a nonnegative integer. A *nondeterministic k-counter machine (NCM)* is a one-way nondeterministic finite automaton, with input alphabet Σ, augmented with k counters. For a nonnegative integer r, we use NCM(k,r) to denote the class of k-counter machines where each counter is *r-reversal-bounded*; i.e., it makes at most r alternations between nondecreasing and nonincreasing modes in any computation; e.g., the following counter value sequence

$$0\ 0\ 1\ 2\ 2\ 3\ \underline{3\ 2}\ 1\ 0\ \underline{0\ 1}\ 1$$

is of 2-reversal, where the reversals are underlined. For convenience, we sometimes refer to a machine M in the class as an NCM(k,r). In particular, when k and r are implicitly given, we call M as a *reversal-bounded NCM*. When M is deterministic, we use 'D' in place of 'N'; e.g., DCM. As usual, $L(M)$ denotes the language that M accepts.

Reversal-bounded NCMs have been extensively studied since their introduction in 1978 [11]; many generalizations are identified; e.g., ones equipped with

multiple tapes, with two-way tapes, with a stack, etc. In particular, reversal-bounded NCMs have found applications in areas like Alur and Dill's [1] time-automata [8, 7], Paun's [19] membrane computing systems [12], and Diophantine equations [22].

In this paper, we show that the information rate of the language L accepted by a reversal-bounded DCM is computable. The proof is quite complex. We first, using automata-theoretic techniques, modify the language into essentially a regular language, specified by an unambiguous regular expression that is without nested Kleene stars, further constrained by a Presburger formula on the symbol counts in the words of the regular language. We show that the information rate of L can be computed through the information rate of the constrained language, where the latter can be reduced to a simple and solvable convex minimization problem. Unfortunately, we are not able to generalize the technique to reversal-bounded NCM. However, it is known [3] that a reversal-bounded NCM can be made to be one with counter values linearly bounded (in input size). Using this fact, we are able to obtain a computable upper bound on the rate when a reversal-bounded NCM is considered. We also consider the case when the reversal-bounded NCM does not make a lot of nondeterministic choices (i.e., sublinear-choice). In this case, the rate is shown computable. The result leads us to study a class of languages accepted by multi-tape DFAs. The information rate of such a multi-tape language is computable as well.

2. Information rate of languages accepted by reversal-bounded counter machines

We now recall a number of definitions that will be used later. Let N be the set of nonnegative integers and k be a positive number. A subset S of N^k is a linear set if there are vectors $\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_t$, for some t, in N^k such that $S = \{\mathbf{v} | \mathbf{v} = \mathbf{v}_0 + b_1\mathbf{v}_1 + \cdots + b_t\mathbf{v}_t, b_i \in N\}$. S is a semilinear set if it is a finite union of linear sets. Let $\Sigma = \{a_1, \cdots, a_k\}$ be an alphabet. For each word $\alpha \in \Sigma^*$, define the Parikh map [18] of α to be the vector $\#(\alpha) = (\#_{a_1}(\alpha), \cdots, \#_{a_k}(\alpha))$, where each symbol count $\#_{a_i}(\alpha)$ denotes the number of symbol a_i's in α. For a language $L \subseteq \Sigma^*$, the Parikh map of L is the set $\#(L) = \{\#(\alpha) : \alpha \in L\}$. The language L is semilinear if $\#(L)$ is a semilinear set. There is a classic result needed in the paper:

Theorem 2. *Let M be a reversal-bounded NCM. Then $\#(L(M))$ is a semilinear set effectively computable from M [11].*

Let Y be a finite set of integer variables. An atomic Presburger formula on Y is either a linear constraint $\sum_{y \in Y} a_y y < b$, or a mod constraint $x \equiv_d c$, where a_y, b, c and d are integers with $0 \leq c < d$. A Presburger formula can always be constructed from atomic Presburger formulas using $\neg$ and $\wedge$. It is known that Presburger formulas are closed under quantification. Let S be a set of k-tuples in N^k. S is Presburger definable if there is a Presburger formula $P(y_1, \cdots, y_k)$ such that the set of nonnegative integer solutions is exactly S. It is well-known that S is a semilinear set iff S is Presburger definable.

Let M be a reversal-bounded deterministic counter machine. The main result of this paper shows that the information rate of $L(M)$ is computable. The proof has four steps. First, we show that the information rate of $L(M)$ can be computed through the information rate of a counting language (defined in a moment) effectively constructed from M. Second, we show that the information rate of a counting language can be computed through the information rate of a counting replacement language (also defined in a moment) effectively constructed from the counting language. Third, we show that the information rate of a counting replacement language can be computed through the information rate of a simple counting replacement language. Finally, we show that the information rate of a simple counting replacement language is computable.

A *counting* language L is specified by a regular language and a Presburger formula such that L is exactly the set of all words w in the regular language with the Parikh map $\#(w)$ satisfying the Presburger formula. For instance, $\{a^n b^{2n} c^{3n} : n \geq 0\}$ and $\{w : \#_a(w) = 2\#_b(w) = 3\#_c(w), w \in (a+b+c)^*\}$ are counting languages.

Lemma 1. *Suppose that M is a reversal-bounded deterministic counter machine. There is a counting language L, effectively constructed from M, such that L and $L(M)$ have the same information rate; i.e., $\lambda_L = \lambda_{L(M)}$.*

Proof. Suppose that M is a DCM(k,r), for some k and r. That is, the counters in M are, say, $x_1, \cdots, x_k$. Without loss of generality, we assume that M starts and accepts with counter values being zero and every counter increments at least once in between. Furthermore, we need only consider the case when $r = 1$ (and every counter makes exactly one reversal). This is because an r-reversal-bounded counter can be easily simulated by $\lceil \frac{r}{2} \rceil$ 1-reversal-bounded counters [11].

We first show that M can be "simulated" by a finite automaton M' as follows. When M runs on an input, every 1-reversal counter x_i in M is simulated by two monotonic (i.e., 0-reversal) counters x_i^+ and x_i^- in M' that counts the number of increments and the number of decrements, respectively, made to x_i. During the run, before the reversal of x_i (M' "knows" the point of reversal), a counter test of "$x_i = 0$?" in M is simulated using M''s finite memory. After the reversal, a counter test of "$x_i = 0$?" in M is always simulated as "no" in M' until M' reads a special symbol $\clubsuit_i$ from the input. After reading this special symbol, the counter test is simulated as "yes" in M' and M' makes sure that there are no further counter increments made to x_i^-. Hence, the special symbol $\clubsuit_i$ resembles the first "time" when x_i decrements to 0 in M. At the end of input, M' accepts when M accepts. Up to now, since the monotonic counters in M' do not contribute to the state transitions in M', M' is indeed a finite automaton. Notice that M' runs on input w' obtained by inserting k (which is a constant) special symbols $\clubsuit_1, \cdots, \clubsuit_k$ into w. Clearly, if w is accepted by M, then there is a way to insert the k special symbols into w such that the resulting w' is accepted by M'. However, the inverse is not necessarily true. This is because

it requires the following Presburger test

$$\bigwedge_{1 \leq i \leq k} x_i^+ = x_i^- \tag{1}$$

to be successful when M' is at the end of w'. We now use a technique that removes the monotonic counters from M'.

When M' runs on an input, on every input symbol b, M' runs from a state s, reading 0 or more ϵ symbols, and then actually reading the b and, after this, entering a state s'. This is called a round. We use a *round symbol* $[s, b, s']$ to denote the round and use $P_{[s,b,s']}$ to denote the set of vectors of net increments made to the monotonic counters during the round. The proof of the following claim is an exercise which constructs a reversal-bounded NCM to accept unary encoding of vectors in the set and uses Theorem 2:

(Claim 1) $P_{[s,b,s']}$ is a Presburger definable set.

Consider an input word $w' = b_0 \cdots b_{n-1}$ in $L(M')$ for some n. Suppose that, when M' runs on the input, a sequence $[w']$ of rounds is as follows:

$$[s_0, b_0, s_1][s_1, b_1, s_2] \cdots [s_{n-1}, b_{n-1}, s_n] \tag{2}$$

where s_0 is the initial state and s_n is an accepting state. The Presburger test in (1), denoted by $Q(Y)$, where Y is the vector of the $2k$ monotonic counters in M', is equivalent to $Q(\Delta)$, for some

$$\Delta = \Delta_{[s_0,b_0,s_1]} + \Delta_{[s_1,b_1,s_2]} \cdots + \Delta_{[s_{n-1},b_{n-1},s_n]}, \tag{3}$$

where each

$$\Delta_{[s_{j-1},b_{j-1},s_j]} \in P_{[s_{j-1},b_{j-1},s_j]}. \tag{4}$$

That is, the counter values in (1) are the accumulated counter net increments Δ in all the rounds as shown in (3) and hence, the Presburger test in (1) can be performed directly over the Δ.

We now use $\#_{[s,b,s']}$ to denote the number of appearances of the round symbol $[s, b, s']$ in (2) and introduce the notation $\#_{[s,b,s']} \cdot P_{[s,b,s']}$ to denote the set of all the summations of $\#_{[s,b,s']}$ number of (not necessarily distinct) vectors in $P_{[s,b,s']}$. Clearly, the Δ in (3) can be re-written as

$$\Delta = \sum_{\substack{[s,b,s'] \text{ appearing in (2)}, \\ \boldsymbol{\Delta}_{[s,b,s']} \in \#_{[s,b,s']} \cdot P_{[s,b,s']}}} \boldsymbol{\Delta}_{[s,b,s']}. \tag{5}$$

We now claim that

(Claim 2) The formula $\boldsymbol{\Delta}_{[s,b,s']} \in \#_{[s,b,s']} \cdot P_{[s,b,s']}$ is Presburger in $\boldsymbol{\Delta}_{[s,b,s']}$ and $\#_{[s,b,s']}$.

