SUCCINCT REPRESENTATION, RANDOM STRINGS, AND COMPLEXITY CLASSES *

Gary L. Peterson
Department of Computer Science
The University of Rochester
Rochester, New York 14627

TR 69

August, 1980

Abstract

A general paradigm for relating measures of succinctness of representation and complexity theory is presented. The measures are based on the new Private and Blindfold Alternation machines. These measures are used to indicate the inherent information (or "randomness") of a string, but with respect to time and space complexity classes. These measures are then used to show that the existence of strings which are random with respect to one measure but not to another can show the relationship between the corresponding complexity classes. The basic hierarchy theorems given allow different and possibly more powerful approaches to these problems.

* This research was supported in part by NSF grant No. MCS79-02971. This paper is to appear in the proceedings of the 21st FOCS Symposium.

Introduction

"The 'computable' numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another."

A.M. Turing [AT]

So begins Turing's famous paper on the "Entscheidungsproblem." Note that the paper focuses on computable numbers as an easy approach to the problems of computable functions. In a similar way, this paper develops the notion of the complexity of numbers (or strings) with respect to familiar complexity measures. This will in turn lead to a paradigm for complexity theory analogous to Turing's number paradigm for computability theory.

The underlying notion is that of succinctness of representation. That is, how can one represent an object (number, string, computation) more succinctly in one model than another. Succinct representation has been studied in several different forms, e.g., automata, grammars, recursive functions, etc. One can even think of such ordinary measures as formula size to be measures of succinct representation. Most of the results in these areas are based on counting arguments. Such arguments are non-constructive and are sometimes easier than constructive arguments. It is hoped that the theorems of this paper will similarly lead to proofs of very difficult complexity theory problems based on non-constructive arguments.

Algorithmic information theory is a relatively new field devoted to measures of information content, and therefore ways of representing information. Turing's non-computable numbers are called random numbers in algorithmic information theory. However, the inherent time or space complexity contained in an encoding of a string is not measured. Therefore, what is needed are sets of measures of "randomness" of strings which indicate the corresponding time or space complexity. An appropriate set of measures will be defined which do exactly that.

The measures are based on the new Private and Blindfold Alternation machines [P&R]. These machines are able to simulate long calculations with very little space, and therefore contain a way of compressing information. Other more familiar models turn out to be inappropriate to studying complexity-based representation.

Using these new measures, hierarchy theorems are proven. These theorems show that the existence

of certain strings which are random with respect to one measure and not random for another indicates the difference between the two corresponding complexity classes. In particular, some standard time vs. time and space vs. space theorems can be expressed in this paradigm. The theorems indicate possible new ways in which time vs. space and similar complexity questions can be approached.

Random Strings

We begin as everyone else does, with a variation of Berry's paradox [W&R]. Consider the fewest number of English words necessary to represent a number. For example, $10^{1,000,000,000}$ can be represented by (the five words) "ten to the one billion." Clearly some numbers can be represented by a hundred words, while most (an understatement) cannot. Consider then "the smallest number requiring more than a hundred words to describe." This number is not well-defined since it has been described with many fewer than a hundred words. (At the heart of the problem is the lack of formal meaning in English, i.e., what is "the smallest even number greater than two not the sum of two primes"?)

Another problem arises in trying to define a "random" number. Consider a state "pick'em" style lottery. If the winning numbers for one month were:

1000000000

2000000000

3000000000

people would have every right to suspect a "fix." Similarly, the sequence:

3141592653

2718281828

1414213562

would arouse suspicion among the more mathematically inclined. Although the above numbers hardly look random, they have just as much probability of appearing as any other sequence.

The relationship between the two problems is succinctness of representation. The sample sequences are very easy to describe. That is, they have short descriptions. This implies that a truly random number is one whose description in an appropriate system is essentially no shorter than the length of the number. But as Berry's paradox shows, such a descriptive system might have lurking dangers.

Church [AC] was apparently the first to propose an algorithmic definition of a random sequence, the

idea being that an infinite sequence of digits is random if it cannot be effectively generated (by a Turing machine, partial recursive function, etc.). It was some time later that researchers such as Kolmogorov [K1,K2], Solomonoff [S1], Chaitin [C1,C2], and Martin-Löf [ML1] began to formalize algorithmic information theory and the concept of random strings. Zvonkin and Levin [ZL] provide a comprehensive overview of the field; their notations will be used here.

Following Kolmogorov [ZL], we can define a measure of the information content of a string. ("String" and "number" will be used interchangeably.) Let F be a partial recursive function. Then the Kolmogorov complexity of string X with respect to F is defined by:

$$K_F(X) = \begin{cases} \text{min size p such that } F(p) = X \\ \text{infinite otherwise.} \end{cases}$$

A basic result is that there exists an "optimal" Fo such that for all G, X:

$$K_G(X) \leq K_{Fo}(X) + O(1).$$

Hence, for any optimal Fo, Go, and any X:

$$K_{Go}(X) = K_{Fo}(X)$$

to within additive constants.

