Computability 1 (2012) 1–15
DOI
IOS Press

Lossiness of Communication Channels Modeled by Transducers[*]

Oscar H. Ibarra[†]
Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Cewei Cui
School of Electrical Engineering & Computer Science
Washington State University, Pullman, WA 99164, USA
ccui@eecs.wsu.edu

Zhe Dang
School of Electrical Engineering & Computer Science
Washington State University, Pullman, WA 99164, USA
zdang@eecs.wsu.edu

Thomas R. Fischer
School of Electrical Engineering & Computer Science
Washington State University, Pullman, WA 99164, USA
fischer@eecs.wsu.edu

Abstract. We provide an automata-theoretic approach to analyzing an abstract channel modeled by a transducer and to characterizing its lossy rates. In particular, we look at related decision problems and show the boundaries between the decidable and undecidable cases. We conduct experiments on several channels and use Lempel-Ziv algorithms to estimate lossy rates of these channels.

Keywords: automata, transducers, Shannon information

1. Introduction

Modern digital communications are realized through channels. A communication system is modeled as a sender, a channel, and a receiver. The channel input is generated by the sender as an encoding of source input information. This process is referred to as channel encoding. The channel output is delivered to a receiver for decoding. Traditional analysis of such a system uses probability and random processes to model channel behavior. In the view of automata theory, the channel is a transducer, which is an automaton (not necessarily of finite states) having both input instructions and output instructions. In automata theory, textbook results [13] focus on formal language aspects of the input-output relationship exhibited in a transducer, without formulation of any probabilistic description of a transducer's behavior. It would be interesting to see if automata theory can be used to investigate certain key characteristics in a communication channel. In this paper, we use an automata-theoretic approach in studying the lossy rate of a channel modeled by a transducer.

A communication channel can be noisy. That is, the input symbols during transmission can be dropped or altered, or unwanted symbols added. As a result, the output of the channel may not be uniquely decoded back to the input. We abstract the problem as an automata theory problem: Given a transducer T, determine whether T is L-lossy. (That is, are there distinct words in L that are translated into the same word with T?) In the paper, the problem is

[*]A conference version of this paper is in the proceeding of CiE 2014.

[†]Supported in part by NSF Grants CCF-1143892 and CCF-1117708.

shown decidable for nondeterministic finite state transducers (NFTs) as well as some NFTs augmented with reversal-bounded counters and their variations, while L is a regular language or in a certain class of nonregular languages. On the other hand, the problem is undecidable in general. Indeed, as shown in the paper, the undecidability remains even under a very restricted case: the T is a deterministic finite state transducer (DFT) augmented with the capability of making one turn on its input and the L is the universe. Hence, the decidability/undecidability boundary of the problem is subtle.

We also study the lossy rate of a channel modeled by a transducer. In the paper, we define the lossy rate based on a notion introduced by Shannon [21], which we call information rate. Using this definition, the input lossy rate (the output lossy, defined accordingly in the paper, as well) of the transducer T can be computed through computing the information rates of the input language, the output language, as well the language of input-output pairs, of T, without, as in traditional communication engineering analysis, explicitly introducing a probabilistic or stochastic model. Later in the paper, among other results, we show that the lossy rates are computable for NFT. We also conduct experiments on several channels and use Lempel-Ziv algorithms to estimate lossy rates of these channels.

2. Decision Problems: Decidable and Undecidable Cases

We first recall reversal-bounded nondeterministic counter machines [14] used subsequently in this paper. 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 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. If M is augmented with a pushdown stack, we call it a reversal-bounded NPCM (resp., DPCM in the deterministic case).

Reversal-bounded NCMs and NPCMs have been extensively studied since their introduction in 1978 [14], and many generalizations have been identified, e.g., ones equipped with multiple tapes, with two-way tapes, etc. In particular, reversal-bounded NCMs and NPCMs have found applications in areas like Alur and Dill's [1] time-automata [8, 9], Paun's [19] membrane computing systems [15], and Diophantine equations [26].

Two fundamental results in the theory of reversal-bounded NCMs and NPCMs are the following [14].

Theorem 2.1. *It is decidable to determine, given a reversal-bounded NPCM M, whether $L(M)$ is empty (resp., infinite).*

A two-way reversal-bounded NCM M is *finite-turn* if, for a given nonnegative integer c, M makes at most c turns on its two-way input tape.

Theorem 2.2. *It is decidable to determine, given a finite-turn two-way reversal-bounded NCM M, whether $L(M)$ is empty (resp., infinite).*

We now formalize the problem under study. A *transducer T* is a nondeterministic automaton that accepts pairs of words; i.e., the set of pairs accepted by T is $L(T) \subseteq \Sigma^* \times \Delta^*$. For a pair $(u,w) \in L(T)$, u is an input word and w is an output word. Suppose that L is the language from which an input u is drawn. T is *L-lossy* if there are u, v, w such that $u \neq v \in L$, and, both (u,w) and (v,w) are in $L(T)$. That is, a lossy transducer can translate distinct input words into the same output word. T is *L-lossless* if it is not L-lossy. If $L = \Sigma^*$ (i.e., the set of all finite-length input strings), then we will just use the terms lossless and lossy (omitting Σ^*). We are interested in algorithmic solutions to the problem of deciding whether a transducer is L-lossy:

Given: A transducer T and an input word language L.

Question: Is T L-lossy?

Clearly, like most decision problems in automata theory, the decidability relies on the exact classes of languages and automata to which L and T, respectively, belong.

Consider a nondeterministic finite transducer (NFT) T, which is an NFA with outputs. An instruction of T is of the form $(p,a) \to (q,b)$, where q,p are states, and a,b are in $\Sigma \cup \{\varepsilon\}$. The instruction means that M in state p reads a, outputs b, and enters state q. (Notice that the instruction can be an ε-instruction; i.e., when a or b is the null symbol ε.) As usual, $L(T)$ denotes the set of pairs (u,w) such that T enters an accepting state after it reads the input word u while it outputs w. It is fairly well known that it is decidable to determine, given an NFT T and a regular language accepted by an NFA M, whether T is L-lossy. We will generalize this.

In the results below, "augmented with reversal-bounded counters" will mean "augmented with a finite number of reversal-bounded counters".

Theorem 2.3. *It is decidable to determine, given an NFT T augmented with reversal-bounded counters and a language L accepted by a reversal-bounded NCM M, whether T is L-lossy.*

Proof. We construct a finite-turn two-way (with end markers on the input) reversal-bounded NCM M' to simulate T on $L = L(M)$. The idea is for M' to accept some string w if there are two distinct strings u and v in L such that they are mapped into w by T.