The proof of the claim will be shown in a moment. We use $\#$ to denote the vector of the counts $\#_{[s,b,s']}$, noticing that there are totally $|S| \times |\Sigma| \times |S|$ round symbols, where $|S|$ is the number of states in M' and $|\Sigma|$ is the size of its input alphabet. From the claim, the equation in (5) and hence (3) is a Presburger

formula in Δ and $\#$, after eliminating quantified variables $\mathbf{\Delta}_{[s,b,s']}$'s. We use $\hat{Q}(\Delta, \#)$ to denote the formula. Therefore, the Presburger test is equivalent to the following Presburger formula

$$\exists \Delta. Q(\Delta) \wedge \hat{Q}(\Delta, \#),$$

which is denoted by $\dot{Q}(\#)$.

We now define a language L' to be the set of all round symbol sequences $[w']$ in (2) for all w' accepted by M'. L' is regular. Let L'' be the counting language obtained from words $[w']$ in L' satisfying Presburger formula $\dot{Q}(\#([w']))$.

In summary, we have the following: w is accepted by M iff there is a w' (after inserting the aforementioned k special symbols $\clubsuit_i$'s into w) accepted by M' and the acceptance is witnessed by the round symbol sequence $[w'] \in L''$ shown in (2). Since M is deterministic, the mapping from $w \in L(M)$ to $[w'] \in L''$ is one-to-one (while it is not necessarily true for the mapping from $w \in L(M)$ to $[w'] \in L'$). Notice also that the length of $[w']$ in (2) equals $|w'| = |w| + k$. Because k is a constant, directly from definition, $\lambda_{L(M)} = \lambda_{L''}$. The result follows.

To complete the proof, we still need show Claim 2. We first assume a unary encoding $[\delta]$ for integers δ; e.g., 00000 for -5 and 11111 for +5, where the 0 and 1 are the basis. From Claim 1, $P_{[s,b,s']}$ is therefore a semilinear set. It is known that, for every semilinear set, one can construct a regular language whose Parikh map is exactly the semilinear set. This can be shown directly using the definition of semilinear set; e.g., the semilinear set $\{(1+t, t) : t \geq 0\} \cup \{(2+2t, 3t) : t \geq 0\}$ corresponds to the regular language $a(ab)^* + aa(aabbb)^*$. Let $L_{[s,b,s']}$ be a regular language (on alphabet, say, $\{c_1, \cdots, c_{2k}\}$), accepted by an NFA $M_{[s,b,s']}$, corresponding to the semilinear set $P_{[s,b,s']}$. We construct a reversal-bounded NCM $\dot{M}$ as follows. Working on an input

$$[\#_{[s,b,s']}] \diamondsuit [\delta_1] \diamondsuit [\delta_2] \diamondsuit \cdots [\delta_{2k}], \tag{6}$$

$\dot{M}$ uses reversal-bounded counters $y_0, \cdots, y_{2k}$, initially being zero. Again, notice that, on the input, the $2k+1$ unary encoding blocks use distinct basis. $\dot{M}$ repeatedly simulates the NFA $M_{[s,b,s']}$ from the NFA's initial state to accepting state. On each simulation, $\dot{M}$ guesses an input (one symbol by one symbol) for the NFA. Along with the simulation, $\dot{M}$ uses monotonic counters $y_1, \cdots, y_{2k}$ to count, respectively, the number of $c_1, \cdots, c_{2k}$ that the NFA has read so far. When a simulation ends, $\dot{M}$ increments the counter y_0. After a number of rounds of simulations, $\dot{M}$ nondeterministically decides to shut down the simulation. At this moment, $\dot{M}$ checks that the values $\#_{[s,b,s']}$, $\delta_1, \cdots, \delta_{2k}$ on the input (6) are exactly the same as the values stored in counters $y_0, \cdots, y_{2k}$, respectively. The checking can be done by reading through each unary block of the input while decrementing the counter corresponding to the block to zero. In this case, $\dot{M}$ accepts the input in (6). Clearly, $\dot{M}$ is indeed reversal-bounded and, $\mathbf{\Delta}_{[s,b,s']} \in \#_{[s,b,s']} \cdot P_{[s,b,s']}$ iff the input in (6) is accepted by $\dot{M}$ with $\mathbf{\Delta}_{[s,b,s']} = (\delta_1, \cdots, \delta_k)$. Since $L(\dot{M})$ is a semilinear language (Theorem 2), there

is a Presburger formula $Q_{[s,b,s']}$ such that the input in (6) is accepted by $\dot{M}$ iff $Q_{[s,b,s']}(\#_{[s,b,s']}, \delta_1, \cdots, \delta_{2k})$. The claim follows since the desired Presburger formula is $Q_{[s,b,s']}(\#_{[s,b,s']}, \mathbf{\Delta}_{[s,b,s']})$. □

The proof of Lemma 1 cannot be generalized to the case where M is a reversal-bounded NCM. This is because, in establishing the one-to-one correspondence in the proof, one requires that M is deterministic.

The second step of the proof for the main theorem is to establish that a counting language can be "converted" into a counting replacement language, which is defined as follows. A *replacement system* G is specified by k levels, for some $k > 0$, where

- for each $1 \leq i \leq k$, the i-level has a distinct alphabet Σ_i (i.e., $\Sigma_i \cap \Sigma_j = \emptyset$ if $i \neq j$);
- the first level contains a finite set of *base words* w on alphabet Σ_1;
- for each $1 < i \leq k$, the i-level contains a finite set of replacement rules in the form of
$$a \leftarrow awa, \tag{7}$$
where $a \in \Sigma_{i-1}$ and w is a word on alphabet Σ_i;
- all of the words w's mentioned above satisfy the following property: there is no symbol appearing simultaneously in any two such words and, for any symbol, if it appears in w, it appears only once.

When a replacement rule in (7) is applied on a word u, the result is to replace an appearance a in u with awa. A word u is *generated by* G if it is the result of the following sequence of replacements (in this order): starting from a basic word in the first level, we apply replacement rules in the second level for zero or more times, replacement rules in the third level for zero or more times, $\cdots$, replacement rules in the k-level for zero or more times. We use $L(G)$ to denote the set of all u's generated by G. Clearly, $L(G)$ is a regular language. A *regular replacement language* is $L(G)$ for some replacement system G.

For instance, consider a replacement system G with basic word abc, and the second level replacement rules $a \leftarrow adea, a \leftarrow afa, b \leftarrow bgb$, and the third level replacement rules $e \leftarrow ehe$. After applying the second level replacement rules for zero or more times, we obtain words in $a(dea + fa)^*b(gb)^*c$. Subsequently, after applying the third level replacement rules, we finally get $L(G) = a(de(he)^*a + fa)^*b(gb)^*c$.

A *counting replacement language* is a counting language specified by a regular replacement language and a Presburger formula.

The proof of Lemma 2 is intuitively not difficult. Suppose that the counting language L is specified by a regular language $L_{\rm R}$ and a Presburger formula P. The proof works on the set of accepting runs (i.e., state-symbol sequences) of a DFA M accepting the regular language $L_{\rm R}$. The runs have an unambiguous way to decompose back to loops, as specified in a replacement system G. This idea

is actually classic, e.g., in the textbook construction from finite automata to regular expressions. Consequently, the Presburger formula needs to be modified after the decomposition.

Lemma 2. *Suppose that L is a counting language. There is a counting replacement language L', effectively constructed from the specification of L, such that $\lambda_L = 2\lambda_{L'}$.*

Proof. Let L be a counting language specified by a regular language L_{R} and a Presburger formula P. By definition, for every w, $w \in L$ iff $w \in L_{\mathrm{R}}$ and $P(\#(w))$ holds. Suppose that L_{R} is accepted by a DFA M (without ϵ-moves) and consider a word $w = a_0 \cdots a_{n-1}$, for some n. A run of M is a sequence

$$s_0 a_0 s_1 \cdots s_{n-1} a_{n-1} s_n \tag{8}$$

such that each $s_i a_i s_{i+1}$ is a transition of M that moves from s_i to s_{i+1} while reading a_i. The run is accepting if s_0 is the initial state and s_n is an accepting state of M (without loss of generality, we assume that the two states are distinct.). Since M is deterministic, there is a unique accepting run $\mathrm{Run}(w)$ for $w \in L(M)$. We use $\mathrm{Run}(M) = \{\mathrm{Run}(w) : w \in L(M)\}$ to denote the set of all accepting runs of M. Define $\mathrm{Run}_P(M) = \{\mathrm{Run}(w) : P(\#(w)) \text{ and } w \in L(M)\}$. Consequently, the information rate of $\mathrm{Run}_P(M)$ is exactly half of the information rate of L. We now convert $\mathrm{Run}_P(M)$ into the desired counting replacement language L'.

We first fix an arbitrary ordering of all the states in M, say, $q_1, q_2, \cdots, q_k$ and consider the run α specified in (8). The run starts with s_0 and ends with s_n and sometimes is simply written as $s_0 \rightsquigarrow s_n$. We say that α passes q *internally* if at least one of $s_1, \cdots, s_{n-1}$ is q. Two runs $q \rightsquigarrow q'$ and $q' \rightsquigarrow q''$ can be concatenated as $q \rightsquigarrow q' \rightsquigarrow q''$. Two sets R_1 and R_2 of runs are concatenated as $R_1 R_2 = \{q \rightsquigarrow q' \rightsquigarrow q'' : q \rightsquigarrow q' \in R_1, q' \rightsquigarrow q'' \in R_2\}$. For a set of runs R, we use $R^{-Q}_{qq'}$ to denote all the runs $q \rightsquigarrow q'$ in R that does not pass any $q'' \in Q$ internally. Let $\mathcal{R}$ be the set of all runs in M. Inspired by the textbook algorithm of computing a regular expression from a DFA, we observe the following equation, where q_{init} and q_{accept} are the initial state and an accepting state, respectively:

$$\mathrm{Run}(M) = \bigcup_{q_{\mathrm{accept}}} \mathcal{R}^{-\emptyset}_{q_{\mathrm{init}} q_{\mathrm{accept}}}\ , \tag{9}$$

where each

$$\mathcal{R}^{-\emptyset}_{q_{\mathrm{init}} q_{\mathrm{accept}}} = \mathcal{R}^{-\{q_1\}}_{q_{\mathrm{init}} q_{\mathrm{accept}}} \cup \mathcal{R}^{-\{q_1\}}_{q_{\mathrm{init}} q_1} (\mathcal{R}^{-\{q_1\}}_{q_1 q_1})^* \mathcal{R}^{-\{q_1\}}_{q_1 q_{\mathrm{accept}}}. \tag{10}$$