Any such optimal Fo is therefore adequate to define the information content of a string. Such Fo's are sometimes called "universal" since they are usually taken to be the function of a universal Tm which accepts as inputs encodings of Tm's which generate strings when started on blank tape. It is easy to see that each such K_{Fo} is not computable, else it would be possible to reproduce Berry's paradox, i.e., to define a small machine which generates a string not generated by any small machine.

The usual universal functions do not in any way indicate the time or space complexity of encoding or decoding strings from their representation, and how this might relate to complexity theory. In this paper, a major result is the application of machines related to familiar complexity classes to define information measures representing the time or space complexity of encoding and decoding. The next sections define the machines and relate them to measures of succinctness of representation.

Multiple-person Alternation Machines

The underlying computational models used in this paper are variations of the multiple-person alternation machines (MPA-Tm's) [P&R]. They are direct extensions of the Chandra, Kozen, and Stockmeyer alternation machines (A-Tm's) [CKS] and Reif's two-person private alternation machines [JHR]. Whereas A-Tm's model two-person games of complete information, MPA-Tm's model multiple person games of incomplete information. The addition of incomplete information gives the machines considerably more power for fixed space resources. These machines can therefore represent a long computation very succinctly. This is the tie-in to algorithmic information theory.

Acceptance for an MPA-Tm is intimately tied into the outcome problem for the corresponding game. The outcome problem is to decide if a player (or team of players) have a strategy (sequence of moves) which take the game from the initial position to a winning position, despite the opponent's countermoves. The corresponding elements of the game in the machine are the initial configuration (input tape, start state, etc.), transitions between configurations are moves, and accepting configurations are wins.

The resources (elements of the state tuple, tapes) are divided among the players. Players may share resources with other players or have them private to themselves. The players make their moves based on the current state of the visible resources and a history of past moves, and in turn alter these resources. As in a game of incomplete information, a strategy for a player cannot depend on information not visible to it, even if the information is visible to a team-mate.

The players are divided into two teams, the A team, trying to win, and the B team, trying to avoid losing. It turns out that the B team needs only one player, so if the A team has k players, this number k determines the degree of complexity of the game. Hence an MPA_k-Tm is a k+1 person game with k players on the A team. These players are numbered and denoted by $\exists_1$-player, $\exists_2$-player, ..., $\exists_k$-player, and the other player is the $\forall$-player. The names correspond to their roles of making existential or universal choices, respectively. More formal definitions can be found in [P&R].

The complexity of space and time bounded MPA-Tm's was studied in [P&R]. Unfortunately, for even MPA_2-Tm's with small space, their complexity is the same as ordinary Tm's with no space restrictions, i.e., the r.e. languages. Restrictions had to be placed on the machines. One is to make the A team *hierarchical*, that is, the resources of the $\exists_k$-player is a subset of the resurces of the $\exists_{k-1}$-player, whose resources are a subset of the $\exists_{k-2}$-player's, etc. Such machines are called *Private Alternation* machines, PA-Tm's. A further restriction is to make the A team *blindfolded*, i.e., no player can communicate to a higher numbered player and the $\forall$-player cannot communicate to any of them, except to indicate turn.

These machines are called *Blindfold Alternation* machines, BA-Tm's. The space bounded versions of these machines gave the following interesting hierarchy, where S(n) is a space constructible function at least logarithmic and $k \geq 1$:

$$PA_k\text{-SPACE}(S(n)) = \text{DTIME}(2^{2^{\cdot^{\cdot^{2^{O(S(n))}}}}\}k+1})$$

$$BA_k\text{-SPACE}(S(n)) = \text{NSPACE}(2^{2^{\cdot^{\cdot^{2^{O(S(n))}}}}\}k}).$$

While the full proofs in [P&R] are too lengthy to be given here, a brief look into how this compactness comes about is necessary to understand some of the later developments. One key idea is that a two-person game of incomplete information can simulate counting on 2^n bits using only n bits of storage. Consider yourself playing in the following simple game: your opponent selects a 2^n bit number, a bit at a time, showing you each bit as it is selected. Afterwards you are supposed to send back a bit at a time, that number plus one. You win if your opponent cannot prove that the number you sent back is not the correct one. Can you cheat and send back a different number and still be assured of winning? The answer is no. Your opponent could have *secretly* remembered a bit of the original number and its location (using only n bits) and when you send your number, he/she could check the same bit against the one you send, taking care to check for carry-in. Since you don't know which bit was saved, in order to ensure winning you must give the correct bit in all places. It turns out that just about the sole difference between PA-Tm's and BA-Tm's is that the former allows the ∀-player to ask the ∃-players to count up or down, while the latter forbids such communication to the ∃-players. This method generalizes to more players in a recursive way so that two ∃-players can be forced to count on 2^{2^n} bits using n bits, etc.