M' has one new 1-reversal counter, C. M', when given input w, makes two sweeps on the input. On the first sweep, M' nondeterministically guesses the symbols comprising some string $u = a_1...a_k$ (but not writing them) and checking that, at the end of the sweep, u is in $L(M)$. Also during the sweep, M' checks that the outputs of T match the symbols in w . Furthermore, M' uses counter C to store a nondeterministically chosen $1 \le i \le k$ (by incrementing the counter) and remembering in its finite control the guessed symbol a_i.

When M and T accept, M' returns to the left end marker and executes the same process as above, but this time guessing the symbols comprising $v = b_1...b_n$. Now, it decrements counter C for every symbol that it guesses. When C becomes zero and the symbol b_i it has guessed is different from a_i and M and T accept, M' accepts w.

Note that the case when u (resp., v) is a proper prefix of v (resp., u) and hence different is taken care of in the above process. Clearly, $L(M')$ is not empty if and only if T is L-lossy. The result follows, since the emptiness problem for finite-turn two-way reversal-bounded NCMs is decidable by Theorem 2.2. □

We can further generalize Theorem 2.3. A two-way reversal-bounded NCM M is *finite-crossing* if for a given nonnegative integer c, M crosses the boundary between any adjacent cells of the input at most c times.

Theorem 2.4. *It is decidable to determine, given an NFT T augmented with reversal-bounded counters and a language L accepted by a two-way finite-crossing reversal-bounded NCM M, whether T is L-lossy.*

Proof. It is known that such an M can effectively be converted to an equivalent reversal-bounded NCM (i.e., one-way) [11]. The result follows from Theorem 2.3. □

A question arises whether Theorem 2.4 still holds when T has a two-way input. We will show that the answer is no, even when T is deterministic and makes only one turn on its input tape: a left-to-right sweep and then a right-to-left sweep (the output is one-way). In the proof, we use the undecidability of the Post Correspondence Problem (PCP).

An instance $I = (u_1,\dots,u_n);(v_1,\dots,v_n)$ of the PCP is a pair of n-tuples of nonnull strings over an alphabet with at least two symbols. A solution to I is a sequence of indices $i_1,i_2,\dots,i_m$ such that $u_{i_1}\dots u_{i_m} = v_{i_1}\dots v_{i_m}$. It is well known that it is undecidable to determine, given a PCP instance I, whether it has a solution. We can define $W(I) = \{x \mid x = u_{i_1}\dots u_{i_m} = v_{i_1}\dots v_{i_m}, m \ge 1, 1 \le i_1,\dots,i_m \le n\}$. Then I has a solution if and only if $W(I) \ne \varnothing$. We shall also refer to a string x in $W(I)$ as a solution to I.

Theorem 2.5. *It is undecidable to determine, given a 1-turn DFT T, whether T is lossy.*

Proof. We first show the undecidability when T is nondeterministic, i.e., a 1-turn NFT.

Let $I = (u_1,\dots,u_n);(v_1,\dots,v_n)$ be an instance of PCP over the alphabet $\{0,1\}$. Let $\Sigma = \{0,1,a,b\}$, and $c_1,...,c_n,\#$ be new symbols. Define a set of tuples:

$S = S_1 \cup S_2$, where

$S_1 = \{(ax, y) \mid x \in (0+1)^+, y = c_{i_1}...c_{i_r}\#x^R, x = u_{i_1}...u_{i_r}, r \geq 1, 1 \leq i_1, ..., i_r \leq n\}$, where R denotes reverse.

$S_2 = \{(bx, y) \mid x \in (0+1)^+, y = c_{i_1}...c_{i_r}\#x^R, x = v_{i_1}...v_{i_r}, r \geq 1, 1 \leq i_1, ..., i_r \leq n\}$

We construct a 1-turn NFT M which accepts a set of transductions $L(T) \subseteq S$ that operates as follows:

Case 1. If the input is ax:
– On the left-to-right sweep of x, M outputs $c_{i_1}...c_{i_r}$ if $x = u_{i_1}...u_{i_r}$ for some $r \geq 1, i_1, ..., i_r$ if they exist. Then on the right-to-left sweep of x, M outputs x^R and accepts.

Case 2. If the input is bx:
– On the left-to-right sweep of x, M outputs $c_{i_1}...c_{i_r}$ if $x = v_{i_1}...v_{i_r}$ for some $r \geq 1, i_1, ..., i_r$ if they exist. Then on the right-to-left sweep of x, M outputs x^R and accepts.

Clearly, if x is a solution to PCP instance I, then two different inputs ax and bx will have the same output. On the other hand, if inputs ax and bz, have the same outputs, then because of the form of the output, $x = z$, and x would be a solution to the PCP instance I. It follows that M is L-lossy if and only if PCP instance I has a solution.

We now modify the construction above to make M a 1-turn DFT M', as follows. In the definitions of S_1 and S_2, ax and bx are now replaced by ax' and bx', respectively, where x' is a 3-track string:

1. The first track contains a string $x \in (0+1)^+$.
2. The second track contains a string $w_1 = c_{i_1}\lambda^{|u_{i_1}|-1}...c_{i_r}\lambda^{|u_{i_r}|-1}$ for some $r \geq 1, 1 \leq i_1, ..., i_r \leq n$, and $|w_1| = |x|$.
3. The third track contains a string $w_2 = c_{i_1}\lambda^{|v_{i_1}|-1}...c_{i_r}\lambda^{|v_{i_r}|-1}$ for some $r \geq 1, 1 \leq i_1, ..., i_r \leq n$, and $|w_2| = |x|$.

Now, $S = S_1 \cup S_2$, where

$S_1 = \{(ax', y) \mid x'$ is a 3-track version of $x \in (0+1)^+, y = c_{i_1}...c_{i_r}\#x'^R, x = u_{i_1}...u_{i_r} r \geq 1, 1 \leq i_1, ..., i_r \leq n\}$

$S_2 = \{(bx', y) \mid x'$ is a 3-track version of $x \in (0+1)^+, y = c_{i_1}...c_{i_r}\#x'^R, x = v_{i_1}...v_{i_r} r \geq 1, 1 \leq i_1, ..., i_r \leq n\}$

Now Σ' is the new input alphabet of M' consisting of 3-track symbols as described above. It is easy to see that track 2 (resp., track 3) of ax' (resp., bx') can be used to guide T to output y deterministically. Then M' is Σ'^*-lossy if and only if PCP instance I has a solution. We omit the details. $\square$

A transducer T is *single-valued* on a language L if for every u in L, there is at most one w such that (u, w) is in $L(T)$. In contrast to Theorem 2.5, it is known that it is decidable, given a finite-crossing two-way NFT M augmented with reversal-bounded counters and a language L accepted by a reversal-bounded NCM, whether T is single-valued on L [12].