Equation (9) repeats the definition $\mathrm{Run}(M)$ of accepting runs, while equation (10) says that either an accepting run does not pass q_1 internally, or, it can be decomposed into the concatenation of the following segments: $q_{\mathrm{init}} \rightsquigarrow q_1$, zero or more loops on q_1, and $q_1 \rightsquigarrow q_{\mathrm{accept}}$, where each segment does not pass q_1 internally. The equation in (10) can be further generalized to, for each $0 \le i < k$,

$$\mathcal{R}^{-Q_i}_{qq'} = \mathcal{R}^{-Q_{i+1}}_{qq'} \cup \mathcal{R}^{-Q_{i+1}}_{qq_{i+1}} (\mathcal{R}^{-Q_{i+1}}_{q_{i+1} q_{i+1}})^* \mathcal{R}^{-Q_{i+1}}_{q_{i+1} q'}, \tag{11}$$

where $Q_i = \{q_1, \cdots, q_i\}$ and $Q_0 = \emptyset$. Notice that $\mathcal{R}_{qq'}^{-Q_k}$ is the set of (one-step) transitions from q to q'. Also notice that, for every run $\alpha \in \mathcal{R}_{qq'}^{-Q_i}$, the way that the α is, exclusively, in either $\mathcal{R}_{qq'}^{-Q_{i+1}}$ or $\mathcal{R}_{qq_{i+1}}^{-Q_{i+1}}(\mathcal{R}_{q_{i+1}q_{i+1}}^{-Q_{i+1}})^*\mathcal{R}_{q_{i+1}q'}^{-Q_{i+1}}$. When it is the latter case, the way that the α is decomposed into the following concatenation is unique: $\mathcal{R}_{qq_{i+1}}^{-Q_{i+1}}$, zero or more loops with each in the form $\mathcal{R}_{q_{i+1}q_{i+1}}^{-Q_{i+1}}$, and followed by $\mathcal{R}_{q_{i+1}q'}^{-Q_{i+1}}$.

We now delete some loops internally in $\alpha \in \mathcal{R}_{qq'}^{-Q_i}$ as follows (in this order): (deletion procedure)

- all the loops in $\mathcal{R}_{q_{i+1}q_{i+1}}^{-Q_{i+1}}$; i.e., loops at q_{i+1} that do not internally pass $q_1, \cdots, q_{i+1}$,
- all the loops in $\mathcal{R}_{q_{i+2}q_{i+2}}^{-Q_{i+2}}$; i.e., loops at q_{i+2} that do not internally pass $q_1, \cdots, q_{i+2}$,
- $\cdots$.

After the deletions, we *derive* a unique basic run β for the given α. We use $[\beta]$ to denote the set of all α's in $\mathcal{R}_{qq'}^{-Q_i}$ that derive the basic run β. Clearly, $[\beta] \cap [\beta'] = \emptyset$ when $\beta \neq \beta'$. The set of all such basic runs obtained from all α's in $\mathcal{R}_{qq'}^{-Q_i}$ is denoted by $B_{qq'}^{-Q_i}$. We then have

$$\mathcal{R}_{qq'}^{-Q_i} = \bigcup_{\beta \in B_{qq'}^{-Q_i}} [\beta], \tag{12}$$

where the union is disjoint. The set $B_{qq'}^{-Q_i}$ is finite, for every $0 \leq i \leq k$. This is because the length of a basic run in $B_{qq'}^{-Q_i}$ is at most $1 + 2 \cdot 2^{k-i}$. We shall emphasize that a basic run can still contain a loop.

Conversely, every α can be generated in a unique way from a basic run β in $B_{qq'}^{-Q_i}$ by the following insertion procedure (which is the result of reversing the aforementioned deletion procedure):

(insertion procedure)

- for each appearance (there is at most one) of q_{i+1} in the original β, we insert zero or more loops in $\mathcal{R}_{q_{i+1}q_{i+1}}^{-Q_{i+1}}$, where each such loop is generated, recursively, from a basic run in $B_{q_{i+1}q_{i+1}}^{-Q_{i+1}}$;
- for each appearance (there are at most two) of q_{i+2} in the original β, we insert zero or more loops in $\mathcal{R}_{q_{i+2}q_{i+2}}^{-Q_{i+2}}$, where each such loop is generated, recursively, from a basic run in $B_{q_{i+2}q_{i+2}}^{-Q_{i+2}}$;
- for each appearance (there are at most four) of q_{i+3} in the original β, we insert zero or more loops in $\mathcal{R}_{q_{i+3}q_{i+3}}^{-Q_{i+3}}$, where each such loop is generated, recursively, from a basic run in $B_{q_{i+3}q_{i+3}}^{-Q_{i+3}}$;

- $\cdots$.

Clearly, two distinct ways A and B that the β uses will generate two distinct runs. For instance, in the first round, if A inserts two q_{i+1}-loops while B inserts one q_{i+1}-loop, no matter what is in the remaining rounds, A (resp. B) will generate a run internally with three (resp. two) q_{i+1}'s. In the first round, suppose that both A and B insert only one loop. However, the q_{i+1}-loop inserted by A is different than the one by B. Clearly, no matter what is in the remaining rounds, both A and B will generate a run that internally has exactly two q_{i+1}'s. However, the two runs generated by A and B respectively are distinct. This is because, between the two q_{i+1}'s, the run generated by A (resp. B) is the q_{i+1}-loop inserted by A (resp. by B). The idea can be similarly used when A and B are distinct in the subsequent runs. In summary, combining the fact in (12), we are safe to conclude that every $\alpha \in \mathcal{R}_{qq'}^{-Q_i}$ can be **unambiguously** generated from a basic run β in $B_{qq'}^{-Q_i}$, using the above insertion procedure.

We now emphasize two points. First, when we expand the recursions inside the insertion procedure, it becomes a procedure of inserting basic runs (which are loops) in $B_{q_jq_j}^{-Q_i}$, $j > i$, repeatedly into a basic run in $B_{qq'}^{-Q_i}$. Second, because of the unambiguity, we can properly renaming each appearance of a symbol in all of the basic runs so that a run generated by a β in $B_{qq'}^{-Q_i}$ is one-to-one correspondent to the run after the renaming. These two points give a replacement system G for $\mathcal{R}_{qq'}^{-Q_i}$ below:

- The first level of G contains all the basic runs $q\gamma q'$ in $B_{qq'}^{-Q_i}$. For every individual appearance of a symbol, say e, in γ, we rename it with a new symbol $(1, \gamma, i, l, e)$, where l is the position of the appearance in γ;
- The second level contains all the replacement rules in the form of

$$(1, \gamma, i, l, q_j) \leftarrow (1, \gamma, i, l, q_j)\gamma'(1, \gamma, i, l, q_j) \tag{13}$$

  where $j \geq i+1$, $(1, \gamma, i, l, q_j)$ is a symbol in the first level, and $q_j\gamma' q_j$ is a basic run in $B_{q_jq_j}^{-Q_j}$. We now, similar to the case of the first level, for every rule in (13), rename every individual appearance of every symbol e in γ' with a new symbol $(2, \gamma', j, l', e)$, where l' is the position of the appearance in γ';
- The third level contains all the replacement rules in the form of

$$(2, \gamma, i+1, l, q_j) \leftarrow (2, \gamma, i+1, l, q_j)\gamma'(2, \gamma, i+1, l, q_j) \tag{14}$$

  where $j \geq i+2$, $(2, \gamma, i+1, l, q_j)$ is a symbol in the second level, and $q_j\gamma' q_j$ is a basic run in $B_{q_jq_j}^{-Q_j}$. We now, similar to the case of the first level, for every rule in (14), rename every individual appearance of every symbol e in γ' with a new symbol $(3, \gamma', j, l', e)$, where l' is the position of the appearance in γ';
- $\cdots$.

In G shown above, a symbol e, which is either a state symbol or an input symbol in M, can be renamed into finitely many new symbols; we use $\langle e \rangle$ to denote the set of the new symbols. Notice that the p and q', respectively as the first and last symbol in a basic run in the first level, are not renamed in G. The language $L(G)$ generated by the replacement system G is "almost" $\mathcal{R}_{qq'}^{-Q_i}$ in the following sense: for every α, we have that $\alpha \in L(G)$ iff $\bar{\alpha} \in \mathcal{R}_{qq'}^{-Q_i}$, where $\bar{\alpha}$ is the result of replacing every symbol in $\langle e \rangle$ in α with e, for all e's that are renamed. Additionally, because of the unambiguity, the mapping from α to $\bar{\alpha}$ is one-to-one and, obviously, length-preserving. Suppose now that P is a Presburger formula over the counts of symbols in the alphabet of $\mathcal{R}_{qq'}^{-Q_i}$. We now replace, for every symbol e renamed in G, $\#_e$ with

$$\sum_{e' \in \langle e \rangle} \#_{e'}, \tag{15}$$

in P and use P' to denote the resulting Presburger formula over the counts of symbols in the alphabet of G. Equation (15) says that the symbols counts in α "remain" in $\bar{\alpha}$. That is, for every α, we have that $\alpha \in L(G) \wedge P'(\#(\alpha))$ iff $\bar{\alpha} \in \mathcal{R}_{qq'}^{-Q_i} \wedge P(\#(\bar{\alpha}))$. Let L' be the counting replacement language specified by $L(G)$ and P'. We then have $\lambda_{\mathcal{R}_{qq'}^{-Q_i}} = \lambda_{L'}$. Now taking $q = q_{\text{init}}$, $q' = q_{\text{accept}}$, $i = 0$ (noting that $Q_0 = \emptyset$), we immediately have $\lambda_{\mathcal{R}_{q_{\text{init}}\, q_{\text{accept}}}^{-\emptyset}} = \lambda_{L'}$. To emphasize that the L' is for the chosen q_{accept}, we write the L' as $L'_{q_{\text{accept}}}$, the G as $G_{q_{\text{accept}}}$, and the P' as $P'_{q_{\text{accept}}}$.