The upper bounds on the complexity of PA/BA-Tm's are a type of subset construction. In the simple PA_1-Tm case, this consists of noting that the ∃-player's strategy need depend only on the inferred set of possible configurations of the ∀-player's private information. If the ∀-player has n bits of private information, then there are 2^n possible configurations, and 2^{2^n} sets of possible configurations. These sets, along with the ∃-player's configurations, form a graph which can be constructed and search in time on the order of its size. For the BA_1-Tm case, the graph need not be constructed and can be searched non-deterministically. The result generalizes to more players in a similar way.

One corollary to this result shown in [P&R] is that the fsa versions of these machines become correspondingly more succinct than ordinary fsa's. For example, for each k, there is a language L_k which is accepted by an O(n) size PA_k-fsa but by no nfsa with fewer than $2^{2^{\cdot^{\cdot^{2^n}}}\}k+1}$ states nor any dfsa with fewer than $2^{2^{\cdot^{\cdot^{2^n}}}\}k+2}$ states. This is the beginning of determining the relationship between succinct

representation and complexity classes.

Random Strings and Complexity Classes

Using the Kolmogorov measure, K_{F_α}, it is quite easy to define a random string to be one whose measure is effectively the same as its length. Therefore no small algorithmic process can generate that string. As noted earlier, this measure cannot reflect the time or space complexity of encoding and decoding the information contained in the string. As will be seen in this section, the PA/BA-fsa's do measure the information content of a string, and relate this to some standard complexity classes. The result is a set of measures of "randomness" of a string with respect to time and space.

The standard way of relating a machine and a string is to consider generators rather than acceptors. A machine which generates a string is considered to be a representation of that string. When discussing complexity classes, however, acceptors are the standard model. One can still create an appropriate definition of a string based on acceptors: any such acceptor must accept a unique string, which is the string the machine represents. Hopefully the model allows us to determine the space/time complexity of the machine from its description; something which might require the use of "clocks" or similar obvious resource limiters. A particularly ugly problem is that the string accepted by such a machine is not effectively constructible from the machine description. Consider for example an LBA which accepts only the unary representation of the smallest even number that is not the sum of two primes. Due to such problems, it is impossible in general to construct the string from the machine description alone. An additional problem is that there is no bound on the relative succinctness of the machine over the string. It will be seen later that bounding the power of succinct representation (which incidentally eliminates the problems of undecidability) gives a major link between succinctness and hierarchy theorems.

The only interesting space or time bounded class of machines that does not have the above problems are the finite state machines. (The reader should be beginning to understand the reason for the discussion on BA/PA-fsa's in the last section.) However, any dfsa or dfsg (finite state generator) which represents a single string of length n will have to have n+1 states (or n+2, depending on your model). (Similarly for nfsa/nfsg's.) Therefore, we are in a dilemma; non-finite state time or space bounded machines do not apply well while ordinary finite state machines give a trivial measure.

The answer, of course, is alternating finite state machines. For the PA/BA-fsa case, we are quickly forced to eliminate acceptors. It turns out that the input head can be used as a counter (even if it is only one-way), which causes confusion between classes. This leaves us with generators. Adapting the results from [P&R, Section V], it is straightforward to derive the following theorem.

Theorem. For every language L in DTIME($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k) and string X of length n, there is an efficiently constructible PA_k-fsg which halts (on blank input tape, without output) iff X is in L. Similarly for languages in NSPACE($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k-1) and BA_k-fsg's.

Proof: This is a simple adaption of Theorem 10 in [P&R]. Using n+1 states, one can have a size n counter. The ∀-player can use this to check counts by an ∃-player on n bits. The general process of recursive counting then follows simply. The ∀-player also can use another n+1 states to remember X. The ∀-player can be secretly checking either the ∃-player's counts or the ∃-player's simulation of the input (X) separately. If it had to do both at the same time, it would need $O(n^2)$ states. This fact becomes important later on. ∎

Similarly, one can simulate PA/BA-fsg's by machines in the respective complexity classes.

Theorem. There exists a Tm in DTIME($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k) which on given a PA_k-fsg of size n will output the string it represents (if it exists) and halt. Similarly for NSPACE($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k-1) and BA_k-fsg's.

Proof: Again this is an adaptation of the PA/BA-SPACE(n) upper bounds in [P&R]. A PA_1-fsg of size n has at most n states; therefore the number of configurations of the ∀-player is at most n; therefore the number of sets of possible configurations is at most 2^n. The resulting graph can be searched in time 2^{cn} for some c. The generalizations to more players and BA-fsg's is similar to before. ∎

An important note: In the first theorem, we count the number of *states*, but in the second theorem we count *size*. Such a discrepancy causes unfortunate "n/log n" terms to appear if the usual encoding of machines is allowed. It turns out that in the cases used here, a simple addition to the standard way of representing machines allows us to drop the distinction between states and size where it matters. Consider an n+1 state dfsg which generates a string of length n. Just to write down the *names* of the states takes about O(n log n) symbols. However, if the states are numbered with qo the initial state and qn the final state, only those two states need be explicitly referred to. All the others can be referred to by saying "the next state." Hence when writing down the "δ" transition function, let "." refer to the current state. Thus ".+1" is the next state (in the enumeration), etc. Using this notation, the dfsg can be written down with O(n) symbols. Thus, the first theorem can be changed to mean O(n) *size* PA/BA-fsg's. (Almost all transitions are simple cycles (which take constant symbols) or counters and input checkers (which take O(n) symbols total).) We will never come across in this paper an n state fsg which cannot be represented by O(n) symbols.