A transducer T is *k-lossy* if for any word w, there are at most k words that are mapped by T into w. T is *finite-lossy* if it is k-lossy for some k. A related notion that has been extensively studied in automata theory is the notion of k-valuedness of transducers (see, e.g., [20], for an early reference). We say that a transducer T is *k-valued* if, for every input word u, there are at most k output words w such that $(u, w) \in T$. That is, T cannot have more than k outputs on any input word. T is *finite-valued* on L if it is k-valued for some k. Given an NFT T, we can construct another NFT T' such that $L(T') = \{(w, u) : (u, w) \in L(T)\}$. Clearly, T is lossless (resp., finite-lossy, k-lossy for a given k) if and only if T' is single-valued (resp., finite-valued, k-valued). The converse is also true.

The case when T is finite-lossy (resp., k-lossy for a given k) is interesting. It implies that for some k (resp., for the given k), *every* output word w received has at most k possible choices of decoded input words (no matter how long w is). Hence, this number k can also be used as an indicator on how lossy the transducer is.

It is decidable to determine, given an NFT T, whether it is finite-valued (i.e., it is k-valued for some k) [24]. It is also decidable to determine whether it is k-valued for a given k [12]. Hence, we have:

Theorem 2.6. *It is decidable to determine, given an NFT T and a regular language L, whether T is finite-lossy on L. In the affirmative case, the minimal k_0 such that T is k_0-lossy on L is computable.*

Currently, we do not know if the first part of Theorem 2.6 holds when M is an NFT augmented with an infinite memory (e.g., a reversal-bounded counter). However, we can prove the following.

Theorem 2.7. *It is decidable to determine, given an NFT T augmented with reversal-bounded counters, a language L accepted by a two-way finite-crossing reversal-bounded NCM M, and an integer $k \geq 1$, whether T is k-lossy on L.*

Proof. First consider an NFT T with no reversal-bounded counters and L accepted by an NFA M. Suppose we are given k. Using the idea in the proof of Theorem 2.3, we construct a $(k+1)$-turn two-way NCM M' with $k(k+1)$ 1-reversal counters: C_{ij} for $1 \leq i,j \leq k+1$ with $i \neq j$. M' accepts a word w in $L = L(M)$ if and only if there are $k+1$ distinct words $u_1, \ldots, u_{k+1}$ that T maps into w. M' makes $k+1$ sweeps on w. On sweep i, M' guesses the symbols comprising u_i and simulates T on u_i and checks that u_i maps into w in $L(M)$. It also nondeterministically stores in each counter $C_{i,j}$ $(i \neq j)$, the position of some symbol a_k in u_i and remembers a_k as s_{ij} in the finite control. After the last sweep, M' accepts if $C_{ij} = C_{ji}$ and $s_{ij} \neq s_{ji}$ for all $i \neq j$. Clearly, T is k-lossy if and only if $L(M')$ is empty, which is decidable.

As in Theorems 2.3 and 2.4, the construction above generalizes to when T is augmented with reversal-bounded counters and L is accepted by a two-way finite-crossing reversal-bounded NCM. $\square$

For deterministic pushdown transducers (DPDTs), the following result can be shown:

Theorem 2.8. *It is undecidable to determine, given a 1-reversal DPDT (i.e., the stack makes exactly one reversal: once it pops it can no longer push), whether T is lossless (resp., k-lossy for a given k, finite-lossy).*

Proof. In [25], it was shown that there is a class of linear context-free grammars (which are equivalent to 1-reversal NPDAs) for which every grammar G in the class is either unambiguous or unboundedly ambiguous (i.e., not finitely ambiguous), but determining which of the two G belongs to is undecidable. It follows from Theorem 2.10 that it is undecidable, given a 1-turn DPDT T, whether it is lossless (resp., k-lossy for a given k, finite-lossy). $\square$

For the case when the NPDT's input is bounded, we have:

Theorem 2.9. *It is decidable to determine, given an NPDT T augmented with reversal-bounded counters whose input comes from $x_1^* \cdots x_k^*$ (where $x_1, \ldots, x_k$ are not necessarily distinct words) and a language L accepted by a reversal-bounded NCM M, whether T is L-lossy.*

Proof. The proof is similar to that of Theorem 2.3. The finite-turn two-way reversal-bounded NCM M' now becomes a finite-turn two-way reversal-bounded NPDA M' over $x_1^* \cdots x_k^*$. The result follows since the emptiness problem for these machines is decidable [14]. $\square$

Again, the theorem above generalizes to the case when L is accepted by a finite-crossing two-way reversal-bounded NCM.

Next, we investigate the subtle relationship between ambiguity in automata and lossiness in transducers. Let M be a (one-way) acceptor, e.g., DFA, NFA, DPDA, NPDA, etc. We say that a transducer T is of the same type as M, if when T's output is suppressed, it reduces to an acceptor in the class where M belongs. So a DFT (resp., NFT, DPDT, NPDT, etc,) is of the same type as DFA (resp., NFA, DPDA, NPDA, etc.) We assume that in an acceptor or transducer, an accepting state is a halting (i.e., the device has no move when it enters an accepting state).

Theorem 2.10. *The following statements are equivalent, where M and T are of the same type:*

1. *It is undecidable, given a nondeterministic acceptor M, whether M is unambiguous (resp., k-ambiguous for a given k, or finitely-ambiguous).*
2. *It is undecidable, given a deterministic transducer T, whether T is lossless (resp., k-lossy for a given k, or finite-lossy).*

Proof. First we prove that if (1) is undecidable, then (2) is also undecidable. Let Σ be the set of rules of M (i.e., each rule is represented by a symbol). We construct a deterministic transducer of the same type as M whose input alphabet is Σ. Given a string w in Σ^* (thus $w = r_1 \cdots r_n$, where each r_i is a rule), T deterministically simulates M's computation by reading w symbol-by-symbol and executes rule r_i and outputting the input symbol or ε involved in rule r_i and making sure that w is an accepting sequence of computation. It follows that if M is unambiguous (resp., k-ambiguous for a given k, or finitely- ambiguous), then T is lossless (resp., k-lossy, or finitely-lossy).