Without of loss of generality, we assume that alphabets of $L'_{q_{\text{accept}}}$ are distinct. We can easily construct a replacement system G, as the "union" of all the $G_{q_{\text{accept}}}$'s, to generate

$$L(G) = \bigcup_{q_{\text{accept}}} L(G_{q_{\text{accept}}}).$$

Now, take P' as

$$\bigvee_{q_{\text{accept}}} P'_{q_{\text{accept}}} \wedge \#_{q_{\text{accept}}} = 1.$$

(Notice that q_{accept} at the end of a basic run in the first level of $G_{q_{\text{accept}}}$ (as well as in G) is not renamed.) Combining the fact that (9) is a disjoint union, the result follows since the counting replacement language L', specified by $L(G)$ and P', satisfies $\lambda_{L'} = \lambda_{\text{Run}_P(M)}$. □

A replacement system is *simple* if it has at most two levels. A regular replacement language is *simple* if it is generated by a simple replacement system. A counting replacement language is *simple* if it is specified by a simple regular replacement language and a Presburger formula. The third step in the proof for the main theorem establishes that the information rate of a counting replacement language can be computed through the information rate of a simple counting replacement language.

The proof of Lemma 3 is difficult. We sketch the ideas used in the proof. In the lemma, L is a given counting replacement language. Hence, L can be specified by a regular expression with nested Kleene stars. In order to obtain the desired simple counting replacement language L', we must "collapse" the nested Kleene stars. There is a straightforward way for the collapsing. For instance, $((ab)^*c)^*$ can be converted into $(ab)^*c^+ + \Lambda$, where the latter one does not have any nested Kleene star and the two expressions have the same Parikh map. However, such a straightforward approach, used in computing the Parikh map of a regular language, has a problem in establishing Lemma 3. Using the approach, the resulting simple counting replacement language L' may not preserve the information rate. Therefore, we need a more sophisticated approach that, roughly speaking, keeps both the information rate (up to a constant ratio of K) and the Parikh map. The proof uses a one-to-one encoding that stretches a word in L (this is where the constant K in the lemma comes from) and moves around the nested loops in the word so that the resulting word contains at most one level of loops.

Lemma 3. *Suppose that L is a counting replacement language. There are a constant K and a simple counting replacement language L', constructed from the specification of L, such that $\lambda_L = K \cdot \lambda_{L'}$.*

PROOF. Let L be specified by $L(G)$ and a Presburger formula P, where G is a replacement system. It suffices to consider the case when the first level of G contains exactly one base word. The reason is as follows. Suppose that $w_1, \cdots, w_m$, for some $m \geq 1$, are all the bases words in G. G_i is a replacement system whose first level contains only one base word w_i and all other levels are exactly the same as those in G. Clearly, $L(G)$ is the disjoint union $\cup_i L(G_i)$. Let L_i be the counting replacement language specified by $L(G_i)$ and a Presburger formula P. Clearly, L is also the disjoint union $\cup_i L_i$ and hence $\lambda_L = \max_i \lambda_{L_i}$. Therefore, it suffices to show that the information rate of L_i, which contains only one base word, satisfies the theorem. Similarly, it suffices to consider the case when every rule in the second level is applied at least once (otherwise, the rule can be deleted).

Assume that G has $k > 2$ levels. Now we modify G into a new replacement system G' with $k-1$ levels. Let w be the only base word in the first level of G. Consider a symbol a that appears in w. By definition, a appears exactly once in w. If there is a rule $a \leftarrow aua$ (for some u) in the second level of G, then the a is called an *active symbol*. Now, let $w' = w$. Then, for every active symbol a and every rule $a \leftarrow aua$ (for some u) in the second level of G, we do the following:

- create a new rule $a \leftarrow a\langle au\rangle a$, where $\langle au\rangle$ is a new symbol;
- for each symbol b (from left to right) in u, append $\clubsuit_{\langle aub\rangle} b$ after w'; i.e., $w' := w'\clubsuit_{\langle aub\rangle} b$, where the delimiter $\clubsuit_{\langle aub\rangle}$ is a new symbol;
- create a new rule, for each symbol b in u, $b \leftarrow \widehat{baubb}$, where $\widehat{aub}$ is a new symbol.

At this moment, we add w' into the first level of G' and all the rules that are created into the second level of G'. Now, for each $i \geq 2$, we add all the rules in the $(i+1)$-level of G into the i-level of G'. Clearly, G' has only $k-1$ levels and has w' as its only base word.

We now explain the intuition underneath the construction of G'. Let a be an active symbol in the base word w of G. An application of a rule $a \leftarrow aua$ in the second level of G is simulated by an application, on the w part of the base word w' of G', of the rule $a \leftarrow a\langle au\rangle a$ in the second level of G'. Notice that, in G', there is no rule that can replace the symbol $\langle au\rangle$. This causes a problem. Suppose that b is a symbol in u and we have a rule $b \leftarrow bvb$ in the third level of G. After the application of $a \leftarrow aua$ in G, there could be zero or more applications of $b \leftarrow bvb$ later on the u. Where are we going to simulate these applications in G'? The key idea of the construction is to simulate these application of $b \leftarrow bvb$, in G', on the part $\clubsuit_{\langle aub\rangle}b$ of w'. Notice that, in w', this part is after the w part. But, there are still problems. There is only one $\clubsuit_{\langle aub\rangle}b$ part in w'. However, in G, multiple applications of $a \leftarrow aua$ will obviously create multiple instances of u's. Where are the u's in G'? In G', we treat the u's as m number of b's, for each b in u, where m is the number of instances of the u's. Hence, there must be a rule in G' to create those b's, which is exactly the job of the rule $b \leftarrow \widehat{baubb}$ created. Certainly, we need a Presburger constraint (defined in a moment) to carefully control the number of applications of each such rule $b \leftarrow \widehat{baubb}$. $m-1$ applications of the rule $b \leftarrow \widehat{baubb}$ result in the following sequence:

$$\widehat{baubb}\widehat{baub}\cdots\widehat{baubb} \tag{16}$$

with m number of b's and $(m-1)$ number of $\widehat{aub}$'s. Now, we go back to the aforementioned rule $b \leftarrow bvb$ in G. This rule can be applied, for zero or more times, over an instance u (suppose that it is the j-th instance) that was generated earlier as a result of applying $a \leftarrow aua$ in G. These applications of the rule $b \leftarrow bvb$ in G are simulated by the same rule in G' applied over the j-th b in (16). By definition of G, symbols a and b can be uniquely associated with the rule $a \leftarrow aua$. Hence, the applications of $b \leftarrow bvb$ in G' can be uniquely translated back to the applications of the rule $b \leftarrow bvb$ in G over the j-th instance of u's. All the higher-level rules, i.e., fourth level, and, subsequently, fifth level, etc., in G are faithfully simulated over the generated v's in G'. Notice that, these v's were initially generated by the rules $b \leftarrow bvb$, along with $b \leftarrow \widehat{baubb}$, from the $\clubsuit_{\langle aub\rangle}b$ in w'. In summary, the construction of G' defines a translation a word $\alpha \in L(G)$ to a word $\alpha' \in L(G')$.

We now look at a simple example. Consider a G where the base word is a, and the second level rule is $a \leftarrow aba$, and the third level rule is $b \leftarrow bcb$. Now, the G' is with base word $a\clubsuit_{\langle abb\rangle}b$, with second level rules: $a \leftarrow a\langle ab\rangle a$, $b \leftarrow \widehat{babbb}$, and $b \leftarrow bcb$. Take a word $\alpha = abcbcbaba \in L(G)$. This word corresponds, uniquely, to the word $\alpha' = a\langle ab\rangle a\langle ab\rangle a\clubsuit_{\langle abb\rangle}bcbcb\widehat{abbb}$ in $L(G')$. The part that is before the delimiter $\clubsuit_{\langle abb\rangle}$ in α' encodes two applications of $a \leftarrow aba$ in G over its base word a (so now we have $\beta = ababa$). The part $bcbcb\widehat{abbb}$ that is

after the delimiter $\clubsuit_{\langle abb\rangle}$ in α' encodes two words $bcbcb$ and b, delimited by the $\widehat{abb}$. The first word $bcbcb$ represents two applications of $b \leftarrow bcb$ in G over the first b in β, while the second word b represents zero application of $b \leftarrow bcb$ in G over the second b in β. The counts in α and α' are the following:

- the second level rules are simulated for the same number of times: $\#_a(\alpha) = \#_a(\alpha')$;
- the third level rules are also simulated for the same number of times: $\#_b(\alpha) = \#_b(\alpha')$, which is also $\#_{\langle ab\rangle}(\alpha') + (\#_b(\alpha') - \#_{\widehat{abb}}(\alpha') - 1)$;
- other counts in α also remain: $\#_c(\alpha) = \#_c(\alpha')$;
- α' should be *consistent*; i.e., be able to translate back to an α:

  > $\#_{\widehat{abb}}(\alpha')$ is correctly related to the number of times that $a \leftarrow aba$ is applied. That is, $\#_{\langle ab\rangle}(\alpha') \geq 1$, and, $\#_{\langle ab\rangle}(\alpha') - 1 = \#_{\widehat{abb}}(\alpha')$ (recall that, as we assumed at the very beginning of the proof, every second level rule is applied at least once).

The translation from an α to the unique α' is straightforward, as shown above. However, not every α' in $L(G')$ can be translated backwards uniquely. In above, the consistent constraint, placed over the symbol counts of α', is a Presburger formula to guarantee the unique backwards translation.