A similarity has now been fully established between PA/BA-fsg's and DTIME/NSPACE. We use "$\simeq$" to denote this similarity, and therefore paraphrase the last two theorems very succinctly. (PA_k/BA_k-fsg(n) denotes the classes of machines of size n which generate strings, where n is a parameter much like in "DTIME(n)," etc.)

Theorem. $PA_k\text{-fsg}(n) \simeq \text{DTIME}(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}\}k)$ and $BA_k\text{-fsg}(n) \simeq \text{NSPACE}(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}\}k-1)$.

We briefly return to one of the original problems: determining if a string is "random." In other words, does the string appear to be the result of a natural, unbiased random process, or does it look like it was generated by a small machine. Using the Kolmogorov measure, if X is random then $K_{F_0}(X)$ is roughly |X|. If another partial recursive function is used, we can have a more practical definition of randomness. Let Pk(Bk) be the (computable) partial recursive function which when given a PA_k-fsg (BA_k-fsg) determines the string it represents. Note that by the previous theorem these functions are exactly in $\text{DTIME}(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}\}k)$ and $\text{NSPACE}(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}\}k-1)$. Consider a measure such as K_{P_1}; there exists a string X of length n which can have $K_{P_1}(X)=n$, but the measure $K_{P_2}(X)$ is much less. This means that any process which finds the shorter description of X must take more than exponential time. Thus, while a string might have a short description (and therefore be non-random), it is entirely intractable to encode the string into its shorter description. For all practical purposes (where practical means polynomial time), such an X is random. Therefore, measures such as K_{P_1}, etc., are sufficient for everyday randomness. Unfortunately they are also intractable to compute. This is another form of Berry's paradox: a measure based on exponential time, for instance, is itself exponential time (though computable). Therefore these measures are mainly useful as an indication of the inherent representation abilities of the respective classes.

Given a long string of pseudo-random numbers generated by a standard random number generator, it is quite easy to represent the string very succinctly: the program and the initial value(s) for the iteration. However, finding the particular random number generator and the starting values could be quite time consuming. Such "one-way trap doors" in computation lie at the center of the determinism/non-determinism questions (P=NP?, etc.). These measures as yet do not reflect this property. See Adleman [LA] for a discussion of P=NP?-type problems (prime testing) and their relation to information theory.

Machine Size and Complexity

There are several papers that have dealt with program/machine/grammar size and its effect on the power of the descriptive system. In early works on program size and complexity there are Blum

[MB], DiPaola [RD], Meyer and McCreight [M&M], Meyer [AM], Constable [RC], and Constable and Borodin [C&B]. The problem of complexity of describing numbers was studied by Simon [JS], Schnorr [CS], and many others. There are some parallels here with the straight line program complexity studied by Lynch [NL]. The descriptive power of automata was studied by Meyer and Fischer [M&F], Sakoda and Sipser [S&S], and Sipser [MS]. Formal languages and their automata have also been studied from this approach; see, for example, Schmidt and Szymanski [SSz], Valiant [LV], Ginsberg and Lynch [G&L], and Hartmanis [JH].

Recently, Hartmanis and Baker [H&B,B&H,TB] have studied problems relating machine size and complexity questions such as P=NP? They rely on an observation concerning completeness results and relative succinctness of description. Consider your favorite NP-Complete (poly-time reduction) problem, such as SAT. Note that for any NP language L there is a P machine, call it M_1, which takes any question for L and converts it into an "equivalent" SAT question. If SAT were in P, i.e., P=NP, then there is a P machine for SAT, call it M_2. Combining the two machines will give a P machine which accepts L and has size $O(|M_1| + |M_2|)$. (Actually, the "O" can be dropped for an additive constant, but for later purposes, we prefer to retain it.) Note that $|M_2|$ is a constant independent of L. The main observation is that P=NP iff every NP machine has an equivalent P machine whose size is at most linearly larger. On the other hand, if $P \neq NP$ then there is no equivalent P machine linearly (or even recursively) larger than many NP machines. Hartmanis and Baker then go on to study the implications of this fact in its relationship to possible proper ways to represent P and NP machines, such as clocked, "provable," etc.

Consider SAT again, and imagine trying to find Boolean formulae of certain sizes which are the "hardest" ones to test for satisfiability among strings of their respective lengths. "Hardest" would naturally mean that they require time exponential (hopefully) in their length on any deterministic machine for SAT. However, every such string is accepting in linear time by a host of machines via table look-up. Such machines, however, have greatly increased size in order to hold the tables. Hence it follows that machine size should be incorporated into studies of "hardest" elements of languages. It is also natural to assume that such "hardest" elements might be random or non-random with respect to appropriate complexity based measures. And in turn, this indicates something about the inherent nature of the string, the language, and finally the class.