Now we show that if (2) is undecidable, then (1) is also undecidable. Suppose T is a deterministic transducer with input and output alphabets Σ and Δ, respectively. We construct a nondeterministic acceptor M with input alphabet Δ. M on input w in Δ^*, guesses a string x in Σ^* symbol-by-symbol (without writing them) and simulates T on x and checks that w is the output of T on input x. M accepts if T accepts. Clearly, since T is deterministic, M is unambiguous (resp., k-ambiguous' for a given k, or finitely- ambiguous) if T is lossless (resp., k-lossy, or finitely-lossy). □

The above result is interesting because it relates the ambiguity question of a *nondeterministic* acceptor to the lossiness question of a *deterministic* transducer of the same type as the acceptor. For example, it is undecidable, given a 1-reversal NPDA (which is equivalent to a linear context free grammar), whether it is unambiguous (resp., k-ambiguous for a given k, unboundedly ambiguous) [25]. Hence, it is also undecidable, given a 1-reversal DPDT (deterministic 1-reversal pushdown transducer), whether it is lossless (resp., k-lossy for a given k, finite-lossy).

Clearly, Theorem 2.10 is not valid if M is deterministic. This is because such an acceptor is always unambiguous. Hence the unambiguity question is trivially decidable (since the acceptor is always unambiguous). However, from Theorem 2.8, the losslessness question for 1-reversal DPDT is undecidable.

Similarly, Theorem 2.10 is not valid if T is nondeterministic. Consider the following example: Let $\mathcal{P}$ be the class of 1-reversal NPDAs M, where M always starts in initial state q_0 and on input ε goes to state q_{01} and q_{02}, and in the next step, the next state from q_{01} or q_{02} are the same. Clearly, any 1-reversal NPDA can be simulated by a machine in $\mathcal{P}$ and, hence, any machine in $\mathcal{P}$ is ambiguous (because, by definition of the class $\mathcal{P}$, any input accepted by the machine has at least two distinct accepting computations). It follows that the unambiguity question for $\mathcal{P}$ is decidable. Now let $\mathcal{T}$ be the class of 1-reversal NPDTs of the type defined in class $\mathcal{P}$. Clearly, any 1-reversal DPDT can be simulated by a transducer in $\mathcal{T}$. Hence, from Theorem 2.8, the losslessness problem for $\mathcal{T}$ is undecidable.

The next result shows that undecidability of losslessness implies undecidability of k-lossiness for any k.

Theorem 2.11. *Let $\mathcal{T}$ be a class of deterministic transducers. Then losslessness for $\mathcal{T}$ is undecidable if and only if k-lossiness for $\mathcal{T}$ is undecidable for any given $k \geq 1$.*

Proof. The "only if" part is obvious. To show the "if" part, let T be a deterministic transducer in the class $\mathcal{T}$ with input and output alphabets Σ and Δ, respectively. Let $k \geq 1$ and $a_1, \ldots, a_k$ be k distinct symbols not in Σ and b be a symbol not in Δ. Construct a deterministic transducer T' in $\mathcal{T}$ which operates as follows. On input $a_i w$ (where $1 \leq i \leq k$, and w in Σ^*), T' reads a_i, outputs b and then simulates T on w. Clearly, T' is k-lossy if and only if T is lossless, and the result follows. □

We now define a form of transducers that are Shannon channels mentioned in the Introduction. Let T be a transducer of any given type. Suppose that (u, w) is in $L(T)$. Thus, on input u, T outputs w. However, if we observe the behavior of T, i.e., we look at exactly the way that we feed T with symbols in u and we observe symbols in w to be sent out, we obtain an observed sequence which is a shuffle of the pair (u, w). For instance, if $u = ABC$ and $w = deffg$, an observed sequence could be *ABdefCfg*. That is, on input A, T runs but emits no output. Then on input B, we have output *def*. Finally, on input C, we have output *fg*. The input distance of the sequence is 3 (the length of *def*), that is the maximal number of output symbols between two consecutive input symbols (B and C).

Formally, define the input distance (resp., output distance) of T on (u,w) to be the maximal number of output (resp., input) symbols between two consecutive input (resp., output) symbols in the shuffled input/output behavior sequence. The input (resp., output) distance of T is the maximal input distance for all (u,w) in $L(T)$. T is k-input Shannon (resp., k-output Shannon) if its input distance (resp., output distance) is at most k. T is finite-input (resp. finite-output) Shannon if it is k-input Shannon (resp., k-output Shannon) for some k.

Theorem 2.12. *The following are decidable, given a reversal-bounded NPCMT T (NPCMT is an NPCM with output):*

1. *Given $k \geq 1$, is T k-input Shannon (resp., k-output Shannon)?*
2. *Is T finite-input Shannon (resp. finite-output Shannon)?*

Proof. Let T be a reversal-bounded NPCMT with input and output alphabets Σ and Δ. Given $k \geq 1$, we construct a reversal-bounded NPCM M which on input ε, accepts if there is a tuple (u,w) in $L(T)$ for which the maximal distance of T on (u,w) is greater than k. This is done by M as follows: M guesses the symbols comprising u (without writing them) and simulates the computation of T on u. During the simulation, M also guesses two positions p_1 and p_2 of two consecutive input symbols in u and counts the number d of symbols T outputs between the two consecutive input symbols. If $d > k$, M continues the simulation and accepts if M accepts after processing u. Thus M does not accept ε if and only if T is k-input Shannon. M on any input different from ε rejects. Thus T is k-input Shannon if and only if $L(M)$ is empty, which is decidable by Theorem 2.1.

To show that it is decidable if T is finite-input Shannon, we construct an NPCM M which accepts the language $L = \{1^k \mid$ there is a tuple (u,w) in $L(T)$ for which the maximal distance of T on (u,w) is greater than or equal to $k\}$. To do this, M on input 1^k operates as describe above, but uses a new counter C to store the number d of symbols T outputs between the two consecutive input symbol and checks, by reading the input 1^k, that $d \geq k$ (by decrementing the counter C). Note that C is 1-reversal. Clearly, T is finite-input Shannon if and only if $L(M) = L$ is finite, which is decidable by Theorem 2.1.

Decidability of k-output Shannon and finite-output Shannon can be shown by similar constructions as above. □

3. Lossy Rates of Transducers

The previous section focuses on the problem of deciding whether a channel modeled as a transducer T is L-lossy for a given input language L. Suppose that T is L-lossy. Without introducing probabilities into T, can we still define a notion that characterizes how lossy T is? Before we proceed further, we first illustrate the intuition behind the definitions.