For a general G, the consistent constraint is similar:

> for each active symbol a and each second level rule $a \leftarrow aua$, $\#_{\widehat{au}}(\alpha')$ is correctly related to the number of times that $a \leftarrow aua$ is applied. That is, $\#_{\langle au\rangle}(\alpha') \geq 1$, and, for every symbol b in u,
>
> $$\#_{\langle au\rangle}(\alpha') - 1 = \#_{\widehat{aub}}(\alpha'). \tag{17}$$

The symbols counts in α also remain in α'; i.e., for every symbol a in G,

$$\#_a(\alpha) = \#_a(\alpha'). \tag{18}$$

If the lengths of α and α' were the same, then we would complete the construction of G' already since we could continue to collapse the G' till it has two levels. Unfortunately, the lengths are not the same. According to (18), α' is longer than α because the new symbols are introduced into w', for each active symbol a in w, each second level rule $a \leftarrow aua$, and for each symbol b in u:

$$\langle au\rangle, \widehat{aub}, \clubsuit_{aub}. \tag{19}$$

The counts of these symbols in α' are in the following, for each active symbol a:

- $\#_{\langle au\rangle}(\alpha')$ is the number of times that the second level rule $a \leftarrow aua$ in G is fired. Hence,

$$\sum_{\text{all second level rules } a\leftarrow aua} \#_{\langle au\rangle}(\alpha') = \#_a(\alpha) - 1; \tag{20}$$

- for every second level rule $a \leftarrow aua$, and for each symbol b in u, $\#_{\widehat{aub}}(\alpha')+1$ is the number of times that the second level rule $a \leftarrow aua$ in G is fired, according to (17). Combining the fact that $\#_{\clubsuit_{aub}}(\alpha') = 1$, we have, for every second level rule $a \leftarrow aua$,

$$\sum_{b \text{ in } u} \#_{\widehat{aub}}(\alpha') + \#_{\clubsuit_{aub}}(\alpha') = |u| \cdot \#_{\langle au \rangle}(\alpha'). \tag{21}$$

Using (21) and (20), for each active symbol a, the total counts of all the symbols in (19) together with a itself, in α', is therefore

$$N_a = \#_a(\alpha) - 1 + \sum_{\text{all second level rules } a \leftarrow aua} |u| \cdot \#_{\langle au \rangle}(\alpha') + \#_a(\alpha). \tag{22}$$

We now make N_a to be a linear term of $\#_a(\alpha)$. Let K_a be the LCM (Least Common Multiple) of the lengths $|u|$, for u in $a \leftarrow aua$; i.e.,

$$K_a = \text{lcm}(|u| : a \leftarrow aua \text{ is a second level rule}),$$

where, by default, any nonnegative integer is a multiple of 0. We modify every rule $a \leftarrow a\langle au \rangle a$, created earlier in G', into the following form:

$$a \leftarrow a\langle au \rangle \underbrace{\sharp_{au} \cdots \sharp_{au}}_{K_a - |u|} a, \tag{23}$$

where $\sharp_{au}$ is a new symbol. (When $|u| = 0$, we do modify the rule; i.e., there is no $\sharp_{au}$ in (23).) In fact, every symbol in the block $\sharp_{au} \cdots \sharp_{au}$ in (23) should be renamed into a new one; for notational convenience, we just leave them unrenamed since it won't cause any confusion. The purpose of the added block of length $K_a - |u|$ in (23) is to make every application of $a \leftarrow a\langle au \rangle a$ created earlier generate additional $K_a - |u|$ symbols. Now, (19) contains one more new symbol $\sharp_{au}$, and therefore, the N_a in (22), as the counts of all these new symbols as well as the a itself, is then modified into

$$N'_a = N_a + \sum_{\text{all second level rules } a \leftarrow aua \text{ with } |u|>0} (K_a - |u|) \cdot \#_{\langle au \rangle}(\alpha'). \tag{24}$$

Using (22) and (20), we can re-write (24) into

$$N'_a = (K_a + 2) \cdot \#_a(\alpha) - (K_a + 1). \tag{25}$$

To make constant term in (25) disappear, we modify the base word w' in G' by appending $K_a + 1$ number of new symbols $\sharp_a$ (again, for notational convenience, we do not rename them). In this case, the total count N_a is now

$$N''_a = (K_a + 2) \cdot \#_a(\alpha). \tag{26}$$

In summary, the length of α' is longer than the length of α, because of the new symbols concerning the active symbols a; i..e,

$$|\alpha'| = |\alpha| + \sum_a (K_a + 1)\#_a(\alpha). \tag{27}$$

Though we can not make the lengths of α and α' be the same, we can, instead, make $|\alpha'|$ be a constant multiple of $|\alpha|$, as follows.

One can imagine that every a in α is "expanded" into K_a+2 many a's in α' so that we have (26). The expansion ratio is not uniform among the a's. To fix this, we take K as the LCM of all the (K_a+2)'s. Now, for every rule in (23), we add additional $K-(K_a+2)$ many $\sharp_{au}$'s. As a result, (26) is accordingly changed to

$$N''_a = (K_a+2)\cdot \#_a(\alpha) + (K-(K_a+2))\cdot(\#_a(\alpha)-1).$$

That is,

$$N''_a = K\cdot \#_a(\alpha) - (K-(K_a+2)).$$

To get rid of the constant term, we can similarly append $K-(K_a+2)$ many $\sharp_a$ to the base word w'. In this case,

$$N''_a = K\cdot \#_a(\alpha). \tag{28}$$

To make $|\alpha'|$ be a constant multiple of $|\alpha|$, we only need to expand every symbol d, that is not an active symbol, into K many d's in the following sense.

- if the d is in the base word w, then in w', we replace the d with $d\underbrace{\sharp_d\cdots\sharp_d}_{K-1}$;
- if the d is in some u with $a\leftarrow aua$ being a second level rule in G and the d appears in the base word w', then in w', we also replace the d with $d\underbrace{\sharp_d\cdots\sharp_d}_{K-1}$;
- if the d is in a rule $d\leftarrow dvd$ of a level l higher than 2 in G, then, according to the construction of G', this rule is now a rule in the $l-1$ level of G'. We change the rule into

$$d\leftarrow d\underbrace{\sharp_d\cdots\sharp_d}_{K-1}vd.$$

Finally, we have $|\alpha'| = K\cdot|\alpha|$.

We shall emphasize that the counts in (18) are still preserved. That is, for a Presburger formula P on the counts of $L(G)$, $P(\#(\alpha))$ iff $P(\#(\alpha'))$. Let L be the counting replacement language specified by $L(G)$ and P, and L' be the counting replacement language specified by $L(G')$ and $P\wedge Q$, where Q is the consistent constraint in the construction of G'. The one-to-one mapping from α to α' shows the following: there is a one-to-one mapping f between L and L' such that, for each $\alpha\in L$, $f(\alpha)$ is with length $K\cdot|\alpha|$. That is, $\lambda_L = K\cdot\lambda_{L'}$. In other words, once the information rate of L' is computed, so is the information rate of L.

The collapsing from G to G' can be continued until G' is simple (i.e., of two levels). The result follows. □

The proof of Lemma 4 is a complex reduction from computing the information rate of a simple counting replacement language to solving a convex minimization problem, which is well-known computable. (It really says that the information rate of a simple counting replacement language is the solution of a convex minimization problem.)

Lemma 4. *The information rate of a simple counting replacement language is computable.*

PROOF. Suppose that L is a simple counting replacement language specified by $L(G)$ and a Presburger formula P on counts of symbols in G, where G is, without loss of generality, of two levels. Similar to the beginning of the proof of Lemma 3, it suffices to consider the case where G contains exactly one base word w.

Recall that, by definition, every symbol a in w appears exactly once in w. We use Γ to denote the set of a's in w such that there is a rule r_a, called an a-rule, in G that is in the form of $a \leftarrow aua$, for some u. Assume that $\Gamma \neq \emptyset$ (otherwise w is the only word generated by G and hence λ_L is computed as 0). The length $|r_a|$ of the rule is the length of au. The vector of symbol counts $\#(r_a)$ is defined as $\#(au)$. There could be multiple a-rules for a given a; we use R_a to denote all of them. The entire set of rules is denoted by $R = \cup_{a\in\Gamma} R_a$. Consider a word $\alpha \in L$. That is, α is generated by G and $P(\#(\alpha))$ holds. Suppose that, in the process of generating the α, each rule $r \in R$ is applied t_r times. Clearly,

$$\#(\alpha) = \#(w) + \sum_{r\in R} t_r \cdot \#(r).$$

We now define a Presburger formula $Q(t_r : r \in R)$ over all the variables t_r with $r \in R$ as $P(\#(w) + \sum_{r\in R} t_r \cdot \#(r))$, noticing that the $\#(w)$ and $\#(r)$'s are all constant vectors. Obviously,

$$\alpha \in L \iff \alpha \in L(G) \wedge Q(t_r : r \in R). \tag{29}$$

The length $N(t_r : r \in R)$ of α can be expressed as a linear polynomial

$$\sum_{r\in R} t_r \cdot |r| + |w| \tag{30}$$

in the t_r's.