Another way of looking at hierarchy theorems is to consider that in addition to there being languages in PSPACE not in LOGSPACE, there may be languages which are in LOGSPACE, but can be represented much more succinctly by PSPACE machines. Such languages do not need to be very complex; they could be a simple class such as regular (or finite state, a familiar phrase by now), perhaps even a finite set of

strings all of the same length. However, there is no effective way to tell apart ordinary LOGSPACE and PSPACE which accept finite languages , even if explicit resource bounds are given. (I.e., a polynomial or log of a constant is still constant.)

There are certain ways to represent some classes which do indicate their true complexity even for finite languages. For example, LOGSPACE can be nicely represented by multi-head two-way automata. However, the alternating finite state machines beat them all; their type (BA/PA-Tm's) and number of players clearly indicate their complexity (for those classes they represent).

Still we have to concern ourselves with one problem if we are to look at finite languages. Suppose one has a language whose LOGSPACE representation is 5000 symbols long while its PSPACE representation is only 50 symbols long. The PSPACE machine is clearly much more succinct. However, we are forgetting the constant factors. It could be that any such PSPACE machine has an equivalent LOGSPACE machine which is no more than a hundred times larger. In general, we therefore need to consider an infinite number of languages; so that for every possible constant factor there is a language whose proportion of the two measures is greater than the constant.

Consider the simple case where the above family of finite languages has the property that each language consists only of strings of the same length. It would be tempting to assume that the union of these languages is a PSPACE language not in LOGSPACE. This is totally wrong. The union could be a language anywhere from LOGSPACE to non-r.e. And in the latter case, the sets of strings has presumably just been shown to exist without explicitly constructing them. This interesting point will be discussed much more later on.

Note that all of this requires that each class of interest have appropriate complete members. This is certainly true of the classes P, NP, PSPACE, in addition to the elementary recursive hierarchy of PA/BA-Tm's. But it is not too difficult to see that every well-defined complexity class has such complete members, even if they are trivial ones such as CSL (LBA) membership being complete for PSPACE.

Basic Hierarchy Theorems

The reader should have noted that the previous section dealt solely with acceptors, while generators have been touted as the preferred model of succinctness. This section ties these two models together and allows us to express some basic hierarchy theorems.

Consider a string X of length 2^n generated by some size n PA_1-fsg. (A PA_1-fsg corresponds to exponential time and can therefore easily generate such a string.) Note that there is a simple finite

language which can be used to convert any such output into an input problem for some machine. In the case of X, let the language L_X be the set of $i_n b$, where b is in {0,1} and i_n is the n-bit binary representation of i with i between 0 and 2^n-1, such that b is the i+1st bit of X. Note that L_X consists of 2^n strings of length n+1. Via simulation, L_X is accepted by some O(n) size EXPTIME machine.

It is also possible to reverse the process. That is, given such an L_X for a size n EXPTIME machine, it is possible to build (efficiently) an O(n) size PA_1-fsg which generates the X. The machine runs through all strings of size n+1 in order and simulates the EXPTIME machine on that input. Note that all of the classes for PA/BA-Tm's are linear space or exponential time and higher, and it is possible to do this without increasing the order of complexity. If the simulation responds "yes" to some $i_n b$, then a "b" is output (being at position i+1). Note that both $i_n 1$ and $i_n 0$ are simulated so that the ∃-players (being non-deterministic) do not cheat and say "no" to one, hoping that the ∀-player will believe the answer is "yes" to the other. The above observations clearly generalize to all PA/BA-fsg types and we can now state our main theorems.

Time Hierarchy Theorem. For a given function f(n), if for all constants c>0, there exists an X of size 2^n, such that X is generated by a PA_{k+1}-fsg of size O(n) but by no PA_k-fsg of size c×f(n), then there is a language in $DTIME(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}{}_{k+1})$ not in $DTIME(2^{2^{\cdot^{\cdot^{2^{O(f(n))}}}}}{}_{k})$.

Space Hierarchy Theorem. For a given function f(n), if for all constants c>0, there exists an X of size 2^n such that X is generated by a BA_{k+1}-fsg of size O(n) but by no BA_k-fsg of size c×f(n), then there is a language in $NSPACE(2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}{}_{k+1})$ not in $NSPACE(2^{2^{\cdot^{\cdot^{2^{O(f(n))}}}}}{}_{k})$.