Consider a pair (u,w) of an input word u and an output word w produced by T. The "information" contained in (u,w) is composed of the information in u and the information in w. However, since u and w are not necessarily independent, there is certain amount of mutual information shared between u and w.

The input lossy rate measures the "number" of inputs to which an average output can be decoded. Intuitively, the input lossy rate, using the classic Venn diagram of Shannon information theory, should be the information contained in the input u, given the output w. Notice that the lengths of the input and the output are in general unbounded and hence, a more scientific measurement would be information rate (in bits per symbol) instead of information (in bits). However, there is a problem. In computing the aforementioned information/mutual information, one usually needs a probability distribution which, unfortunately, the transducer T does not have and which, in practice, would be very hard to obtain.

Without an explicit probabilistic model, can we still define an information rate? There has already been a fundamental notion shown below, proposed by Shannon [21] and later Chomsky and Miller [3], that we have evaluated through experiments over C programs [5, 6, 10, 18, 27], fitting our need for the aforementioned complexity. For a number n, we use $S_n(L)$ to denote the number of words in a language L whose length is n. The *information rate* λ_L of L is defined as $\lambda_L = \lim \frac{\log S_n(L)}{n}$. Where the limit does not exist, we take the upper limit, which always exists for a finite alphabet.

The following result is fundamental.

Theorem 3.1. *The information rate of a regular language L is computable [3].*

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 , yet the notion of information rate has been applied to software testing [5, 23]. However, in such a case, computing the information rate is difficult (sometimes even not computable [16]) in general. Existing results (such as unambiguous context-free languages [17], Lukasiewicz-languages [22], 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. A recent important result, using a complex loop analysis technique, is as follows.

Theorem 3.2. *The information rate of the language accepted by a reversal-bounded DCM is computable [7].*

Note that the case for a reversal-bounded NCM is open.

We now return to our definitions. Assume that T is *length-preserving*. That is, for all $(u,w) \in L(T)$, we have $|u| = |w|$. Example channels modeled by such transducers are binary channels that can alter a bit but never drop one. We now consider $L(T,L) = \{(u,w) : (u,w) \in L(T), u \in L\}$. Recall that the information rate $\lambda_{L(T,L)}$ is the average bit rate (number of bits per symbol) of (the string encoding of) a pair $(u,w) \in L(T,L)$. We use a simple shuffle encoding $[u,w]$ of (u,w); e.g., $[aaa,bbb] = ccc$, where c is a symbol representing the pair (a,b). Hence, the length of $[u,w]$ is the same as $|u|$ (as well as $|w|$). It is not hard to imagine that the bit rate $\lambda_{L(T,L)}$ of $[u,w]$ is "contributed" by the average bit rate λ_L in u and the average bit rate $\lambda_{T(L)}$ in w. Herein, $T(L) = \{w : (u,w) \in L(T,L)\}$. Notice that u and w are not completely independent, since $(u,w) \in L(T,L)$. What is the meaning of the bit rate amount $\lambda_{L(T,L)} - \lambda_{T(L)}$? It characterizes, for $(u,w) \in L(T,L)$, the average bit rate amount in u that is independent of w. Notice that, if T is L-lossless, the amount is simply zero. This is because, in this case, the output w completely decides the input u. Now, we define the *input lossy rate* $\lambda_{\text{in}}(L,T)$ to be $\lambda_{L(T,L)} - \lambda_{T(L)}$. Symmetrically, we define the *output lossy rate* $\lambda_{\text{out}}(L,T)$ to be $\lambda_{L(T,L)} - \lambda_L$. Notice that $\lambda_{\text{in}}(L,T) = \lambda_{\text{out}}(T(L),T^{-1})$, where T^{-1} is the inverse of T. Hence, for theoretical purposes, it suffices for us to consider only the input lossy rate in many cases.

We first consider the case when T is a length-preserving NFT (i.e., without ε-instructions).

Theorem 3.3. *The input and output lossy rates are computable when T is an NFT without ε-instructions and L is a regular language.*

Proof. Notice that L, $T(L)$, and the shuffle encoding of $L(T,L)$ (i.e., the set $\{[u,w] : (u,w) \in L(T,L)\}$) are all regular languages. The result follows from the definitions of $\lambda_{\text{in}}(L,T)$ and $\lambda_{\text{out}}(L,T)$, using Theorem 3.1. $\square$

We now consider a DFT T augmented with reversal-bounded counters. In every instruction of T, if the instruction reads a non-null inout symbol, it will also output a non-null symbol and vice versa. We call such a T *non-null* and obviously it is length-preserving. The following result uses Theorem 3.2.

Theorem 3.4. *The output lossy rate is computable when T is a non-null DFT T augmented with reversal-bounded counters and L is the language accepted by a reversal-bounded DCM.*

Proof. It is an exercise to show that the shuffle encoding of $L(T,L)$ can be accepted by a reversal-bounded DCM. The result follows from the definition of $\lambda_{\text{out}}(L,T)$, using Theorem 3.2. $\square$

We currently do not know if Theorem 3.4 can be generalized to the input lossy rate. This is because in computing the input lossy rate, one needs $\lambda_{T(L)}$, where $T(L)$ can be accepted by a reversal-bounded NCM (instead of a DCM) and hence Theorem 3.2 is not applicable.

Currently, we are not clear on how to generalize the definitions of input and output lossy rates to the case when T is not necessarily length-preserving. The difficulty is that, in this case, T can map a low (resp., high) bit rate input to a high (resp., low) one, even when T is one-to-one. Hence, it is not obvious how information rates used in the definitions can faithfully catch the intuitive meaning of lossy rates. We leave this generalization for future work.

4. Experiments

Automata are a fundamental model for all modern programs. Therefore, using the results presented so far, we would be able to compute the input and output lossy rates for a channel modeled by a transducer, which, in practice, is implemented by a program. However, there are difficulties since, as we shown earlier, there are only limited cases when the lossy rates are computable. In general, when the channel is complex enough, it is treated as a black-box (i.e., its internal implementation is unknown). In the rest of the section, we provide a practical approach that uses Lempel-Ziv compression algorithm to estimate the lossy rate of a black-box channel. In the following, our experiments consist of two parts. In the first part, we will do experiments and estimate the lossy rates of several simple black-box channels. In the second part, experiments will be conducted on complex black-box channels and the corresponding lossy rates also will be estimated. We will explain the meaning of simple and complex in the corresponding subsection.