By definition,

$$\lambda_L = \lim_{N\to\infty} \frac{\log S_N}{N}, \tag{31}$$

where S_N is the number of words in L with length N. In below, $S_{t_r:r\in R}$ denotes the number of words α in L such that, in generating the α from G, each rule $r \in R$ is applied t_r times. Since G is a replacement system,

$$S_N = \sum_{N(t_r:r\in R)=N} S_{t_r:r\in R}. \tag{32}$$

Observing that the number of vectors $(t_r : r \in R)$ satisfying $N(t_r : r \in R) = N$ is at most N^k where $k = |R|$ is the number of rules in G, we therefore have, from (32),

$$\max_{N(t_r:r\in R)=N} S_{t_r:r\in R} \leq S_N \leq N^k \cdot \max_{N(t_r:r\in R)=N} S_{t_r:r\in R}. \tag{33}$$

By default, the max in a set of numbers is 0 if the set is empty. Using (33), the definition in (31) can be rewritten into:

$$\lambda_L = \lim_{N\to\infty} \max_{N(t_r:r\in R)=N} \frac{\log S_{t_r:r\in R}}{N}. \tag{34}$$

Let $\hat{S}_{t_r:r\in R}$ be the number of words α generated by G where each rule r is applied for t_r times. From (29), we can rewrite (34) as

$$\lambda_L = \lim_{N\to\infty} \max_{Q(t_r:r\in R)\wedge N(t_r:r\in R)=N} \frac{\log \hat{S}_{t_r:r\in R}}{N}. \tag{35}$$

By definition, $\hat{S}_{t_r:r\in R}$ can be computed as follows. Recall that every position of the base word w has a distinct symbol. Consider an $a \in \Gamma$. Each a-rule r_a in R_a will be applied for t_{r_a} times. Then, what is the number $\hat{S}_{t_{r_a}:r_a\in R}$ of words that can be generated from the a appearing in w? We first look at an example. Consider two rules $a \leftarrow aba$ and $a \leftarrow aca$, which fires t_1 and t_2 times, respectively. Starting from a, there are many words that can be generated; e.g., $\underbrace{abab \cdot ab}_{t_1}\underbrace{acac\cdots ac}_{t_2}a$. The total number of such words is $\frac{(t_1+t_2)!}{t_1!t_2!}$. In general, we have a similar form,

$$\hat{S}_{t_{r_a}:r_a\in R} = \frac{(\sum_{r_a\in R} t_{r_a})!}{\prod_{r_a\in R} t_{r_a}!}. \tag{36}$$

Continuing with the previous example, we consider two additional rules $d \leftarrow ded$ and $d \leftarrow dfd$, which fires t_3 and t_4 times, respectively. Starting from ad, what is the total number of words generated? The total number is $\frac{(t_1+t_2)!}{t_1!t_2!} \cdot \frac{(t_3+t_4)!}{t_3!t_4!}$. In general, we also have a similar form

$$\hat{S}_{t_r:r\in R} = \prod_{a\in\Gamma} \hat{S}_{t_{r_a}:r_a\in R}. \tag{37}$$

Consider the term $\frac{\log \hat{S}_{t_r:r\in R}}{N}$ in (35), which, using (37), can now be written in the form of

$$\frac{\log \hat{S}_{t_r:r\in R}}{N} = \sum_{a\in\Gamma} \frac{\log \hat{S}_{t_{r_a}:r_a\in R_a}}{N}. \tag{38}$$

The term $\frac{\log \hat{S}_{t_{r_a}:r_a\in R_a}}{N}$ in (38) is actually, using (36),

$$\frac{\log \hat{S}_{t_{r_a}:r_a\in R_a}}{N} = \frac{\log(\sum_{r_a\in R_a} t_{r_a})!}{N} - \frac{\sum_{r_a\in R_a} \log t_{r_a}!}{N}. \tag{39}$$

Using the Stirling approximation

$$\ln n! = n \ln n - n + O(\ln n) \tag{40}$$

in every factorial in (39), we obtain

$$\frac{\log \hat{S}_{t_{r_a}:r_a \in R}}{N} = \frac{(\sum_{r_a \in R_a} t_{r_a}) \log(\sum_{r_a \in R_a} t_{r_a})}{N} - \frac{\sum_{r_a \in R_a} t_{r_a} \log t_{r_a}}{N} + \epsilon, \tag{41}$$

where

$$\epsilon = \frac{O(\log \sum_{r_a \in R} t_{r_a}) - \sum_{r_a \in R_a} O(\log t_{r_a})}{N}$$

satisfying $\epsilon \to 0$ as $N \to \infty$, since $\sum_{r_a \in R_a} t_{r_a} \leq N$ according to (30) and (35). Therefore, it is safe to remove the ϵ from (41) without affecting the value of λ_L in (35) that uses (38), which is now, using (41),

$$\frac{\log \hat{S}_{t_r:r \in R}}{N} = \sum_{a \in \Gamma} \left(\frac{(\sum_{r_a \in R_a} t_{r_a}) \log(\sum_{r_a \in R_a} t_{r_a})}{N} - \frac{\sum_{r_a \in R_a} t_{r_a} \log t_{r_a}}{N}\right). \tag{42}$$

From now on, we only look at (35) and (42). The N and the t_r's in (42) are specified in (35) as values satisfies $Q(t_r : r \in R) \wedge N(t_r : r \in R) = N$. The semilinear set that is defined by the Presburger formula Q is, by definition, a finite union of linear sets $\hat{Q}_1, \cdots, \hat{Q}_m$, for some m. That is, Q can be expressed as $Q_1 \cup \cdots \cup Q_m$ where each Q_i is a Presburger formula that defines the linear set $\hat{Q}_i$. Let λ_i be the λ_L in (35) when the Q in (35) is replaced with Q_i. Then, since $\lambda_L = \max_i \lambda_i$, it suffices for us to show that each λ_i is computable. Therefore, we assume that the Q in (35) is already a Presburger formula that defines a linear set $\hat{Q}$. In below, we will prove that λ_L in (35) is computable.

Since $\hat{Q}$ is a linear set, there are nonnegative integer variables $s_1, \cdots, s_m$, for some m, and, for each $r \in R$, a linear polynomial (with nonnegative coefficients $c_{1,r}, \cdots, c_{m,r}, c_r$),

$$l_r(\mathbf{s}) = c_{1,r} s_1 + \cdots + c_{m,r} s_m + c_r, \tag{43}$$

where $\mathbf{s}$ denotes the vector $(s_1, \cdots, s_m)$, such that $Q(t_r : r \in R)$ iff there is $\mathbf{s}$ satisfying, for each $r \in R$,

$$t_r = l_r(\mathbf{s}). \tag{44}$$

Therefore, using (44), the equation in (35) can be written as

$$\lambda_L = \lim_{N \to \infty} \max_{N(\mathbf{s}) = N} \frac{\log \hat{S}_{l_r(\mathbf{s}):r \in R}}{N(\mathbf{s})}, \tag{45}$$

where $N(\mathbf{s})$ is the linear polynomial in (30), after replacing each t_r with $l_r(\mathbf{s})$, written

$$N(\mathbf{s}) = d_1 s_1 + \cdots + d_m s_m + d, \tag{46}$$

and $\frac{\log \hat{S}_{l_r(\mathbf{s}):r \in R}}{N}$ is the result of replacing each t_{r_a} in (42) with $l_{r_a}(\mathbf{s})$. More precisely,

$$\lambda_L = \lim_{N \to \infty} \max_{N(\mathbf{s}) = N} \sum_{a \in \Gamma} \left(\frac{l_a(\mathbf{s}) \log l_a(\mathbf{s})}{N(\mathbf{s})} - \frac{\sum_{r_a \in R_a} l_{r_a}(\mathbf{s}) \log l_{r_a}(\mathbf{s})}{N(\mathbf{s})}\right), \tag{47}$$

where $l_a(\mathbf{s})$ defined as

$$l_a(\mathbf{s}) = \sum_{r_a \in R_a} l_{r_a}(\mathbf{s}), \tag{48}$$

is a linear polynomial, written

$$l_a(\mathbf{s}) = c_{1,a}s_1 + \cdots + c_{m,a}s_m + c_a. \tag{49}$$

We can further simplify (47) into

$$\lambda_L = -\lim_{N\to\infty} \min_{N(\mathbf{s})=N} \sum_{a\in\Gamma} \sum_{r_a \in R_a} \frac{l_{r_a}(\mathbf{s})}{N(\mathbf{s})} \log \frac{l_{r_a}(\mathbf{s})}{l_a(\mathbf{s})}. \tag{50}$$

Recall that $N(\mathbf{s})$ is the length of a word α generated from the base word w in G, while each l_{r_a} is the number of times that the a-rule r_a is applied in the generation. Therefore,

$$\sum_{a\in\Gamma} \sum_{r_a \in R_a} l_{r_a}(\mathbf{s}) \le N(\mathbf{s}). \tag{51}$$

Using (43), (49) and (46), we replace $l_{r_a}(\mathbf{s})$, $l_a(\mathbf{s})$, and $N(\mathbf{s})$ in (50) and, after simplification, we have

$$\lambda_L = -\lim_{N\to\infty} \min_{d_1s_1+\cdots+d_ms_m+d=N} \sum_{a\in\Gamma} \sum_{r_a \in R_a}$$

$$\frac{\frac{c_{1,r_a}s_1+\cdots+c_{m,r_a}s_m}{d_1s_1+\cdots+d_ms_m} + \frac{c_{r_a}}{d_1s_1+\cdots+d_ms_m}}{1+\frac{d}{d_1s_1+\cdots+d_ms_m}} \log \frac{\frac{c_{1,r_a}s_1+\cdots+c_{m,r_a}s_m}{d_1s_1+\cdots+d_ms_m} + \frac{c_{r_a}}{d_1s_1+\cdots+d_ms_m}}{\frac{c_{1,a}s_1+\cdots+c_{m,a}s_m}{d_1s_1+\cdots+d_ms_m} + \frac{c_a}{d_1s_1+\cdots+d_ms_m}}. \tag{52}$$

Notice that, in (52), $d_1s_1 + \cdots + d_ms_m + d = N$. So, as $N \to \infty$, all the terms $\frac{c_{r_a}}{d_1s_1+\cdots+d_ms_m}$, $\frac{d}{d_1s_1+\cdots+d_ms_m}$, and $\frac{c_a}{d_1s_1+\cdots+d_ms_m}$ go to 0. Hence, it is safe to replace the terms with 0 in (52) without disturbing the value λ_L. Also, the constraint $d_1s_1 + \cdots + d_ms_m + d = N$ in (52) can be changed to $d_1s_1 + \cdots + d_ms_m = N$ since d is a constant. That is

$$\lambda_L = -\lim_{N\to\infty} \min_{d_1s_1+\cdots+d_ms_m=N} \sum_{a\in\Gamma} \sum_{r_a \in R_a}$$

$$\frac{c_{1,r_a}s_1 + \cdots + c_{m,r_a}s_m}{d_1s_1 + \cdots + d_ms_m} \log \frac{c_{1,r_a}s_1 + \cdots + c_{m,r_a}s_m}{c_{1,a}s_1 + \cdots + c_{m,a}s_m}, \tag{53}$$

where we shall emphasize that all the coefficients c's and d's are nonnegative. In particular, because the inequality in (51) holds for all $\mathbf{s}$, whenever, in (53), the coefficient $d_i = 0$ (for some i), we also have, for each $a \in \Gamma$, that $c_{i,r_a} = 0$ for every $r_a \in R_a$ and hence $c_{i,a} = 0$. Therefore, without loss of generality, we assume that none of the coefficients $d_1, \cdots, d_m$ is 0.