Proofs: Consider the PA_2 vs. PA_1 case. Assume otherwise, that $DTIME(2^{O(f(n))})$ equals $DTIME(2^{2^{O(n)}})$. Then by the completeness approach, for every $DTIME(2^{2^{O(n)}})$ machine, there is an equivalent $DTIME(2^{O(f(n))})$ machine which is at most linearly larger. Thus, for every size n PA_2-fsg, there is an equivalent size c×f(n) size PA_1-fsg (where c is now the constant of the theorem). But this contradicts the hypothesis that there is a PA_2-fsg with no equivalent PA_1-fsg of the given size. Therefore the classes are different. The same argument applies to all PA/BA-fsg classes and their respective time and space classes. ∎

Note that the theorems effectively require an infinite sequence of such X's. These X's (or their respective languages L_X) do not necessarily form a language in one class and not in the other. A simple diagonalization argument allows us to construct such sequences of X's. Consider, for example, trying to construct an X of size 2^n which is generated by an O(n) size PA_2-fsg but by no n log n size PA_1-fsg. Of

the $2^{n \log n}$ such machines, some generate strings which start with 1, while the others start with 0 (if any string is generated at all). Let X begin with the first bit of the fewer of the two groups. At least half of the machines have been eliminated as not generating strings starting with the same bit as X. Of the remaining machines, let X's second bit be the same as the second bit of the fewer of the next two groups of machines, based on their second bit generated, etc. By the (n log n)+1st bit, all machines have been eliminated and X can be padded out to its 2^n bits. A size O(n) PA_2-fsg can simulate the above process and generate X. (A DTIME($2^{2^{O(n)}}$) machine can simulate DTIME($2^{O(n \log n)}$ machines.) Clearly this will work for any sub-exponential size PA_1-fsg. This gives the following standard general theorems (for time and space).

Corollary. DTIME($2^{2^{\cdot^{\cdot^{2^{f(n)}}}}}$}k) is properly contained in DTIME($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k+1) and NSPACE($2^{2^{\cdot^{\cdot^{2^{f(n)}}}}}$}k-1) is properly contained in NSPACE($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k), where f(n) is asymptotically smaller than any exponential.

Hopefully this is not too surprising to any reader. Note that we do not yet distinguish say 2^n and 3^n time or space. A closer look at the completeness idea to see where machine sizes change by additive constants, rather than constant factors, might be helpful.

Note that the above method is *constructive*. However, once any such sequence is proven to exist, we can non-constructively assume its existence without further need to explicitly give it. Still no technique has been found that is explicitly non-constructive, which might be a limit to proving more interesting theorems about time vs. space. Speaking of which, the corresponding hierarchy theorems can now be given as corollaries of the previous theorems.

Theorem. For a given function f(n), if for all c>0, there exists an X of size 2^n such that X is generated by a PA_k-fsg of size O(n) but by no BA_k-fsg of size c×f(n), then there is a language in DTIME($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k) not in NSPACE($2^{2^{\cdot^{\cdot^{2^{O(f(n))}}}}}$}k-1).

Theorem. For a given function f(n), if for all c>0, there exists an X of size 2^n such that X is generated by a BA_{k+1}-fsg of size O(n) but by no PA_k-fsg of size c×f(n), then there is a language in NSPACE($2^{2^{\cdot^{\cdot^{2^{O(n)}}}}}$}k) not in DTIME($2^{2^{\cdot^{\cdot^{2^{O(f(n))}}}}}$}k).

According to popular belief, f(n) is nearly exponential in n in both cases. All that is known for sure is that it is at least linear. Several attempts were made to deduce what possible sequences of X's might work, but the underlying techniques turned out to be constructive and of course failed. It is probably just as hard as the two way dfsa/nfsa succinctness questions [S&S,MS], which themselves are closely related to the DLOG=NLOG? question.

Note that the theorems can be "translated" down into something approaching more familiar questions such as P=PSPACE?

Non-constructive Methods

One of the threads running through this paper has been the recurring principle of non-constructive methods. The Hierarchy Theorems allow, in theory, non-constructive methods to be used to prove complexity results. In fact, non-constructive arguments are favored. Only existence of certain sets of strings each with certain properties need be demonstrated. There is no requirement that the strings form a language in a nice recursive set. Another way of phrasing this is that there is no "uniformity" requirement of any kind. (Karp and Lipton [K&L] have recently derived several results concerning uniformity and P=NP? questions.)

This brings up a perfect analogy to Boolean circuit complexity. By counting arguments (which are non-constructive) it is easy to show that there must be Boolean function of n variables whose circuit complexity is roughly exponential. However, when it is required that the functions be "natural," that is, easily constructed or "uniform," then the lower bounds are only linear, and a small constant factor at that. Hence, non-constructive arguments seem intrinsically more powerful.

The work of Baker, Gill, and Solovay [BGS], Lipton [RL], and DeMillo and Lipton [D&L], all indicate certain limitations, or possible limitations, of current techniques in solving P=NP? type questions. The first shows that diagonalization and other simulation arguments are probably not powerful enough, while the other two point out that our lack of proofs may be due to overly weak logical systems in use. Hence, other avenues of exploration may be more fruitful, especially if the techniques do not relativize and are not restricted to simple proof systems. The non-constructive approaches left open in this paper are a step in these directions.

