Theoretical Computer Science 577 (2015) 125–140

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Sampling automata and programs

Qin Li[a,*], Zhe Dang[a,b]

[a] School of Computer, Anhui University of Technology, Ma'anshan, Anhui, China
[b] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA

A R T I C L E   I N F O

Article history:
Received 16 August 2013
Received in revised form 19 February 2015
Accepted 4 March 2015
Available online 9 March 2015
Communicated by J.-F. Raskin

Keywords:
Finite automata
Information rate
Sampling

A B S T R A C T

We study sampling schemes for executions of automata and programs, which can be used, based on Shannon information theory, to measure how much information flows from one variable to another. We show that the information rates of periodically sampled executions of a nondeterministic finite automaton and of a reversal-bounded nondeterministic multicounter machine are computable. Finally, through experiments, we use Lempel–Ziv compression algorithm to approximate information leakage in programs.

1. Introduction

Sampling is a basic technique in Electrical Engineering where values of a continuous signal are recorded at some discrete time points so that part or all of the information of the signal remains. The idea of sampling can be used in an execution of a software system as well. That is, when the system runs, one records a sequence of values of all or some of its state variables; e.g. PC (program counter), variable values, pointer locations, stack frames, I/O, etc. Such a sequence, called a trace, is essentially the result of sampling the system at every step. In reality, it is either difficult or inefficient to sample the system at such a high frequency, considering the fact that the system may run at a speed that is higher than one can effectively sample. Certainly, a sampled trace is "good" if it carries a high amount of the information in the unsampled trace. Therefore, it is a problem to define the amount of information contained in an (un)sampled trace. Theoretically, such a trace is a word, drawn from a language. A fundamental notion has already been defined, namely information rate, due to Shannon [27], to characterize this amount, which has been confirmed at least in coding theory [9]. It measures, asymptotically, the bit rate needed to losslessly encode words in the language.

Automata are a fundamental model for all modern software systems. Therefore, at the fundamental level, we study how to compute the information rates of sampled traces for various automata. For periodic sampling, the information rate is computable for a nondeterministic finite automaton (NFA). However, it is not obvious whether one can find a period such that the sampled traces and unsampled traces share, roughly, the same information rate, since here, the period is a parameter instead of a given constant. We show that the problem is decidable by showing that there are only finitely many regular languages of sampled traces using different periods. However, it is difficult to compute the information rate for sampled traces obtained from an infinite state automaton. This is because such traces are almost always accepted by a nondeterministic infinite state automaton and, unfortunately, we have only limited results concerning information rates for non-regular languages; such results almost always focus on languages accepted by deterministic automata [3,20,29]. We show that the

* Corresponding author.
E-mail address: linuxos2@163.com (Q. Li).

http://dx.doi.org/10.1016/j.tcs.2015.03.012
0304-3975/© 2015 Elsevier B.V. All rights reserved.

information rate of sampled runs (traces that record the control states), using periodic sampling, obtained from a nondeterministic reversal-bounded multicounter machine (reversal-bounded NCM [18]) is computable. This result is quite surprising since the sampled runs are known to be accepted by a reversal-bounded NCM (instead of a reversal-bounded DCM), and it is currently open whether the information rate of the language accepted by a reversal-bounded NCM is computable or not [11]. The proof is quite complex which uses a technique where counters are indirectly simulated. As for aperiodic sampling, as we will point out in the paper, it is generally difficult to have a case where the information rate of sampled traces is computable; we leave this for future studies.