Now, we use a trick to make the N disappear in (53). Define, for each $1 \le i \le m$,

$$\theta_i = \frac{d_i s_i}{N}. \tag{54}$$

Now, the constraint $d_1 s_1 + \cdots + d_m s_m = N$ in (53) is equivalent to

$$\sum_i \theta_i = 1 \text{ and each } \theta_i \geq 0. \tag{55}$$

Under the same constraint, (53) can be represented in θ_i's:

$$\lambda_L = -\lim_{N\to\infty} \min_{\substack{\sum_i \theta_i = 1 \text{ and} \\ \text{each } \theta_i \geq 0}} \sum_{a\in\Gamma} \sum_{r_a \in R_a}$$

$$(\hat{c}_{1,r_a}\theta_1 + \cdots + \hat{c}_{m,r_a}\theta_m) \log \frac{\hat{c}_{1,r_a}\theta_1 + \cdots + \hat{c}_{m,r_a}\theta_m}{\hat{c}_{1,a}\theta_1 + \cdots + \hat{c}_{m,a}\theta_m}, \tag{56}$$

where $\hat{c}_{i,r_a} = \frac{c_{i,r_a}}{d_i}$ and $\hat{c}_{i,a} = \frac{c_{i,a}}{d_i}$ are nonnegative rational constants. Now, the limit in (56) can be removed[1] and, finally, we have the following: λ_L equals the negative of

$$\min H(\theta_1, \cdots, \theta_m), \tag{57}$$

subject to

$$\sum_i \theta_i = 1 \text{ and each } \theta_i \geq 0, \tag{58}$$

where $H(\theta_1, \cdots, \theta_m)$ is the following function

$$\sum_{a\in\Gamma} \sum_{r_a \in R_a} (\hat{c}_{1,r_a}\theta_1 + \cdots + \hat{c}_{m,r_a}\theta_m) \log \frac{\hat{c}_{1,r_a}\theta_1 + \cdots + \hat{c}_{m,r_a}\theta_m}{\hat{c}_{1,a}\theta_1 + \cdots + \hat{c}_{m,a}\theta_m}. \tag{59}$$

The function $H(\theta_1, \cdots, \theta_m)$ defined in (59) satisfying (58) is convex, which can be checked as follows. Let $L_{r_a}(\boldsymbol{\theta}) = \hat{c}_{1,r_a}\theta_1 + \cdots + \hat{c}_{m,r_a}\theta_m$, and $L_a(\boldsymbol{\theta}) = \hat{c}_{1,a}\theta_1 + \cdots + \hat{c}_{m,a}\theta_m$. Therefore, (59) can be written as

$$\sum_{a\in\Gamma} \sum_{r_a \in R_a} L_a(\boldsymbol{\theta}) \mathcal{H}(\frac{L_{r_a}(\boldsymbol{\theta})}{L_a(\boldsymbol{\theta})}), \tag{60}$$

[1] There is a subtle argument in here, since the definition of θ_i uses N. Or more precisely, the θ_i should be $\theta_i(N)$. For each fixed N, the min in (56) is achieved as $H(\theta_1^*(N), \cdots, \theta_m^*(N))$ by $\theta_i^*(N)$. Therefore, the λ_L in (56) is $-\lim_{N\to\infty} H(\theta_1^*(N), \cdots, \theta_m^*(N))$ and clearly, $\lambda_L \leq -H(\theta_1^*, \cdots, \theta_m^*)$, where $H(\theta_1^*, \cdots, \theta_m^*)$ is the min achieved in (57). Actually, from the values $\theta_1^*, \cdots, \theta_m^*$, for every small $\epsilon > 0$, one can find infinitely many N's such that $H(\theta_1^*(N), \cdots, \theta_m^*(N)) \leq H(\theta_1^*, \cdots, \theta_m^*) + \epsilon$. This is because $H(\theta_1, \cdots, \theta_m)$ is continuous in the region defined by (58) and the set

$$\{(\frac{d_1 s_1}{N}, \cdots, \frac{d_m s_m}{N}) : d_1 s_1 + \cdots + d_m s_m = N\}$$

is exactly the (dense) set of all rational points in the region. This already gives $\lambda_L \geq -H(\theta_1^*, \cdots, \theta_m^*) - \epsilon$. Sending ϵ to 0, we have $\lambda_L \geq -H(\theta_1^*, \cdots, \theta_m^*)$. Hence, the limit can be removed, since $\lambda_L = -H(\theta_1^*, \cdots, \theta_m^*)$.

where $\mathcal{H}(x) = x \log x$ is a convex function. (By convention, $0 \log 0 = 0$ and $\frac{0}{0} = 0$. Also notice that, from (48) and (49), when $L_a(\boldsymbol{\theta}) = 0$, $L_{r_a}(\boldsymbol{\theta})$ has to be 0.) It suffices for us to show each $L_a(\boldsymbol{\theta})\mathcal{H}(\frac{L_{r_a}(\boldsymbol{\theta})}{L_a(\boldsymbol{\theta})})$ is convex; i.e., for each $0 < \delta < 1$,

$$\delta L_a(\boldsymbol{\theta}_1)\mathcal{H}(\frac{L_{r_a}(\boldsymbol{\theta}_1)}{L_a(\boldsymbol{\theta}_1)}) + (1-\delta)L_a(\boldsymbol{\theta}_2)\mathcal{H}(\frac{L_{r_a}(\boldsymbol{\theta}_2)}{L_a(\boldsymbol{\theta}_2)})$$

$$\geq L_a(\delta\boldsymbol{\theta}_1 + (1-\delta)\boldsymbol{\theta}_2)\mathcal{H}(\frac{L_{r_a}(\delta\boldsymbol{\theta}_1 + (1-\delta)\boldsymbol{\theta}_2)}{L_a(\delta\boldsymbol{\theta}_1 + (1-\delta)\boldsymbol{\theta}_2)}). \tag{61}$$

Both L_a and L_{r_a} are linear functions. When $L_a(\boldsymbol{\theta}_1)$ or $L_a(\boldsymbol{\theta}_2)$ is zero, the inequality in (61) is obvious. When $L_a(\boldsymbol{\theta}_1) \neq 0$ and $L_a(\boldsymbol{\theta}_2) \neq 0$, the inequality in (61) follows directly using the linearity of L_a and L_{r_a} as well as the following result (derived easily from Jensen inequality): if F is convex and each $c_i > 0$ with $1 \leq i \leq q$, for some q, then

$$\sum_{1 \leq i \leq q} c_i F(z_i) \geq (\sum_{1 \leq i \leq q} c_i)\ F(\frac{\sum_{1 \leq i \leq q} c_i z_i}{\sum_{1 \leq i \leq q} c_i}),$$

where we can take F as $\mathcal{H}$, $q = 2$, $c_1 = \delta L_a(\boldsymbol{\theta}_1)$, $c_2 = (1-\delta)L_a(\boldsymbol{\theta}_2)$, $z_1 = \frac{L_{r_a}(\boldsymbol{\theta}_1)}{L_a(\boldsymbol{\theta}_1)}$ and $z_2 = \frac{L_{r_a}(\boldsymbol{\theta}_2)}{L_a(\boldsymbol{\theta}_2)}$.

Hence, computing λ_L is reduced to the convex minimization problem in (57) subject to linear (in)equalities in (58), which is well-known to be polynomial time solvable, numerically; e.g., using the interior point methods [17]. □

Directly from Lemmas 1, 2, 3, and 4, we have the following main theorem.

Theorem 3. *The information rate of the language accepted by a reversal-bounded deterministic counter machine is computable.*

We currently do not have a precise time complexity of computing $\lambda_{L(M)}$ where M is the reversal-bounded DCM in Theorem 3. However, the lower bound of the complexity is $\Omega(2^m)$, where m is the number of states in M. The reason is as follows. Consider an M with no counters and with $m-1$ nested loops, each with length 2. That is, the replacement system G obtained in Lemma 2 is of m levels. Hence, in Lemma 3, after $m-2$ rounds of collapsing, G becomes a replacement system of two levels, where the length of each basic word is at least 2^{m-2}.

Currently, we are not able to generalize the approach used in the above proof to the case when M is nondeterministic. The reason was already mentioned right after Lemma 1. However, when M is nondeterministic, a word in $L(M)$ may correspond to multiple accepting runs. Therefore, λ_L in the statement of Lemma 1 now satisfies

$$\lambda_L \geq \lambda_{L(M)}, \tag{62}$$

and hence, the information rate computed throughout the proof serves as a computable upper bound of $\lambda_{L(M)}$.

Suppose that M is nondeterministic. Consider the one-to-many mapping from input word $w = a_0 \cdots a_{n-1}$ in M to the "run" w' (also of length n). We use $g(n)$ to denote the maximal number of distinct w' one could possibly obtain. Clearly, when M is highly nondeterministic, the number $g(n)$ should be large. Clearly, we have $g(n) \cdot S_n(L(M)) \geq S_n(L)$. Hence, when

$$\lim_{n\to\infty} \frac{\log g(n)}{n} = 0, \tag{63}$$

we have $\lambda_L \leq \lambda_{L(M)}$. Combining (62), we have $\lambda_{L(M)} = \lambda_L$ and hence it is computable when condition (63) holds.