4.1. Simple black-box channels

In this subsection, our preliminary approach works on simple black-box channels. Herein, we say a black-box channel is simple if the internal structure of the channel is a k-lossy and k'-valued transducer with k and k' being integers. Notice that, our experiments are conducted on black-box channels means that the internal structures of these channels are unknown. These structures, shown in Figure 1, are only used to help readers understand our experimental setting and results, not for doing experiments.

The procedure of our experiments has four steps.

First, for a given (black-box) transducer, a sequence of symbols with length 100,000 is generated using a pseudo-random number generator as an input to the transducer. We use Lempel-Ziv compression algorithm to compress the input sequence and obtain its compression rate.

Second, the input sequence is fed to the transducer, one symbol by one symbol, to generate the output sequence. A nondeterministic choice in the transducer is simulated by a uniform probabilistic choice. Lempel-Ziv compression algorithm is used to compute the compression rate of the output sequence.

Third, while the input sequence is fed to the transducer, we record every input symbol and the corresponding output symbol to obtain an input-output pair. As a result, a sequence of input-output pairs is generated. However, this process may not be length-preserving. Thus, to solve this problem, we use aforementioned shuffle coding to translate every input-output pair into a single symbol so that a new sequence, called transducer sequence, is formed. Consequently, the compression rate of the transducer sequence is computed using Lempel-Ziv.

Fourth, we use the compression rates, in inverse proportion, of the input sequence, the output sequence and the transducer sequence as an approximation to, respectively, the information rates. Directly from their definitions, the input lossy rate and output lossy rate of the transducer are then calculated.

The experimental results are shown below.

Table 1. Input/ouput lossy rates of various transducers

Transducer Type	Input	Output	Transducer	Output Lossy	Input Lossy
(a) lossless and single-valued	0.160	0.160	0.160	0	0
(b) 2-lossy and single-valued	0.296	0.159	0.296	0	0.137
(c) lossless and 2-valued	0.160	0.295	0.295	0.135	0
(d) 2-lossy and 2-valued	0.160	0.160	0.296	0.136	0.136
(e) 3-lossy and single-valued	0.381	0.160	0.381	0	0.221
(f) lossless and 3-valued	0.160	0.382	0.382	0.222	0
(g) 3-lossy and 3-valued	0.239	0.238	0.465	0.226	0.227

In Table 1, the term "Input" (resp., "Output", and "Transducer") represents the information rate (i.e., compression rate) of the input (resp., output and transducer) sequence. The term "Output Lossy" (resp., "Input Lossy") is the output (resp., input) lossy rate of the corresponding transducer. From the previous definitions, it is known that the output (resp., input) lossy rate of a transducer equals the information rate of its transducer sequence minus the information rate of its input (resp., output) sequence. Fig. 1 shows the internal structures of all black-box transducers used in our experiments.

The transducers are implemented in Python. All of the experiments are performed on Ubuntu 12.04 operating system.

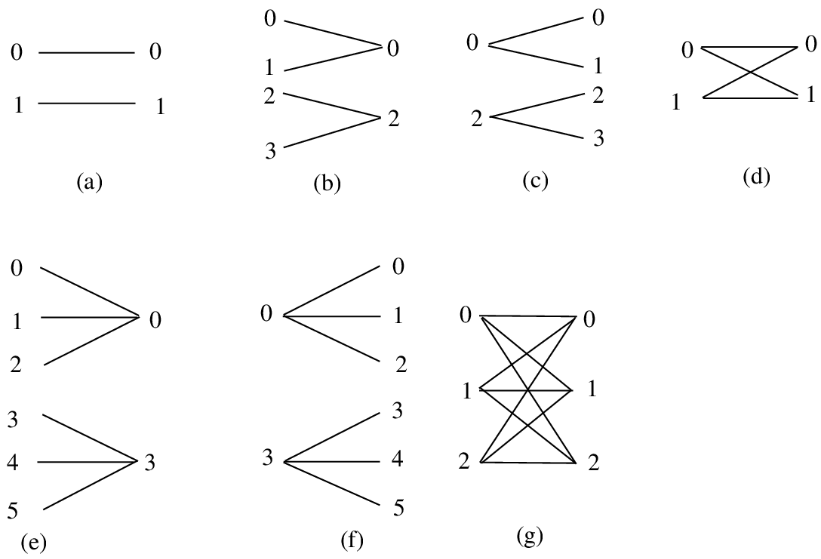

Figure 1. Structures of all transducers used in our experiments: (a) is a lossless and single-valued transducer; (b) is a 2-lossy and single-valued transducer; (c) is a lossless and 2-valued transducer; (d) is a 2-lossy and 2-valued transducer; (e) is 3-lossy and single-valued transducer; (f) is a lossless and 3-valued transducer; and (g) is a 3-lossy and 3-valued transducer.

We summarize our findings from the experimental results as follows.

1. When the output lossy rate of a transducer is 0 (resp., positive), the transducer is likely single-valued (resp., k-valued ($k > 1$)). This finding is consistent with our definition. In a single-valued transducer (e.g., (a), (b) and (e) in Fig. 1), when the input sequence is given, the output sequence is unique. Thus, the information rates of input sequence and the transducer sequence (using shuffle coding) are equal so that the output lossy rate is 0. On the other hand, in a k-valued transducer, it is easy to show that its output lossy rate is positive.

2. When the input lossy rate of a transducer is 0 (resp., positive), the transducer is likely lossless (resp., k-lossy ($k > 1$)). The definition of a lossless transducer in this paper also reflects this fact. In a lossless transducer (e.g., (a), (c) and (f) in Fig. 1), two different input sequences cannot generate the same output sequence through the transducer. Similar to the previous finding, using the shuffle coding, one-to-one mapping can be built between the transducer sequence and the output sequence. Hence, the information rates of the transducer sequence and the output sequence are the same, i.e., the input lossy rate is 0. Similarly, the positive input lossy rate implies a k-lossy ($k > 1$) transducer.