In reality, a control state itself can be complex; e.g. in a UML activity diagram, a control state can be a tuple of (finite state) variables. From the information rate of a trace, we can further define information flow among the variables, which are essential in studying software security [21,22]. Notice that our definition is quite different from traditional definitions [2,5,25] that use static code analysis and/or an explicit probability model to calculate information dependency of variables between two execution points (e.g. initial and final). Our definition is in fact, theoretically, a special case of min-entropy in [28] when the probability distribution is uniform. However, we look at the information rate (entropy per symbol) instead of entropy. This is because we are initially inspired by "long-term information flow" [22] that tries to describe information flow within an unbounded time duration [22,23]. Compared to the traditional definitions, as pointed out in [22], our definition catches asymptotic (instead of "sudden") information flow and dependency. In Section 5, we derive a practical approach that uses the Lempel–Ziv compression algorithm to estimate information rates of sampled traces of Python programs and obtain an approximation on information flow from the traces. Our approach is efficient (since Lempel–Ziv is almost linear time) and effective in many cases, as shown in our experiments. This is quite exciting since previous known approaches in quantitative information flow analysis of programs for software security almost always rely on an explicit probability model or static code analysis, as we have mentioned earlier, while ours does not. (Strictly speaking, our approach still relies on a probability model. However, the model is implicitly embedded inside a run of Lempel–Ziv, which adaptively builds a probabilistic model of the string under compression.) We believe that our approach is new.

In the past, sampling has been used in some forms of continuous-time automata; e.g. hybrid automata and timed automata (see [4,17]), while this paper focuses on sampling classic automata (i.e. with discrete steps) and, in particular, studies the information rate of sampled traces. We have shown [10,12] that, in some cases, information rates for executions of various automata are decidable, in particular for some restricted forms of two-way finite automata. Notice that an execution, being a sequence of instructions, contains both controls states *and* input symbols, along with possible operations on storage devices (counters, stack, etc.), which is different from a run and trace (a sequence of values of state variables) studied in this paper. Lempel–Ziv has been used in our recent paper [13] in estimating the information rate of a C program. In this paper, we employ a new "sample-on-change" scheme in sampling out the interesting events for state variables in the program and, after compression with Lempel–Ziv, we are able to estimate information flow. In the future, we may also study how information flow can help in fault finding [30].

The rest of the paper is organized as follows. In Section 2, we briefly introduce the concept of information flow. In Sections 3 and 4, we show the theoretical results on computing information rate for traces in an NFA and runs in a reversal-bounded NCM, using periodic sampling. We also study problems on computing information flow in an NFA. In Section 5, we present experimental results on using Lempel–Ziv compression over sampled traces of Python programs and show how to estimate information flow in such traces in practice. Section 6 concludes the paper.

2. Information flow in a trace

Finding a relationship among the variables within a (sampled) trace of a software system is not a new problem at all. Traditional approaches (see [7] for a survey) employ static analysis and/or probabilistic models, where Shannon conditional and mutual information can be used. As pointed out in [7], scalability for static analysis and the assigning of probabilities in a probabilistic model are common issues using these approaches. Herein, we propose a new definition, also based on a notion by Shannon, that assumes a simple uniform distribution.

We now go back to the origin of the question: what is dependency? In a concurrent system, it is well-known that two events are independent if they have the freedom of interleaving in any order. This suggests that dependency is a metric that measures how much the space of the (sampled) trace is restricted. What would be a mathematical way to define a metric of the restriction? There has already been a fundamental notion shown below, proposed by Shannon [27] and later used by Chomsky and Miller [8], that, as we believe, serves the metric. Let L be a set of words on a given finite and non-empty alphabet Σ, and $S_n(L)$ be the number of words with length n in L. The *information rate* λ_L of L is defined as

$$\lambda_L = \lim \frac{\log S_n(L)}{n} \tag{1}$$

When the limit does not exist, we take the upper limit, which always exists for a finite alphabet. Also, when $S_n(L) = 0$, $\frac{\log S_n(L)}{n}$ is defined to be 0. Throughout this paper, the logarithm is to base 2.

Before we proceed further, we first explain the intuition behind Shannon's definition. $\frac{\log S_n(L)}{n}$ specifies the average number of bits needed per symbol, i.e. bit rate, if one losslessly compresses a word of length n in L, while the information rate λ_L is simply the asymptotic bit rate. In other words, λ_L is the average amount of information per symbol contained in a word in L.