We say that M is $f(n)$-choice if, during every execution of M over input word of length n, M makes at most $f(n)$ nondeterministic choices. M is *sublinear-choice* if M is $f(n)$-choice for some f satisfying $\lim \frac{f(n)}{n} = 0$. Suppose that M contains K instructions (which is a constant). For an input word $w \in L(M)$ of length n, there are at most $K^{f(n)}$ number of accepting executions that witness the fact $w \in L(M)$. Recalling the definition $g(n)$, we have $g(n) \leq K^{f(n)}$, and therefore, when $\lim \frac{f(n)}{n} = 0$, condition (63) holds. Hence,

Theorem 4. *The information rate of the language accepted by a sublinear-choice reversal-bounded nondeterministic counter machine is computable.*

Kuich and Maurer [16] investigate computation of information rate of tuple languages from pseudolinear tuple grammars that, intuitively, cannot generate more than one copy of a nonterminal symbol. It is worth studying the relationship between the notion of "pseudolinear" and our notion of "sublinear-choice", noticing that our notion essentially limits the nondeterminism in a nondeterministic machine.

Though currently it is open whether the information rate of a reversal-bounded nondeterministic counter machine is computable, we can compute one more upper bound as follows. It is known [3] that if M is a reversal-bounded nondeterministic counter machine, we can effectively construct an equivalent reversal-bounded nondeterministic counter machine M' that runs in dn time for some effectively computable constant d. Without loss of generality, we assume that M does not stay; i.e., on a move of M', it either changes a counter value, or reads an input symbol. Again, we can also assume that each counter in M' makes exactly one reversal and M' accepts with all counters being 0. Let w be an input word of length n accepted by M'. That is, there is an accepting run $s_1 b_1 \cdots s_t b_t$ over w, with $t = dn$. In above, each s_i is a state, and each b_i is either a symbol that M' reads from w, or a counter increment or decrement symbol. Even though M' is a nondeterministic machine, the accepting runs can be accepted by a reversal-bounded deterministic counter machine M''. Hence, the information rate $\lambda_{L(M'')}$ is computable. Notice that the accepting run (of length $2dn$), after dropping all states and counter increment or decrement symbols, becomes the input word w (of length n). Immediately, we have $S_{2dn}(L(M'')) \geq S_n(L(M'))$. Hence, $\lambda_{L(M')} \leq 2d\lambda_{L(M'')}$. Recalling that M and M' are equivalent (i.e., accepting the same language), we have $\lambda_{L(M)} \leq 2d\lambda_{L(M'')}$ and the latter is computable.

Currently, it is unclear whether this upper bound or the one obtained in (62) is better.

A 2-tape NFA M is an NFA with two input tapes. If M is a 2-tape NFA, let $T(M) = \{(x, y) : M \text{ on input } (x, y) \text{ accepts } \}$. At each step, the transition of M is of the form $q : (a, b) \to p$, where $a, b \in \Sigma \cup \{\epsilon\}$ (ϵ is the null symbol). The transition means that M in state q reading a and b on the two tapes enters state p. A deterministic 2-tape NFA is denoted by 2-tape DFA.

If M is a 2-tape DFA, let $R(M) = \{xy : (x, y) \in T(M)\}$ and $S(M) = \{x\bar{y} : (x, y) \in T(M)\}$, where $\bar{y}$ is the reverse of string y. We first show that if M is a 2-tape DFA, the information rate of $R(M)$ is computable. We describe an NFA M' accepting a regular language that simulates the 2-tape DFA M. If M uses transition $q : (a, b) \to p$, then M' in state q reads a and enters state s (a new intermediate state), then reads b' (a marked version of b) in state s and enters state p (and thus the symbols in the second tape of M are marked). Clearly, the number of words of length n accepted by M' is "almost" the number of words in $R(M)$ of length n, where "almost" here refers to a ratio of $n + 1$ (there are at most $(n + 1)$ ways to decode an xy in $R(M)$ back to a pair (x, y) in $T(M)$); i.e.,

$$S_n(R(M)) \leq S_n(L(M')) \leq (n + 1) \cdot S_n(R(M)).$$

This immediately gives $\lambda_{L(M')} = \lambda_{R(M)}$. Similarly, one can obtain $\lambda_{L(M')} = \lambda_{S(M)}$. Since the information rate of the language accepted by the NFA M' is computable, we have

Theorem 5. *For 2-tape DFA M, the information rates of $R(M)$ and $S(M)$ are computable.*

The proof ideas can be generalized. Let M be a 2-tape DFA with reversal-bounded counters. As before, $R(M) = \{xy : (x, y) \in T(M)\}$ and $S(M) = \{x\bar{y} : (x, y) \in T(M)\}$. Clearly, we can construct a deterministic reversal-bounded counter machine M' simulating M by interleaving the symbols in the two tapes as before. Since the information rate of the language accepted by a deterministic reversal-bounded machine is computable (Theorem 3), it follows that the information rate of $R(M)$ as well as $S(M)$ is computable.

The above can further be generalized to k-tape DFA with reversal-bounded counters ($k \geq 2$), where

$$T(M) = \{(x_1, ..., x_k) : M \text{ on input } (x_1, ..., x_k) \text{ accepts}\},$$

and

$$R(M) = \{(x_1^{\text{op}_1} ... x_k^{\text{op}_k}) : (x_1, ..., x_k) \in T(M)\}.$$

where each op_i is either "does nothing" or "reverse x_i" (the choice depends only on R).

From this we can see that there are rather complicated examples of languages for which the information rate is computable. For example, the language $L = \{x\#y\#xy : x, y \in (a + b)^*\}$ has computable information rate, since the set of triples $T = \{(x\#, y\#, xy) : x, y \in (a + b)^*\}$ can be accepted by a 3-tape DFA

(even without reversal-bounded counters). Note that L is not even a context-free language.

As we have shown above, the information rate of the language accepted by a multi-tape DFA is computable. In contrast, for multi-head DFAs, we have the following.

A 2-head DFA is a DFA with two one-way heads. The move of the machine depends on the state and the symbols scanned by the two heads. In a move, the machine changes state and moves each head (independently) at most one cell to the right.

Proposition 6. *The information rate of the language accepted by a 2-head DFA is not computable.*

Proof. The proof idea follows Kaminger [14]. It is known (using the undecidability of the halting problem for Turing machines) that the emptiness problem (i.e., is the language accepted empty?) for 2-head DFAs is undecidable. Given a 2-head DFA M, we modify it to a 2-head DFA M' such that $L(M') = \{xw : x \in L(M), w \in (a+b)^*\}$, where a, b are new symbols. M' simply simulates M on x and when M accepts, M' reads w and accepts. Hence $L(M')$ is empty if and only if $L(M)$ is empty, and moreover, $L(M')$ is infinite (and with rate 1) if and only if $L(M)$ is not empty. Then information rate $\lambda_{L(M')} = 0$ if and only if $L(M')$ is empty, which is undecidable. □

3. Conclusions and discussions

We have shown that the information rate of a language accepted by a reversal-bounded deterministic counter machine is computable. For the nondeterministic case, we have provided computable upper bounds. We also considered the cases when the reversal-bounded NCM is sublinear-choice. For the class of languages accepted by multi-tape DFAs, the information rate is computable as well, as we have shown.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] E. Asarin and A. Degorre. Volume and entropy of regular timed languages: Discretization approach. In *CONCUR*, pages 69–83, 2009.

[3] B. S. Baker and R. V. Book. Reversal-bounded multipushdown machines. *J. Comput. Syst. Sci.*, 8(3):315–332, June 1974.

[4] N. Chomsky and G. A. Miller. Finite state languages. *Information and Control*, 1:91–112, 1958.

[5] C. Cui, Z. Dang, T. R. Fischer, and O. H. Ibarra. Execution information rate for some classes of automata. In *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, volume 7810 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2013.

[6] C. Cui, Z. Dang, T. R. Fischer, and O. H. Ibarra. Similarity in languages and programs. *Theor. Comput. Sci.*, 498:58–75, 2013.

[7] Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 1-3(302):93–121, 2003.

[8] Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *CAV'00: Proceedings of International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 1855, pp. 69–84, Springer, 2000.

[9] Z. Dang, O.H. Ibarra, and Q. Li. Sampling a two-way finite automaton. In *Automata, Universality, Computation*. Springer, 2014.

[10] G. Hansel, D. Perrin, and I. Simon. Compression and entropy. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, volume 577 of *Lecture Notes in Computer Science*, pages 513–528. Springer Berlin Heidelberg, 1992.

[11] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

[12] O. H. Ibarra, Z. Dang, O. Egecioglu, and G. Saxena. Characterizations of Catalytic Membrane Computing Systems. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, volume 2747 of *Lecture Notes in Computer Science*, pages 480–489. Springer, 2003.

[13] Oscar H. Ibarra, Cewei Cui, Zhe Dang, and Thomas R. Fischer. Lossiness of communication channels modeled by transducers. In *Language, Life, Limits - 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014. Proceedings*, volume 8493 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2014.

[14] F. P. Kaminger. The noncomputability of the channel capacity of context-senstitive languages. *Inf. Comput.*, 17(2):175–182, September 1970.

[15] W. Kuich. On the entropy of context-free languages. *Information and Control*, 16(2):173–20, 1970.

[16] W. Kuich and H. Maurer. The structure generating function and entropy of tuple languages. *Information and Control*, 19(3):195–203, 1971.

[17] Y. Nesterov, A. S. Nemirovskii, and Y. Ye. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994.

[18] Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, October 1966.

[19] Gh. Paun. *Membrane Computing, An Introduction.* Springer-Verlag, 2002.

[20] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[21] L. Staiger. The entropy of lukasiewicz-languages. In *Revised Papers from the 5th International Conference on Developments in Language Theory*, DLT '01, pages 155–165, London, UK, UK, 2002. Springer-Verlag.

[22] G. Xie, Z. Dang, and O. H. Ibarra. A solvable class of quadratic Diophantine equations with applications to verification of infinite state systems. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 668–680. Springer, 2003.

[23] L. Yang, C. Cui, Z. Dang, and T. R. Fischer. An information-theoretic complexity metric for labeled graphs. 2011 (in review).

[24] L. Yang, Z. Dang, and T. R. Fischer. Information gain of black-box testing. *Form. Asp. Comput.*, 23(4):513–539, July 2011.