Other Classes

So far the discussion has focused only on DTIME and NSPACE hierarchies. (Actually the latter is a DSPACE hierarchy also for $k \geq 2$, but the fundamental function is non-deterministic.) This leaves NTIME and DSPACE unrepresented. There are no such hierarchies to be found in [P&R]. In fact, DSPACE has not been represented appropriately at any nontrivial level. NTIME does appear in [P&R], but only for one level. *Markov Alternation* machines (MA-Tm's) are ordinary PA-Tm's (with one A team player) where the $\exists$-player's strategy can depend on *visible* information alone, and not on any history. It was

shown in [P&R] that MA-SPACE(n) is the same as NEXPTIME (non-deterministic exponential time). Hence we would expect that MA-fsg's are similar to NTIME(n), but not quite.

Theorem. For every L in NTIME(n) and input X of size n, there is an efficiently (log-space) constructible MA-fsg of size O(n) which halts iff X is in L. Every size n MA-fsg can be simulated in NTIME(n^2).

Proof: For the first part, the ∃-player has n states, and will guess the sequence of n moves of the NTIME machine for L. A move is the old state, old symbol, new state, new symbol, head motion. The ∀-player uses O(n) private states as a counter to either check the ∃-player's picture of input against X, or to check a position, and every time ∃-player returns to that position checks the previous new symbol against the current old symbol, etc. This proof is also applicable to machines with more than one tape. Note that each player can have O(n) states, for $O(n^2)$ possible machine states but that the description remains size O(n) since each is given separately. The upper bound follows by constructing an NTIME(n^2) machine which simulates MA-fsg's. First it will guess in time O(n) the moves of the ∃-player. Then it simulates in parallel the moves of the ∀-player against that sequences of guesses. Since the ∀-player has no more than n states, the set of possible states is at most n long, and can be updated in time O(n). This gives a total time of $O(n^2)$. Note that the latter simulation could be significantly improved, but it doesn't matter here. ∎

Corollary. MA-fsg($n^{O(1)}$) ≃ NP.

Multi-player MA-Tm's do not form a non-deterministic time hierarchy, so no further levels are known. However, ordinary A-Tm's do allow us to find a similar class for P.

Theorem. A-fsg($n^{O(1)}$) ≃ P.

Proof: The lower bound is a straightforward simulation of a DTIME(n) machine on an input of size n by an A-fsg of size O(n). Similarly the upper bound is much like the previous theorem except that *all* machine states are written down and searched. There are $O(n^2)$ such states (n by n) and the resulting graph can be searched in no more than $O(n^4)$ time. ∎

The next logical step would be to phrase the P=NP? question using A-fsg's and MA-fsg's. Unfortunately, this is not easily do-able. However, using PA_1-fsg's, it is possible to make some weak conjectures about NP containing exponential time problems, etc., but nothing as tight as the previous hierarchy theorems.

Other possible models for finite state versions of these classes come to mind. For example, some

limited form of an A-PDA [LLS] might work out, but only for very low level complexity classes. Vector machines [P&S] might have reasonable fsg versions. A similar hierarchy of complexity for regular expressions with complements [LS] does not appear to relate significantly to this approach.

Clearly, it must be that in any generalization all deterministic/non-deterministic space/time, etc., classes should be represented. And not just the simple hierarchies given here. All "in-between" classes should also be represented. This would undoubtedly require that the PA/BA-Tm basis be either scrapped or thoroughly revised.

Summary

We have shown a direct relationship between complexity theory and succinctness of representation. The measures of succinctness developed are computable and also measure inherent time and space complexity. The major hierarchy theorems are especially designed to allow non-constructive methods to be applied in familiar ways. The possible non-constructive techniques remain to be found, however. The next logical steps are to determine what such methods might be like and to generalize the theorems to more complexity classes.

Bibliography

[LA] L.M. Adleman, "Time, space and randomness," M.I.T./LCS/TM-131, April 1979.

[TB] T.P. Baker, "On 'Provable' analogs of P and NP," *Math. Sys. Theory, 12* (1979), 213–218.

[B&H] T.P. Baker and J. Hartmanis, "Succinctness, verifiability and determinism in representations of polynomial time languages," *Proc. of the 20th Annual IEEE Symp. on Foundations of Computer Science* (1979), 392–396.

[BGS] T.P. Baker, J. Gill and R. Solovay, "Relativizations of the P=NP? question," *Siam J. Comp., 4*, 431–442.

[MB] M. Blum, "On the size of machines," *Info. and Control, 11* (1967), 257–265.

[C1] G.J. Chaitin, "On the length of programs for computing finite binary sequences," *JACM, 13* (1966), 547–569.

[C2] G.J. Chaitin, "On the length of programs for computing finite binary sequences: statistical considerations," *JACM, 16* (1969), 145–159.

[CKS] A. Chandra, D. Kozen, and L. Stockmeyer, "Alternation," IBM Research Report RC 7489, Yorktown Heights, N.Y., January 1978.

[AC] A. Church, "On the concept of random sequence," *BAMS, 46* (1940), 254–260.

[RC] R. Constable, "On the size of programs in subrecursive formalisms," *Proc. of the 2nd Annual ACM Symp. on theory of Computing* (1970), 1–9.