3. In a k-lossy ($k > 1$) and k'-valued ($k' > 1$) transducer (e.g., (d) and (g) in Fig. 1), the information rate of the transducer sequence are greater than the information rate of the input sequence and the information rate of the output sequence. However, the experimental results also show another interesting property in a k-lossy and k'-valued transducer: the information rates of the input sequence and the output sequence are almost equal and also almost half of the information rate of the transducer sequence. After we inspect the source code of the transducers, we find that this phenomenon is related to two reasons. First, k equals k' in our experiments. Second, it is caused by the pseudo-random number generator used in our implementations. By default, the random numbers are uniformly generated by the pseudo-random number generator in our programs. According to the binary symmetric channel (which is a 2-lossy and 2-valued transducer) in [4], when the probability is uniformly distributed, the information

rates of the input sequence and output sequence are almost same and their summation is the information rate of the transducer sequence. Thus, if we adjust the probability assignment in transducers, the input lossy rates and output lossy rates of this type of transducers may change but still stay positive.

4.2. Complex black-box channels

In the previous subsection, we have shown how to use Lempel-Ziv algorithm to estimate the input and output lossy rates of simple black-box channels. However, the channels used in the previous subsection are only ideal channels in textbooks. In real world, it is rare to find such simple channels in computer programs. Thus, in this subsection, we try to conduct experiments on some complex black-box channels, which are modeled by real world computer programs, rather than pure k-lossy and k'-valued transducers.

Before going further, we need to solve one difficulty first. In previous proofs and experiments, inorder to computer the lossy rates of a channel, we always require the channel is a length-preserving channel. However, in practice, this requirement is too hard to achieve for most computer programs. Our previous approaches only works on length-preserving channels. If we encounter non-length-preserving channels, how can we compute the lossy rates of these channels? Actually, we can use a subtle approximation technique to overcome this difficulty.

In this subsection, we will do our experiments on a compiler channel (i.e., a channel modeled by a compiler program). Thus, in the following, we use a compiler channel as an example to explain this approximation approach for non-length-preserving channels. In practice, it is known, for a compiler channel, the length of its input is always less than or equal to the length of its corresponding output. Obviously, it is not length preserving. If we still use previous approaches, we cannot compute its lossy rates. To solve this problem, we convert it into a length preserving channel.

The input of a compiler channel consists of a sequence of input symbols. Meanwhile, in its input side, every input symbol represents exactly one character (i.e. an ASCII character in the original input). Through the channel, in the output side, every input symbol is received as exactly one output symbol. Every output symbol may correspond to one or more characters in the output side. Now, we may raise a question how many characters a output symbol can represent. The answer depends programs. We assume that, for a program, every output symbol represent the same amount of characters in its output side. (Notice that the output symbol has a larger size of alphabet. For example, if a output symbol can represent two characters, then, the size of output symbols' alphabet is four times as that of characters' alphabet). Hence, the number of characters represented by a output symbol is simply decided by the ratio of the length of the compiler's output and the length of the compiler's input. We call this value **compiler ratio** Following these settings and assumptions, the compiler channel is length-preserving in input/output symbols format.

Now, we have a length-preserving channel and a compiler ratio for the channel. How can we compute the lossy rates? The key is how to compute the information rate of the output and the transducer (i.e., input-output pair) since the information rate of the input is as same as previous experiments. We use the output side as an example to explain this. In the previous subsection, we have shown that Lempel-Ziv algorithm works for approximating the information rate of the input/output on a length-preserving channel. Now, we use the same technique to compute the information rate of the output of the compiler channel in character format, rather than in output symbol format. But, under the length-preserving channel assumption, the output consists of output-symbols, not characters. Thus, we need to find a relation between the information rate of the output in character format and the information of the output in output symbol format. Then, we notice that every character-format sequence can be transformed into a output-symbol sequence by one-to-one mapping. When a output-symbol represent m characters, the number of words in the output in characters format with length n_{oc}, (say $S_{n_{oc}}$), equals the number of words in the output in output symbol format with length $n_{os} = \frac{n_{oc}}{m}$, (say $S_{n_{os}}$). Hene, the information rate of the output in character format is as $\frac{1}{m}$ as the information rate of the output in output symbol format. The, we have the following equation,

$$\lim \frac{\log S_{n_{oc}}(L_{oc})}{n_{oc}} = \frac{1}{m} \times \lim \frac{\log S_{n_{os}}(L_{os})}{n_{os}}, \quad (1)$$

where L_{oc} is the set of output words in character format, $S_{n_{oc}}(L_{oc})$ denotes the number of words with length n_{ol}, L_{os} is the set of output words in output symbol format, $S_{n_{os}}(L_{os})$ denotes the number of words in L_{os} with length n_{os}, and m is the compiler ratio, i.e., the number of characters represented by a output-symbol. The way to compute the information rate of transducer (input-output pairs) is almost same. Therefore, we can use (1) and the definitions of lossy rates in the following experiments to estimate the lossy rates of a real world compiler.

Table 2. ARM

Name	Input Lossy	Output Lossy
cat.c	0.3000	0.3837
cp.c	0.3059	0.4106
cut.c	0.3436	0.5366
date.c	0.3257	0.3915
ls.c	0.3080	0.4917
pwd.c	0.4167	0.5725
sort.c	0.3512	0.5240
tail.c	0.3699	0.6490
wc.c	0.3849	0.7422
who.c	0.3217	0.5937

Table 3. MIPS

Name	Input Lossy	Output Lossy
cat.c	0.3124	0.4214
cp.c	0.3235	0.4455
cut.c	0.3515	0.5432
date.c	0.3471	0.4264
ls.c	0.3174	0.5063
pwd.c	0.4522	0.7460
sort.c	0.3649	0.5852
tail.c	0.3786	0.6630
wc.c	0.4086	0.7799
who.c	0.3340	0.6012

Table 4. PowerPC

Name	Input Lossy	Output Lossy
cat.c	0.3757	0.6846
cp.c	0.3386	0.5856
cut.c	0.3334	0.5914
date.c	0.4876	0.7658
ls.c	0.2784	0.5046
pwd.c	0.4369	0.6777
sort.c	0.3197	0.5842
tail.c	0.3365	0.6012
wc.c	0.3090	0.5723
who.c	0.3741	0.6274

Table 5. x86

Name	Input Lossy	Output Lossy
cat.c	0.3937	0.6840
cp.c	0.3554	0.6131
cut.c	0.3373	0.6491
date.c	0.4598	0.7016
ls.c	0.2642	0.4719
pwd.c	0.4078	0.6198
sort.c	0.3241	0.5729
tail.c	0.3424	0.6119
wc.c	0.3253	0.5978
who.c	0.3663	0.6233

Experiment subjects and setting:

`GCC` is one of most commonly used compiler in the world. It is natural to use this compiler as our experiment subject. But, due to our device limitations, we cannot directly compile our programs on different architectures. Instead, we used a tool `crosstool-ng`, which also includes a `GCC` compiler inside, to conduct cross-compiling on a Linux machine so that `crosstool-ng` outputs assembly code that uses different architectures' instruction sets.

The GNU coreutils is a collection of basic tools in Unix-like operating systems. It is also appropriate to use them as the inputs of the compiler. However, this toolset contains more than 100 programs, so that we don't have enough space to show results for all programs. Then, we randomly chose 10 commonly used unix shell commands to present our results.

Experiment procedure:

First, before compilation, each program is compressed by Lempel-Ziv compression algorithm. Using its original length and compression length, the compression rate is gained. Similar to the previous experiments, the information rate of the input is approximated by the compression rate.

Second, we use `crosstool-ng` to compile all programs on four architectures: ARM, MIPS, PowerPC, and x86. It means that, for every `C` program, we have different versions of assembly code outputs on the four platforms.

In the process of compilation, two kinds of outputs are generated. One kind of outputs is the assembly code on the corresponding architecture while the other kind of outputs is the combination of C source code and its corresponding assembly code on the same architecture, i.e. input-output pair in the transducer. These outputs are used, in next step, to compute the information rates of the output side and transducer side, respectively.

Third, after compilation, for each program, we can compress its combined output (i.e., the output in the trandcuer side) and its pure assembly output to get corresponding compression rates. Then, the information rates on the transducer and the output in character format are gained. Because compiler is non-length-preserving, we then compute compilation ratios, for two types of outputs. Using equation (1), the information rates of transducer side and the output side, in output symbol format, are obtained.

Fourth, using definitions of the input and output lossy rates, for each program, its lossy rates are also computed.

Experimental results and findings:

The experimental results are presented in Tables 2–5.

In each table, the term "Name" represents the program being compiled and the program is treated as a black-box channel. The term "Input Lossy" (resp., "Output Lossy) is the input (resp., output) lossy rate of the corresponding channel. Each table's name represents the architecture on which programs are compiled.

We summarized our findings from the experimental results as follows.

1. On each architecture, both input lossy rate and output lossy rate of each program are positive. It indicates that the compiling process is equivalent to a k-lossy ($k > 1$) and k'-valued ($k' > 1$) channel (or transducer). (But, we may not find the integers k and k' to satisfy lossy rates in the tables.) This finding is consistent with our understanding on a compiler. In the compiling process, two distinct symbols may be translated to the same symbol and the same symbol in different locations may be translated to different symbols.

2. As shown in Tables 2–5, on every architecture, for every program, it is obvious that the output lossy rate is greater than the input lossy rate. This finding may surprise ourselves at first. But, inspecting the compiling process and input/output lossy rates' definitions, we found that it is not surprising.

Recalling the definitions of the input and output lossy rates, we can find the reason behind this phenomenon is that the information rate of input side is less than the information rate of the output side. Why does this phenomenon happen? For a compiler, the input is just a `C` program while the output side is an assembly program. It is well known that `C` is a high-level programming language while assembly language is a low-level programming language. To implement the same functionality, a `C` program only focuses on the control flow of its program and its high-level data structures, and many details in low-level (i.e. related to hardware architectures) are ignored, while all high-level control flows, data structures and low-level details are included in a assembly program. Hence, we can say that a `C` program only includes high-level information and a assembly program combines high-level and low-level information together. It also indicates that, for the same functionality, assembly programs programs contain more information than `C` programs.

3. Observing Tables 2–5, we also find that, in Tables 2 and 3, for the same program, the input lossy rates on ARM architecture and MIPS architecture are very close; in Tables 4 and 5, the input lossy rates on PowerPC architecture and x86 architecture are also close. Reviewing all data generated in our experiments, we try to infer the reason behind this phenomenon. For a channel, the input lossy rates are decided by two terms: the information rate of its transducer side and the information rate of its output. Both the transducer side and the output contains information from the output. Then, the amount of information in the output may be a key factor in the input lossy rate. We also know that the output mainly depends on architecture and hence the input lossy rate. It really says the input lossy rate of a channel reflects its architecture characteristics. Both ARM and MIPS use RISC instruction set and are mainly used in mobile devices and embedded systems. Thus, their platform characteristics are similar. For PowerPC and x86, although they use different instruction sets, but they are mainly designed for desktop PCs and server computers, they also have some similar platform characteristics.

Through the above experiments, we demonstrate that lossy rates in a channel is a useful analysis tool in practice. In the previous experiments, the implementation of a compiler is unknown, and its structure is modeled as a black-box channel. We still can use its external behaviors to compute the lossy rates of the channel and gain some knowledge regarding the channel's properties only using its lossy rates.

Acknowledgements

We would like to thank Klaus Wich for pointing out to us that the finite-ambiguity problem for linear context-free grammars was shown undecidable in his PhD thesis, and Eric Wang for comments that improved the presentation of our results.

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

[4] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley-Interscience, second edition, 2006.

[5] C. Cui, Z. Dang, and T. R. Fischer. Bit rate of programs. 2013. Submitted.

[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] C. Cui, Z. Dang, T. R. Fischer, and O. H. Ibarra. Information rate of some classes of non-regular languages: An automata-theoretic approach. In *MFCS'14*. Lecture Notes in Computer Science, volume 8634, Springer, 2014.

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

[9] 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.

[10] Z. Dang, O.H. Ibarra, and Q. Li. Sampling a two-way finite automaton. A book chapter in *Automata, Universality, Computation*, volume 12, *Emergence, Complexity and Computation*, Springer, 2014.

[11] E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22:220–229, 1981.

[12] E. M. Gurari and O. H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Math. Systems Theory*, 16:61–66, 1983.

[13] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, first edition, 1979.

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

[15] 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.

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

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

[18] Q. Li and Z. Dang. Sampling automata and programs. 2013. Submitted.

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

[20] M. P. Schützenberger. Sur les relations rationnelles. *Lecture Notes In Comput. Sci*, 33:209–213, 1975.

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

[22] 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.

[23] E. Wang, C. Cui, Z. Dang, T. R. Fischer, and L. Yang. Zero-knowledge blackbox testing: Where are the faults. *Int'l. J. Foundations of Computer Science*, 25(2): 196-218, 2014.

[24] A. Weber. On the valuedness of finite transducers. *Acta Inf.*, 27(9):749–780, August 1990.

[25] K. Wich. Ambiguity functions of context-free grammars and languages. PhD thesis, 2004.

[26] 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.

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