[C&B] R. Constable and A. Borodin, "On the efficiency of programs in subrecursive formalisms," *Proc. of the 11th Annual IEEE Symp. on Switching and Automata Theory* (1970), 60–67.

[D&L] R. DeMillo and R. Lipton, "On the consistency of 'P=NP' and related problems with fragments of arithmetic," *Proc. of the 12th Annual ACM Symp. on theory of Computing* (1980), 45–57.

[RD] R. DiPaola, "Random sets in subrecursive hierarchies," *JACM, 16* (1969), 621–630.

[G&L] S. Ginsberg and N.A. Lynch, "Comparative complexity of grammar forms," *Proc. of the 7th Annual ACM Symp. on theory of Computing* (1975), 153–158.

[JH] J. Hartmanis, "On the succinctness of different representations of languages," *SIAM J. Comp., 9* (1980), 114–120.

[H&B] J. Hartmanis and T.P. Baker, "Relative succinctness of representations of languages and separation of complexity classes," *MFOCS 1979, Lecture Notes in Computer Science 74*, Springer–Verlag, 70–88.

[K&L] R. Karp and R. Lipton, "Some connections between nonuniform and uniform complexity classes," *Proc. of the 12th Annual ACM Symp. on theory of Computing* (1980), 302–309.

[K1] A.N. Kolmogorov, "Three approaches to the quantitative definition of information," *Prob. Info. Transmission, 1* (1965), 1–7.

[K2] A.N. Kolmogorov, "Logical basis for information theory and probability theory," *IEEE Trans. Info. Theory, 14* (1968), 662–664.

[LLS] R. Ladner, R. Lipton, and L. Stockmeyer, "Alternating pushdown automata," *Proc. of the 19th Annual IEEE Symp. on Foundations of Computer Science* (1978), 92–106.

[RL] R. Lipton, "Model theoretic aspects of computational complexity," *Proc. of the 19th Annual IEEE Symp. on Foundations of Computer Science* (1978), 193–200.

[NL] N.A. Lynch, "Straight-line program length as a parameter for complexity measures," *Proc. of the 10th Annual ACM Symp. on theory of Computing* (1978), 150–161.

[ML1] P. Martin-Löf, "The definition of random sequences," *Info. and Control, 9* (1966), 602–619.

[AM] A.R. Meyer, "Program size in restricted programming languages," *Info. and Control, 21* (1972), 382–394.

[M&F] A.R. Meyer and M.J. Fischer, "Economy of description by automata, grammars, and formal systems," *Proc. of the 12th Annual IEEE Symp. on Switching and Automata Theory*

(1971), 188–191.

[M&M] A.R. Meyer and E.M. McCreight, "Computationally complex and pseudo-random zero-one valued functions," *Int. Symp. on Theory of Machines and Computation* (1970), 19–42.

[P&R] G.L. Peterson and J.H. Reif, "Multiple-person alternation," *Proc. of the 20th Annual IEEE Symp. on Foundations of Computer Science* (1979), 349–363.

[P&S] V.R. Pratt and L. Stockmeyer, "A characterization of the power of vector machines," *JCSS, 12* (1976), 198–221.

[JHR] J.H. Reif, "Universal games of incomplete information," *Proc. of the 11 Annual ACM Symp. on theory of Computing* (1979), 288–308.

[S&S] W.J. Sakoda and M. Sipser, "Nondeterminism and the size of two way finite automata," *Proc. of the 10th Annual ACM Symp. on theory of Computing* (1978), 275–286.

[SSz] E.H. Schmidt and T.G. Szymanski, "Succinctness of description of unambiguous context-free languages," *SIAM J. of Computing, 6* (1977), 547–553.

[CS] C.P. Schnorr, "Process complexity and effective random tests," *Proc. of the 4th Annual ACM Symp. on theory of Computing* (1972), 168–176.

[JS] J. Simon, "On feasible numbers," *Proc. of the 9th Annual ACM Symp. on theory of Computing* (1977), 195–207.

[MS] M. Sipser, "Lower bounds on the size of sweeping automata," *Proc. of the 11th Annual ACM Symp. on theory of Computing* (1979), 360–364.

[S1] R.J. Solomonoff, "A formal theory of inductive inference," *Info. and Control, 7* (1964), 1–22.

[LS] L. Stockmeyer, "The complexity of decision problems in automata theory and logic," M.I.T. MAC TR-133 (Ph.D. thesis), July 1974.

[AT] A.M. Turing, "On computable numbers with an application to the Entscheidungsproblem," *Proc. London Math. Soc., 2* (1936), 230–265.

[LV] L.G. Valiant, "A note on the succinctness of description of deterministic languages," *Info. and Control, 32* (1976), 139–145.

[W&R] A.N. Whitehead and B. Russell, *Principia Mathematica, I*, Cambridge University Press, London (1925) 61.

[ZL] A.K. Zvonkin and L.A. Levin, "The complexity of finite objects and the development of the concepts of information theory and randomness by means of the theory of algorithms," *Russian Math. Surveys, 25* (1970), 83–124.